ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ УЧРЕЖДЕНИЕ "ФЕДЕРАЛЬНЫЙ НАУЧНЫЙ ЦЕНТР НАУЧНО-ИССЛЕДОВАТЕЛЬСКИЙ ИНСТИТУТ СИСТЕМНЫХ ИССЛЕДОВАНИЙ РОССИЙСКОЙ АКАДЕМИИ НАУК"

На правах рукописи

Гиацинтов Александр Михайлович

МЕТОДЫ И АЛГОРИТМЫ ВИЗУАЛИЗАЦИИ РАЗНОРОДНЫХ ДАННЫХ В ТРЕНАЖЕРНО-ОБУЧАЮЩИХ СИСТЕМАХ ПРОМЫШЛЕННОГО ПРИМЕНЕНИЯ

05.13.01 - Системный анализ, управление и обработка информации (промышленность)

Диссертация на соискание ученой степени
кандидата технических наук

Научный руководитель -
кандидат технических наук
Мамросенко К.А.

Москва 2016

ОГЛАВЛЕНИЕ

Введение

Актуальность работы. В ряде отраслей (например, в авиационно-космической промышленности, электроэнергетике) в настоящее время ощущается нехватка квалифицированных специалистов, способных профессионально управлять сложными техническими системами. Причем потребности в квалифицированных специалистах по некоторым отраслям, в ближайшие годы будут только расти. Необходимо учитывать, что процент возникновения особых, внештатных, аварийных ситуаций по причине неверных действий персонала составляет до 80%.

Требуется многократное увеличение производительности учебных центров, в том числе за счет внедрения новых методик подготовки, создание более совершенной тренажерной базы.

В промышленности находят применение тренажеры по управлению сложными техническими системами. На Сочинской ТЭС установлен и эксплуатируется компьютерный тренажерно-аналитический комплекс для парогазовой установки 39 МВт, моделирующий работу основного и вспомогательного оборудования ПГУ-39 МВт, алгоритмов защиты, управление с операторских станций [1]. В Санкт-Петербурге организован и успешно действует тренажерный центр по подготовке специалистов по управлению атомными энергетическими установками [2].

Тренировки операторов на реальных установках и в реальных условиях слишком опасны и дороги, а часто и не реализуемы. Альтернативой являются компьютерные тренажерно-обучающие системы (ТОС), которые в максимально возможной степени моделируют реальные комплексы и позволяют тренирующимся приобрести правильные и устойчивые навыки.

Под *тренажерно-обучающей системой* (ТОС) оператора сложной технической системы (СТС) будем понимать техническое средство для подготовки операторов в едином информационном окружении, отвечающее требованиям методик подготовки, создающее условия для получения знаний,

навыков и умений, реализующее модель таких систем и обеспечивающее контроль над действиями обучаемого, а также для исследований.

Обучение операторов включает как теоретическую часть (изучение соответствующих математических моделей, критериев надежности и т.д.), так и практическую, без которой невозможно получение устойчивых навыков управления. Подготовка с применением ТОС позволяет отрабатывать навыки в условиях, которые в реальной эксплуатации могут привести к нештатным ситуациям.

Для повышения эффективности подготовки на ТОС целесообразно использовать методы обучения посредством взаимодействия обучаемого с образовательными ресурсами, индивидуализированного обучения, а также методы, для которых характерно активное взаимодействие между всеми участниками процесса подготовки [3].

Для этапа теоретической подготовки ТОС должна обеспечить следующие возможности:

- получение фундаментальных знаний (для обучаемых с уровнем подготовки от 0 и выше);
- обсуждение с другими обучаемыми (в том числе и зарубежными) проблем, возникающих на этапах подготовки;
- работа с дополнительными источниками информации;
- проведение наблюдений;
- коррекция деятельности обучаемого со стороны преподавателя.

Использование тренажерных систем может давать ряд преимуществ по сравнению с обучением на реальных системах:

- безопасная отработка действий операторов при различных отказах;
- возможность отработки до 90 % выполняемых задач;
- возможность многократного повторения выполнения задач;
- гибкость учебного плана;
- снижение расхода ресурсов реальной системы;
- возможность работы тренажера до 23 часов в сутки;

- экологическая чистота.

В зависимости от объема решаемых задач тренажеры могут быть комплексными, специализированными, процедурными.

Комплексные тренажеры предназначены для совместной подготовки операторов СТС в полном объеме алгоритмов их деятельности или одного оператора, деятельность которого в СТС осуществляется по нескольким специальностям.

Специализированные тренажеры обеспечивают формирование и отработку оператором навыков и умений по одному из видов деятельности либо обучение и подготовку одного оператора в объеме его функциональных обязанностей (при эксплуатации СТС двумя и более операторами).

Процедурные тренажеры обеспечивают формирование и отработку оператором навыков выполнения процедур эксплуатации отдельных систем СТС и выполнения отдельных элементов различных видов деятельности. Данная классификация тренажерных систем обусловлена следующими обстоятельствами:

- стоимостью самого комплексного тренажера СТС и стоимостью эксплуатации тренажера (эта величина весьма велика);
- отработкой отдельных частных процедур управления, задействующей малую часть функционала комплексного тренажера.

Таким образом, отработка отдельных частных процедур на комплексном тренажере, как правило, экономически невыгодна. Это и является предпосылкой к созданию специализированных и процедурных тренажеров.

Одним из важнейших компонентов ТОС является система визуализации, обеспечивающая отображение результатов моделирования внешней среды и объекта управления с помощью устройств отображения информации. Применительно к теоретическому этапу подготовки, система визуализации должна отображать следующие виды учебных материалов: динамические графики развития процессов; диаграммы, гистограммы для анализа массивов данных; графические материалы изучаемых объектов; трехмерные модели объектов, их частей; функциональные схемы взаимодействия отдельных подсистем,

обобщенные схемы работы изучаемой системы в целом; результаты работы моделирующих комплексов, с сохранением управляемости приложения; видеоматериалы реальных объектов.

Следовательно, возникает необходимость создания ТОС, позволяющих объединять разнородную мультимедийную информацию в едином информационном пространстве.

Разработке тренажерных систем посвящены работы Бюшгенса Г.С., Бюшгенса А.Г., Решетникова В.Н., Косарева В.А., Дьякова А.Ф., Семенова Н.А., Шибаева В.М., Кольцова С.Е., Кубланова М.С. и др [4–19].

Известна система автоматизированного обучения, которая может быть использована для комплексного группового и/или индивидуального обучения и переподготовки персонала для эксплуатации и обслуживании сложных технических комплексов в условиях возникновения экстремальных и аварийных ситуациях на морских нефтегазодобывающих платформах (Бирюков Ю.Б., Бондарь Е.М.)[20; 21]. Также известна интерактивная автоматизированная система междисциплинарного обучения, позволяющая в режиме реального времени проводить междисциплинарное групповое обучение и тестирование обучаемых в условиях реальных технико-технологических, экологических и организационно-управленческих процессов, происходящих в сложных природно-технических комплексах (Мартынов В.Г., Владимиров А.И.).

Еще одна система может быть использована для формирования навыков, умения и способностей, необходимых в реальных условиях деятельности при работе с аппаратурой технических информационных систем (Фомченко В.Н., Кошкин В.В.) [22].

Результаты проведенного анализа показывают, что вопросы разработки технологий и методов отображения разнородных мультимедийных материалов в системах подготовки персонала раскрыты не достаточно полно. Следовательно, учитывая все вышеизложенное, задача создания тренажерно-обучающих систем, отвечающих современным требованиям, является актуальной[23], [24].

Цель диссертационной работы: разработка архитектуры автоматизированной системы обучения, методов и алгоритмов визуализации,

трансформации и анализа информации в тренажерно-обучающих системах подготовки персонала сложных промышленных объектов, а также проработка вопросов практической реализации системы на примере курса подготовки по обследованию и диагностике щеточно-контактных аппаратов турбогенераторов ГРЭС.

Задачи исследования: Для достижения поставленной цели решались следующие основные задачи:

- провести анализ современных систем визуализации и методов хромакеинга;
- разработать методы и алгоритмы одновременного воспроизведения видеоматериалов в виртуальном трехмерном окружении, позволяющие отображать параллельно выполняемые процессы, как реальные, так и моделируемые;
- создать методы воспроизведения потоковых мультимедиа материалов в виртуальном трехмерном окружении, обеспечивающие получение и отображение данных с моделирующих комплексов с минимальной задержкой;
- создать метод хромакеинга, обеспечивающий выделение объектов переднего плана в изображении из однородного фона в реальном масштабе времени с использованием вычислительных ресурсов видеокарт;
- разработать архитектуру подсистемы визуализации тренажерно-обучающих систем, применяемую для отображения разнородных данных в едином информационном окружении в реальном масштабе времени;
- разработать архитектуру подсистемы воспроизведения мультимедийных материалов, обеспечивающую отображение видеоматериалов высокой четкости на гранях объектов виртуальной трехмерной сцены;
- разработать модель языковой структуры программируемых сценариев, содержащую информацию о командах динамического языка программирования и командах, используемых для управления функциональностью ТОС.
- проработать вопросы практической реализации системы для подготовки промышленно-производственного персонала.

Методы исследования: поставленные задачи решались с использованием методов математического и имитационного моделирования, визуализации,

трансформации, кластерного, регрессионного анализа данных, методов системного анализа.

Методологическую и теоретическую основу диссертационной работы составили научные труды отечественных и зарубежных авторов в области моделирования сложных технологических процессов, визуализации данных, дистанционного обучения, создания компьютерных тренажерных систем.

Научная новизна:

- установлена взаимосвязь порядка поступления данных и команд в память видеокарты и изменения производительности подсистемы визуализации при отображении видеоматериалов;
- разработан метод, впервые позволивший одновременно воспроизводить несколько разнородных видеоматериалов в разрешении 4K (3840*2160) в режиме реального времени в виртуальной трехмерной сцене;
- предложен метод хромакеинга, который по сравнению с существующими решениями, имеет меньшее число этапов и применим для реализации в реальном времени на мощностях видеокарт, без использования специализированных аппаратных средств.

Основные научные положения, выносимые на защиту:

- архитектура автоматизированной системы обучения, позволяющая осуществлять подготовку персонала в едином виртуальном трехмерном окружении;

- методы и алгоритмы одновременного воспроизведения нескольких видеоматериалов высокой четкости в виртуальной трехмерной сцене, обеспечивающие возможность отображения параллельно выполняемых процессов, как реальных, так и моделируемых;

- метод хромакеинга, позволяющий отделить объекты переднего плана от однородного фона с использованием вычислительных мощностей современных видеокарт.

Теоретическая значимость работы заключается в разработке нового подхода к созданию информационных технологий визуализации больших объемов разнородной информации в тренажерно-обучающих системах промышленного применения, позволившего единовременно использовать

виртуальное трехмерное окружение, разнородную мультимедийную информацию, виртуальный графический образ инструктора.

Практическая значимость диссертационного исследования состоит в том, что разработанные архитектура систем, методы, алгоритмы и модели используются при создании тренажерно-обучающих систем сложных технических комплексов в различных отраслях промышленности, курсов теоретической подготовки промышленно-производственного персонала.

Достоверность результатов основывается на корректном использовании методов системного анализа и математического моделирования, а также на основании данных, полученных в ходе создания мультимедийных курсов по проектированию подсистем связи космических аппаратов, по обследованию и диагностике щеточно-контактных аппаратов турбогенераторов ГРЭС и внедрения результатов исследования в деятельность ООО «ЭФ-КОНТЭЛ».

Реализация и внедрение результатов работы осуществлено в МАИ (МАТИ-РГТУ им. К.Э. Циолковского) при создании мультимедийного курса по проектированию подсистем связи космических аппаратов. Основные практические результаты диссертационной работы использовались ООО «ЭФ-КОНТЭЛ» в 2015-2016 годах при проведении обучения персонала предприятий электроэнергетики по обследованию и диагностике щеточно-контактных аппаратов турбогенераторов ГРЭС.

Апробация результатов работы: результаты демонстрировались на авиасалоне МАКС 2013, на выставке "Телевидение высокой четкости России", заседании секции Информационных технологий НП «Научно-технический совет Единой энергетической системы» в 2015 году.

Результаты исследований докладывались на следующих конференциях:

- Международная конференция «Гагаринские чтения» в 2008, 2009, 2010, 2011, 2012, 2013 годах, Москва;
- Международная конференция «Научные исследования по проблемам открытого и дистанционного образования», проводимая Комитетом министров образования стран АСЕАН в 2012, 2015 годах, Ханой, Вьетнам;
- VI Всероссийская научно-практическая конференция «Компьютерная интеграция производства и ИПИ-технологии», 2013 год, Оренбург;

- VII Международная конференция «Авиационные тренажеры и учебные центры 2015»;

В 2007 году коллективу, в котором работает автор, присуждена премия «Кристалл знаний», учрежденная Комитетом министров образования стран АСЕАН, в 2009 году автору присуждена золотая медаль Российской академии наук за лучшую научную работу 2008 года молодых ученых России.

Публикации основных результатов: по теме диссертации опубликовано 11 научных печатных работ, в том числе 7 в журналах «Программные продукты и системы», «Информационные ресурсы России», «Энергетик», «Программная инженерия», рекомендованных ВАК для публикации основных научных результатов диссертаций.

Структура работы: Диссертационная работа состоит из введения, 5 глав, заключения, библиографического списка из 117 наименований, содержит 20 рисунков и 3 таблицы.

В главе 1 производится обзор современных технологий визуализации в реальном времени, а также анализ их возможностей и применимости в тренажерно-обучающих системах (ТОС). В пункте 1.1. производится обзор лидирующих технологий визуализации в реальном времени – Unreal Engine (пункт 1.1.1), CryEngine (пункт 1.1.2), Source (пункт 1.1.3). В пункте 1.2. производится обзор современных методов рир-проекции в реальном времени. В пункте 1.3. производится анализ приведенных технологий визуализации и их применимость в ТОС. В пункте 1.4. проводится анализ методов рир-проекции и возможность применения в подсистеме визуализации ТОС.

В главе 2 приводится разработанная автором информационная архитектура автоматизированной системы обучения. Приводятся требования к архитектурам информационных систем. Описаны разработанные компоненты архитектуры и их взаимосвязи.

В главе 3 приводится предлагаемая архитектура подсистемы визуализации. В пункте 3.1. приводятся требования, предъявляемые к подсистеме визуализации тренажерно-обучающих систем. В пункте 3.2. описывается архитектура базового компонента Horde3D, лежащего в основе архитектуры подсистемы визуализации, приводятся его характеристики, достоинства и недостатки. В пункте 3.3. описываются разработанные методы анимации виртуальной камеры, а также формат хранения анимационных данных камеры. В пункте 3.4. предлагаются методы отображения изображений с частичной прозрачностью в подсистеме визуализации, и приводятся необходимые настройки подсистемы визуализации и шейдерных программ.

В главе 4 описывается разработанная архитектура воспроизведения мультимедийных материалов в трехмерном виртуальном окружении. В пункте 4.1. приводятся методы и алгоритмы, позволяющие одновременно отображать несколько видеоматериалов высокой четкости на гранях объектов виртуальной трехмерной сцены из файлов. В пункте 4.2. приводятся методы и алгоритмы, обеспечивающие воспроизведение потоковых видеоматериалов высокой четкости, поступающих с внешних устройств, таких как видеокамеры или моделирующие комплексы. В пункте 4.3. приведена разработанная архитектура подсистемы хромакеинга, обеспечивающая обработку поступающих изображений с задействованием мощностей графической карты.

В главе 5 приведены методы взаимодействия пользователя с виртуальным окружением с помощью программируемых сценариев. В пункте 5.1. произведено описание существующих технологий программируемых сценариев и применяемых для этого динамических языков программирования. В пункте 5.2. представлена разработанная модель языковой структуры программируемых сценариев, описаны виды хранения информации в модели, обрабатываемые команды и варианты взаимодействия пользователя с моделью языковой структуры. В пункте 5.3. представлена практическая реализация методов и алгоритмов на примере теоретического электронного курса по обследованию и

диагностике щеточно-контактных аппаратов турбогенераторов ГРЭС. Библиографический список содержит 117 наименований.

Глава 1 . Современное состояние исследований в области технологий визуализации данных

Проблеме визуализации данных в реальном масштабе времени посвящено множество работ, как отечественных авторов, так и зарубежных. В частности, можно отметить работы Бобкова А.Е. [25–28], Игнатенко А.В. [29–32], Jorge Jimenez [33–35]. В [36] описано применение методов визуализации в медицинских целях. В [37] приведены методы визуализации данных о мировом океане.

Применительно к тренажерно-обучающим системам, система визуализации - это тренажерный имитатор, воспроизводящий визуальную обстановку, соответствующую реальной [38]. Для осуществления такой имитации система должна обеспечивать визуализацию высоко-реалистичных виртуальных сцен большой сложности в реальном режиме времени.

Анализ применяемых в РФ видов тренажерно-обучающих систем проведен в [39] и [40; 41]. Совершенствованию тренажерно-обучающих систем и систем визуализации посвящены работы Решетникова В.Н., Мамросенко К.А. [13; 15; 42–47], Манакова Д.В. [48–50].

Известна система автоматизированного обучения, которая может быть использована для комплексного группового и/или индивидуального обучения и переподготовки персонала для эксплуатации и обслуживании сложных технических комплексов в условиях возникновения экстремальных и аварийных ситуациях на морских нефтегазодобывающих платформах (Бирюков Ю.Б., Бондарь Е.М.)[20; 21]. Также известна интерактивная автоматизированная система междисциплинарного обучения, позволяющая в режиме реального времени проводить междисциплинарное групповое обучение и тестирование обучаемых в условиях реальных технико-технологических, экологических и организационно-управленческих процессов, происходящих в сложных природно-технических комплексах, каковыми, в том числе, являются скважины, сооружаемые без непосредственного доступа человека к объекту воздействия (забою, горным породам, стволу скважины, продуктивному пласту). (Мартынов В.Г., Владимиров А.И.).

Также известна автоматизированная система обучения, обеспечивающая решение задач обучения принципам действия и работе со сложными устройствами и системами автоматики. Основным результатом ее использования является расширение возможностей и повышение эффективности обучения за счет обеспечения детального изучения основ функционирования и структуры построения различных автоматических устройств и систем управления технологическим процессом (Фомин В.И., Федоров А.В.) [51].

Результаты проведенного анализа показывают, что вопросы разработки технологий и методов отображения разнородных мультимедийных учебных материалов в системах подготовки персонала раскрыты не достаточно полно.

Основной частью системы визуализации является графический движок (англ. *graphics engine*) или графический модуль — программное обеспечение (англ. *middleware*), основной задачей которого является визуализация (рендеринг) двухмерной или трёхмерной компьютерной графики [52]. Может существовать как отдельный продукт или в составе информационной системы. Может использоваться для визуализации статических изображений или видеоряда. Графические движки, использующееся в программах по работе с компьютерной графикой (таких, как 3ds Max, Maya, Cinema 4D, Zbrush, Blender), обычно называются «рендерерами» или «визуализаторами» [53–57].

Основное и важнейшее отличие графических движков от программных рендереров состоит в том, что первые должны обязательно работать в режиме реального времени, тогда как вторые могут тратить по несколько десятков часов на вывод одного изображения. Вторым существенным отличием является то, что начиная приблизительно с 1995-1997 года, графические движки производят рендеринг с помощью графических процессоров (англ. *GPU*), которые установлены на отдельных платах — видеокартах. Большинство программных рендереров используют только центральные процессоры (англ. *CPU*).

Одним из основных аспектов реалистичности генерируемого системой визуализации изображения является освещение трехмерной сцены. Существует четыре основных вида источников света: отраженный свет (ambient light),

направленный источник свет (directional light), всенаправленный источник света (point light) и источник света типа «прожектор» (spot light).

Отраженный свет используется для грубой аппроксимации освещения в виртуальной среде. Отраженный свет равномерно освещает объекты трехмерной сцены с равной интенсивностью.

Направленный источник света испускает свет в одном направлении и не затухает с увеличением расстояния. Подобные тип источников света часто применяется для имитации солнца.

Всенаправленный источник света излучает свет из одной точки во все стороны с одинаковой интенсивностью. Интенсивность света уменьшается в зависимости от расстояния по закону обратных квадратов.

Если источник света находится в точке P, тогда интенсивность света в точке пространства Q рассчитывается по следующей формуле:

$$C = \frac{1}{k_c + k_l d + k_q d^2} C_0,$$

где C - интенсивность источника света в точке Q, C_0- цвет источника света, d – дистанция между источником света и точкой Q ($d = \|\mathbf{P} - \mathbf{Q}\|$), k_c – константа неизменного затухания света, k_l – константа линейного затухания света, k_q – константа квадратичного затухания света.

Источник света типа «прожектор» (spot light) во многом схож с всенаправленным источником света, но имеет направление. Для источника света с началом в точке P и направлением R, интенсивность в точке пространства Q будет рассчитываться по следующей формуле:

$$C = \frac{\max\{-\mathbf{R} \cdot \mathbf{L}, 0\}^p}{k_c + k_l d + k_q d^2} C_0,$$

где C - интенсивность источника света в точке Q, C_0- цвет источника света, d – дистанция между источником света и точкой Q, k_c, k_l, k_q – константы затухания света, L – вектор единичной длины, направленный из точки Q в центр источника света:

$$\mathbf{L} = \frac{\mathbf{P} - \mathbf{Q}}{\|\mathbf{P} - \mathbf{Q}\|}.$$

Коэффициент p определяет фокусировку источника света. Чем больше значение p, тем меньше световое пятно.

Диффузной поверхностью является поверхность, на которой часть падающего света рассеивается, равномерно отражая определенный цвет (отражаемый цвет диффузной поверхности) во всех направлениях. Подобное поведение света была описано И. Ламбертом в 1760 году (Закон Ламберта) [58]. Кроме того, из-за того, что свет отражается равномерно во всех направлениях, отраженный цвет не зависит от позиции наблюдателя. Для расчета цвета диффузного отражения световой волны K_{DIFF}, отраженной от поверхности в точке Q в направлении наблюдателя, с использованием нескольких источников света, применяется следующая формула:

$$K_{DIFF} = DTA + DT \sum_{i=1}^{n} C_i \max\{\mathbf{N} \cdot \mathbf{L}_i, 0\},$$

где D – диффузный отраженный свет, A – интенсивность рассеянного света, C_i – цвет i-го источника света в точке Q, $\mathbf{N}$ – вектор нормали, $\mathbf{L}_i$ – вектор направления i-го источника света, T – цвет пикселя используемой текстуры.

В дополнение к равномерно рассеиваемому диффузному отражению, поверхность отражает свет под углом, равным углу падения. В отличие от диффузного отражения, обычное отражение зависит от позиции наблюдателя. Для расчета отраженного от поверхности света применяется следующая формула:

$$K_{SPEC} = SG \sum_{i=1}^{n} C_i \max\{\mathbf{N} \cdot \mathbf{H_i}, 0\}^m, (\mathbf{N} \cdot \mathbf{L}_i > 0),$$

где S – цвет отраженного света, C_i – цвет i-го источника света в точке Q, $\mathbf{N}$ – вектор нормали в точке Q, m – коэффициент отражения света, G – цвет пикселя из текстуры освещения, $\mathbf{H}_i$ – биссектриса угла между вектором направления к наблюдателю и вектором направления источника света $\mathbf{L}_i$:

$$\mathbf{H}_i = \frac{\mathbf{L}_i + \mathbf{V}}{\|\mathbf{L}_i + \mathbf{V}\|}.$$

Выражение $(\mathbf{N} * \mathbf{L}_i > 0)$ является булевым выражением, которое может принимать значения 0 (ложь) и 1 (истина), и позволяет избежать появления отражений на точках поверхности, которых не достигает свет от конкретного источника света.

Учитывая приведенные выше рассуждения, получаем итоговую формулу расчета цвета поверхности в точке Q, освещенной несколькими источниками света, по модели Blinn-Phong [59]:

$$K = EM + DTA + \sum_{i=1}^{n} C_i[DT\max(\mathbf{N} \cdot \mathbf{L}_i) + (SG\max\{\mathbf{N} \cdot \mathbf{H}_i, 0\}^m, (\mathbf{N} \cdot \mathbf{L}_i > 0))],$$

где D – отраженный диффузный свет, S – отраженный свет, m – коэффициент отражения света, A – цвет рассеянного света, E – излучаемый цвет, T – цвет текстуры, G – цвет карты отражений, M – цвет карты излучения, C_i – цвет i-го источника света в точке Q, $\mathbf{L}_i$ – вектор направления i-го источника света, $\mathbf{H}_i$ – биссектриса для i-го источника света, **N** – вектор нормали.

В последнее время в мультимедийных проектах все чаще начинают использовать освещение, основанное на физических принципах (Physical Based Rendering, PBR). Основными отличиями от традиционных моделей освещения является учет сохранения энергии луча от источника света и учет неровностей поверхности, что влияет на отражающую способность материала. Использование PBR позволяет генерировать более реалистичное изображение, однако требует изменений в существующих технологиях визуализации и процессах создания контента [60; 61].

1.1. Обзор лидирующих технологий визуализации в реальном времени

1.1.1. Unreal Engine

Unreal Engine — трехмерный движок, разрабатываемый и поддерживаемый компанией Epic Games [62]. Развивается с 1998 года, на данный момент для мультимедийных проектов используется четвертая версия движка (Unreal Engine 4). На данном движке было создано более 100 мультимедийных

проектов и, на сегодняшний день, он является в индустрии основным движком для создания мультимедийных проектов [63].

Движок написан на языке C++ и позволяет создавать проекты для операционных систем Microsoft Windows, Linux и Mac OS X, Разделение функциональности на модули позволяет поддерживать различные системы рендеринга (Direct3D, OpenGL, Metal), воспроизведения звука (EAX, OpenAL, DirectSound3D; FMOD, WWISE), модули для работы с сетью и различными устройствами ввода.

Для сетевой работы поддерживаются технологии Windows Live, способные одновременно обрабатывать данные для 64 клиентов. Несмотря на то, что официально средства разработки не включают в себя поддержку большого количества клиентов на одном сервере, движок использовался для создания MMO-проектов.

Основным классом в Unreal Engine является класс UObject, описывающий набор характеристик, доступных всем объектам в движке. Большая часть классов в движке является дочерними классами к классу UObject. Можно выделить следующие основные классы в Unreal Engine:

- Актор (*actor*) — родительский класс, характеризующий положение объекта в пространстве. Является базовым для всех классов, имеющих отношение к игровому процессу.
    - Пешка (*pawn*) — класс, позволяющий задавать физическую модель объекта, а также сценарий, задающий его поведение. Без заданной модели объект представляется в виде пешки (pawn (англ.)–.пешка).
- Мир, уровень (*world, level*) — класс, позволяющий задавать параметра виртуального пространства, такие как силу тяжести и наличие тумана, а также осуществляющий управление всеми акторами. Может содержать в себе различные параметры игрового процесса, такие как игровой режим для текущего уровня [64].

При визуализации трехмерных объектов в Unreal Engine применяется принцип зонирования – виртуальная трехмерная сцена разделяется

«заполненную» и «пустую» части при помощи двоичного разбиения пространства. В «пустой» части пространства располагаются все объекты трехмерной сцены, в том числе виртуальная камера, используемая при прорисовке сцены. Объекты могут быть полностью или частично помещены в «заполненную» часть пространства, однако это может привести к некорректным расчетам физического взаимодействия между объектами или прорисовке объектов с графическими артефактами.

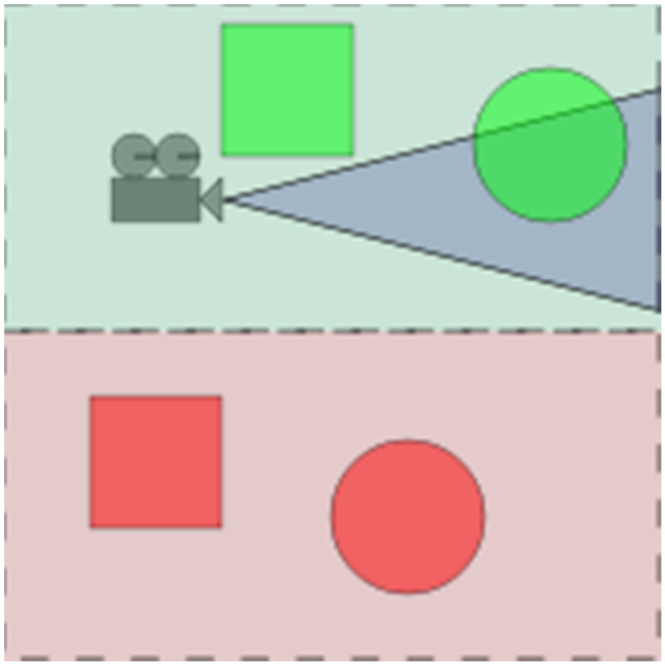

Рис. 1.1. Зонирование. В угол обзора виртуальной камеры не попадают объекты в красной зоне (портале), соответственно, объекты в этой зоне не обрабатываются.

Поверхность (*surface*) является основным элементом двоичного дерева пространства, которые создаются на границе между «заполненной» и «пустой» частями пространства. Группа элементов двоичного дерева пространства называется узлом. Количество одновременно отображаемых узлов является одним из основных факторов, влияющих на производительность движка при прорисовке виртуальной сцены. В случае, когда узел не попадает в угол обзора виртуальной камеры или другие узлы полностью закрывают этот узел, обработка узла не производится — это позволяет повысить производительность визуализации, особенно в закрытых пространствах. Для этого иногда используются *порталы* —

невидимые поверхности, применяемые для разделения узла на два меньшего размера.

Для описания «заполненных» и «пустых» частей пространства применяются замкнутые трехмерные объекты, которые составлены из не пересекающихся поверхностей — кистей (*brush*, русск. *кисть*). Этот принцип построения пространства называется конструктивной сплошной геометрией. Пространство может быть представлено в двух вариантах: изначально пустое (аддитивная геометрия) или изначально заполненное материей пространство (вычитательная геометрия). Основной упор сделан на *additive*-геометрию, однако от поддержки вычитательной геометрии не отказываются. Основным недостатком вычитательной геометрии является более ресурсоемкий и длительный расчет освещения.

Кисти можно условно разделить на три типа:

- Сплошные (*solid*) — полноценно участвуют в двоичном разбиении пространства.
    - Аддитивные (*additive*) — «заполняют» двоичное пространство.
    - Разностные (*substractive*) — «вырезают» объёмы в пространстве.
- Полу-сплошные (*semi-solid*) — влияют только на физическую модель пространства. Служат для создания «невидимых» препятствий, а также уменьшения числа видимых полигонов и узлов.
- Пустые (*non-solid*) — только создают поверхности, не влияют на двоичное дерево пространства. Используются преимущественно для создания *объёмов* (*volume*) — пространств, обладающих свойствами, отличными от свойств остальной трехмерной сцены. Объёмы имеют приоритет обработки, свойства объёма с большим приоритетом применяются к находящимся в нём акторам. При помощи объёмов возможно изменять параметры в отдельно взятой части трехмерной сцены, такие как гравитация, или добавлять различные эффекты, например, туман, и т.д [64].

Unreal Engine был разработан для использования возможностей современных интерфейсов взаимодействия с графическим адаптером – DirectX 11,

OpenGL 4, Apple Metal. В связи с широким распространением многоядерных систем движок использует два параллельных главных потока — основной поток, применяемый для обработки событий приложения и команд пользователя, и поток визуализации. Также могут быть созданы дополнительные потоки для выполнения разовых задач. Начиная с третьей версии движка появилась поддержка многопоточной динамической загрузки данных (*streaming*), которая может применяться для визуализации обширных открытых пространств без отображения экранов загрузки.

Графическая подсистема Unreal Engine поддерживает большинство современных технологий, включая тесселяцию, HDR, динамическое освещение, шейдеры версии 5.0. В качестве физической подсистемы применяется PhysX от компании NVIDIA. За анимацию лиц персонажей отвечает механизм FaceFX. Для генерации деревьев может применяться технология SpeedTree. [65]

1.1.2. CryEngine 3

CryEngine 3 — движок, разработанный немецкой компанией Crytek. На момент выхода является самым технологически продвинутым и фотореалистичным движком по сравнению с конкурентами. CryEngine 3 используется фирмой Crytek для создания мультимедийных продуктов, а также предлагается для лицензирования другим компаниям.

Движок CryEngine 3 предназначен для создания трёхмерных открытых (англ. *outdoor*) и закрытых (англ. *indoor*) локаций, используя интерфейс программирования приложений (англ. *API*) DirectX 9-й или 11-й версии. CryEngine 3 — полнофункциональный инструментарий для разработки трехмерных проектов; помимо графического движка, движок включает в себя следующие компоненты: систему анимации персонажей, физический движок CryPhysics, систему воспроизведения звука, систему управления интеллектом, редактор сцен Sandbox 3, а также инструмент PolyBump. Все компоненты движка являются собственными разработками Crytek. Для повышения

производительности движок CryEngine 3 применяет многопоточность на многоядерных процессорах [66].

Особенности:

- Инструментарий для анализа производительности

Встроенные в движок инструменты для анализа производительности позволяют отслеживать расход оперативной и видео памяти, определять загрузку процессорных ядер и влияние каждого отдельного трехмерного объекта на общую производительность.

- Оффлайн визуализация

CryEngine 3 позволяет создавать изображения или видео в режиме оффлайн. Этот режим применяется, когда необходимо максимальное качество, а не скорость визуализации. Настройки режима могут быть произведены через командную строку. Примерами настройки могут служить определение разрешения генерируемого изображения или видео, соотношения сторон.

- Модульное построение

В движке применяется модульная архитектура. Каждый модуль движка, например, модуль физической обработки или модуль генерации изображения, компилируется в отдельный файл-библиотеку. За счет подобного разделения возможно использовать для различных проектов только необходимые компоненты. При необходимости внесения изменений модификации подвергаются только отдельные модули, а не весь движок. CryEngine 3 написан на языке программирования C++.

- Поддержка многопоточности

Для эффективного использования возможностей современных многоядерных процессоров, многие подсистемы CryEngine 3, такие как подсистема физической обработки, способны применять многопоточность.

- Компилятор ресурсов

Активы (англ. *assets*) компилируются из первоначальных форматов в формат, оптимизированный под используемую платформу, при помощи

встроенного компилятора ресурсов. Применение компилятора ресурсов позволяет вносить глобальные изменения в выходные данные (например, изменять порядок байтов для ресурсов под платформы с другой архитектурой процессора) в зависимости от предварительных установок (англ. *presets*) и целевых платформ, а также избежать необходимости хранения версий активов под каждую платформу.

- Система потоковой загрузки ресурсов

Применение потоковой загрузки позволяет загружать ресурсы во время выполнения программы в реальном времени. Это дает возможность отображать виртуальные трехмерные сцены больших размеров (более нескольких квадратных километров), загружая объекты сцены блоками, и экономить оперативную и видео память.

- Сетевая система «Клиент-сервер»

Для CryEngine 3 была написана новая система сетевого взаимодействия, применяющая многопоточность для повышения производительности. которая управляет всеми подключениями в режиме многопользовательской работы. Новая система основана на архитектуре «клиент-сервер», обеспечивает низкое время ожидания (англ. *low-latency*) и характеризуется низкими требованиями к ширине полосы пропускания.

- Динамическое освещение и затенение в реальном времени

Графический модуль CryEngine 3 позволяет визуализировать мягкие тени, динамически реагирующие на движение источников света и возникновение преград между светом и затеняемой поверхностью. Тени визуализируются объёмными, с мягкими краями и в высоком разрешении.

- Технология объемного тумана (англ. *Volumetric, Layer and View Distance Fogging*)

При помощи данной технологии возможна визуализация объемных облаков, дымки и тумана, которые способны частично перекрывать другие трехмерные объекты, такие как ландшафт. При визуализации тумана учитывается влияние окружающих источников света и теней.

- Технология Terrain 2.5D и карты затенения (англ. *Terrain 2.5D Ambient Occlusion Maps*)

Технология Screen Space Ambient Occlusion, являющуюся развитием технологии Ambient Occlusion, была впервые применена именно в CryEngine 3. Данная технология на попиксельно аппроксимирует влияние окружающего света на трехмерный объект. Аппроксимация зависит от количества окружающих преград, созданных, например, листвой.

- Карты нормалей и параллакс-маппинг

Карты нормалей (англ. *Normal Maps*) применяются для имитации деталей на трехмерных объектах без изменения количества полигонов. Для сжатия карт нормалей в CryEngine 3 применяется алгоритм *3DC/BC5*. CryEngine 3 стал первым графическим движком, использующим технологию parallax occlusion mapping — усовершенствованный вариант parallax mapping. Благодаря данной технологии все вычисления, связанные с имитацией деталей трехмерных объектов, производятся на графическом адаптере. Примером работы технологии может служить создание изображения сложного ландшафта с перепадами высот, в то время как трехмерная поверхность состоит всего из двух треугольников.

- Подповерхностное рассеивание (англ. *Subsurface Scattering*)

Подповерхностное рассеивание моделирует диффузию (распространение) и дифракцию света, прошедшего через прозрачные объекты, такие как, например, лёд или мрамор. Данная методика может использоваться для создания естественно выглядящей кожи и растительности.

- Моделирования зрения и освещение в расширенном динамическом диапазоне

Технология моделирования зрения применяется для аппроксимации процесса адаптации человеческого глаза к внезапным изменениям условий освещения, например, переход из темноты в ярко освещенное помещение. Рендеринг с использованием HDR повышает реалистичность сцен с широким динамическим диапазоном.

- Световые лучи и волны (англ. *Light Beams & Shafts*)

Технология световых лучей и волн используется для визуализации эффектов, которые возникают при пересечении светового потока с трехмерной геометрией. Примером работы данной технологии может служить визуализация света, проходящего через листву деревьев.

- Эффективная работа с шейдерами

Система сценариев позволяет комбинировать текстуры и математические функции для создания визуальных эффектов, таких как грязь, обледенение поверхностей, эффект невидимости. Подобные эффекты могут применяться как отдельно, так и в сочетании со стандартными шейдерами, отвечающих за вид трехмерных объектов (например, металлическая поверхность). Среди поддерживаемых эффектов можно отметить смазанные отражения (англ. *bumpy reflections*), преломление света (англ. *refractions*), объёмные эффекты свечения.

- Технология визуализации океана

Технология визуализации океана позволяет визуализировать высоко-детализированную поверхность океана, которая учитывает направление ветра и волн. Кроме того, данная технология обеспечивает корректную визуализацию перехода между океаном и берегом, принимая во внимание контур береговой линии и глубину океана. Также возможна визуализация подводного пространства.

Рис. 1.2. Изображение, созданное движком CryEngine 3. Хорошо видно мягкие сложные тени от деревьев на земле, а также сложные высокополигональные модели деревьев. У верхней части изображения на примере неба можно видеть работу освещения в расширенном динамическом диапазоне (HDR).

- Размытие изображения при движении и глубина резкости (англ. *Motion Blur & Depth of Field*)

Размытие изображения при движении используется для визуализации эффекта смазывания у быстро движущихся объектов или при резких движениях камеры. Размытие можно применять как к определённым объектам, так и ко всей сцене. Глубина резкости (англ. *Depth of Field*) используется для фокусировки на определенном объекте, в то время как остальные объекты подвергаются размытию, сила которого зависит от их удаления от наблюдаемого объекта.

- Управление уровнем детализации ландшафта

Технология управления уровнем детализации ландшафтов позволяет снизить нагрузку на центральный процессор и графический адаптер, а также снизить количество используемой оперативной памяти. Близкие к

наблюдателю объекты и ландшафт прорисовываются в максимальном качестве, в то время как с удалением от наблюдателя детализация объектов и ландшафта падает. Используя уровни детализации, возможно отображения ландшафта на дальности до 8 километров от наблюдателя.[67][68]

1.1.3. Source

Source (в переводе с английского **"Источник"**, официальное название **Valve Source Engine**) — 3D движок, разработанный корпорацией Valve [69]. Его особенностями считаются модульная основа и гибкость, технология выражения эмоций и система физики, работающая по сети. В качестве формата хранения моделей используется общий для продуктов Valve формат mdl. Физическая часть движка Source основана на Havok Physics. Первая версия движка была представлена 16 ноября 2004 [70].

Движок Source регулярно обновляется, а также имеет широкие возможности к модификации и улучшению без изменения ключевых архитектурных особенностей движка благодаря модульной архитектуре. Благодаря системе цифрового распространения Steam движок может получать обновления сразу после их выхода. Таким образом, при внедрении новой функциональности все проекты, основанные на Source и распространяемые через Steam, начнут их поддерживать. Ярким примером этого служит добавление разработчиками в движок поддержки HDR (High Dynamic Range).

Следующие ключевые технологии реализованы в Source:

- **Лицевая анимация.** Лицевая анимация позволяет компьютерным персонажам реалистично выражать эмоции, а также синхронизировать движение губ с воспроизводимой звуковой информацией. Технология использует исключительно ресурсы видеоадаптера, снимая нагрузку с центрального процессора.
- **Динамическое освещение и затенение.** В Source добавлена возможность использования HDR-рендеринга, затенения с использованием

динамических теней, учитывающие самозатенение объектов и другие параметры, в трехмерную сцену.

- **Кинематографическая физика.** Кинематографическая физика поддерживает систему ключевых кадров. Аниматоры могут создавать длительные сцены с небольшим количеством ключевых кадров, промежуточные же кадры рассчитываются физической подсистемой. В итоге разработчики получают технологию, которая позволяет им создавать гораздо более сложные сцены, чем раньше, затрачивая на это те же самые ресурсы [71].

1.2. Обзор современных методов рир-проекции в реальном времени

В настоящее время технология рир-проекции (кеинг) применяется в теле- и киноиндустрии для создания детализированных виртуальных трёхмерных сред. Основной идеей технологии кеинга является выделение объекта с однородного фона. Данный процесс можно описать как процесс создания маски, содержащей информацию о прозрачности изображения, отделяющей объект от остального изображения. В компьютерной графике маска является изображением с одним каналом, используемым для определения прозрачных областей переднего плана и фона на скомбинированном изображении. Также, маска прозрачности часто называется «альфа каналом».

Создание качественной маски прозрачности является трудоемкой задачей. Наиболее простым решением является создание маски, которая делает фон изображения полностью прозрачным, а объекты переднего плана полностью видимыми. Однако, такой подход неприменим для сложных сцен – объекты переднего плана будут иметь резкие края и, соответственно, будут выглядеть нереалистично. Для отображения таких объектов, как волосы и дым необходимо, чтобы маска прозрачности поддерживала не только дискретные значения прозрачности (0 – прозрачный объект, 1 – непрозрачный объект), но и значения в пределах от 0 и 1 для обеспечения частичной прозрачности объекта.

Существует несколько подходов, используемых для выделения объектов с фона. Наиболее распространенным подходом является использование однородного фона, например, синего или зеленого. Недостатком является необходимость точной калибровки света для получения однородного фона. Еще один подход состоит в использовании нескольких изображений, на которых присутствует объект, который необходимо выделить. Однако данный подход требует строгих условий съемки, и применим только для статических объектов. Третий подход основывается на том, что объекты фона и переднего плана заранее известны. Пользователь выделяет объекты на переднем плане и фоне, а оставшаяся часть изображения – границы объекта на переднем плане – считаются зоной перехода. Данный подход не может быть использован в реальном масштабе времени, соответственно, неприменим для использования в подсистеме визуализации ТОС.

Существует большое количество методов кеинга, каждый со своими достоинствами и недостатками. Одним из наиболее простых методов кеинга является *люма-кеинг* (англ. Luma keying – кеинг по яркости). Так как большинство видеосигналов (таких как PAL, NTSC, SECAM) предоставляют информацию о яркости изображения в отдельности от информации о цвете, то задача создания маски прозрачности на основе яркости является достаточно простой. Если изображение представлено в цветовом пространстве RGB, то целесообразным является приведение цвета к цветовой модели HLS (Hue, Luminance, Saturation - цвет, яркость, насыщенность) для получения яркостной составляющей. Установив пороговое значение яркости можно создать простую маску прозрачности, которая определяет видимость пикселя – полностью видим или полностью невидим. Более совершенные методы люма-кеинга позволяют определять значения мягкости и допустимого предела. Мягкость описывает переход от полностью видимого пикселя к невидимому (что необходимо для кеинга полупрозрачных объектов таких как стекло или вода, а также для сохранения частичной видимости таких объектов как волосы). Допустимый предел описывает диапазон значений вокруг заданного значения яркости. Параметры мягкости и допустимого предела

применяются также и в других видах кеинга. Из-за отсутствия возможности использовать информацию о цвете изображения люма-кеинг неприменим для большинства ситуаций, однако может быть использован для таких работ, как извлечение текста (черный текст на белом фоне) [72].

Другим методом кеинга является так называемый *хромакеинг (*англ. Chroma keying – кеинг по цветовой составляющей). Под термином хромакеинг подразумевается процесс создания маски прозрачности на основе цветовой информации изображения, в то время как информация о яркости и насыщенности не используется. Изображения объектов, снятых на более-менее однородном синем или зеленом фоне, могут быть выделены, используя только цветовую информацию.

Однако невозможно обеспечить полностью однородный фон у изображения (в т.ч. из-за особенностей освещения). Из-за этого маска М создается путем задания предельно допустимого значения по цвету Т для выбранного значения цвета Н. Если значение Н пикселя Р находится в допустимых пределах, то значение маски пикселя Мп выставляется в 1, иначе 0.

$$Mn = \begin{cases} 1, & если\ (H - T) < P < (H + T) \\ 0, & если\ P <= (H - T)\ или\ P => (H + T) \end{cases}. \qquad \textbf{(1.1)}$$

Следовательно, хромакеинг может быть произведен с использованием любого выбранного цвета фона. Однако, качество хромакеинга обычно невысокое, особенно если используется сжатие изображения. Например, популярный видеоформат DV сжимает кадры видео в формат YCbCr 4:1:1 (для NTSC, для PAL применяется 4:2:0), что ведет к артефактам в цветовом канале. Хромакеинг сжатого изображения приведет к созданию маски худшего качества, чем несжатого [73].

Метод хромакеинга может быть улучшен при помощи добавления таких параметров, как оттенок и яркость изображения, а также их предельно допустимых значений. HLS кеинг обрабатывает значения цвета (Hue - H), яркости (Luminance – L) и насыщенности (Saturation – S) изображения, и использует

полученные результаты для создания маски прозрачности (М). При использовании в методе параметра мягкости также повышается качество итогового изображения.

$$H_{маска} = \begin{cases} 1, & если\ (Hф - T) < Hn < (Hф + T) \\ 0, & если\ Hn <= (Hф - T)\ или\ Hn => (Hф + T) \end{cases}$$
$$L_{маска} = \begin{cases} 1, & если\ (Lф - T) < Ln < (Lф + T) \\ 0, & если\ Ln <= (Lф - T)\ или\ Ln => (Lф + T) \end{cases}$$
$$S_{маска} = \begin{cases} 1, & если\ (Sф - T) < Sn < (Sф + T) \\ 0, & если\ Sn <= (Sф - T)\ или\ Sn => (Sф + T) \end{cases}, \qquad \textbf{(1.2)}$$

$$Mn = \alpha H_{маска} + \beta S_{маска} + \delta L_{маска}$$

где $\alpha + \beta + \delta = 1$, Нф – цвет фона, Lф – яркость фона, Sф – насыщенность фона, Нп – цвет пикселя, Lп – яркость пикселя, Sп – насыщенность пикселя.

Несмотря на то, что у HLS кеинга существуют такие недостатки, как острые грани, для смягчения которых требуется размытие или другие техники обработки граней, существует несколько приложений, использующих HLS кеинг. Например, компания Autodesk предоставляет два модуля HLS кеинга в своем программном обеспечении для монтажа Combustion: Discreet Keyer и Diamond Keyer.

На сегодняшний день термин «хромакеинг» используется для многочисленных методов выделения объектов переднего плана с фона изображения, которые не являются хромакеингом в первоначальном смысле этого слова. Причиной этого можно считать то, что изображение в цифровом изображении представляется в цветовой модели RGB. Алгоритмы кеинга, работающие с изображением напрямую в цветовой модели RGB, используют информацию трех цветовых каналов, а не информацию о яркости или насыщенности изображения.

Примером кеинга, работающего в цветовом пространстве RGB, может являться методика определения различий между цветами. Данный метод был предложен Петро Влахосом (Petro Vlahos) в 1964 году. Изначально метод был разработан для химической и оптической обработки цвета отснятой кинопленки. Основной идеей определения различий между цветами изображения I является

определение значения прозрачности на основе разности между цветовыми каналами R, G и B. Например, при использовании синего фона значение прозрачности рассчитывается как разность значения синего канала (I_B) и максимального значения двух цветовых каналов (I_R и I_G):

$$\alpha = I_B - MAX(I_R, I_G). \quad \textbf{(1.3)}$$

Изначально данный метод был разработана для создания масок для кинопленки (а не для цифровых изображений), затем она была использована для создания аппаратуры по обработке аналогового видео, и с 1995 года существует в виде программного обеспечения, распространяемого компанией Ultimatte.

Еще одним методом кеинга является 3D кеинг. В 3D кеинге маска прозрачности создается с использованием поверхностей и трёхмерных объектов для отделения фоновых пикселей от остального изображения. Изображение представляется в виде точек определенного цвета (пикселей) в трехмерном пространстве RGB или HLS. Пример изображения в трехмерном пространстве RGB представлен на Рис. 1.3. Фон изображения представлен как компактное облако синего цвета, в то время как объект переднего плана представляется в виде кривой с различными цветами (от темно-красного до бледно желтого).

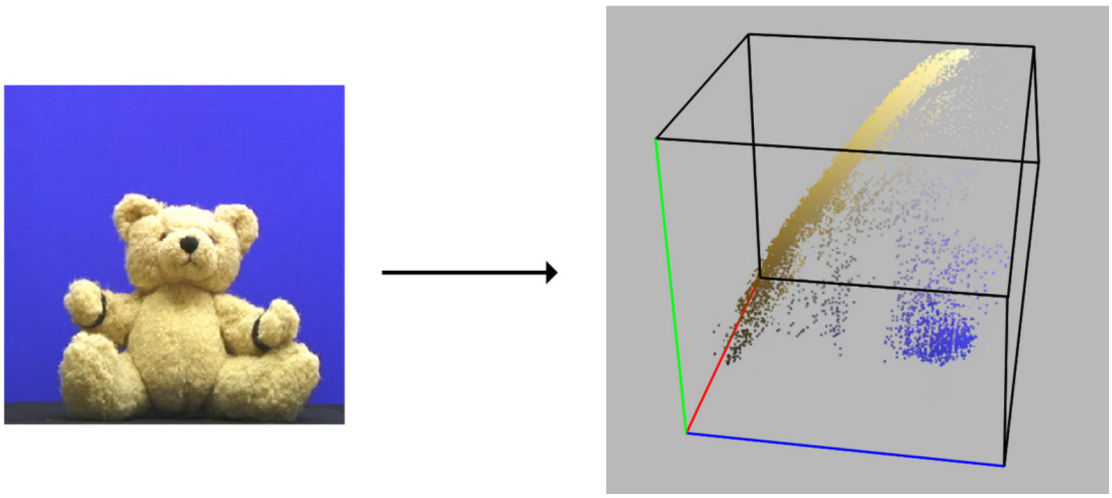

Рис. 1.3. Представление изображения в трехмерном пространстве RGB

Обычно применяют два объекта: первый объект (O1), определяющий гарантированно фоновый цвет, перекрывается вторым объектом (O2),

определяющим зону перехода. Одним из простейших объектов, определяющим границы цветов, является сфера. Поместим сферу S1 в пространстве так, чтобы она перекрывала облако цветов фона, то есть определяла диапазон значений цвета, которые будут считаться полностью прозрачными. Координаты этой сферы могут быть заданы автоматически на основе анализа цветов изображения, либо вручную указаны пользователем.

Для получения плавной зоны перехода создается вторая сфера большего радиуса S2, центр которой располагается в той же точке, что и центр первой сферы. Как показано на Рис. 1.4, пиксели, расположенные внутри сферы S1, являются полностью прозрачными. Пиксели, находящиеся вне сферы S2, являются полностью видимыми, а пиксели, находящиеся между границами сфер S1 и S2, являются полупрозрачными. Значения прозрачности рассчитываются на основе координат пикселя в трехмерном пространстве и центра сфер либо поверхностей S1 и S2.

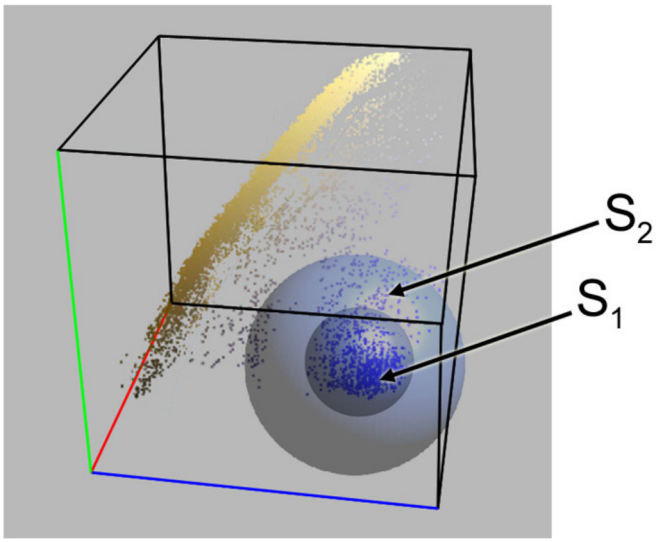

Рис. 1.4. 3D кеинг

Практически любой тип трёхмерного объекта может быть использован для отделения цвета фона от цвета объектов переднего плана, однако в большинстве случаев применяются выпуклые каркасы (convex hull). Соответственно, необходимо чтобы пользователь выделил несколько пикселей фона с целью их перекрытия выпуклым каркасом. На сегодняшний день существует два

приложения на рынке, выполняющих 3D кеинг при помощи трехмерных поверхностей: Primatte Keyer компании Imagica и Modular Keyer компании discreet/autodesk. Они различаются типом применяемых трехмерных объектов и алгоритмом взаимодействия с пользователем.

Существуют и другие методы кеинга, которые требуют использования специального оборудования для захвата дополнительных данных, используемых для создания маски прозрачности. Так, один из методов подразумевает использование поляризованного света, который, отражаясь от кожи или поверхностей, становится неполяризованным, однако специальный материал фона отражает поляризованный свет. Разница в отраженном сигнале используется для создания маски прозрачности. Преимуществом данного подхода является независимость цвета объекта переднего плана от цвета фона. Нет необходимости в точной настройке света для того, чтобы избежать получения определенных цветов на объекте переднего плана, и постобработке изображения для удаления засветки объекта переднего плана цветом фона. Недостатком данного метода являются значительные финансовые затраты на отражающий материал и оборудование для записи и обработки поляризованного сигнала.

Другой метод использует инфракрасное излучение для расчета дистанции каждого пикселя до камеры, поэтому данный метод также называется «z-кеинг». Используя информацию о глубине изображения возможно задать определенную границу и выделять только те объекты, которые соответствуют заданным параметрам глубины.

Еще один метод кеинга использует информацию о температуре объектов и создает маску прозрачности путем отделения холодных участков изображения от теплых. Недостатком этого метода является то, что только живые или теплые объекты будут представлены как объекты переднего плана, в то время как некоторые детали, такие как волосы, не будут правильно вырезаны из фонового изображения [74].

1.3. Анализ современных технологий визуализации

Проанализировав современные системы визуализации можно сделать следующие выводы:

- Описанные системы визуализации предлагаются компаниями-разработчиками за высокую стоимость (например, стоимость Unreal Engine 3 может доходить до 900 тысяч долларов, стоимость CryEngine 3 может доходить до 1.2 миллиона долларов) [75]. Несмотря на то, что в настоящее время часть этих систем стала бесплатной или открыли исходный код, их применение для неигровых проектов ограничено и по-прежнему требует покупки лицензии [76].
- Описанные системы визуализации являются проприетарными, что усложняет или делает невозможной их модификацию (изменение базовой функциональности или исправление ошибок) без согласования с разработчиками этих систем
- Высокий порог вхождения для работы с данными системами, обширное количество кода и документации для изучения
- Необходимость в мощном аппаратном обеспечении для разработки ПО (например, для CryEngine 3 рекомендуется использование конфигурации Intel Core i7, Nvidia 780)
- Данные системы визуализации не предоставляют возможности воспроизведения разнородных видеоматериалов, а также не позволяют воспроизводить потоковые видеоматериалы; внесение такой функциональности может быть затруднительно в связи с причинами, описанными выше.

На основе проведенного анализа можно сделать вывод, что описанные выше системы неприменимы в качестве основы подсистемы визуализации для системы предтренажерной подготовки. Предлагается использовать решения с открытым исходным кодом. В качестве основы для подсистемы визуализации выбран графический модуль Horde3D, характеристики которого будут детально описаны в следующих главах.

1.4. Анализ современных методов рир-проекции

Проанализировав существующие методы рир-проекции можно создать следующую сводную таблицу функциональности:

Таблица 1-1. Сравнение функциональности различных методов кеинга

Название метода	Преимущества	Недостатки
Luma-кеинг	Простота реализации	Неприменим для большинства ситуаций, может быть использован для таких работ, как извлечение текста
Хромакеинг	Может быть произведен с использованием любого выбранного цвета фона	Невысокое качество, острые грани при обработке изображения, для смягчения которых требуется размытие или другие техники обработки граней
3D кеинг	Простота настройки параметров обработки, точная настройка обрабатываемых цветов с помощью произвольных трёхмерных фигур	Относительная сложность реализации
Кеинг с использованием поляризованного света	Независимость цвета объекта переднего плана от цвета фона и от качества освещения	Значительные финансовые затраты на отражающий материал и оборудование для записи и обработки сигнала
Z-кеинг	Определение объекта по глубине изображения, независимость от цвета объекта и качества освещения	Значительные финансовые затраты на оборудование для записи и обработки сигнала
Кеинг с	Независимость от цвета	Значительные финансовые

использованием температуры объектов	объекта и качества освещения	затраты на оборудование для записи и обработки сигнала, некоторые детали, такие как волосы, не будут правильно вырезаны из фонового изображения

Проведенный анализ показывает преимущества 3D кеинга перед другими методами рир-проекции. Вывод - метод 3D кеинга применим в ТОС.

Глава 2 . Информационная архитектура автоматизированной системы обучения

Автоматизированная система обучения (АСО) – модуль тренажерно-обучающей системы, обеспечивающий теоретическую подготовку операторов сложных технических комплексов. Теоретическая подготовка является базой, на которой в процессе тренажерной подготовки обучаемый формируется как специалист.

Архитектура автоматизированной системы - это наиболее абстрактное ее представление, которое включает в себя идеализированные *модели компонентов* системы, а также модели взаимодействий между компонентами. Элементы архитектуры находятся во взаимосвязи, образуя единую автоматизированную систему и обеспечивая решение поставленной задачи автоматизации на архитектурном уровне. В то же время архитектура оставляет достаточно свободы для выбора конкретных технических решений. Поэтому корректно спроектированная *архитектура допускает множество технических реализаций* путем выбора различных компонентов архитектуры и методов взаимодействия между ними.

При построении архитектуры должны быть заложены следующие свойства будущей автоматизированной системы:

- *слабая связанность* элементов архитектуры между собой (т. е. декомпозицию системы на части следует производить так, чтобы поток информации через связи был минимален и через них не замыкались контуры автоматического регулирования);
- *тестируемость* (возможность установления факта правильного функционирования);
- *диагностируемость* (возможность нахождения неисправной части системы);
- *ремонтопригодность* (возможность восстановления работоспособности за минимальное время при экономически оправданной стоимости ремонта);

- *надежность* (например, путем резервирования);
- *простота обслуживания* и эксплуатации (минимальные требования к квалификации и дополнительному обучению эксплуатирующего персонала);
- *безопасность* (соответствие требованиям промышленной безопасности и технике безопасности);
- *защищенность системы* от неквалифицированных пользователей;
- *экономичность* (экономическая эффективность в процессе функционирования);
- *модифицируемость* (возможность перенастройки для работы с другими технологическими процессами);
- функциональная *расширяемость* (возможность ввода в систему дополнительных функциональных возможностей, не предусмотренных в техническом задании);
- *наращиваемость* (возможность увеличения размера автоматизированной системы при увеличении размера объекта автоматизации);
- *возможность переконфигурирования* системы для работы с новыми технологическими процессами;
- максимальная *длительность жизненного цикла* системы без существенного морального старения, достигаемая путем периодического обновления аппаратных и программных компонентов, а также путем выбора долгоживущих промышленных стандартов;
- *минимальное время на монтаж и пуско-наладку* (развертывание) системы [77].

Информационная архитектура АСО была спроектирована с учетом данных требований. Основные компоненты архитектуры приведены на Рис. 2.1.

Предложенная архитектура АСО состоит из: ядра системы, содержащего базовые компоненты, необходимые для работы системы; подсистем, отвечающих за определенные функции (рендеринг, воспроизведение мультимедиа материалов, анимацию и т.д.); интерфейса взаимодействия, обеспечивающего связь различных

компонентов ядра и подсистем между собой; графического интерфейса пользователя, управляющего функциональностью системы через интерфейс взаимодействия.

Подобная организация архитектуры позволяет удовлетворить большинство требований, предъявляемых к построению архитектур.

Зачастую архитектуры АСО создаются под определенную область задач. Преимуществом предложенной автором архитектуры является ее универсальность – данная архитектура может быть использована для подготовки персонала практически в любой предметной области.

Ядро содержит базовые компоненты, на которых строится функциональность всей остальной системы. Оно включает в себя как методы взаимодействия с операционными системами, так и компоненты, отвечающие за определенные функции, такие как управление виртуальной сценой, управление ресурсами и т.д. (менеджеры).

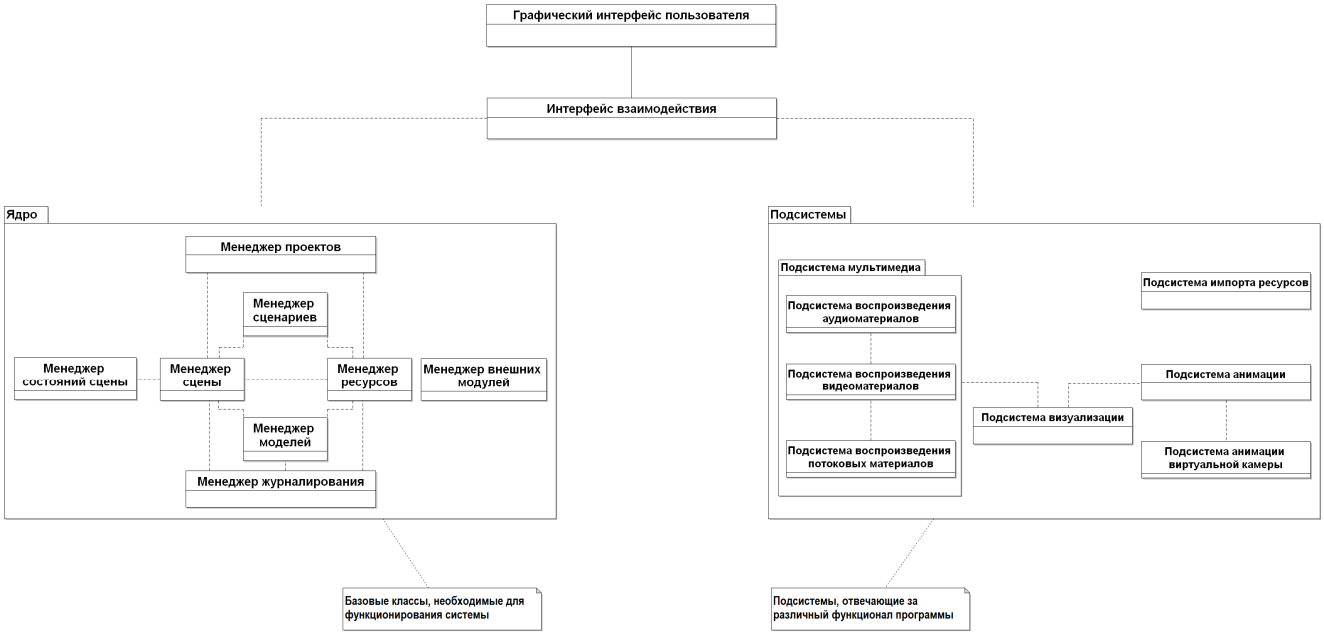

Рис. 2.1. Предложенная архитектура системы предтренажерной подготовки

Ядро содержит следующие типы менеджеров: менеджер журналирования, менеджер внешних модулей, менеджер ресурсов, менеджер проектов, менеджер моделей, менеджер состояний сцены, менеджер сцены и менеджер сценариев.

Менеджер журналирования используется для записи событий, происходящих в системе. События делятся на четыре категории: ошибка, предупреждение, информация, отладочная информация для разработчика. Полученные сообщения могут записываться в файл журнала и отображаться в графическом интерфейсе пользователя. Примерами событий могут являться ошибка открытия файла, предупреждения о выходе определенного параметра за заданные границы значений, информация об успешности выполнения определенных операций.

Менеджер ресурсов отвечает за поиск, загрузку и передачу ресурсов другим компонентам системы. Менеджер позволяет работать с такими типами ресурсов, как аудио файлы, видео файлы, изображения, файлы сценариев, текстовые файлы, файлы с информацией о виртуальной сцене (геометрия, материалы, шейдеры и т.д.). Поддерживаются все популярные форматы для файлов аудио, видео и изображений. Менеджер ресурсов содержит список ресурсов и обеспечивает, чтобы ресурсы были загружен только один раз и в последствие использовались повторно. Для более эффективного управления ресурсами, менеджер применяет счетчик ссылок на ресурс. Ресурс может быть удален, только если на него больше не ссылаются объекты, такие как узел сцены или другой ресурс. Существует возможность сбора мусора для выгрузки и удаления всех неиспользуемых ресурсов.

Менеджер сцены обеспечивает работу с различными типами узлов виртуальной трехмерной сцены. Основными типами узлов сцены являются: камера, источник света, геометрическая модель, группа объектов. Менеджер сцены обеспечивает создание графа сцены, реализованного в виде дерева. Каждый узел может иметь неограниченное число дочерних узлов, но существуют некоторые ограничения на то, какие типы узлов могут быть присоединены к какому типу родительского узла. Только такие сущности, как модели, частицы и т.д., имеющие трансформации и, соответственно, местонахождение в виртуальном мире, представляются как узлы. Все другие абстрактные сущности, такие как материалы, представлены как свойства узлов. Данный подход позволяет

сократить количество узлов в графе и повысить производительность операций над графом.

Менеджер сценариев позволяет задавать работу системы предтренажерной подготовки при помощи программируемых сценариев, записываемых на динамическом языке программирования Lua. Применение языка Lua дает возможность реализовывать различные алгоритмы в пользовательских сценариях, например, расчет динамических воздействий на основе физических законов на объекты трехмерной сцены. Менеджер позволяет получить доступ к компонентам системы предтренажерной подготовки из сценариев, и обеспечивает выполнение обработки сценария отдельно от основных вычислений, производимых системой предтренажерной подготовки. Существует возможность импорта функций, записанных в одном файле сценария, в другие файлы, что бывает необходимо при больших объемах пользовательских программ.

Менеджер состояний сцены сохраняет каждое внесенное пользователем изменение в виртуальную трехмерную сцену. Данный менеджер обеспечивает возможность отмены внесенных изменений, восстановление ранее выполненных действий или откат виртуальный сцены к первоначальному состоянию.

Менеджер моделей осуществляет управление моделями, используемыми для взаимодействия с графическим интерфейсом пользователя. Модели получают данные из нижестоящих компонентов, таких как менеджер сцены или менеджер ресурсов, и представляют данные в виде, задаваемом пользователем. Примером модели может служить модель подстановщика подсказок. Она получает данные из других моделей, для отображения подсказок пользователю при создании программируемых сценариев на основании введенных данных.

Менеджер проектов осуществляет создание, загрузку и сохранение проектов системы предтренажерной подготовки. В проекте сохраняются параметры системы, в том числе используемая виртуальная трехмерная сцена и ссылки на пользовательские данные.

Менеджер внешних модулей позволяет осуществлять поддержку компонентов, разработанных сторонними разработчиками. Менеджер

обеспечивает связь между внешними модулями и внутренними компонентами системы, предоставляя доступ ко всем необходимым внутренним интерфейсам. Примером внешнего модуля может служить модуль поддержки плат захвата данных с внешних устройств.

Для расширения базовой функциональности системы предтренажерной подготовки используются подсистемы. На данный момент существуют подсистемы импорта ресурсов, визуализации, мультимедиа и анимации.

Подсистема импорта обеспечивает внесение ресурсов с трехмерными виртуальными сценами в систему предтренажерной подготовки, а также их конвертирование в формат, используемый системой. Трехмерная виртуальная сцена создается в приложениях создания цифрового контента, таких как Autodesk 3ds Max, Maya, Modo, Lightwave 3D, Blender. Ресурс экспортируется из приложений создания цифрового контента в формате Collada. Этот формат является открытым, основанным на языке разметки XML, и, на данный момент, является стандартом индустрии для обмена трехмерными данными между различными приложениями [78]. Ресурс в формате Collada конвертируется подсистемой импорта ресурсов в формат, используемый системой предтренажерной подготовки: создаются файлы геометрии, файлы графа сцены и файлы материалов. Кроме того, если указанные в файле Collada файлы текстур, используемые геометрическими моделями виртуальной сцены, были найдены, то они будут скопированы в папку виртуальной сцены.

Подсистема визуализации обеспечивает отображение виртуальной трехмерной сцены в режиме реального времени (не менее 25 кадров в секунду). В основе подсистемы лежит современный графический модуль Horde3D, основанный на графическом интерфейсе OpenGL. Применяется современный метод рендеринга с помощью шейдеров, позволяющий реализовывать такие эффекты, как размытие движения, отражения, засветки, туман, режим ночного видения, HDR и т.д. Поддерживается работа с системами частиц, скелетной анимацией, морфингом.

Подсистема анимации является дополнением к подсистеме визуализации, и позволяет дополнить систему низкоуровневой анимации, встроенную в подсистему визуализации. Подсистема позволяет осуществлять высокоуровневое управление анимацией виртуальных камер и объектов трехмерной сцены. Одной из особенностей подсистемы анимации является возможность воспроизведения определенных кадров анимации, а не всей анимации целиком. Кроме того, присутствует возможность воспроизведения анимации в обратном порядке расположения кадров.

Подсистема мультимедиа осуществляет воспроизведение аудио- и видеоматериалов в системе предтренажерной подготовки. Подсистема способна воспроизводить все современные форматы мультимедиа, такие как mp3, wma, ogg, flac, ogv, mp4, avi, mkv и т.д. Воспроизведение возможно как из файлов, так и потоковых источников, таких как видеокамеры и системы приема звуковых данных. Подсистема позволяет воспроизводить одновременно несколько видеоматериалов высокой четкости в виртуальной трехмерной сцене. Количество воспроизводимых видеоматериалов ограничено только производительностью аппаратной части системы.

Глава 3 . Архитектура подсистемы визуализации

Архитектура подсистемы визуализации состоит из графического модуля Horde3D, являющегося ядром подсистемы, и расширений, отвечающих за работу с анимацией камеры, полупрозрачными изображениями и системами координат, разработанных автором, направленных на приведение графического модуля к требованиям к подсистеме визуализации ТОС. Расширения представляются в виде отдельных от графического модуля надстроек, и являются отдельными подсистемами внутри подсистемы визуализации.

3.1. Требования к подсистеме визуализации

Одним из основных компонентом тренажерно-обучающих систем является подсистема визуализации. К подсистеме визуализации предъявляются жесткие требования:

- Подробная виртуальная модель объектов, а также окружающей среды (звезды, водная и земная поверхность и т.д.);
- Возможность применения сглаживания (антиалиасинг) для устранения артефактов на границах объектов, исчезновения или мигания мелких деталей;
- Работа в реальном режиме времени, т.е. визуализация сцены со скоростью не менее 25 кадров в секунду;
- Имитация внешних условий исследуемого процесса или объекта;
- Имитация приборов и специальных средств наблюдения, а также реальных условий работы, включающих помехи, засветки, блики от оптических приборов и т.д.;
- Синхронизация каналов, если их несколько, то есть сшивка соседних изображений без искажений.

Также подсистема визуализации должна обладать следующими свойствами:

- Современная архитектура, основанная на шейдерах;

- Работа на нескольких платформах (Windows, Linux, Macintosh) с использованием графического слоя OpenGL;
- Низкая зависимость от внешних компонентов;
- Модульная архитектура и высокий уровень абстракции позволяют легко встраивать графический движок в системы;
- Возможность работы в режиме стереоизображения.

Управление сценами и ресурсами

- Система управления ресурсами со встроенным сборщиком мусора;
- Наличие интерфейса для загрузки данных из файлов, потоков или архивов;
- Быстрая перезагрузка ресурсов для повышения продуктивности при разработке;
- Легкая структура дерева сцены, поддерживающая трансформации иерархии;
- Единая система сцены, где мир и модели являются листьями дерева сцены и не являются специальными объектами;
- Загрузка объектов сцены из файлов XML;
- Использование аппаратной техники occlusion culling (отсечение поверхностей, находящихся за полностью видимыми поверхностями для повышения производительности);
- Поддержка уровня детализации для геометрии моделей и материалов;
- Возможность присоединять объекты сцены к костям персонажа (например, к руке);
- Доступ к параметрам вершин для определения столкновений и взаимосвязи с физическими модулями;

Рендеринг

- Использование системы ubershader для автоматической компиляции шейдеров при изменении параметров (используются GLSL шейдеры);
- Изменяемый конвейер рендеринга, основанный на XML, используемый для различных техник рендеринга;
- Пост-обработка: эффекты свечения (bloom), глубины резкости или смазывания движения;
- Поддержка прямого рендеринга и различных техник рендеринга с отложенным освещением;
- Поддержка освещения и текстур HDR (High Dynamic Range);
- Поддержка различных техник рендеринга, включая normal-mapped, освещение по Фонгу и parallax mapping;
- Поддержка отражений в реальном времени и других техник, требующих нескольких камер для рендеринга;
- Поддержка теней в реальном времени при помощи Parallel Split Shadow Maps (PSSM);
- Программное и аппаратное создание кожи (skinning) в вершинном шейдере для визуализации сотен анимированных персонажей;
- Системы частиц, способные отбрасывать тени и обрабатывать эффекты типа смазывания движения (motion blur);
- Наложение (overlays) для визуализации элементов интерфейса пользователя и шрифтов.

Анимация

- Единая низкоуровневая система анимации, работающая напрямую с графом сцены;
- Ключевая анимация для костей и моделей;
- Возможность применения до четырех весов на вершину для сочлененных моделей (скелетная анимация);

- Смешивание анимаций по уровням, а также с использованием масок и каналов аддитивности (additive channels);
- Интерполяция между кадрами для получения плавной анимации;
- Получение параметров костей для динамической анимации и ragdoll-физики;
- Морфинг для лицевой анимации и синхронизации губ.

Работа с ресурсами

- Собственные форматы моделей и анимации для получения максимальной производительность
- Использование бинарных форматов и XML для компромисса между производительностью и продуктивностью
- Поддержка формата текстур DDS и других форматов изображений
- Импорт ресурсов из приложений моделирования через формат Collada
- Вычисление касательного пространства для использования normal mapping
- Редактируемый конвейер рендеринга для быстрого переключения между различными техниками рендеринга

- Редактор для создания сцен и разработки шейдеров, оперирующий с файлами сцены в формате движка

3.2. Архитектура базового компонента подсистемы визуализации – графического модуля Horde3D

3.2.1. Концепции Horde3D

В Horde3D применена стратегия интеграции, отличная от других хорошо известных графических движков, таких как OGRE 3D или Irrlicht. Данные графические движки предоставляют объектно-ориентированную библиотеку классов, используя которые пользователь может создать собственные реализации. Horde3D можно скорее считать программным компонентом – по сравнению с другими движками, этот графический модуль использует более высокий уровень абстракции, а основная функциональность доступна через небольшой процедурный интерфейс, в некоторых аспектах схожий с Microsoft Win32 API. Преимуществом простого С-интерфейса является большая наглядность и простота изучения функциональности и особенностей движка. Другим преимуществом является лучшая портируемость. Практически в любом языке программирования, в том числе и скриптовых, есть механизм доступа к функциям внешних библиотек, в то время как импорт классов намного сложнее или вовсе невозможен. Horde3D может использоваться в таких языках, как C#, Java, LUA, Python и многих других.

Процедурный интерфейс не мешает использовать движок в объектно-ориентированных языках программирования. Сам движок является полностью объектно-ориентированным. Объекты движка, такие как узлы графа сцены или ресурсы предоставляются приложению через дескрипторы. Дескриптор во многом похож на указатель, но работает не напрямую с объектом для большей безопасности. Horde3D использует специальные функции для создания объектов, которые всегда возвращают дескриптор созданного объекта или нулевой дескриптор (аналогичный нулевому указателю) в случае ошибки. Дескриптор должен храниться в приложении и может быть использован для изменения свойств объекта или его выгрузки.

Из-за высокого уровня абстракции интерфейса прикладного программирования (API) пользователь не может добавлять новую функциональность, например, новый тип узла сцены, без модификации исходных кодов движка. Для преодоления этого недостатка, Horde3D предоставляет механизм расширений, который дает пользователю полный доступ к внутренней структуре движка. Расширение статически присоединяется к библиотеке движка и предоставляет свою функциональность через процедурный интерфейс.

Horde3D основан на графической библиотеке OpenGL. Основой для этого API является ядро, которое обрабатывает примитивные объекты (как правило, треугольники). Сам интерфейс (для программиста) содержит несколько сотен процедур и функций, необходимых для создания высококачественных графических изображений. Чтобы аппаратно реализовать многие функции (например, сглаживание или текстурирование) от видеокарты требуется наличие кадрового буфера, причем некоторые операции могут выполняться исключительно в нем.

OpenGL ориентируется на следующие две задачи:

- Скрыть сложности адаптации различных 3D-ускорителей предоставляя разработчику единый API.
- Скрыть различия в возможностях аппаратных платформ, требуя реализации недостающей функциональности с помощью программной эмуляции.

Основным принципом работы OpenGL является получение наборов векторных графических примитивов в виде точек, линий и многоугольников с последующей математической обработкой полученных данных и построением растровой картинки на экране и/или в памяти. Векторные трансформации и растеризация выполняются графическим конвейером (graphics pipeline), который по сути представляет из себя дискретный автомат. Абсолютное большинство команд OpenGL попадают в одну из двух групп: либо они добавляют графические примитивы на вход в конвейер, либо конфигурируют конвейер на различное исполнение трансформаций.

OpenGL является низкоуровневым процедурным API, что вынуждает программиста диктовать точную последовательность шагов, чтобы построить результирующую растровую графику (императивный подход). Это является основным отличием от дескрипторных подходов, когда вся сцена передается в виде структуры данных (чаще всего дерева), которое обрабатывается и строится на экране. С одной стороны императивный подход требует от программиста глубокого знания законов трёхмерной графики и математических моделей, с другой стороны — даёт свободу внедрения различных инноваций.

Для работы с библиотекой программа сначала должна открыть окно в кадровом буфере (где будет происходить все операции), и потом уже контекст библиотеки связывается с этим окном. Такую операцию нужно проделать один раз, а далее все действия определяются через параметры обработки (рисование простых геометрических объектов: линии, точки, полигоны; рендеринг примитивов, включая их положение и цвет).

Сама же библиотека реализована в виде информации о структуре, определяющей, как объекты будут нарисованы в кадровом буфере, причем некоторые из структур доступны пользователю, который может создавать различные вызовы процедур для изменения параметров [79].

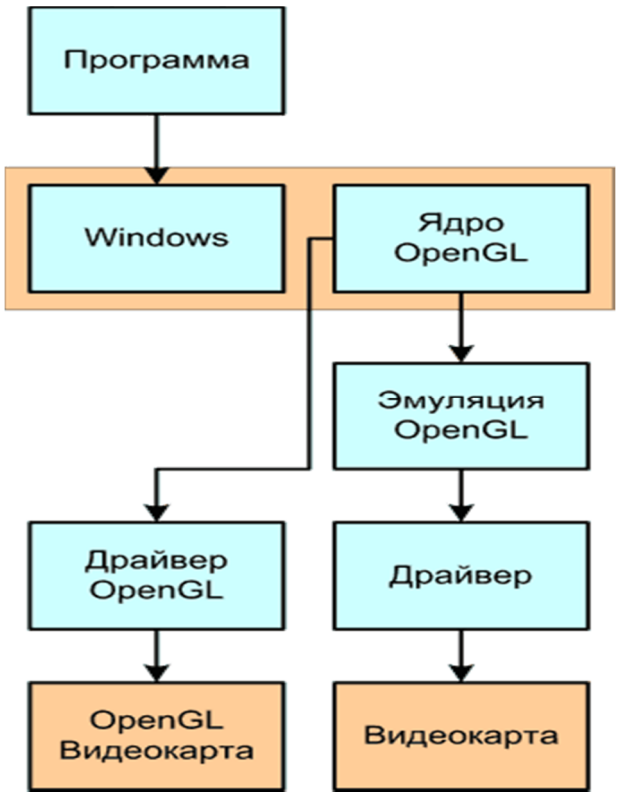

Рис. 3.1. Структурная схема работы с OpenGL приложениями

3.2.2. Концепция программируемого конвейера

Одной из основных причин гибкости движка Horde3D является глубокая интеграция систем шейдеров и программируемого конвейера, позволяющего реализовывать различные техники рендеринга. В Horde3D ресурсом программируемого конвейера является XML документ, описывающий этапы процесса рендеринга. Программируемый конвейер может определять цели рендеринга, служащие выходными буферами и хранящими временные результаты рендеринга. Цели рендеринга могут быть различного размера и формата, а также могут содержать несколько буферов цвета и глубины. Несколько команд программируемого конвейера определяют, какая геометрия рендерится в какой буфер и каким образом. Выбор прорисовываемой геометрии определяют классы материалов, а техника рендеринга определяется выбранным контекстом шейдера. Контексты шейдеров будут рассмотрены подробнее немного позже. Команды объединены в этапы, каждый из которых можно включить или выключить. После заполнения буфера, его можно использовать снова в качестве входной текстуры

на следующем этапе рендеринга. Такое использование входных и выходных буферов позволяет реализовывать множество различных эффектов пост-обработки и улучшенные техники рендеринга.

3.2.3. Описание аппаратных шейдеров

Одним из основных элементов системы рендеринга Horde3D являются шейдеры. Шейдеры – это программы, выполняемые на графической карте. У всех видеокарт есть аппаратный конвейер, обрабатывающий входящие полигональные данные и выводящий их на экран. Этот конвейер состоит из нескольких этапов, таких как трансформация входящих вершин, растеризация треугольников и расчет окончательного цвета пикселей в буфере кадра. Ранее этот конвейер был фиксирован, и настраивался в очень узких пределах (fixed function pipeline). Поколения видеокарт после NVidia GeForce 2 и ATI Radeon 9000 имели программируемый конвейер, позволяющий управлять различными этапами обработки изображения при помощи программных команд.

Существует шесть типов шейдеров, настраивающих этапы рендеринга изображения: вертексные (вершинные), геометрические, управления тесселяцией и тесселирования, фрагментные (пиксельные). Также существуют вычислительные шейдеры, которые являются отдельным классом шейдеров.

Вершинный шейдер выполняется для каждой вершины входных данных. Основной его задачей является трансформация позиции вершины в трехмерном пространстве и вычисление ее позиции на двумерной поверхности для выведения на экран. Вершинный шейдер также может оперировать такими свойствами вершин, как цвет или текстурные координаты, а также передавать результаты своей работы в следующий этап рендеринга. Вершинный шейдер не может создавать или удалять вершины.

Геометрические шейдеры могут создавать новые примитивы, такие как треугольники или линии. Они выполняются после вершинных шейдеров. Геометрические шейдеры полезны для реализации таких алгоритмов, как

тесселяция и управления просчетом теней, полностью на видеокарте. Геометрические шейдеры поддерживаются только на видеокартах, использующих DirectX 10 или OpenGL 3.3.

Фрагментный шейдер (также известный как пиксельный) рассчитывает цвет и глубину каждого пикселя в выходном буфере. Из-за того, что в настоящий момент большинство операций освещения выполняются для пикселей, а не для вершин, пиксельные шейдеры используются для реализации большинства визуальных эффектов.

Шейдеры тесселяции применяются для динамического изменения уровня детализации трехмерных объектов. Ярким примером использования шейдеров тесселяции является отображение виртуальных ландшафтов без резкого появления/удаления объектов или видимого изменения их детализации.

Вычислительные шейдеры применяются для проведения общих вычислений на видеокартах. Могут применяться для расчета позиций большого количества частиц, или например, управления прорисовкой освещения трехмерной сцены.

Первые графические карты с программируемым конвейером имели серьезные ограничения на количество операций, и программировались с использованием ассемблера. Сегодня, все основные графические API предоставляют языки высокого уровня для написания шейдеров с синтаксисом, схожим с Си. Основными языками написания шейдеров являются: HLSL, используемый в Direct X, GLSL, используемый в OpenGL, и Cg от NVidia. Horde3D использует язык GLSL для написания шейдеров.

3.2.4. Шейдеры и материалы в Horde3D

В Horde3D понятие шейдера расширено, по сравнению с аппаратным шейдером, описанным выше. Шейдер в Horde3D – это группа шейдерных программ, где пара вершинных и пиксельных шейдеров называется контекстом шейдера. Контексты шейдера были введены потому, что полигональную сетку необходимо по-разному обрабатывать на различных этапах рендеринга. Расчет

карты теней требует шейдер, отличный от необходимого для расчета влияния источника света на объект, или, например, для записи информации о движении в буфер данных для создания эффекта смазывания движения. Шейдер в Horde3D предоставляет все необходимые шейдерные программы как контексты, а программируемый конвейер выбирает необходимый для конкретного этапа рендеринга контекст.

Каждый шейдерный контекст имеет уникальное имя. Большинство команд программируемого конвейера позволяют задавать используемый для рендеринга контекст шейдера. Следует отметить, что у источников света также есть два свойства, определяющий шейдерные контексты, используемые для расчета теней и освещения при использовании прямого освещения. Если заданный контекст шейдера не найден или не указан в шейдере, то связанная с этим контекстом геометрия игнорируется и, соответственно, не прорисовывается.

Аппаратные шейдеры, указываемые в шейдерных контекстах, являются всего лишь описанием того, как следует обрабатывать данные. Для работы им необходимы входные данные. Одной частью входных данных является информация о геометрии, которую рендерер извлекает из графа сцены и соответствующих ресурсов геометрии. Другой частью являются текстурные карты и константы. В языке программирования шейдеров OpenGL (GLSL) эти данные получаются шейдером через переменные, называемые uniforms. Материалы в движке Horde3D служат промежуточным звеном между шейдером и значениями переменных. Материал ссылается на шейдер, которому он передает данные текстур. Более того, при помощи материала возможно задание переменных векторного типа, используемых для обмена данными между шейдером и приложением. Примером использования векторной переменной может служить установка свечения объекта из приложения.

Материалы образуют классовую иерархию. Некоторые команды программируемого конвейера позволяют при помощи этого свойства рендерить только определенную геометрию. Например, при помощи классов можно прорисовывать только полупрозрачную геометрию на определенном этапе

программируемого конвейера. Необходимо отметить, что система классов материала иерархична, то есть материал может быть частью нескольких подклассов.

Еще одной особенностью материалов является возможность ссылаться на другой материал. Связь между материалами позволяет импортировать переменные и привязки текстур из других материалов, что очень удобно для объявления глобальных параметров в одном единственном месте. Например, возможно задать интенсивность глобального освещения сцены в одном материале, на который ссылаются все остальные материалы. Это позволяет избежать необходимости обновления значения интенсивности освещения во всех материалах при его изменении.

3.2.5. Тени и освещение

По умолчанию Horde содержит два подхода обработки освещения – прямой и отложенный. Прямое освещение - это стандартная техника, используемая в большинстве приложений. При использовании данной техники геометрия прорисовывается для каждого источника света, используя специальный пиксельный шейдер, рассчитывающий влияние света. Главным недостатком этого подхода является необходимость перерисовывать геометрию столько раз, сколько в сцене источников света, что может значительно снизить производительность в сценах с большим количеством полигонов и источников света. Отложенное освещение рассчитывает освещение сцены как шаг пост обработки. Необходимая информация о всех пикселях экрана хранится в специальном буфере, называемом G-Buffer. Обычно в эту информацию входят позиции фрагмента, нормали и цвет. Для применения освещения к геометрии необходимо только нанести рассчитанную модель света для кадра на текстуру геометрии, хранящуюся в G-Buffer. Преимуществом данного подхода является независимость производительности расчета света от сложности геометрии сцены, т.к влияние всех источников света на геометрию рассчитывается один раз. Это позволяет реализовать сложное динамическое освещение в сценах, в отличие от

статического освещения, рассчитываемого заранее в карты освещения. Примером может служить реализация на движке Horde сцены с более чем 200 перемещающимися источниками света в реальном времени. Недостатками данного метода является сложность реализации механизма сглаживания и обработки полупрозрачных объектов, а также высокие требования к производительности и объему видеопамяти видеокарты.

Источники света в движке Horde определяются как узлы графа сцены. Каждый источник света имеет два особых свойства – контекст освещения (lightingContext) и контекст теней (shadowContext). Значения этих свойств связаны с именами контекстов, определенных в шейдерах. При получении движком команды на прямую обработку света, сначала происходит создание списка геометрии, которую необходимо вывести на экран. После этого каждый объект из этого списка прорисовывается с учетом контекста шейдера, указанного для данного источника света. Если контекст, указанный в материале, не найден, объект, использующий данный материал, игнорируется и, соответственно, не рендерится. Те же шаги применяются при прорисовке освещения и построении карты теней. При использовании отложенного освещения необходимо также указывать материал для источников света. Шейдер, указанный в материале, используется для прорисовки квадов в плоскости экрана (квад – закрашенный прямоугольник, состоящий из двух треугольников), а свойство источника света lightingContext определяет используемый контекст шейдера. Расчет теней происходит по схожей схеме при использовании прямого освещения, когда вся геометрия рендерится в карту теней, используя контекст шейдера, указанный в shadowContext. При использовании прямого освещения вы также можете задать материал для источника света. В этом случае шейдер будет проигнорирован, но появится возможность привязать несколько текстур, которые могут быть использованы, например, для создания эффекта проектора слайдов.

Благодаря системе контекстов шейдеров возможно определение произвольных (пользовательских) видов источников света. Поведение источника света (например, направленный или световое пятно) и его взаимодействие с

материалами полностью определяются в шейдерах. При создании нового типа света необходимо присвоить имя свойству lightingContext, а также определить соответствующие контексты в шейдере, описывающем функциональность данного источника света, для всех материалов (и шейдеров), взаимодействующих с данным типом источников света.

3.2.6. Граф сцены

Граф сцены используется для представления логической или пространственной структуры сцены, которую необходимо визуализировать. Обычно он также используется для ускорения поиска по узлам сцены и таких операций, как определение столкновений между узлами и отсечение по пирамиде видимости. На практике, существует множество реализаций графа сцены, каждая со своими достоинствами и недостатками. Некоторые движки применяют направленные и ненаправленные графы, а некоторые даже используют циклы. Другие же используют более простой вариант деревьев. Другим важным вопросом является отношения объектов и их свойств. Например, как следует реализовывать материал полигональной сетки? Следует ли ему быть родительским узлом полигональной сетки, дочерним узлом или просто свойством? Возможно создать очень мощную систему, но, зачастую, это приведет к потере гибкости системы и снижению производительности, ведь более сложные структуры графов требуют больше процессорного времени для обработки. Так как Horde3D был изначально предназначен для рендеринга большого количества анимированных персонажей, в нем найден баланс между гибкостью и связанными с ней падениями производительности.

В Horde3D граф сцены реализован в виде дерева. Каждый узел может иметь неограниченное число дочерних узлов, но движок накладывает некоторые ограничения на иерархию, и определяет, какие типы узлов могут быть присоединены к какому типу родительского узла. Только такие сущности, как модели, частицы и т.д., имеющие трансформации и, соответственно, местонахождение в виртуальном мире, представляются как узлы. Все другие

абстрактные сущности, такие как материалы, представлены как свойства узлов. Данный подход позволяет сократить количество узлов в графе и повысить производительность операций над графом сцены.

Для каждого узла применимо два типа трансформаций – локальные и глобальные. Локальные трансформации, иногда так же называемые трансформациями объекта, относительны к родительскому объекту и могут быть явно заданы в приложении. Глобальные трансформации (мировые трансформации) рассчитываются автоматически - путем прохода по иерархии родительского – дочерних объектов и накопления трансформаций. То есть, когда происходит трансформация узла, все его дочерние объекты также трансформируются. Например, если необходимо повернуть всю сцену, достаточно повернуть только корневой узел, и движок автоматом повернет все объекты сцены. Также у каждого узла есть центрированный по осям ограничивающий куб, который является объединением ограничивающих кубов его дочерних объектов. Подобная иерархия используется движком для оптимизации алгоритмов определения видимости.

Следующие типы узлов сцены доступны в Horde3D:

- **Группа (Group).** Группа – это контейнер для других узлов. Например, типом корневого узла является группа. У узла группы есть два параметра, определяющих минимальную и максимальную дистанцию, при которой узел (и все его дочерние объекты) видны. Эти свойства полезны для реализации функционала простого уровня детализации, основанного на дистанции.
- **Камера (Camera).** Сцена всегда прорисовывается с позиции виртуальной камеры, также являющейся узлом сцены в Horde3D. Благодаря этому камерам доступна вся гибкость графа сцены, что позволяет присоединять камеру к сочленению или анимированной модели, чтобы позиция камеры автоматически трансформировалась вместе с анимацией модели.
- **Свет (Light).** Источники света используются для задания освещения в сцене. У узла есть такие параметры, как цвет света, радиус действия и зона

охвата - для источников света типа spot light (размытое пятно). С помощью этих параметров также могут быть настроены тени. Возможно присоединение материалов к источникам света. Это необходимо для отложенного освещения, когда освещение выполняется как этап пост-обработки при помощи шейдера. Для осуществления прямой обработки света, источники света используют свойство контекста освещения, определяющего используемый контекст шейдера для расчета интенсивности освещения.

- **Модель (Model).** Модель – это полигональный трехмерный объект, обычно экспортируемый из приложений моделирования, таких как 3ds Max или Blender. Модель может быть статической или динамической. Узел модели ссылается на ресурс геометрии, хранящий полигональные данные. Узлы служат контейнером для полигональных сеток и сочленений, и могут быть анимированы с помощью ресурса анимации. Модель сама по себе не отображается, и будет видима только, если присутствует полигональная сетка. Обычно модели с полигональными сетками и сочленениями автоматически создаются приложениями экспорта и конвертирования, и хранятся в ресурсе графа сцены.
- **Полигональная сетка (Mesh).** Узлы полигональных сеток определяют группы полигонов с единым материалом. Каждая полигональная сетка является частью ресурса геометрии и прорисовывается движком, используя указанный материал. Так как полигональные сетки ссылаются на ресурс геометрии, им необходим родительский объект – модель, находящийся выше по иерархии.
- **Сочленение (Joint).** Иерархия узлов сочленений используется для представления скелета, используемого в скелетной анимации. Как и полигональные сетки, узлам сочленений необходим родительский объект – модель. У каждого сочленения есть индекс, указывающий на соответствующую матрицу в ресурсе геометрии модели. Эта матрица

используется движком для применения скиннинга (привязка вершин модели к анимированному скелету).

- **Генератор частиц (Emitter).** Узлы генераторов частиц используются для создания систем частиц. Частицы создаются на основе информации из ресурса частиц, и обладают такими параметрами, как максимальное количество частиц и объем генерации.

Каждый тип узла имеет свою функцию создания, принимающую различные параметры. Большинство параметров можно изменить после создания, но некоторые настройки, затрагивающие внутреннюю структуру, изменить нельзя. Базовые параметры, общие для всех типов узлов сцены, изменяются стандартными функциями API. Для получения значений свойств, специфичных для конкретного типа, API предоставляет специальные функции, в которые, в большинстве случаев, необходимо передавать дескриптор обрабатываемого узла сцены.

3.2.7. Управление ресурсами

Ресурс – это объект данных, необходимый для рендеринга сцены, например, текстура или шейдер. Одним важным свойством ресурсов является возможность их повторного использования. Это значит, что на ресурс может ссылаться несколько объектов, таких как узлы графа сцены, но загружен он будет только один раз. Каждому ресурсу присваивается имя, уникальное для каждого типа ресурсов, при помощи которого этот ресурс смогут использовать различные объекты.

Все ресурсы управляются менеджером ресурсов. Менеджер ресурсов содержит список ресурсов и обеспечивает, чтобы ресурсы были загружен только один раз и в последствие использовались повторно. Менеджер ресурсов также используется для поиска ресурсов, доступа к ним и удаления. В отличие от некоторых других движков, Horde3D использует единый менеджер для всех типов ресурсов, а не отдельные менеджеры для каждого типа ресурсов. Для более

эффективного управления ресурсами, менеджер применяет счетчик ссылок на ресурс. Ресурс может быть удален, только если на него больше не ссылаются объекты, такие как узел сцены или другой ресурс. Существует возможность сбора мусора для выгрузки и удаления всех неиспользуемых ресурсов.

Horde3D использует отложенную загрузку ресурсов. Это означает, что создание ресурса и его загрузка являются отдельными этапами. Преимуществом такого разделения является то, что ресурс не должен быть доступен сразу же после создания, а может быть загружен в фоне в отдельном потоке. При создании ресурса, он инициализируется с данными по умолчанию, специфичными для конкретного типа данных, а другие объекты получают возможность ссылаться на этот ресурс. После создания ресурс может быть загружен и заполнен необходимыми данными. При создании ресурса задается его базовая структура. После загрузки ресурса в большинстве случаев есть возможность изменить значения данных в ресурсе, но не поменять его структуру. Например, для файла текстуры это означает, что есть возможность поменять значение цвета всех пикселей, но поменять размеры текстуры или глубину цвета изображения вы не сможете. После загрузки ресурса, его нельзя загрузить снова, но есть возможность выгрузить ресурс и, тем самым, привести его в начальное состояние, в котором он был сразу после создания.

В Horde3D загрузка ресурсов полностью виртуализирована. Движок не производит никакого доступа к файлам, а ожидает указатель на блок памяти от приложения. Именно в приложении указывается, откуда получить данные – из памяти, из файла, по сети. Преимуществом этой системы является независимость от конкретного формата архивов. Ресурсы можно получить из любого источника и даже получать в виде потоковых данных по сети. Возможно также процедурное генерирование ресурсов (например, геометрии).

Следующие типы ресурсов используются ядром Horde3D:

- **Граф сцены.** Ресурс графа сцены – это XML файл, содержащий ветвь графа сцены. Каждый элемент XML файла считается узлом сцены согласно его

имени. Иерархия, указанная в XML файле, копируется в иерархию сцены. Все свойства узлов сцены, доступные через API движка, могут быть настроены в XML файле. Все трёхмерные модели, статические и динамические, представлены в Horde3D как файлы графа сцены. Это позволяет осуществлять доступ к компонентам модели, таким как полигональная сетка или сочленения, используя API графа сцены, без использования дополнительных функций.

- **Геометрия.** Ресурсы геометрии загружаются из бинарных файлов, хранящих полигональные данные, используемые моделями и полигональными сетками (meshes). Ресурс содержит индексы треугольников, а также такие свойства вершин, как позиции, нормали, касательные, координаты текстур и вес сочленений. Также в ресурсе может храниться информация о скелете модели и отдельные данных для морфинга (трансформации позиций вершин геометрической модели).
- **Анимация.** Ресурс анимации хранит данные ключевых кадров для анимированных моделей. Horde3D поддерживает анимацию полигональных сеток и сочленений. Для всех анимированных узлов, определяемых по имени, записывается информация о каждом кадре анимации: вектор трансформации, вектор масштабирования и кватернион вращения.
- **Программируемый конвейер.** Программируемый конвейер – это XML документ, определяющий этапы рендеринга сцены. Программируемый конвейер состоит из списка команд рендеринга, организованных в этапы, определяющих какая геометрия рендерится в какой буфер. Каждый файл программируемого конвейера имеет произвольное количество различных буферов, в которые выводятся данные, называемых целями рендеринга. Цели рендеринга могут хранить промежуточные результаты рендеринга, необходимые на следующих этапах. При использовании с шейдерами, программируемый конвейер является мощным средством, позволяющим реализовать многие современные техники рендеринга, такие как

отложенное освещение и многие другие эффекты пост-обработки, такие как HDR, parallax mapping и смазывание движения.

- **Материал.** Материалы определяют внешний вид полигональной сетки или объекта рендеринга в целом. Материал содержит ссылку на шейдер и задает значения переменных, используемых в шейдере. Такими переменными (uniforms) могут быть текстурные карты или вектора, передающие необходимые данные в шейдер.
- **Шейдер.** Шейдер в Horde3D, по сравнению с шейдером стандарта GLSL, отличается расширенным функционалом, так как это не просто код, исполняемый на графической карте. Основной идеей шейдера в Horde3D является то, что модель необходимо прорисовывать на различных этапах процесса рендеринга при помощи различных шейдеров. Например, код расчета карты теней отличается от кода расчета рассеянного света. В Horde3D шейдер является группой программ, исполняемых в различных контекстах. Пара подпрограмм исполняемого кода (вершинная и пиксельная) называется контекстом шейдера. Программируемый конвейер выбирает нужный для конкретного этапа контекст шейдера. Код шейдера в контексте, исполняемый графической картой, должен соответствовать синтаксису OpenGL Shading Language.
- **Код.** Ресурс кода – текстовый файл, содержащий произвольный код. На файлы кода могут ссылаться шейдеры. Обычно используются для определения фрагментов программ шейдеров.
- **Текстуры.** Ресурсы текстуры содержит двумерную карту текстуры или кубическую карту. Двумерные текстуры являются изображениями, используемыми в шейдерах для определения вида поверхности, но они также могут содержать и не визуальные данные, такие как нормали или таблица преобразований. Кубические карты являются разновидностью текстурных карт, состоящие из шести различных изображений для каждой стороны куба. Кубические карты используются для отображения окружения вокруг некоторой точки в пространстве. Благодаря этому возможно

использовать трехмерный вектор для поиска текселя (пикселя текстуры), лежащего в нужном (соответствующем) направлении.

- **Эффекты частиц.** Ресурс эффектов частиц используется для настройки систем частиц. Ресурс определяет такие свойства частиц, как цвет, размер и скорость, изменяемая за время существования частицы. Ресурсы частиц обычно присваиваются генераторам частиц.

3.2.8. Система анимации Horde3D

Horde3D использует систему анимации, позволяющую производить анимацию твердых тел и скелетную анимацию, а также лицевую анимацию при помощи морфинга. Система поддерживает плавный переход от одного кадра анимации к другому, а также позволяет смешивать анимации (animation blending). Система анимации работает напрямую с графом сцены, поэтому возможна ручная трансформация объектов для динамической анимации, такой как обратная кинематика или Ragdoll-физика.

3.2.9. Комбинирование анимаций

Плавный переход от одной анимации к другой достигается за счет комбинирования анимаций. При изменении состояния персонажа от бездействующего к шагающему, или от шагающего к бегущему, комбинирование анимаций необходимо для получения приемлемых визуальных результатов. Для достижения этого модели в Horde3D могут применять одновременно несколько анимаций, на основе которых создается финальная анимация. У каждой анимации есть параметр «вес смешивания», определяющий степень влияния анимации на общий результат. Изменение анимации персонажа, например, от шага к бегу, может быть реализовано с помощью, так называемого, перекрестного исчезновения (cross-fade). В начале у анимации шага вес равен 1.0, а у анимации бега соответственно 0.0. Для выполнения перехода между анимациями вес анимации шага линейно уменьшается, в то время как весу анимации бега

прибавляется такое же значение. Наконец, вес анимации бега составляет 1.0, а вес анимации шага равен 0.0. Видимым результатом является бег персонажа.

3.2.10. Использование слоев анимации

Horde3D поддерживает слои анимации для облегчения переключения между анимациями. Слою анимации присваивается номер и идентификатор слоя, определяющий его приоритет. Слои с большим индексом имеют больший приоритет, и их анимации обрабатываются раньше, чем анимации слоев с меньшими индексами. Все анимации на одном слое имеют нормализованный вес смешивания. Например, если есть две анимации на слое 0, и вес каждой составляет 1.0, то финальная анимация будет состоять из 50 процентов обеих анимаций. Если сумма весов анимаций одного слоя меньше 1.0, то остаток передается нижним слоям. Например, если есть два слоя – 1 и 0, и сумма весов двух анимаций на слое 1 равна 0.6, то оставшийся вес, равный 0.4, доступен для анимаций слоя 0.

Предположим, что у нас есть персонаж, выполняющий на одном слое зацикленный переход от анимации шага к анимации бега, в зависимости от задаваемой скорости. Мы хотим заменить результирующую анимацию на анимацию выстрела при нажатии клавиши наблюдателем. Благодаря системе слоев это легко достижимо – достаточно поместить анимацию выстрела на более высокий слой, чтобы она имела более высокий приоритет перед другими анимациями. Вес анимации выстрела может быть быстро увеличен от 0.0 до 1.0. Так как у анимации выстрела будет больший приоритет, чем у цикла шаг-бег, которому будет доступен оставшийся вес анимации. В результате, цикл шаг-бег вначале полностью активен, затем его влияние на конечную анимацию значительно снижается до тех пор, пока в конце он не будет полностью заменен анимацией выстрела. Большим преимуществом данного подхода является то, что необходимо изменить только вес анимации выстрела, в то время как анимации шага и бега можно оставить нетронутыми.

3.2.11. Группирование различных анимаций

Другим обычным явлением, кроме перехода от одной анимации к другой, является необходимость выполнять персонажем одновременно нескольких перемещений. И хотя в теории возможно создание анимаций для всех комбинаций действий, это будет неэффективно как в плане использования памяти, так и в плане продуктивности художников и дизайнеров. Более эффективным решением является объединение анимаций движком в реальном времени. В отличие от смешивания анимаций, при группировании анимации обычно независимы друг от друга, и применяются к различным частям скелета. Поэтому анимации должны не комбинироваться, а проигрываться одновременно с полным весом. Horde3D предоставляет 2 подхода к реализации группирования анимаций.

Первым подходом является определение первого узла, с которого система анимации начнет применять анимацию. Сочленения, находящиеся в иерархии скелета выше стартового узла, игнорируются и, соответственно, не анимируются. Предположим, что необходимо одновременно проигрывать две анимации для персонажа – анимации шага и волновая анимация. Для применения обеих анимаций в полном объеме, стартовым узлом для анимации шага можно выставить узел на уровне таза, чтобы анимация применялась только к нижней части тела. Волновой анимации не нужен стартовый узел, и она может применяться ко всему скелету. Если анимация шага находится на более низком слое, чем волновая анимация, сумма весов нижней части тела уже составляет 1.0, и волновая анимация будет воздействовать только на верхнюю часть тела и игнорировать нижнюю часть.

Более простой реализацией группировки анимаций являются аддитивные анимации. Когда этап аддитивен, движок рассчитывает величину трансформации текущего кадра от первого кадра анимации и добавляет эту величину к трансформации сочленений, независимо от веса этапа. Для указанного примера это будет значить, что анимация шага будет присвоена всему телу персонажа. Волновая анимация может быть настроена аддитивной, что позволит применять

трансформации, например, только к руке. При корректной настройке результатом будет идущий персонаж с помахивающей рукой.

3.2.12. Применение процедурной анимации

Другим важным аспектом являются процедурная анимация, типа обратной кинематики или Ragdoll-физики. Сочленения являются простыми узлами сцены, и у вас есть полный контроль над трансформациями, благодаря функциям API. Так как система анимации работает напрямую с графом сцены, возможно получить трансформации после применения анимации, и использовать эти расчеты в системах обратной кинематики или Ragdoll-физики. Результаты, полученные из внешних систем, могут быть снова применены сочленениям для получения видимых результатов.

3.2.13. Структура шейдеров Horde3D

Шейдеры движка Horde3D хранятся в текстовых файлах с расширением *.shader*. Файл шейдера состоит из нескольких секций: секции FX и произвольного количества именованных кодовых секций. Секция начинается с тега [id], где *id* является уникальным идентификатором, используемым в качестве имени секции. Специальным идентификатором является **FX,** который характеризует начало секции FX.

Horde3D поддерживает технологию ubershader при помощи статического ветвления (директив компилятора), и может автоматически компилировать различные комбинации шейдеров в зависимости от выставленных флагов шейдера. Этот подход позволяет решить известную проблему: представим, что у вас есть один шейдер, выполняющий процедуру скиннинга, и второй шейдер, выполняющий parallax mapping. Предположим, что вам необходим шейдер, выполняющий обе эти функции. Вам пришлось бы создать файл шейдера, в котором объединен код двух первых шейдеров. Очевидно, что при добавлении новых функций написание шейдеров, учитывающих все необходимые

комбинации, становится проблематичным или даже невыполнимым. С использованием технологии ubershader такой проблемы нет. Вам необходим всего один файл шейдера, в котором все функции имеют директивы компилятора. Функция включается или отключается согласно значению соответствующего флага (например, в файле материала). Horde3D может автоматически проверять, существует ли требуемая комбинация функций, и, если нет, компилировать ее налету. Необходимо учитывать, что, несмотря на гибкость системы ubershader, ее необходимо применять с осторожностью, так как существует возможность создания огромного количества комбинаций функций, что приведет к длительному времени загрузки.

3.2.14. Описание секции FX шейдеров Horde3D

Секция FX определяет и настраивает контексты шейдера, пользовательские текстурные сэмплеры (структура, определяющая используемую для прорисовки объекта текстуру и ее параметры) и переменные (uniforms), используемые в аппаратных шейдерах. Необходимо отметить, что все пользовательские текстурные сэмплеры и переменные (uniforms), используемые в аппаратных шейдерах, обязательно надо указывать в секции FX, чтобы они были доступны системе материалов Horde3D.

Синтаксис Horde3D FX схож с синтаксисом языком CgFX. Поддерживаются стандартные блоки C++ и однострочные комментарии. Все ключевые слова и перечни параметров (enums) не чувствительны к регистру. Однако, идентификаторы, такие как имя контекста, чувствительны к регистру. Идентификаторы могут содержать только цифробуквенные символы и знак подчеркивания. Дробные числа не должны содержать пробелов.

Элементы могут опционально хранить метаинформацию в качестве аннотаций. Эта функция полезна для обеспечения лучшей поддержки. Например, переменная (uniform) может принимать значения только в некотором интервале. Подразумевается, что аннотации соответствуют требованиям синтаксиса CgFX, то

есть <*тип данных*> <*идентификатор*> = <*значение*>. Однако, это не требование, так как аннотации пропускаются парсером[1] Horde3D.

Поддерживается следующий синтаксис. Опциональные элементы помещены в квадратные скобки, а значения по умолчанию выделены полужирным. Альтернативные значения разделены косой чертой.

Для работы описания сэмплеров применяется следующая синтаксическая конструкция:

sampler [< аннотация(и) **>] id [** = sampler_state { }

Таблица 3-1 Параметры текстурного сэмплера

Название	**Описание**	**Возможные значения**
TexUnit	Используемый текстурный блок	**-1** / 0 / 1 / 2 / 3 / 4 / 5 / 6 / 7 / 8 / 9 / 10 / 11.
Filter	Применяемая фильтрация текстур Билинейная / Трилинейная	Bilinear / **Trilinear**
MaxAnisotropy	Степень применяемой к текстуре анизотропной фильтрации	1 / 2 / 4 / **8** / 16
Address	Поведение системы при выходе за границы текстуры Wrap – начать с начала текстуры Clamp – оставаться на границе текстуры	**Wrap** / Clamp

[1] парсер: синтаксический анализатор; анализатор. Часть компилятора или программы, выполняющая чтение предоставленного текста (или исходного текста программы), проверку её синтаксиса и создание промежуточного файла, который служит для блоков, выполняющих дальнейшие стадии компиляции. Другими словами, он превращает входной поток символов в поток лексем.

Примечание:

- TexUnit определяет текстуру, используемую сэмплером; значение -1 используется для автоматического присвоения
- Доступно совмещение имен для текстур – сэмплеры могут использовать одинаковые текстуры, если только они не используются в одном и том же контексте шейдера

Для задания переменной, передаваемой из управляющего приложения и состоящего из четырех или менее компонентов, применяется следующая синтаксическая конструкция:

float4 [< аннотация(и) **>] id [= {** *a, b, c, d* **}];**

По умолчанию все объявленные переменные инициализируются со значением 0.0.

Для описания контекста шейдера, объединяющего в себе применяемые шейдерные программы и их параметры, применяется следующая синтаксическая конструкция:

context [< аннотация(и) >] **id {}**

Таблица 3-2 Параметры контекста шейдера

Название	**Описание**	**Возможные значения**
VertexShader	Используемый вершинный шейдер	**compile GLSL** *название секции кода;*
PixelShader	Используемый пиксельный шейдер	**compile GLSL** *название секции кода;*
ZWriteEnable	Возможность записи в буфер/текстуру глубины	false / **true**
ZFunc	Функция, используемая при сравнении двух значений	Always / Equal / Less / **LessEqual** / Greater /

	глубины Значения: Не сравнивать, Равно, Меньше, **Меньше либо равно,** Больше, Больше либо равно	GreaterEqual
BlendMode	Режим смешивания Значения: **Замена**, Смешивание, Добавление, Добавление смешанного, Перемножить	**Replace** / Blend / Add / AddBlended / Mult
AlphaToCoverage	Применяется при сглаживании объектов, таких как листва и трава, для которых используются полупрозрачные текстуры Значения: **ложь /** истина	**false** / true

Примечание:

- Ссылки на секции кода, указываемые в секции FX, должны быть определены в том же файле

3.2.15. Описание секции кода шейдеров

Секция кода содержит код GLSL. Кроме ключевых слов GLSL, Horde3D поддерживает дополнительную команду препроцессора: **#include.** Директива *include* служит для вставки кодового ресурса в указанное место. Имя кодового файла должно быть указано в кавычках.

Кодовая секция также может содержать флаги шейдеров для автоматического создания комбинаций эффектов. У флагов шейдеров свой

синтаксис: **_F<***число***><***число***>_<***имя***>.** Пример: _F06_MyFlag. Флаг должен иметь число от 01 до 32 (если порядковый номер флага меньше 10, необходимо указывать 0 в порядковом номере). Этот номер служит для идентификации флага. Имя опционально и используется для удобства с целью повышения читабельности кода.

3.2.16. Текстурные карты

Horde3D поддерживает двумерные и кубические текстуры. Двумерные текстуры можно загрузить из формата Direct3D DDS или из любого другого указанного ниже формата изображений. Кубические карты необходимо хранить в формате DDS. В отличие от других форматов изображений, формат DDS может хранить мипмапы (текстуры с варьируемым разрешением) и сжатые DXT данные. Поэтому, флаги текстур для генерации мипмапов и сжатия игнорируются для DDS текстур. Для оптимальной производительности при работе с DDS текстурами, Horde3D использует систему координат Direct3D, где началом координат считается верхний левый угол изображения. В OpenGL началом координат изображения считается левый нижний угол. Из-за этого, координаты текстур для двумерных текстур необходимо вручную переворачивать в шейдерах.

Движок может загружать следующие форматы данных. Существуют некоторые ограничения для определенных форматов изображений, таких как текстуры 1 bpp.

- DDS (форматы: RGB8, RGBA8, RGBA16F, DXT1, DXT3, DXT5 и другие)
- JPEG (не прогрессивный)
- PNG
- TGA
- BMP (не-RLE)
- PSD (только RGB)
- HDR

3.2.17. Структура материалов в Horde3D

Расширение: .material.xml

Материалы используются для передачи параметров шейдерам. Они ссылаются на шейдер и связывают текстурные сэмплеры с изображениями. Они также могут задавать значения переменных шейдеров (uniforms), являющихся четырехмерными дробными векторами. В материалах также указываются используемые флаги шейдера, при помощи которых выбирается необходимая комбинация шейдерных программ из файла шейдера.

Материал может быть использован, например, для определения вида поверхности. Каждый материал может иметь класс, полезный для доступа к геометрии со специфическими свойствами, такой как полупрозрачная геометрия. Так как система классов материалов иерархична, то возможно применение подклассов, отделяемых от родительского класса точкой. Существует возможность присоединить материал к другому материалу для использования его текстурных сэмплеров и переменных. Эта возможность очень удобна для определения глобальных данных (например, глобального освещения) в одном единственном материале.

Данные материалов движка хранятся в формате XML.

3.2.18. Спецификации программируемого конвейера рендеринга

Программируемый конвейер определяет шаги, необходимые для выведения абстрактных трехмерных данных в виде видимого результата на экран. Horde3D использует гибкий программируемый конвейер, позволяющий использовать различные техники рендеринга, включая прямой и отложенный рендеринг. Система позволяет определить несколько целей рендеринга (буферов) и задавать команды для заполнения целей данными. Это позволяет создавать большинство эффектов пост-обработки, таких как HDR (High Dynamic Range – Расширенный динамический диапазон), эффект смазывания движения или глубины резкости, а также применять различные алгоритмы рендеринга.

3.2.19. Настройка и синтаксис программируемого конвейера

Гибкий программируемый конвейер позволяет определять цели рендеринга и команды, определяющие последовательность рендеринга сцены. Команды задаются в XML файле. Большинство команд рендеринга используют свойства class и context. Класс указывается в материалах и определяет, какую геометрию необходимо прорисовать. Возможно использование оператора ~, который дает команду прорисовать всю геометрию за исключением указанного класса. Context определяет применяемую технику рендеринга для конкретного вызова прорисовки.

3.2.20. Недостатки выбранного графического модуля

Основными выявленными недостатками выбранного графического модуля Horde3D являются отсутствие возможности анимации виртуальных камер, а также невозможность отображения полупрозрачных объектов в виртуальной трехмерной сцене. С целью устранения выявленных недостатков автором разработаны методы и алгоритмы, описанные в следующих разделах.

3.3. Методы анимации виртуальной камеры в подсистеме визуализации

Задача анимации неуправляемых объектов трехмерной сцены является одной из задач достижения реалистичности моделируемой виртуальной среды ТОС. Неуправляемые движущиеся объекты – это объекты с заранее определенной траекторией, не зависящие от управляющих воздействий. Движение таких объектов отрабатывается в общем случае самой системой визуализации. Основным подходом, при обработке движения объектов системой визуализации, является технология ключевой анимации. Суть ее состоит в том, что вместо полного определения координат и ориентации объекта в каждом из моментов времени, что потребовало бы значительных затрат памяти, выделяются так называемые ключевые кадры, разделяющие фазы движения объекта. Параметры объектов (координаты, ориентация и т.д.) задаются только в ключевых кадрах, а в промежуточные моменты времени осуществляется интерполяция значений. В ключевых кадрах задаются положения и ориентации исходных локальных систем координат объектов относительно локальной системы координат родителя объекта, причем ориентации задаются с помощью пары $(\varphi,\vec{u})$, указывающей на поворот вокруг указанного вектора $\vec{u}$ на заданный угол φ. Поэтому матрица анимации, переводящая исходную систему в родительскую, будет заменять матрицу M_L в равенстве $M_O = M_O^P \cdot M_L$, где *Mo* – модельная матрица для объекта, M_O^P - модельная матрица для родителя, M_L - матрица перехода из локальных координат объекта в систему координат родительского объекта.

Наша задача заключается в вычислении с помощью интерполяции вектора $\vec{V}_L$ перемещения и матрицы M_{rot}^L поворота объекта для некоторого кадра визуализации. Для интерполяции координат положения объекта обычно используются эрмитовы кривые, TCB-сплайны или кривые Безье. В соответствии с выбранным типом кривой в момент времени t можно вычислить текущую позицию $\vec{V}_L$ исходной локальной системы координат объекта в системе координат его родителя.

Ориентацию, заданную в виде пары $(\varphi,\vec{u})$, удобно перевести в кватернион. Из определения кватерниона следует, что он будет иметь вид:

$$q = \left(\cos\frac{\varphi}{2}, \vec{u}\sin\frac{\varphi}{2} \right). \qquad \textbf{(3.1)}$$

Для интерполяции кватернионов обычно применяется метод сферической линейной интерполяции (slerp), который выполняет интерполяцию по большой дуге единичной сферы, соединяющей ключевые кватернионы в четырехмерном пространстве. Также возможно применение сферической квадратичной интерполяции (squad). В результате мы получаем некоторый кватернион $q = (w,x,y,z)$. Анимационная матрица поворота M_{rot}^{L} вычисляется по кватерниону q по формуле:

$$M_{rot}^{n} = \begin{pmatrix} 1-2(y^2+z^2) & 2(xy-wz) & 2(xz+wy) & 0 \\ 2(xy+wz) & 1-2(x^2+z^2) & 2(yz+wx) & 0 \\ 2(xz-wy) & 2(yz+wx) & 1-2(x^2+y^2) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \qquad (3.2)$$

Таким образом, для каждого анимируемого узла иерархического дерева мы имеем анимационный вектор $\vec{V}_L$ и анимационную матрицу поворота M_{rot}^{L}, по которым в соответствии с равенством $M_O = \begin{pmatrix} M_{rot} & \vec{V} \\ \vec{0} & 1 \end{pmatrix}$, где M_O – текущая модельная матрица объекта, описывающая текущее положение объекта, M_{rot} - матрица поворота размера 3x3, $\vec{V}$ - вектор перемещения, вычисляется матрица анимации M_L [80].

Графический движок Horde3D не поддерживает анимацию камеры. Для обеспечения данной возможности автор модернизировал алгоритм импорта графических ресурсов в движок, с целью создания файла анимации камеры, а также разработаны алгоритмы и программный модуль управления и воспроизведения анимации.

Экспорт объектов и анимации в формат Collada происходит через модуль (плагин) OpenCollada для пакета моделирования Autodesk 3ds Max. Для

корректной обработки анимации приложением-импортером ресурсов, экспортируемый из приложения моделирования файл формата Collada должен содержать дискретные кадры анимации.

Анимация камеры преобразовывается из формата Collada в формат XML, поддерживаемый Horde3D. Для каждой камеры, присутствующей в сцене, при наличии у них анимации, записывается информация о кадрах анимации. Структура файла анимации:

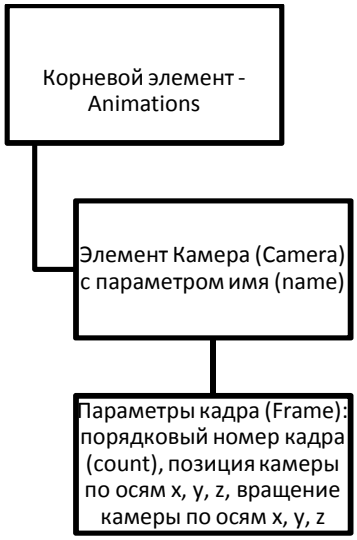

Рис. 3.2. Структура файла анимации

Алгоритм основан на методах ключевой анимации. Задаются номера начального и конечного кадров анимации камеры, а также скорость их воспроизведения. Заданные кадры должны быть в пределах массива кадров анимации, данные которого берутся из файла анимации камеры. Если запрашиваемый сегмент выходит за границы массива кадров анимации, и хотя бы одна из границ сегмента меньше нуля, то анимация не воспроизводится. В первоначальной версии алгоритма скорость задается в кадрах в секунду, как правило, 25, 30, 50, 60. В последней версии алгоритма значение количества кадров в секунду для анимации считывается при импорте ресурсов в систему. Это позволило задавать скорость анимации в виде коэффициента, например, 0.5, 1, 1.3, 2.0 и т.д.

Кадры анимации для всех камер хранятся в одном массиве, поэтому необходимо делать выборку используемых кадров для каждой из камер. Предположим, что в массиве хранятся кадры анимации для двух камер. Для каждой камеры существует по 60 кадров анимации. Необходимо воспроизвести анимацию второй камеры. После задания требуемого сегмента воспроизводимых кадров для второй камеры (от 0 до 60) происходит выборка начального и конечного кадров для получения корректной анимации. В массиве данные анимации для второй камеры хранятся, начиная с 60 кадра, следовательно, нулевым кадром анимации для второй камеры является 60 кадр массива анимации. Соответственно, при воспроизведении анимации для второй камеры будут воспроизводиться кадры массива от 60-го до 120-го.

Разработанный алгоритм способен обрабатывать анимацию камеры не только в случае, когда последовательность воспроизводимых кадров возрастающая (предположим, происходит воспроизведение анимации камеры с 0-го кадра по 60-й), но и в случае, когда последовательность является убывающей (с 60-го по 0-й). Данная функциональность позволяет избежать необходимости создания дополнительных кадров анимации камеры в приложениях трёхмерного моделирования.

Введем следующие условные обозначения:

- S – скорость воспроизведения анимации (кадров в сек)
- T_1 – время генерации кадра подсистемы визуализации (мс)
- T_2 – время кадра подсистемы визуализации с начала анимации (мс)
- T_3 – время предыдущего кадра подсистемы визуализации с начала анимации (мс)
- Pos – промежуточная позиция анимации в пределах одного ключевого кадра. Может принимать значения в отрезке [0, 1]
- $prevPos$ – значение Pos на предыдущем кадре подсистемы визуализации
- K_T – текущий кадр анимации камеры
- K_H – начальный кадр анимации камеры

- $К_К$ – конечный кадр анимации камеры

Алгоритм:

1) Получение номеров начального $К_Н$ и конечного $К_К$ кадров анимации из управляющего приложения, а также идентификатора камеры, к которой будет применена анимация.
2) Проверка номеров $К_Н$ и $К_К$ на выход за границы массива. ЕСЛИ номера кадров находятся в границах массива, ТО происходит выделение используемых кадров анимации из массива кадров и флаг, разрешающий воспроизведение анимации, выставляется в 1, ИНАЧЕ анимация не воспроизводится. Анимация также не воспроизводится, если $К_Н = К_К$.
3) Получение данных о положении и ориентации исходных локальных систем координат виртуальной камеры из начального и следующего кадров.
4) Сохранение текущего времени кадра подсистемы визуализации с начала анимации $T_3 = T_2$
5) Цикл (обновление текущей позиции камеры происходит раз в кадр подсистемы визуализации):

- Проверка флага, разрешающего воспроизведение анимации;
- Расчет времени генерации кадра по формуле: $T_1 = T_2 - T_3$;
- Расчет промежуточной позиции анимации: $Pos = T_1 / \left(\frac{1}{S} \right)$;
- Сохранение текущего времени кадра подсистемы визуализации с начала анимации $T_3 = T_2$
- Проверка номера текущего кадра:

  - ЕСЛИ текущий кадр равен последнему кадру, ТО окончание анимации (применение значений позиции и ориентации камеры, указанных в последнем кадре анимации);

  - ЕСЛИ текущий кадр не равен последнему кадру, ТО:

    - ЕСЛИ $Pos + prevPos > 1$, ТО:

* Расчет K_T по формуле: $K_T = K_T + \lfloor Pos + prevPos \rfloor$;

* Получение данных кадра $K_T + 1$ из массива кадров (если последующий кадр не последний);

* Расчет $prevPos$ по формуле: $prevPos = (Pos + prevPos) - \lfloor Pos + prevPos \rfloor$;

* Расчет позиции виртуальной камеры в трехмерной сцене при помощи линейной интерполяции значений, записанных в текущем и следующем ключевых кадрах, и промежуточной позиции Pos между этими кадрами;

- ИНАЧЕ:

* Расчет $prevPos$ по формуле: $prevPos = Pos + prevPos$;

* Расчет позиции виртуальной камеры в трехмерной сцене при помощи линейной интерполяции значений, записанных в текущем и следующем ключевых кадрах, и промежуточной позиции Pos между этими кадрами;

Окончание цикла.

Используя данный алгоритм, возможно достичь воспроизведения анимации камеры, созданной в приложении моделирования, подсистемой визуализации ТОС [81].

3.4. Методы отображения изображений с частичной прозрачностью в подсистеме визуализации

Альфа-канал (Alpha channel) - дополнительный канал в изображении, который, используется для создания эффекта полной или частичной прозрачности при комбинировании изображений [82]. Широко применяется при создании окончательных изображений в 3D-графике. Термин альфа-канал впервые введён в оборот Алви Смитом в конце 1970-х и детально проработан в статье Томаса Портера и Тома Даффа 1984 года.

Основные растровые форматы, поддерживающие альфа канал:

psd - формат Adobe Photoshop, широко используется во многих областях компьютерной графики;

tiff – один из самых распространенных растровых форматов, широко используется в полиграфии;

gif – формат позволяющий создавать изображения с прозрачными областями для web, не поддерживает частичную прозрачность;

iff – формат для обмена данными, используется программой Maya.

tga – растровый формат графики, созданный компанией Truevision в 1984 году. Применяется в компьютерной графике и при обработке видео.

png – растровый формат, предназначенный для хранения и передачи растровых изображений: черно-белых до 16 бит на пиксель (для односоставных), а цветных - до 48 бит (для многосоставных). Как и gif, широко используется в web, но в отличии от него, поддерживает частичную прозрачность. Он использует прогрессивный метод сжатия без потерь, позволяет сохранять в файле палитру, текстовую информацию.

Для достижения частичной прозрачности изображения в подсистеме визуализации Horde3D был использован растровый формат png. Основными преимуществами формата являются:

- алгоритм сжатия, позволяющий избежать потерь информации при сжатии;

- возможность хранения индексированных изображений с палитрой до 256 цветов;
- прогрессивное отображение чересстрочных данных;
- поддержка альфа-канала и прозрачности;
- независимость от аппаратной конфигурации и платформы;
- наличие CRC - метода обнаружения ошибок в потоке данных.

В формате PNG определено 4 типа стандартных блоков, иначе именуемых *критические блоки*, которые должны поддерживаться любой программой чтения и записи PNG. Далее следует перечень этих блоков:

- заголовочный блок (IHDR). Заголовочный блок содержит основную информацию о данных изображения и должен быть первым блоком в потоке данных PNG (не допускается более одного заголовочного блока);
- блок палитры (PLTE). Палитра несёт в себе данные таблицы цветов, связанные с данными изображения. Этот блок присутствует, только если данные изображения используют палитру и должен находиться перед этими данными;
- данные изображения (IDAT). Блок данных изображения содержит в себе само изображение, и допускается несколько таких блоков в потоке данных, причём все они должны следовать друг за другом;
- замыкающий блок изображения (IEND). Замыкающий блок изображения должен находиться в конце файла или потока данных PNG.

Среди этих блоков, IHDR, IDAT и IEND должны присутствовать в любом потоке данных PNG.

Существуют также вспомогательные блоки. В них хранится такая информация, как фактический размер изображения в пикселях и прозрачность изображения.

Блок Прозрачности содержит значение прозрачного (ключевого) PNG изображения, не содержащего соответствующих альфа данных. Значения пикселей для полноцветных и чёрно-белых изображений, совпадающих с

прозрачным цветом, считаются прозрачными (альфа значение - 0), остальные же считаются непрозрачными.

Изображения с индексированными цветами содержат массив альфа значений, максимум по одному на элемент палитры. Эти значения прозрачности обрабатываются как альфа значения. Элементам палитры, не имеющим значений прозрачности, присваивается значение по умолчанию 255 (абсолютно непрозрачные). Хотя реализация прозрачности без использования альфа канала дает менее качественное изображение, она вполне применимо для большинства случаев, и требует меньше места для хранения данных.

Альфа значение определяет уровень прозрачности пикселя. Минимальное значение битовой глубины (всегда 0) указывает на абсолютную прозрачность, а максимальное значение, либо отсутствие альфа маски, указывает на полную непрозрачность [83]. Расчет цвета результирующего пикселя выполняется по следующей формуле:

$$P_R = P_B(1 - A) + P_F A, \quad \textbf{(3.3)}$$

где P_R – цвет результирующего пикселя, P_B – цвет фонового пикселя, A – значение альфа, P_F – цвет накладываемого пикселя.

Для реализации функционала прозрачности автором был модифицирован базовый шейдер, используемый для визуализации трехмерных моделей. В секции FX шейдера был создан новый контекст шейдера с TRANSLUCENT. Название контекста должно быть указано латинскими прописными буквами. В данном контексте определены используемые шейдерные подпрограммы (вершинная и пиксельная), а также их свойства.

Используемая подпрограмма вершинного шейдера описана в поставляемом вместе с пакетом разработчика движка Horde3D шейдере model.shader, и называется VS_GENERAL. Префикс VS обозначает, что это вершинный шейдер.

Подпрограмма пиксельного шейдера имеет два входных параметра – текстура, с которой предстоит работать, и ее координаты. Изменение параметра

прозрачности трехмерного объекта производится на основе данных из входящей текстуры. Также была добавлена возможность управления степенью прозрачности объекта при помощи дополнительного входного параметра, что позволяет изменять прозрачность даже при использовании текстуры без альфа-канала. Результатом работы пиксельного шейдера является измененный цвет каждого пикселя отображаемого трехмерного объекта в итоговом изображении.

Необходимо также отметить следующие свойства контекста: запись в Z буфер должна быть отключена (функция ZWriteEnable = false), а режим смешивания должен быть выставлен в Blend (смешивание) [81].

Глава 4 . Архитектура подсистемы воспроизведения мультимедийных материалов в трехмерном виртуальном окружении

4.1. Методы и алгоритмы воспроизведения мультимедийных материалов из файлов

Подсистема визуализации обеспечивает отображение результатов моделирования внешней среды и объекта управления с помощью устройств отображения информации. Отображение разнородных видеоматериалов, в том числе графического образа инструктора, в виртуальной трёхмерной сцене является одним из требований к ТОС [84].

Для отображения видеоматериалов в подсистеме визуализации была выбрана кросс-платформенная система декодирования видео файлов FFmpeg, распространяемая по лицензии LGPL. Данная система является базовым средством декодирования видеоматериалов в операционных системах семейства Linux, а также используется в операционных системах Windows (в составе пакета FFDShow) и MacOS X. Основой системы FFmpeg является библиотека libavcodec, в которой хранятся различные кодеры и декодеры для работы с аудио и видео информацией.

Структура видео файла - контейнер, содержащий общую информацию о содержимом контейнера (количество видео- и аудио-потоков, наличие субтитров, информацию о разделах, метаданные - название, год создания, автор и т.д.) и различных потоках информации (видео, аудио, субтитры и т.д). Некоторые контейнеры допускают использование только определенных кодеров для представления информации. Наиболее популярными контейнерами являются AVI, OGV, MP4, MKV, MOV, WMV.

В большинстве контейнеров данные представляются в виде пакетов (в некоторых форматах называемые chunks (куски) или segments (сегменты)). Пакеты хранят закодированную информацию определенного типа – видео, аудио или субтитры и т.д. В большинстве видео файлов количество аудио пакетов значительно меньше, чем количество видео пакетов – приблизительно 75% видео пакетов и 25% аудио пакетов. В зависимости от типа контейнера и используемого

кодера расположение пакетов внутри контейнера может быть разным. Зачастую пакеты расположены в виде последовательности очередей пакетов разного типа, например, около 20 видео пакетов, расположенных подряд, затем несколько аудио пакетов, после которых снова подряд расположены около 20 видео пакетов.

Пакеты хранят в себе определенный сегмент аудио или видео информации. В большинстве случаев в видео пакете хранится целый видеокадр, но в некоторых форматах (в частности, в формате MPEG 2) один видеокадр может состоять из нескольких пакетов. Это связано с тем, как кодер сохраняет информацию о видеокадре. Например, в формате MPEG 2 присутствует три типа кадров – независимо сжатые кадры (I-кадры), кадры, сжатые с использованием предсказания движения в одном направлении (P-кадры) и кадры, сжатые с использованием предсказания движения в двух направлениях (B-кадры). Соответствующие группы кадров от одного I-кадра до другого образуют GOP — Group Of Pictures — группу кадров.

Большинство форматов видео файлов представляют видео информацию в цветовой модели YUV (Y – яркость, U и V – цветоразностные составляющие).

В пакетах, кроме закодированных данных и информации о размере и типе данных данного пакета, также присутствует информация о времени, когда необходимо декодировать данный пакет (последовательности декодирования пакетов), и времени, когда декодированная информация должна быть отображена или воспроизведена.

Аудио информация видео файла может быть закодирована с использованием различных форматов сжатия, таких как MP3, AAC, AC3, WMA и т.д. Для корректного воспроизведения аудио информации необходимо знать количество каналов (моно, стерео, 5.1, 7.1 и т.д) и частоту дискретизации. После декодирования звукового пакета, декодированные данные представляются в формате PCM (Pulse Code Modulation, импульсно-кодовая модуляция), которые передаются звуковой карте для воспроизведения звука.

Пакет FFmpeg не предоставляет возможностей для визуализации декодированных видеоизображений и воспроизведения звуковых данных. Для

воспроизведения звуковых данных из декодируемого видео файла в разработанной автором архитектуре подсистемы воспроизведения мультимедиа был использован пакет SFML, распространяемый по лицензии zlib/png. Пакет SFML является кросс-платформенным и предоставляет возможности по работе с графикой, звуком, различными системами ввода информации и передачей данных по сети.

Воспроизведение видеоматериалов внутри виртуальной трехмерной сцены является сложной задачей, так как необходимо учитывать факторы, не актуальные при воспроизведении в медиаплеерах. Графическая подсистема должна визуализировать трехмерную сцену с приемлемой частотой кадров (не менее 25 кадров в секунду), и должна быть способна при этом реагировать на внешние воздействия, в том числе на изменения параметров трёхмерной сцены или загрузка дополнительных объектов.

Разработанная автором архитектура подсистемы воспроизведения мультимедиа материалов позволяет декодировать и отображать одновременно несколько видео высокой четкости в трехмерной сцене. Состав архитектуры включает: декодер видео, в котором происходит декодирование видео и аудио пакетов; подсистема воспроизведения декодированного звука; управляющая структура, необходимая для запуска видео, паузы воспроизведения, выставления громкости воспроизводимого видео и т.д.; интерфейс взаимодействия с движком, необходимый для обновления видеокадров; интерфейс управления графической подсистемой.

Общая схема работы декодера представлена на Рис. 4.1.

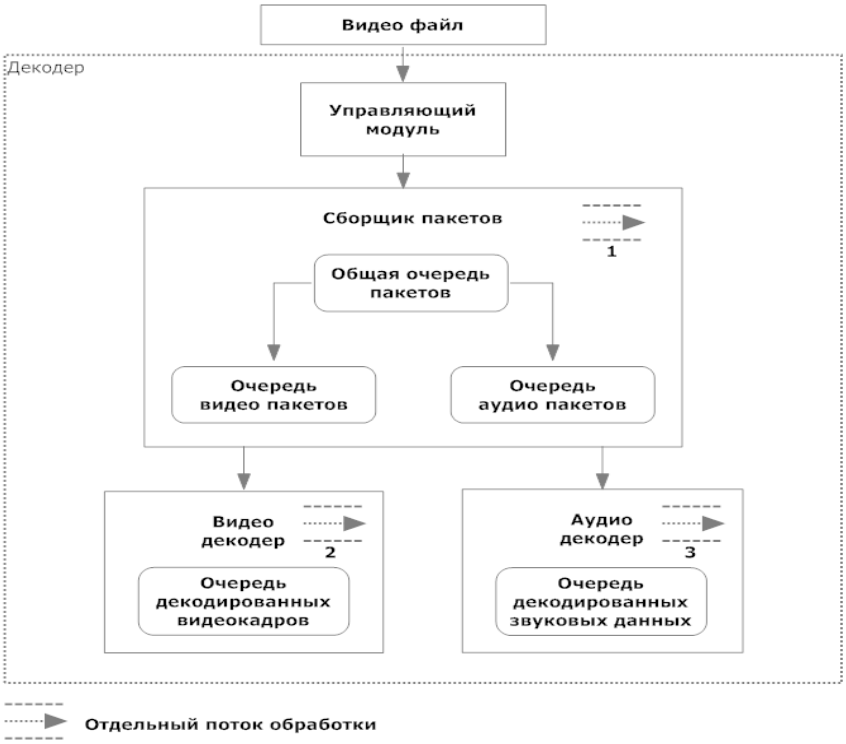

Рис. 4.1. Общая схема работы декодера видео файла

Во время инициализации декодера считываются все необходимые параметры о видео файле, такие как: кодеки, используемые для сжатия видео и аудио информации, количество звуковых и видео потоков в файле, ширина и высота (разрешение) видео, количество каналов звука и т.д.

В зависимости от формата видео файла размещение пакетов в файле различается, следовательно, сложно предсказать в каком порядке будут следовать видео и аудио пакеты. Это может привести к ситуации, когда очередь видео пакетов заполнена, а очередь аудио пакетов будет пуста. Узнать, какого типа будет следующий пакет, невозможно до его чтения. Если следующий видео пакет окажется закодированным видеокадром, то либо этот пакет необходимо помещать в очередь видео пакетов, т.е. превышать заданные рамки очереди, либо удалять пакет, что приводит к артефактам на конечном изображении или пропущенным кадрам и рывкам воспроизводимого видео. Для решения этой проблемы было принято решение создать третью очередь пакетов, в которую помещаются пакеты различного типа из видео файла, а затем сортируются по типу пакета (аудио или

видео пакет) в одну из очередей. Пакеты сортируются по методу FIFO («первый вошел, первый вышел»).

Видео декодер проверяет наличие пакетов в очереди видео пакетов и, при наличии пакетов, забирает пакет из начала очереди. Процесс декодирования видео пакета состоит из двух этапов – декодирование пакета и преобразование декодированного кадра в необходимую цветовую модель. После декодирования и преобразования в другую цветовую модель, кадр помещается в очередь декодированных кадров. Аналогично, аудио декодер проверяет наличие пакетов в очереди и, при наличии пакетов, декодирует первый пакет из очереди. После декодирования звуковые данные в формате PCM помещаются в очередь декодированных звуковых данных.

Так как подсистема визуализации принимает изображения только в цветовой модели BGRA, то после декодирования видеокадр будет преобразован в эту цветовую модель. Цветовая модель BGRA выбрана не случайно – большинство форматов изображений и большая часть оборудования хранят изображения в формате BGRA. При использовании других цветовых моделей происходит преобразование используемой цветовой модели в цветовую модель BGRA, выполняемое на центральном процессоре. Так как любое лишнее преобразование снижает производительность графической подсистемы, то использование цветовой модели BGRA является оптимальным.

Процесс декодирования является ресурсоёмким процессом. Например, декодирование одного видео пакета видео с разрешением 1280 на 720 (720p) на процессоре Intel Core 2 Quad 2,66 ГГц занимает от пяти до десяти миллисекунд, для видео размером 1920 на 1080 (1080p) – от 10 до 20 мс. Декодирование пакета звуковых данных занимает значительно меньше времени (0,5 – 2 мс), но, если одновременно декодируется звук для нескольких воспроизводимых видео, это также может серьезно снизить производительность. Скорость получения пакетов из видео файла ограничивается производительностью системы хранения данных (СХД), что в свою очередь влияет на производительность подсистемы визуализации. Задержки образуются, если в момент считывания пакетов из

видеофайла СХД обрабатывала другие команды, либо файл являлся фрагментированным, что заставило бы СХД искать фрагменты данного файла. Для преодоления проблем с производительностью была использована многопоточность.

Многопоточность - свойство платформы (например, операционной системы) или приложения, состоящее в том, что запущенный процесс может состоять из нескольких *потоков*, выполняющихся «параллельно», то есть без предписанного порядка во времени. При выполнении некоторых задач такое разделение позволяет достичь более эффективного использования ресурсов вычислительной машины.

Все потоки выполняются в адресном пространстве процесса. Выполняющийся процесс имеет как минимум один (главный) поток. Применение многопоточности наиболее эффективно при использовании многоядерных процессоров. На одноядерных процессорах многопоточное приложение также будет работать, но повышение производительности будет незначительным; возможно даже снижение производительности, если разные потоки одного процесса выполняют ресурсоёмкие операции[85].

Хотя многопоточность способна существенно повысить производительность приложения, применять ее следует с осторожностью. Внедрение многопоточности существенно повышает сложность самого приложения и его отладки, а также, при некорректном использовании, может привести к таким проблемам, как «состояние гонки» (когда работа системы зависит от того, в каком порядке выполняются части кода), взаимной блокировке потоков (несколько потоков находятся в состоянии бесконечного ожидания ресурсов, занятых самими этими потоками), и трудностям при обработке ошибок (исключений) внутри приложения.

Для оптимизации процесса декодирования видео используются отдельные потоки для сбора пакетов, декодирования видео- и аудио пакетов. Процесс воспроизведения звука также происходит в дополнительном потоке. Первоначально, в качестве системы управления потоками был выбран пакет

SFML, который предоставляет базовую функциональность управления многопоточностью на системах семейств Windows и Unix (Linux, MacOS). В целом, система управления потоками SFML работала без сбоев, но было выявлено несколько недостатков. Система не позволяет выставить приоритет обработки создаваемому потоку, а также привязать создаваемый поток к конкретному процессорному ядру. Система предоставляет только один способ приостановки потока – «сон», что не позволяет точно прогнозировать, через какое время потоку вновь будет выделено процессорное время. Например, если потоку дано указание «заснуть» на 5 миллисекунд, то нет гарантий, что через 5 мс потоку будет выделено процессорное время, оно может быть выделено через 15 мс. Данное обстоятельство зависит от степени загруженности процессора, реализации планировщика задач операционной системы и приоритета выполнения потока. С целью преодоления описанных выше недостатков было принято решение интегрировать библиотеку управления потоками thread системы boost.

Система boost – это набор кросс-платформенных библиотек, расширяющих стандартные инструменты языка C++, распространяемых по лицензии Boost Software License. Часть библиотек являются кандидатами на включение в следующий стандарт языка C++.

Библиотека управления потоками boost thread обладает большей функциональностью, чем библиотека SFML, и предоставляет такие возможности, как управление потоками, созданными с помощью библиотеки thread, средствами операционной системы; использование условных переменных (conditional variables) и т.д. Использование средств операционной системы для управления созданным потоком позволяет реализовывать такую функциональность, как задание приоритета выполняемого потока, а также привязка потока к определенному ядру процессора. Применение условных переменных позволяет приостанавливать поток при определенных условиях и, в отличие от функции “сон”, мгновенно восстанавливать работу потока.

Обновление видеотекстуры в подсистеме визуализации происходит в основном потоке программы (потоке рендеринга). Каждый кадр подсистемы

визуализации происходит проверка – следует ли обновлять видеотекстуру в трехмерной сцене или нет. Если обновление требуется, то видеокадр берётся из очереди декодированных кадров и помещается в память видеокарты в качестве текстуры, которая может быть наложена на поверхность любого трёхмерного объекта.

При запуске видео происходит создание пустой текстуры с размерами видеокадра, которая затем применяется к материалу трёхмерного объекта (материал объекта определяет, какие текстуры и шейдеры использует данный трёхмерный объект). Перед применением новой текстуры к материалу происходит резервное копирование старого материала для возможности возвращения к исходным текстурам объекта после окончания воспроизведения видео. Также, для повышения производительности, используются два т.н. пиксельных буфера. Пиксельные буферы применяются для хранения пиксельных данных (текстуры) в видеопамяти, и позволяют значительно снизить временные затраты на обновление видеотекстуры в подсистеме визуализации. При использовании стандартной функциональности движка для обновления текстур в видеопамяти сначала происходит копирование данных из видеокадра в промежуточный массив, затем данные из промежуточного массива помещаются в текстуру. При обновлении текстуры в видеопамяти сначала происходит выделение нового места под данные и только затем происходит копирование данных из промежуточного массива в видеопамять. Все описанные выше действия выполняются центральным процессором, что значительно снижает производительность подсистемы визуализации. Например, при воспроизведении FullHD видео (1920 на 1080) время копирования данных из видеокадра в промежуточный массив составляет приблизительно 3-4 мс, а обновление текстуры из промежуточного массива в видеопамяти составляет от 5 до 8 мс. Преимущество пиксельных буферов состоит в том, что затраты процессорного времени необходимы только на помещение информации в буфер, а обновление текстуры из буфера происходит практически мгновенно (около 0.1 мс) за счёт

того, что обработка перемещения данных из пиксельного буфера в текстуру возлагается на видеокарту.

Проверка необходимости обновления видеотекстуры в подсистеме визуализации состоит из нескольких частей. Первая проверка заключается в определении состояния видео – проигрывается, приостановлено, остановлено. Если видео остановлено или приостановлено, то дальнейших проверок не происходит. Приостановленное видео от остановленного отличается тем, что все потоки декодирования созданы, но остановлены, а очереди пакетов и декодированных кадров заполнены. Если видео остановлено, то оно загружено (декодер инициализирован, информация о видеофайле получена), но его потоки декодирования не созданы и очереди пакетов пусты.

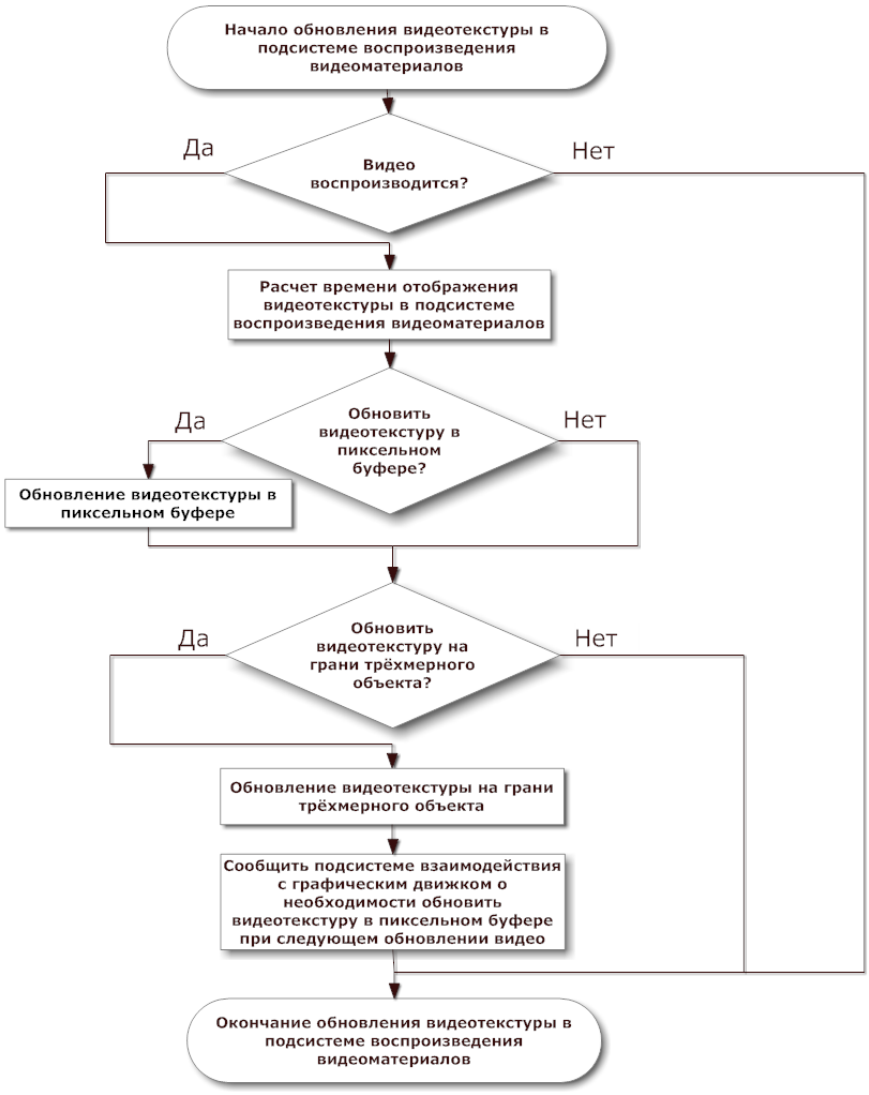

Рис. 4.2. Алгоритм обновления видеотекстуры в подсистеме визуализации.

Составной частью первой проверки является проверка на кэширование (выполняется или нет). В начале воспроизведения видео запускается процесс кэширования, который заполняет очереди пакетов декодера, очередь декодированных звуковых данных и очередь декодированных кадров. Кэширование пакетов и управление этим процессом происходит в потоке декодера, чтобы избежать проблем с производительностью (падением количества отображаемых подсистемой визуализации кадров в секунду), а декодирование аудио и видео пакетов происходит в соответствующих потоках декодера. Во время кэширования данных основной поток приложения раз в итерацию цикла генерации кадров (далее ИЦ) подсистемы визуализации проверяет состояние кэшируемого видео. Если хотя бы один видеокадр уже был декодирован, то он помещается в пиксельный буфер, а затем удаляется из очереди декодированных кадров. После заполнения всех пиксельных буферов данного видео основной поток продолжает раз в кадр проверять состояние видео, пока процесс кэширования не будет завершен. После завершения процесса кэширования происходит запуск аудио потока и состояние видео меняется на «проигрывается». Также запускается таймер, по которому происходит отсчет времени для проигрываемого видео.

После того, как первая проверка пройдена, происходит вторая проверка на необходимость обновления видеотекстуры в подсистеме визуализации. В ходе данной проверки происходит сравнение времени, прошедшего с начала воспроизведения видео, со временем, когда необходимо отобразить следующую видеотекстуру.

Для определения необходимости обновления видеоматериала в подсистеме визуализации решается следующее неравенство:

$$\left|TS - I_V \frac{1}{F}\right| \leq \left|\frac{1}{10}\sum_{i=1}^{10}(TF_i) - I_V \frac{1}{F}\right|, \qquad \textbf{(4.1)}$$

где TS - время с начала воспроизведения видео, I_V - количество обработанных кадров видеоматериала, F - частота кадров воспроизводимого видео, TF_i - время генерации одного кадра подсистемы визуализации.

Неравенство **(4.1)** может быть представлено в виде следующего алгоритма:

T1 – время с начала воспроизведения видео (мс);

T2 – предполагаемое время генерации следующего кадра подсистемы визуализации (с начала воспроизведения видео) (мс);

T3 – время генерации текущего кадра подсистемы визуализации (мс). Для первого кадра после начала воспроизведения видео предполагаемое время следующего кадра равно времени генерации текущего кадра (T3). Для расчета последующих значений применяются значения T3, которые сохраняются в массиве из 10 элементов. Для расчета T2 применяется среднее арифметическое значений из массива. При заполнении массива, хранящего значения T3, первое значение удаляется;

T4 – время отображения следующей видеотекстуры. Для первой отображаемой видеотекстуры равно T5 (мс);

T5 – 1/частоту кадров воспроизводимого видео (мс);

ЕСЛИ: |T1 - T4| <= |T2 - T4|

ТО: { необходимо обновить видеотекстуру в подсистеме визуализации;

T4 = T4 + T5 }

ИНАЧЕ: обновление видеотекстуры в подсистеме визуализации не требуется

Вне зависимости от результата описанной выше проверки, после нее происходит проверка на наполненность пиксельных буферов. Если хотя бы один пиксельный буфер не заполнен, то происходит добавление в него нового видеокадра.

Если после второй проверки необходимо обновить видеотекстуру в подсистеме визуализации, то обновление видеотекстуры происходит из соответствующего пиксельного буфера. Индекс используемого буфера определяется каждый раз при необходимости обновления текстуры. Так как для каждого видео предусмотрено 2 пиксельных буфера, то индекс следующего используемого буфера определяется по формуле: индекс следующего буфера = остаток от деления по модулю 2 суммы индекса предыдущего буфера и единицы.

При внедрении функциональности воспроизведения видео в подсистему визуализации возникли некоторые проблемы. Основными проблемами являлись рассинхронизация звука и видео во время воспроизведения и значительное падение производительности при воспроизведении видео высокой чёткости.

Алгоритм воспроизведения видео и синхронизации видео и звука в приложениях-медиаплеерах зачастую проще, чем разработанный алгоритм, так как в этих приложениях фактор производительности менее значим. Наиболее часто применяемым методом синхронизации является синхронизация видео по звуку, то есть скорость воспроизведения звука считается постоянной величиной (определяется на основании информации о частоте дискретизации и количестве каналов, указанной в видеофайле), а время отображения видеокадра рассчитывается на основании времени, записанном в пакетах, и времени, прошедшим с начала воспроизведения звука. Гораздо реже применяется синхронизация звука по видео, так как время отображения видеокадра всегда больше подвержено погрешностям, нежели воспроизведение звука. Соответственно, использование синхронизации звука по видео может приводить к «рваному» воспроизведению звука: скорость воспроизведения может замедляться или ускоряться.

В разработанном алгоритме скорость воспроизведения звука считается величиной постоянной, а время вывода видеотекстуры на экран рассчитывается с учетом производительности (кадров в секунду) подсистемы визуализации. Для корректной синхронизации видео и звука необходимо, чтобы их воспроизведение началось одновременно. Если отображение видео возможно практически сразу

после окончания процесса кэширования (максимальная задержка может составлять одну ИЦ подсистемы визуализации), то после запуска звукового потока проходит определенное время, необходимое для генерации буферов, используемых звуковым потоком для воспроизведения звука. В зависимости от количества передаваемых в звуковой поток данных в момент его запуска задержка может изменяться. В среднем, время, необходимое для генерации буферов, составляет от 150 до 250 миллисекунд. Если не компенсировать данную задержку, то рассинхронизация видео и звука будет заметна. Данная задержка присутствует только при первом запуске видео; если воспроизведение звука приостановить, а затем возобновить, то задержка отсутствует.

Для обеспечения синхронизации видео и звука, после кэширования и запуска аудиопотока, вводится задержка исполнения потока декодера. После указанной задержки запускается процесс воспроизведения видео. Первоначально величина задержки была константой, что для различных видеофайлов могло давать различные результаты – для одних видеофайлов видео и звук шли синхронно, при просмотре других была заметна рассинхронизация. В данном случае наиболее правильным методом было бы отслеживание состояния аудиопотока и, когда поток запущен, оповещать об этом управляющее приложение. Пакет SFML не предоставляет функциональности для отслеживания состояния, когда звуковые буферы сгенерированы и звук запущен, поэтому пакет SFML был модернизирован для отслеживания данного состояния. Благодаря доработке появилась возможность с приемлемой точностью определить момент запуска звука для обеспечения синхронизации звука и видео.

Второй серьёзной проблемой при воспроизведении видео в подсистеме визуализации стало снижение производительности при воспроизведении видео разрешением 1280 на 720 и более. Причина снижения производительности – время загрузки одной видеотекстуры в память видеокарты, которое составляло от 15 до 20 миллисекунд.

Первоначально для загрузки видеотекстуры в подсистему визуализации применялась базовая функциональность обновления текстур. Она включает в себя

получение указателя на содержимое существующей текстуры, копирование данных нового видеокадра в текстуру и помещение в видеопамять. Одним из предположений по причинам длительной загрузки текстуры являлось то, что видеокарте не хватает времени на обработку обновляемой в видеопамяти видеотекстуры, которая будет использована на следующем кадре подсистемы визуализации. Учитывая указанное предположение, был создан механизм заменяемых текстур.

Принцип данного механизма состоит в том, что при запуске видео создавалось две пустых текстуры, которые бы обновлялись заранее (за кадр или несколько кадров подсистемы визуализации до отображения), а при необходимости обновления видеотекстуры в подсистеме визуализации используемая текстура бы заменялась в материале трёхмерного объекта. Замена используемой текстуры в материале занимает доли миллисекунды, однако применение данной системы не решило проблему, так как скорость обновления текстур продолжала составлять от 15 до 20 мс.

Следующим вариантом решения проблемы производительности могло стать обновление текстуры в другом потоке. Применение дополнительного потока могло также привнести проблему, присущую многопоточности - синхронизация потоков. Например, в одном варианте реализации, при обновлении текстуры основной поток рендеринга ожидал бы окончания операции обновления, что сводило бы на нет всю эффективность данной системы. Поэтому механизм использования дополнительного потока был объединен с механизмом заменяемых текстур, когда одна текстура всегда содержит видеокадр, а вторая текстура обновляется в дополнительном потоке. Однако данный подход также себя не оправдал, так как при использовании дополнительного потока текстура просто не отображалась в подсистеме визуализации, несмотря на то, что весь код обновления текстуры выполнялся правильно и ошибок не выдавал. Всё дело в том, что графический интерфейс OpenGL, на котором основан графический модуль, как и графический интерфейс DirectX, являются по своей специфике однопоточными, то есть, команды, полученные из других потоков, будут

игнорироваться. При создании окна, в котором будет генерироваться изображение, создаётся так называемый контекст, который обеспечивает связь между созданным окном и графическим интерфейсом (DirectX, OpenGL). Контекст привязан к тому потоку, в котором был создан, поэтому все команды графического интерфейса будут выполняться только в этом потоке. Соответственно, все команды подсистемы визуализации, которые связаны с обновлением или заменой данных в памяти видеокарты, могут быть выполнены только в том потоке, в котором был создан контекст. Зачастую это основной поток программы или так называемый поток рендеринга.

Существует возможность передать управление контекстом другому потоку. Благодаря этому, возможно сделать другой поток способным управлять ресурсами видеокарты, однако все команды из основного потока будут игнорироваться. В теории, существует возможность быстро переключить контекст между потоками для обновления текстуры в видеопамяти в дополнительном потоке, однако на практике операция по изменению родительского потока контекста довольно ресурсоёмка. В результате проблема с производительностью с помощью данного метода так и не была решена.

Еще одним вариантом применения потоков и контекстов было создание второго контекста для дополнительного потока. В этом случае, возможно произвести объединение ресурсов для двух контекстов, что позволит управлять ресурсами подсистемы визуализации из двух независимых потоков. Однако, в большинстве случаев, для использования второго контекста необходимо создавать второе окно. Так как используемая оконная подсистема GLFW не позволяет создавать более одного окна, а также учитывая сложность внедрения данной функциональности и отсутствие возможности проверить эффективность методики до её внедрения, было принято решение не использовать данную функциональность.

После различных попыток использования дополнительных потоков для обновления текстуры в подсистеме визуализации было принято решение вернуться к использованию однопоточного обновления текстур, но с

использованием пиксельных буферов. Пиксельные буферы позволяют сократить время обновления текстур в видеопамяти за счёт того, что центральный процессор необходим только для обновления информации в самом пиксельном буфере, а передача информации из пиксельного буфера в текстуру происходит с помощью видеокарты, без участия центрального процессора. Однако, даже с использованием пиксельных буферов, скорость обновления текстур в видеопамяти продолжала колебаться в интервале от 10 до 20 мс.

Во время использования пиксельных буферов было сделано одно важное наблюдение, которое впоследствии и позволило решить проблему производительности. Рассмотрим стандартный цикл работы графического приложения:

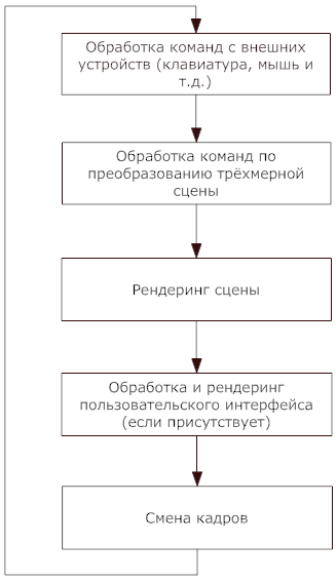

Рис. 4.3. Стандартный цикл работы графического приложения.

В большинстве случаев вся обработка и преобразование трёхмерной сцены происходит до генерации кадра для того, чтобы в кадре отображались последние состояния всех визуализируемых систем. Только графический интерфейс пользователя обрабатывается и генерируется после создания кадра, так как интерфейс должен накладываться уже на готовый кадр, иначе он будет «затёрт».

Во всех выше описанных способах обновления видеотекстур в подсистеме визуализации обновление видео происходило до рендеринга сцены.

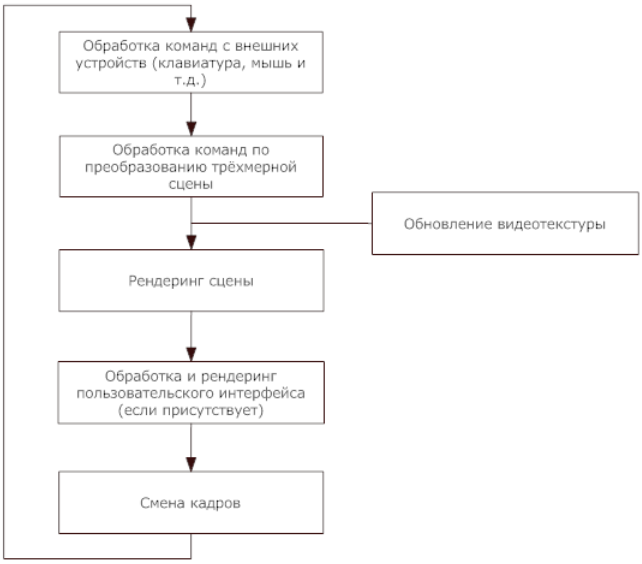

Рис. 4.4. Процесс обновления видеотекстур до рендеринга кадра не позволяет достичь необходимой производительности.

Проведение процесса обновления видео **после** рендеринга кадра позволило снизить время обновления текстуры в видеопамяти с 10-20 миллисекунд до 0.1-0.2 миллисекунд. Это позволило достичь стабильных шестидесяти кадров в секунду, генерируемых подсистемой визуализации, даже при воспроизведении видео с разрешением 1920 на 1080 (FullHD).

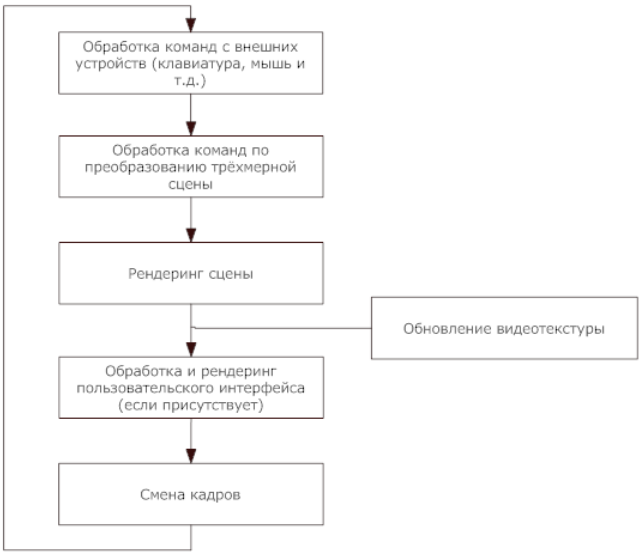

Рис. 4.5 Процесс обновления видеотекстур после рендеринга кадра позволяет достичь необходимой производительности.

Причин значительного снижения производительности может быть несколько. Первое: видеокарта не успевает обработать поступающие команды. По сути, процесс рендеринга изображения является конвейером. Процессор асинхронно задает команды видеокарте, которая сохраняет команды в очередь и последовательно их выполняет. Существуют некоторые команды, которые могут вызвать синхронизацию процессора и видеокарты, т.е. ситуацию, когда одно устройство будет ожидать другое. Основными командами, способными вызвать синхронизацию, являются команды получения данных из видеопамяти и загрузки новых данных в видеопамять. Так как для того, чтобы получить данные из видеопамяти или загрузить новые данные, видеокарте необходимо выполнить все стоящие в очереди задачи, то центральный процессор ожидает, пока они будут выполнены. Соответственно, данная задержка ведет к снижению производительности подсистемы визуализации. В случае с обновлением видео, конвейер видеокарты мог не успевать выполнить операцию по смене кадров. Так как данная операция является асинхронной, а обработка команд внешних устройств и трансформация сцены может занимать незначительное время (около одной миллисекунды), то перед загрузкой новых данных в видеопамять, видеокарте приходилось выполнять операцию по смене кадра. В то же время, после рендеринга кадра видеокарта в большинстве случаев оказывается свободной до операции смены кадра.

Второй причиной может служить использование многопоточности - в одном процессе становится больше потоков одного приоритета, то время, через которое процессу рендеринга будет отданы ресурсы процессора, может быть увеличено, а количество предоставляемого времени, наоборот, уменьшено. Был проведен следующий эксперимент. Использовали стандартную функциональность обновления текстур подсистемы визуализации при отсутствии дополнительных потоков. Текстура размером 1920 на 1080 загружается в видеопамять за 5-8 миллисекунд. Также, несколько раз было замечено, что задержка происходила не на этапе загрузки текстуры в видеопамять, а на этапе копирования нового видеокадра в пиксельный буфер. Данный этап никак не связан с работой

видеокарты и выполняется только центральным процессором. Это может означать то, что выполнение потока было приостановлено для выполнения других потоков с более высоким приоритетом выполнения. Данную проблему можно решить, если выставить потоку, в котором происходит обновление пиксельных буферов, более высокий приоритет, чем у других потоков приложения.

Результаты измерения производительности подсистемы визуализации с использованием различных методов обновления видеотекстур сведены в следующую диаграмму:

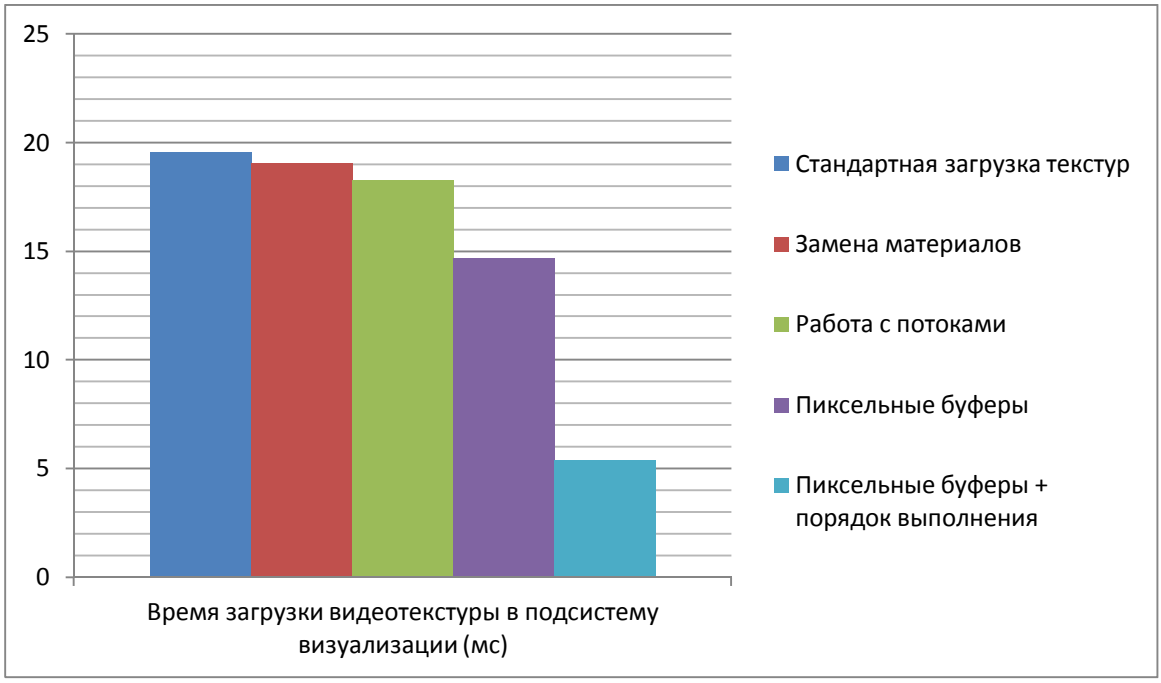

Рис. 4.6 Сравнение времени загрузки видеотекстуры для различных методов обновления

Описанные выше методы и алгоритмы применимы для воспроизведения одного видео с разрешением вплоть до 1920 на 1080 (FullHD) в подсистеме визуализации, либо нескольких видео с разрешением не выше 720 на 576 (576p, SD). Однако они не применимы для одновременного воспроизведения нескольких видео высокой чёткости из-за значительного падения производительности.

Для преодоления проблем с производительностью при одновременном воспроизведении нескольких видео высокой чёткости было создано несколько методов. Первый метод состоял в том, чтобы копировать данные новых

видеокадров в пиксельные буферы в отдельном потоке, а затем, в основном потоке, передавать эти данные в память видеокарты. Однако данный метод себя не оправдал, так как «узким местом» было не копирование информации в оперативной памяти, а передача данных в память видеокарты.

Второй метод основывался на использовании единого пиксельного буфера, который бы содержал кадры всех воспроизводимых видео. Преимуществом данного метода должно было стать то, что пиксельный буфер обновляется только один раз для всех воспроизводимых видео. Однако данный метод имел недостатки: была необходимость сохранять предыдущие видеокадры, чтобы избежать рассинхронизации видео и аудио, а также сложность реализации динамического расширения и уменьшения пиксельного буфера при добавлении или удалении видео в ходе работы приложения. Данный метод также не смог предоставить необходимой уровень производительности при обновлении нескольких видео.

Метод, который позволил получить оптимальную производительность при воспроизведении нескольких видео высокой чёткости, основан на применении приоритетов к обновлению воспроизводимых видео файлов.

Для каждого видео применяется весовой коэффициент, отражающий общий приоритет видео и весовой коэффициент выполняемого действия. Общий приоритет видео основывается на разрешении видео – чем ниже разрешение видео, тем выше приоритет, так как видео меньшего размера обновляется быстрее. Существует три общих приоритета видео – высокий (8), средний (4), низкий (0). Для видео с разрешением до 720 на 576 выставляется высокий приоритет, до 1280 на 720 - выставляется средний приоритет, для видео с разрешением до 1920 на 1080 выставляется низкий приоритет. Весовой коэффициент выполняемой операции меняется каждый раз при обновлении видео в зависимости от действия, которое необходимо будет выполнить при следующей ИЦ подсистемы визуализации. Выполняемые операции имеют следующие весовые коэффициенты (обновление видеотекстуры имеет наибольший коэффициент): обновление видеотекстуры (3), обновление видеотекстуры и

пиксельного буфера (2), обновление пиксельного буфера (1), исключение видео (0). Операция «исключение видео» выполняется в случае, когда на следующей ИЦ подсистемы визуализации нет необходимости обновлять видеотекстуру, а также все пиксельные буферы заполнены. Видео исключается из списка обновляемых, однако весовой коэффициент выполняемого действия меняется на высший для того, чтобы состояние видео было проверено на следующей ИЦ подсистемы визуализации.

Соответственно, расчет позиции видеоматериала в очереди обновления производится по следующей формуле:

$$P_V = B_V + K_V, \quad \textbf{(4.2)}$$

где P_V - позиция обрабатываемого видеоматериала в очереди обновления, B_V - базовый приоритет видеоматериала, K_V - весовой коэффициент выполняемой операции.

Список обновляемых видео определяется на каждой ИЦ подсистемы визуализации. На основе общего приоритета видео и весового коэффициента выполняемого действия, определяется позиция видео в списке обновления.

Алгоритм обновления видеотекстуры в подсистеме визуализации был также изменен для повышения производительности. Теперь первым производится обновление видеотекстуры, а не обновление пиксельного буфера. Так как операция обновления текстуры из пиксельного буфера является асинхронной, то она выполняется очень быстро, в то время как операция обновления информации в пиксельном буфере может вызвать синхронизацию процессора и видеокарты, что приведёт к задержкам. Также видеокарте после обновления пиксельного буфера может понадобиться некоторое время для обработки полученных данных. Поэтому, при обновлении текстуры, сразу после обновления пиксельного буфера может появиться задержка. Изменение алгоритма обновления видеотекстуры даёт больше времени на обработку данных пиксельного буфера. После обновления пиксельного буфера, одновременно с обработкой данных в пиксельном буфере видеокартой, происходит расчет весового коэффициента действия для следующей

ИЦ подсистемы визуализации и переход к обновлению следующей видеотекстуры [86–88].

Для реализации воспроизведения видеоматериалов в подсистеме визуализации ТОС были решены следующие задачи:

- разработаны методы и алгоритмы воспроизведения видеоинформации в виртуальном трехмерном окружении;
- разработаны методы и алгоритмы для одновременного воспроизведения нескольких видео, в т.ч. высокой чёткости;
- проведена оптимизация алгоритмов для достижения требуемого (не более 20 мс) времени генерации кадров в подсистеме визуализации при воспроизведении видеоинформации;
- разработана подсистема визуализации ТОС, внедрен программный модуль для декодирования видео, позволяющий обрабатывать видео файлы на различных программно-аппаратных платформах.

4.2. Методы и алгоритмы воспроизведения потоковых мультимедийных материалов

Воспроизведение потоковых видеоматериалов отличается от воспроизведения видеоматериалов, хранящихся в виде файлов. Основной задачей при воспроизведении потоковых видеоматериалов является достижение наименьшей задержки между получением информации с внешнего устройства и отображением уже преобразованной информации. При воспроизведении видеоматериалов из файлов, для обеспечения плавности воспроизведения применяется технология кэширования данных в оперативной памяти. Однако использование кэширования в значительной степени ограничено либо вовсе неприменимо при воспроизведении потоковых видеоматериалов в ТОС. При кэшировании данных образуется дополнительная задержка между получением данных с внешнего устройства и их отображением. Например, при воспроизведении потоковых видеоматериалов в формате HDV, задержка воспроизведения может составлять более секунды даже без использования кэширования. При кэшировании большого количества данных данная задержка будет увеличиваться, что неприемлемо для тренажерно-обучающих систем. Например, по требованиям ICAO к тренажерным системам, задержка отображения информации с момента подачи управляющего воздействия не должна превышать 100 миллисекунд [89]. Следовательно, использование технологии кэширования при воспроизведении потоковых видеоматериалов в ТОС следует минимизировать.

Поток данных с внешнего устройства (с видеокамеры) представляет собой поступающие с равной периодичностью массивы данных одинакового размера. В зависимости от формата передаваемых данных количество массивов данных в секунду и их размер могут различаться. Например, для видеоформата DV количество передаваемых массивов данных равно частоте кадров в секунду, соответственно, при 25 кадрах в секунду массивы данных с камеры поступают каждые 1 / 25 кадров = 40 миллисекунд. Для формата Mpeg 2, применяемом во многих камерах, поддерживающих видео высокой четкости, количество

передаваемых массивов данных может быть более 80, так как технология Mpeg 2 позволяет передавать не целый кадр, а только его изменения по сравнению с предыдущим. Также, для получения целого видеокадра может потребоваться несколько массивов данных (пакетов).

Зачастую поток данных, приходящий с видеокамеры, совмещает в себе звуковую и видеоинформацию. Такой поток называется мультиплексированным. Как формат DV, так и формат Mpeg 2 передают свои данные мультиплексированными. Для их воспроизведения сначала необходимо демультиплексировать поступающую информацию, т.е. произвести разделение видео и аудио данных на отдельные пакеты.

Основной проблемой получения пакетов с видеокамеры является отсутствие единого стандартного интерфейса для работы с внешними устройствами. Так, на операционных системах семейства Windows для работы внешними мультимедийными устройствами применяется интерфейс DirectShow, на операционных системах семейства Linux – Video For Linux 2 (v4l2), на операционных системах семейства Macintosh (MacOS) – QuickTime. Каждый из этих интерфейсов имеет собственную структуру и в значительной степени отличается от других интерфейсов.

Из-за необходимости осуществления захвата изображений на всех указанных выше платформах, было принято решение использовать пакет декодирования FFmpeg, способный работать на всех целевых платформах. Благодаря использованию пакета FFmpeg, большая часть кода декодера данных, поступающих с видеокамеры, остается неизменной, а меняется только та часть кода, которая специфична для используемой платформы.

В качестве базовой платформы была выбрана операционная система Windows. Выбор базовой платформы обусловлен ее широкой распространенностью и относительной стабильностью. Для захвата данных с камеры был использован интерфейс DirectShow. Это API (интерфейс программирования приложений), позволяющий Windows-приложениям управлять широким спектром устройств аудио/видео ввода, включающий (но не

ограниченный) DV камеры, веб-камеры, DVD-устройства, карты TV-тюнеров. DirectShow основан на технологии Component Object Model (COM).

DirectShow предлагает как высокоуровневую модель программирования, позволяющую быстро разрабатывать цифровые медиаприложения, так и низкоуровневую классовую модель, позволяющую сторонним производителям создавать собственные компоненты аудио и видео обработки.

Основным объектом в DirectShow является фильтр. Существуют три вида фильтров: фильтры захвата (получения данных), фильтры преобразования и фильтры визуализации (рендеринга).

Фильтры захвата позволяют получать данные из файла или, например, видеокамеры, веб-камеры, TV-тюнера, сетевого потока и т.д.. DirectShow тесно связана с моделью драйверов Windows (Windows Driver Model - WDM); любое медиаустройство с правильно реализованным WDM-драйвером автоматически предоставляется для приложения как DirectShow фильтр захвата.

Фильтры преобразования получают входящие данные от некоторого другого фильтра, обрабатывают их и передают следующему фильтру. Фильтры преобразования способны анализировать потоки, кодировать их и декодировать и т.д., т.е. проводить анализ или манипуляции над аудио и видеоданными. DirectShow предоставляет множество фильтров преобразования для управления различными форматами входных данных, включая аналоговые и телевизионные сигналы.

Фильтры визуализации принимают данные от фильтров захвата или преобразования и выводят их на внешние устройства (отображение информации на дисплее, вывод звука на колонки, запись данных в файл, передача данных на другие устройства и т.д.). Часть "Direct" в названии "DirectShow" отражает тот факт, что фильтры визуализации используют технологии DirectDraw и DirectSound для передачи данных в видео- и звуковую карту [90].

Для работы с фильтрами DirectShow создается граф фильтров, на который добавляются все необходимые фильтры захвата, преобразования и визуализации.

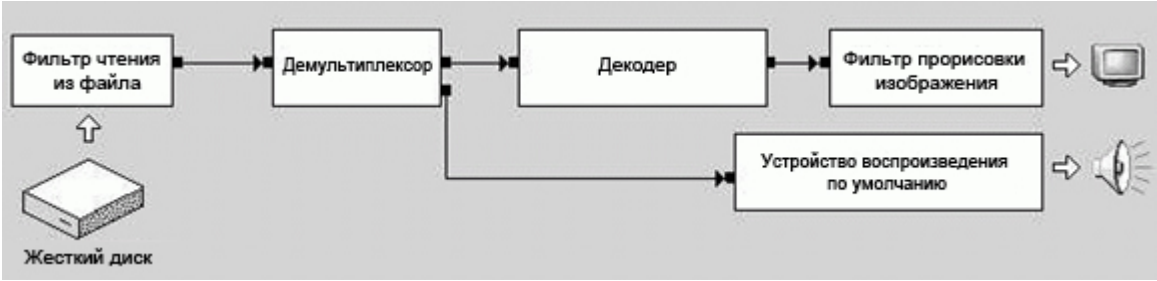

Рис. 4.7. Пример использования графа фильтров для воспроизведения видеофайла

Вместе с DirectShow поставляется множество стандартных фильтров для получения данных с внешних источников, преобразования этих данных и их отображения. Например, существуют отдельные фильтры захвата данных как для видеокамер, которые выводят видеоинформацию в формате DV, так и для видеокамер, выводящих информацию в формате Mpeg 2. Однако среди множества представленных стандартных фильтров отсутствует фильтр, который бы позволял получить неизмененные данные с видеокамеры для их дальнейшего преобразования внутри приложения. Из-за этого было принято решение разработать собственный фильтр получения данных с видеокамеры.

Задачей разработанного автором фильтра является получение и хранение определенного количества пакетов, поступивших с видеокамеры. Фильтр подсоединяется к выходу фильтра захвата видеокамеры и принимает пакеты данных через определенные интервалы времени (зависит от выходного формата видеоинформации).

Разработанный фильтр состоит из трёх основных частей: вход фильтра, который принимает данный с видеокамеры; интерфейс фильтра, позволяющий настраивать работу фильтра; выход фильтра, с которого происходит получение пакетов, полученных с камеры, приложением. Фильтр способен хранить до 20 поступивших с камеры пакетов. При превышении этого порога наиболее старый пакет удаляется из очереди.

Получение данных с видеокамеры и определение параметров входного потока состоит из нескольких этапов. Первым этапом является инициализация COM-модели и поиск внешних устройств захвата. Такими устройствами могут

являться видеокамеры, платы видеозахвата или устройства захвата звука (например, звуковые карты). Если хотя бы одно устройство было найдено, то происходит переход ко второму этапу. В первую очередь осуществляется поиск видеокамер и плат видеозахвата.

Второй этап состоит в определении типа выходного устройства, определении наличия и типа выходов фильтра захвата данного устройства, а также определение формата выходного сигнала с данного фильтра. Не для каждого типа выходного сигнала возможно определение параметров выходного сигнала, таких как используемая цветовая модель, высота и ширина изображение и т.д. Например, параметры сигнала, поступающего с видеокамеры в формате DV, могут быть определены сразу же после определения выходов фильтра захвата. Однако для формата MPEG 2 это невозможно, т.к. информация о параметрах выходного сигнала хранится не в каждом пакете, а только в пакетах, предшествующих началу группы изображений (Group of Pictures), в т.н. заголовках начала последовательности (sequence header). Для получения информации о параметрах выходного сигнала необходимо декодировать несколько пакетов. При обнаружении выходного сигнала в формате MPEG 2, потоку данных присваиваются параметры по умолчанию, а конкретные параметры определяются на следующих этапах.

Третьим этапом является создание экземпляра разработанного фильтра для получения данных с видеокамеры и присоединение его входа к выходу фильтра захвата видеокамеры. После выполнения третьего этапа возможно получение пакетов данных с видеокамеры.

Четвертым этапом является определение параметров входного сигнала и выставление параметров подсистемы декодирования FFmpeg. Для определения параметров входного сигнала происходит получение нескольких пакетов с видеокамеры. В зависимости от формата входного сигнала требуется различное количество пакетов. Например, для формата DV достаточно всего одного пакета для определения всех необходимых параметров декодирования подсистемы FFmpeg. Для формата MPEG 2 требуется порядка 130-140 пакетов для

корректного определения параметров входного сигнала. Во время определения параметров обрабатываемые пакеты автоматически демультиплексируются и сохраняются во внутренних буферах FFmpeg. После определения параметров входного сигнала все использованные пакеты удаляются из очереди хранения и внутренних буферов FFmpeg, чтобы исключить повреждение отображаемого при воспроизведении изображения из-за несовпадения изображений, хранимых в старых и новых пакетах.

По завершению четвертого этапа видеоматериалы, полученные с видеокамеры, могут быть воспроизведены в виртуальной трёхмерной сцене.

Рассмотрим подробнее структуру декодера поступающих с видеокамеры данных и его алгоритм работы.

Декодер состоит из трёх независимых частей:

- получение пакетов с видеокамеры, их демультиплексирование и сортировка видео- и аудио пакетов по соответствующим очередям
- декодирование видеопакетов
- декодирование аудиопакетов

Каждая независимая часть декодера работает в собственном потоке. Использование многопоточности позволяет значительно повысить производительность подсистемы декодирования при использовании многоядерного процессора.

После команды запуска воспроизведения видеоматериалов, поступающих с видеокамеры, создается три независимых потока и запускается фильтр получения пакетов с видеокамеры. В потоке получения пакетов с видеокамеры после запуска происходит кэширование определенного количества пакетов. Количество кэшированных пакетов зависит от формата входного сигнала. Например, для формата DV кэшируется до 4х пакетов, для формата MPEG 2 – до 10. После окончания кэширования заданного числа пакетов запускается рабочий цикл потока, в котором последовательно происходит получение пакетов с видеокамеры, их демультиплексирование и сортировка полученных видео- и

аудиопакетов по соответствующим очередям. В этом же рабочем цикле происходит проверка на необходимость кэширования декодированных аудио и видео данных. Если воспроизведение видеоматериалов было только что запущено, то при каждой итерации рабочего цикла потока происходит проверка на количество декодированных видеокадров и аудиоданных. Если количество декодированных видеокадров меньше установленного порога, то отправляется запрос на декодирование потоку декодирования видеопакетов. Аналогично проверяется количество декодированных аудиоданных.

Потоки декодирования работают в непрерывном режиме до тех пор, пока не превысят заданный порог. Например, поток декодирования видеопакетов работает непрерывно до тех пор, пока не будет декодировано 2 видеокадра. После этого выполнение потока останавливается до тех пор, пока от управляющего приложения не придет сигнал о необходимости декодирования следующего видеокадра.

Существует возможность ускорения процесса декодирования поступающих потоковых данных благодаря использованию вычислений на видеокарте (GPGPU). Данная возможность способна значительно повысить скорость декодирования, однако у нее есть недостатки. Во-первых, происходит снижение качества изображения[91]. Во-вторых, декодирование данных с использованием GPGPU даст дополнительную нагрузку на видеокарту, которая уже используется для визуализации виртуальной трехмерной сцены. Если применяемая трехмерная сцена является «тяжелой» для визуализации, то видеокарта может не успеть своевременно декодировать поступающие данные, что может привести к рассинхронизации видео и аудио данных, задержкам и рывкам изображения. Несмотря на перечисленные ограничения, при необходимости декодирования нескольких потоковых видеоматериалов с разрешением 3840x2160 и выше, использование мощностей видеокарты является оправданным.

Первоначально планировалось использовать стандартные фильтры, поставляемые вместе с ОС Windows, для демультиплексирования сигнала,

поступающего с видеокамеры. Однако у данного подхода были выявлены существенные недостатки.

Во-первых, для каждого выходного формата требуется свой фильтр демультиплексирования. Тестирование всех возможных выходных форматов является трудоемкой задачей и требует значительных финансовых затрат на приобретение оборудования для тестирования. Соответственно, существует риск, что разработчик не сможет определить и предоставить вместе со своим приложением все необходимые фильтры, что повлечет за собой невозможность воспроизведения потоковых видеоматериалов на компьютере пользователя при определенной конфигурации оборудования.

Во-вторых, для настройки работы фильтров демультиплексирования могут потребоваться дополнительные фильтры. Например, для фильтра-демультиплексора формата Mpeg 2 требуется дополнительный PSI-фильтр, который по входящему на фильтр-демультиплексор сигналу определяет наличие видео- и аудио сигналов, и создает соответствующие выходы у фильтра-демультиплексора. Данный PSI-фильтр не поставляется вместе с ОС Windows и должен быть включен в комплект поставки приложения.

В-третьих, при использовании фильтров демультиплексирования могут возникнуть существенные проблемы с определением параметров входящего сигнала. Например, для успешного декодирования входного сигнала в формате Mpeg 2 необходимо инициализировать подсистему декодирования FFmpeg с корректными параметрами. Для того чтобы получить данные из входного сигнала необходимо в каждом поступающем с демультиплексора видеопакете проверять наличие т.н. заголовка начала последовательности (sequence header). В данном заголовке хранятся данные о высоте и ширине изображения, о формате видеокадра (независимо сжатые кадры (I-кадры), кадры, сжатые с использованием предсказания движения в одном направлении (P-кадры) и кадры, сжатые с использованием предсказания движения в двух направлениях (B-кадры)), скорость передачи данных и т.д. Кроме того, специфика формата Mpeg 2 состоит в том, что в отличие от формата Mpeg 1, где все данные о входном потоке

хранились только заголовках начала последовательности, в формате Mpeg 2 присутствует механизм расширений, который позволяет хранить в заголовке начала последовательности только часть необходимой информации. Другая часть хранится в т.н. расширении заголовка начала последовательности (sequence extension). Для получения корректных данных о входном сигнале необходимо вручную, побитно (в форматах Mpeg 1 и Mpeg 2 многие данные хранятся в нестандартных типах данных для экономии места, например, переменная, определяющая высоту изображения, занимает 12 бит) декодировать входящие видеопакеты, объединить данные, полученные из основного заголовка и его расширения, и только затем применить полученные данные для инициализации подсистемы FFmpeg.

Закономерным является вопрос о необходимости использования подсистемы FFmpeg вместе с DirectShow, когда интерфейс DirectShow предоставляет все необходимые возможности для декодирования потоковых данных без использования FFmpeg. Во-первых, подсистема FFmpeg уже была успешно внедрена в подсистему визуализации и используется для воспроизведения видеоматериалов из файлов в виртуальной трехмерной сцене. Во-вторых, многие фильтры DirectShow, такие как FFDShow, используют для декодирования видео подсистему FFmpeg. Соответственно, использования таких фильтров как FFDShow дублировало бы уже внедренную и испытанную подсистему декодирования. В-третьих, использование только интерфейса DirectShow для декодирования потоковых видеоданных не избавило бы от проблемы подбора и тестирования фильтров, описанной выше.

Использование описанной выше методики позволяет воспроизводить потоковые видеоматериалы практически любого формата в подсистеме визуализации тренажерно-обучающей системы [92].

4.3. Архитектура подсистемы кеинга

Метод кеинга, реализованный в подсистеме визуализации ТОС, базируется на методе 3D кеинга. Так как любой алгоритм реализации кеинга является ресурсоемким, то при обработке изображений большого размера на центральном процессоре затруднительно достигнуть плавной работы подсистемы визуализации. В то же время одним из базовых требований к подсистеме визуализации является работа в реальном масштабе времени. Учитывая данное обстоятельство, было принято решение использовать вычислительные мощности графического процессора для реализации кеинга.

Проведен анализ алгоритма кеинга, указанного в [93]. Рир-проекция выполняется для каждого пикселя изображения, которое необходимо обработать. Пиксели изображения представляются в цветовой модели RGB, в то время как компонент альфа используется для определения прозрачности пикселя.

Первой проверкой является сравнение компонентов цвета пикселя друг с другом. Каждая компонента цвета представляется в диапазоне от 0 до 255. В зависимости от настроек параметров алгоритма происходит выбор базовой компоненты, доминирующей в фоне обрабатываемого изображения. Рассмотрим случай, когда базовой является компонента зеленого цвета (т.е. фон, на котором снимается объект, зеленый). Осуществляем проверку на преобладание зеленой компоненты цвета пикселя над остальными, то есть (G > R) и (G > B). Однако данный подход подвержен ошибкам, например, если G = x, R = x - 1 и B = x – 1, то данный цвет является оттенком серого. По вышеуказанному критерию данный пиксель будет считаться не принадлежащим объектам переднего плана, что неверно. Одним из способов повышения точности выбора пикселей с преобладающей зеленой компонентой является введение параметра дистанции.

$$d = \sqrt{(G-R)^2 + (G-B)^2}, d > dmax\ , \qquad \textbf{(4.3)}$$

где d - дистанция, определяющая степень преобладания зеленой компоненты пикселя над остальными; G - зеленая компонента пикселя; R - красная компонента пикселя, B - синяя компонента пикселя; *dmax* - дистанция, задаваемая пользователем, свыше которой пиксели признаются фоном.

Если условие $d > d_{max}$ выполняется, то пиксель не принадлежит объектам переднего плана и, соответственно, должен быть прозрачным в итоговом изображении. Рис. 4.8 показывает RGB куб, содержащий усеченную пирамиду OGYWC. Для каждой точки внутри пирамиды с координатами RGB выполняется условие $G \geq R$ и $G \geq B$.

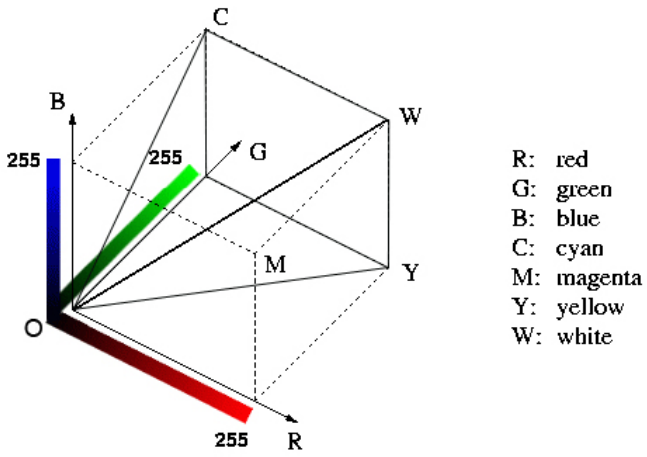

Рис. 4.8. Пирамида OGYWC

Рис. 4.9 показывает усеченную пирамиду OGYWC, пересеченную плоскостью S, параллельной диагонали OW. Необходимо отметить, что уравнение (4.3) применимо только для точек, расположенных внутри усеченной пирамиды (пиксели, в цвете которых преобладает зеленая компонента).

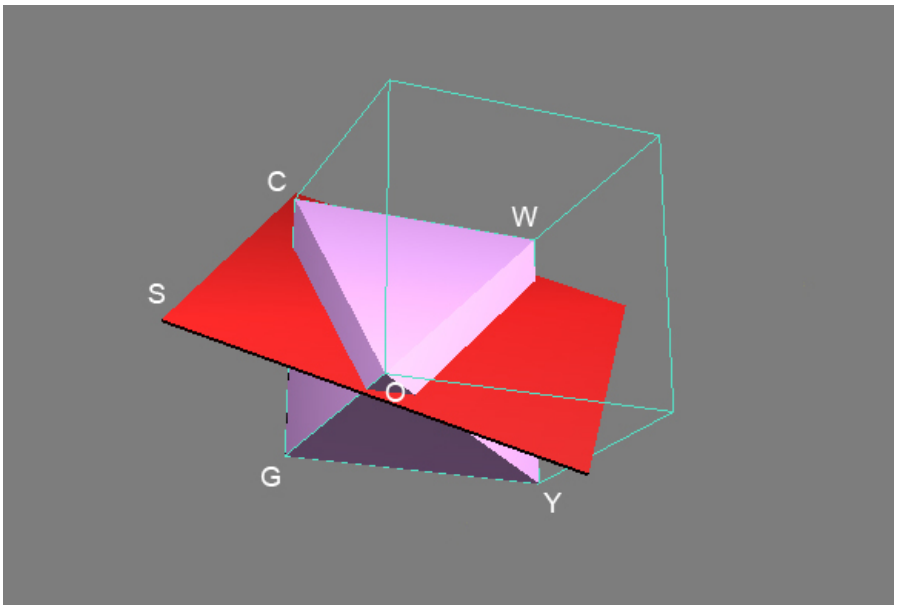

Рис. 4.9. Плоскость S определяет границу допустимых значений для цвета

Положение плоскости S определяется значением параметра d_{max}. Для каждого пикселя изображения производится расчет значения d, и маска прозрачности строится на основе сравнения значений d и d_{max}.

Генерируемая маска прозрачности позволяет задавать частичную прозрачность пикселя изображения. При объединении обработанного кеингом изображения с изображением, генерируемым подсистемой визуализации, итоговый пиксель будет являться линейной интерполяцией цвета пикселей двух изображений, а рассчитанное значение альфа будет выступать в качестве параметра.

Использование частичной прозрачности позволяет добиться плавного перехода между объектами переднего плана и фоновым изображением, генерируемым подсистемой визуализации. Дистанция d, рассчитанная по формуле (4.3), используется в качестве входящего параметра для функции расчета значения альфа. На основе параметров d, d_{max} и параметра мягкости (используемого в качестве порога, после которого пиксель считается лежащим в зоне перехода) рассчитывается значение альфа для конкретного пикселя. Формула расчета значения альфа основана на линейной интерполяции значений. Функция расчета значения альфа возвращает значение в отрезке [0:1] (0 – цвет полностью прозрачный, 1 – полностью непрозрачный). В OpenGL цвет представляется в виде значений от 0 до 1, а не от 0 до 255. Уравнение (4.3) возвращает значение на отрезке от 0 (полностью серый) до 510 (полностью зеленый), так как компоненты цвета R, G и B приведены к отрезку от 0 до 255 [93].

Описанный алгоритм не учитывает наличие отраженного света от фона на объектах переднего плана. Например, в области попадания отраженного света значение зеленой компоненты оказывалось меньше, чем значение красной компоненты, соответственно, такой пиксель признавался принадлежащим объектам переднего плана. Такие цвета близки к светло-желтому, поэтому хорошо заметны на коже инструктора. Для преодоления данной проблемы автором был изменен алгоритм путем добавления цветокоррекции. Пользователь выставляет пороговые значения, на основе которых алгоритм проверяет необходимость

выполнения цветокоррекции для данного пикселя. Если значение зеленой компоненты превышает или равно значению красной компоненты минус первое пороговое значение, задаваемое пользователем, и значение зеленой компоненты равно или превышает значение синей компоненты плюс второе пороговое значение, задаваемое пользователем, то значение зеленой компоненты данного пикселя будет изменено на значение, лежащее в середине отрезка между значениями красной и синей компонент пикселя (4.4).

$$P_G = \frac{P_R + P_B}{2}, \qquad \textbf{(4.4)}$$

где P_G – зеленая компонента пикселя, P_R – красная компонента пикселя, P_B – синяя компонента пикселя

Алгоритм кеинга реализован автором в виде шейдера посредством языка GLSL и выполняется при помощи видеокарты.

Таким образом, для реализации кеинга в подсистеме визуализации были решены следующие задачи:

- разработаны методы и алгоритмы кеинга в виртуальном трехмерном окружении;
- реализован кеинг с использованием вычислительных мощностей графического процессора с незначительным снижением производительности подсистемы визуализации (по сравнению с реализацией кеинга на центральном процессоре);
- введена цветокоррекция для устранения отраженного света от фона на объектах переднего плана.

Разработанная реализация алгоритма кеинга была внедрена в разрабатываемую подсистему визуализации и протестирована на нескольких видеоматериалах, воспроизводимых в реальном времени. Была измерена производительность подсистемы визуализации на видеоматериалах с разрешениями 1280*720 (HD), 1920*1080 (FullHD), 3840*2160 (4K). Результаты приведены на следующем графике:

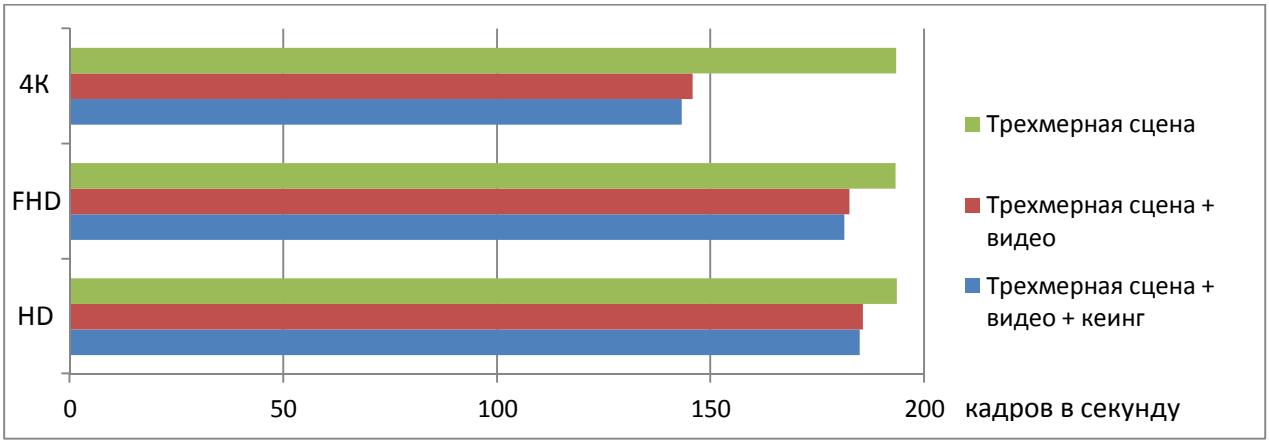

Рис. 4.10 Влияние разработанной реализации хромакеинга на производительность подсистемы визуализации

По полученным результатам можно судить, что влияние работы шейдера хромакеинга на частоту кадров подсистемы визуализации незначительно и составляет доли процента. Применение разработанного шейдера кеинга позволит отказаться от применения дорогостоящих аппаратных реализаций хромакеинга [94].

С учетом того, что обработка мультимедийных материалов происходит в режиме реального времени, реализован режим подготовки персонала промышленного предприятия территориально-распределённой структуры с обеспечением коммуникации инструктора и обучаемого персонала.

Глава 5 . Методы взаимодействия пользователя с виртуальным окружением при помощи программируемых сценариев

5.1. Описание технологий и концепций, применяемых в программируемых сценариях

Одним из основных методов взаимодействия пользователя с системой предтренажерной подготовки является использование программируемых сценариев. Пользователь создаёт сценарии, в которых определяет алгоритмы взаимодействия компонентов системы между собой и системы в целом. Например, сценарий может содержать алгоритмы для определения взаимодействий объектов виртуальной трехмерной сцены между собой, такие как просчет динамики столкновений объектов друг с другом в соответствии с физическими законами.

Сценарии задаются пользователем в графическом интерфейсе и обрабатываются менеджером сценариев (см.. Информационная архитектура автоматизированной системы обучения). Синтаксис команд сценариев основан на динамическом языке программирования Lua. Этот интерпретируемый язык программирования разработан подразделением Tecgraf Католического университета Рио-де-Жанейро (*Computer Graphics Technology Group of Pontifical Catholic University of Rio de Janeiro in Brazil*). В настоящее время Lua является популярным скриптовым языком в индустрии мультимедиа приложений и используется в ряде приложений в других предметных областях, например в Adobe Lightroom и компоненте FreePCRF (Policy and Charging Rules Function)[95].

Lua является сравнительно новым языком и позаимствовал черты и идеи из других языков:

- синтаксис структур управления логикой программы — из Modula;
- семантику более поздних версий — из Scheme;
- концепцию локальных переменных — из C++;
- концепцию наличия единственной встроенной структуры данных, используемой несколькими способами — из Lisp;
- использование ассоциативных массивов — из SNOBOL;

- множественные присвоения и возвраты из функций — из CLU и т.д.

Основополагающим принципом Lua является расширяемость семантики, т.е. предоставление мета-механизмов для реализации переменного набора инструментов вместо предоставления фиксированного набора инструментов. Это позволяет языку быть относительно простым, в то же время, сохраняя функциональность. Lua можно считать мультипарадигменным языком, поскольку он позволяет вести разработку в различных стилях.

Lua поддерживает логические, числовые (по умолчанию — числа с плавающей точкой двойной точности) и строковые атомарные типы данных. Единственным сложным типом данных является таблица — гетерогенный ассоциативный массив, позволяющий использовать разные типы данных для разных пар ключей и значений. Функции являются объектами первого класса, т.е. ими можно манипулировать точно так же, как переменными, передавать и получать как аргументы и т.д. [96].

По умолчанию динамический язык Lua использует интерпретатор для обработки введенных команд. Однако использование интерпретатора в значительной степени снижает производительность обработки сценариев и способно снизить производительность всей системы предтренажерной подготовки в целом. Чтобы избежать проблем с производительностью было принято решение использовать обработку сценариев при помощи так называемой компиляции «на лету» (Just-In-Time compilation, JIT compilation). Для этого в систему предтренажерной подготовки автором предполагается использовать пакет LuaJIT, способный в значительной мере повысить скорость обработки команд Lua [97].

Архитектурно, взаимодействие пользователя с системой предтренажерной подготовки при помощи программируемых сценариев реализовано согласно концепции (паттерну) «модель – представление – контроллер» (Model-View-Controller, MVC). Паттерн распределяет обработку взаимодействия с пользовательским интерфейсом между тремя участниками: моделью, представлением и контроллером. Он был разработан Тригве Реенскаугом (Trigve

Reenskaug) для платформы Smalltalk в конце 70-х годов прошлого столетия. С тех пор паттерн сыграл значительную роль в разработке множества инфраструктур и лег в основу целого ряда концепций проектирования пользовательских интерфейсов.

Типовое решение модель-представление-контроллер подразумевает выделение трех отдельных ролей. Модель - это объект, предоставляющий некоторую информацию об объекте. У модели нет визуального интерфейса, она содержит в себе все данные и поведение, не связанные с пользовательским интерфейсом. В объектно-ориентированном контексте наиболее "чистой" формой модели является объект модели предметной области (Domain Model).

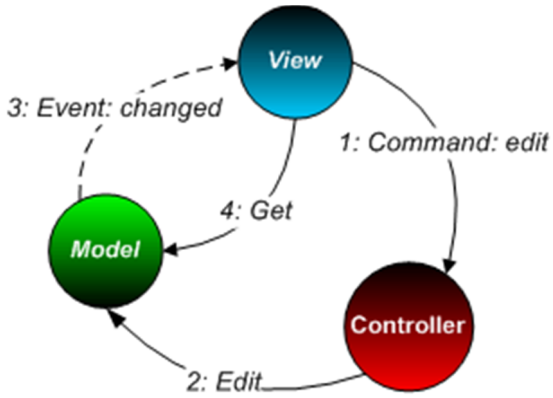

Рис. 5.1. Структура паттерна MVC [98]

Представление отображает содержимое модели средствами графического интерфейса. Функции представления заключаются только в отображении информации на экране. Все изменения информации обрабатываются контроллером. Контроллер получает входные данные от пользователя, выполняет операции над моделью и указывает представлению на необходимость соответствующего обновления. В этом плане графический интерфейс можно рассматривать как совокупность представления и контроллера.

Существует два принципиальных типа разделения: отделение представления от модели и отделение контроллера от представления. Отделение представления от модели - это один из фундаментальных принципов проектирования программного обеспечения. Наличие подобного разделения весьма важно по ряду причин.

В зависимости от ситуации возникает необходимость отображения одной и той же информации разными способами. Отделение представления от модели позволяет разработать для одной и той же модели несколько представлений, а точнее, несколько абсолютно разных интерфейсов. Наиболее наглядно данный подход проявляется в том случае, когда одна и та же модель может быть представлена с помощью Web-обозревателя, удаленного API или интерфейса командной строки.

Объекты, не имеющие визуального интерфейса, гораздо легче тестировать, чем объекты с интерфейсом. Отделение представления от модели позволяет легко протестировать всю логику модели. Ключевым моментом в отделении представления от модели является направление зависимостей: представление зависит от модели, но модель не зависит от представления. Это существенно облегчает разработку модели и одновременно упрощает последующее добавление новых представлений. Кроме того, это означает, что изменение представления не требует изменения модели. Данный принцип тесно связан с распространенной проблемой. При использовании программы с множеством диалоговых окон на экране могут одновременно находиться несколько представлений одной и той же модели. Если пользователь внесет изменения в модель посредством одного представления, эти изменения должны быть отражены и во всех остальных представлениях. Чтобы это было возможным при отсутствии двунаправленной зависимости, необходимо реализовать паттерн наблюдатель (Observer). В этом случае представление будет исполнять роль "наблюдателя" за моделью: как только модель будет изменена, представление генерирует соответствующее событие и все остальные представления обновляют свое содержимое.

Отделение контроллера от представления не играет такой важной роли, как предыдущий тип разделения. Классическим примером необходимости подобного разделения является поддержка редактируемого и не редактируемого поведения [99].

Для использования программируемых сценариев в подсистеме предтренажерной подготовки автором были разработаны модель языковой структуры, представление модели и контроллер. Модель языковой структуры содержит информацию о командах языка Lua и командах, используемых для управления функциональностью СПП. Представлением служит текстовое поле в графическом интерфейсе, в которое пользователем вносятся команды сценария. Контроллером служит менеджер сценариев, обеспечивающий обработку сценариев и изменение данных в модели.

5.2. Модель языковой структуры программируемых сценариев

Модель языковой структуры является основой для функциональности программируемых сценариев.

Основной составляющей модели является узел или элемент. Элемент содержит имя, текст, а также любую дополнительную пользовательскую информацию. Элементы также содержат данные о родительском и дочерних узлах, что позволяет выстраивать иерархию узлов в необходимом формате.

Существует три основных вида хранения информации в модели: в виде списка, в виде таблицы и в виде дерева. В модели языковой структуры используется хранение информации в виде дерева. Каждая команда управления являются элементом, который присоединяется как дочерний узел к главному узлу модели. Параметры команд управления также являются элементами и присоединяются к узлам команд управления в качестве дочерних узлов.

При инициализации модели производится определение всех команд управления СПП. Команды условно разделены на три категории: базовые операторы языка Lua, дополнительные команды языка Lua (например, функции по работе со строками, математические функции) и команды управления функциональностью СПП. Для каждой команды задается имя команды, родительский элемент команды, количество вводимых пользователем параметров, тип вводимых параметров и символ-разделитель. Символ-разделитель (обычно точка или скобка) используются алгоритмами подстановщика подсказок для

отделения команд от их параметров и вывода соответствующих подсказок. На данный момент все параметры в модели создаются вручную, однако, в дальнейшем все параметры модели можно будет сохранять в файл и загружать из файла.

Команды управления функциональностью СПП делятся на несколько категорий: журналирование, ресурсы, сцена, мультимедиа, система. Команды, относящиеся к журналированию, позволяют выводить пользовательскую информацию из сценария в лог-файл и окно менеджера журналирования в графическом интерфейсе. Существует возможность вывести сообщение об ошибке, предупреждение или обычное информационное сообщение.

Команды управления ресурсами обеспечивают получение доступа данным в сценариях, а также загрузку дополнительных пользовательских ресурсов напрямую из сценария.

Команды управления виртуальной сценой позволяют получить доступ к определенному узлу сцены. Возможно получение и задание таких параметров узла, как позиция, вращение, масштаб и видимость объекта. Кроме того, существуют команды для управления анимацией объектов трехмерной сцены.

Команды управления мультимедиа дают возможность работы с мультимедийными возможностями СПП. Поддерживается работа с изображениями, а также аудио- и видеоматериалами во всех популярных форматах. Воспроизведение мультимедиа материалов возможно не только из файлов, но и из потоковых источников, таких как видеокамеры. Присутствует возможность одновременного воспроизведения нескольких аудио- и видеоматериалов, в том числе высокой четкости. Количество одновременно воспроизводимых видеоматериалов ограничено только аппаратными возможностями СПП. Видеоматериалы можно воспроизводить на любом объекте виртуальной трехмерной сцены. В будущем планируется внедрить возможность воспроизведения мультимедиа материалов напрямую из сети Интернет.

Команды управления системой осуществляют базовое управление СПП во время исполнения сценария. Одной из основных функций являются приостановка

исполнения сценария до выполнения пользователем определенных действий, например, нажатия клавиши на клавиатуре.

Модель языковой структуры также получает данные от других моделей, используемых в СПП. Данные из других моделей используются в качестве значений параметров команд управления. При обновлении любой из моделей в СПП происходит передача новых данных в модель языковой структуры. После этого модель языковой структуры обновляет значения параметров в командах управления.

Данные из модели языковой структуры могут также использоваться для облегчения процесса ввода команд пользователем. Например, данные могут использоваться для подсветки и выведения подсказок для вводимых пользователем команд.

Проверка необходимости подсветки текста осуществляется при вводе каждого символа. В модуль обработки текста передается измененный пользователем блок, который форматируется при помощи регулярных выражений. Если модулем найдены совпадение введенного пользователем текста с командами, хранящимися в модели, то данные команды или их параметры будут подсвечены в соответствии с заданными правилами подсветки. Для каждого типа команд, таких как команды языка Lua или команды управления СПП, пользователь может задать свои правила подсветки.

Процесс отображения подсказок пользователю в зависимости от введенного текста значительно сложнее и состоит из большего числа этапов. Для отображения подсказок используется так называемый подстановщик подсказок (Completer). В подстановщике подсказок происходит обработка введенного пользователем текста и, при совпадении текста с данными из модели, выдается соответствующий список с подсказками.

Рассмотрим алгоритм отображения подсказок. Первым этапом является ввод пользователем текста в окно сценариев в графическом интерфейсе. Каждое нажатие клавиши пользователем происходит проверка на соответствие следующим правилам: нажатая клавиша не является клавишей-модификатором,

такой как Control или Shift; нажатая клавиша не является «горячей» клавишей для определенного функционала, например, CTRL+E; введенный символ не является символом окончания слова, например, символы равно, минус, плюс и т.д.. Если проверка пройдена, то текущая вводимая пользователем строка передается в подстановщик подсказок на дальнейшую обработку.

Вторым этапом является выделение подстановщиком подсказок отдельных команд и параметров этих команд из строки. Каждое слово в строке, разделенное пробелом или такими символами, как точка или скобка, проверяется в модели языковой структуры на соответствие. Если соответствие найдено, то элемент берется на дальнейшую обработку. Из элемента берутся данные об используемом символе-разделителе. Используемый символ-разделитель зависит от типа вводимой информации: точка является разделителем между типом команд и конкретной командой (например, типом команды может являться мультимедиа, конкретной командой может являться запуск воспроизведения); скобки являются разделителем между командой и параметрами команды; запятая является разделителем между параметрами команды.

Если найденный разделитель во введенной пользователем строке не соответствует символу-разделителю, указанному в элементе, то обработка данного элемента прекращается, и подсказки выведены не будут. Это сделано во избежание неправильного ввода команд сценария. Если же разделители совпадают, то по положению курсора в строке определяется, какой именно элемент требуется использовать в качестве подсказки (тип команды, команда или параметр команды).

Третий этап наступает после выбора пользователем необходимой подсказки и заключается в создании строки, содержащей ранее введенную информацию и выбранную пользователем подсказку. Итоговая строка получается из подстановщика подсказок и заменяет текущую строчку целиком. Курсор выставляется сразу после текста выбранной подсказки [100; 101].

Рассмотрим работу алгоритма на примере команды сценария для запуска воспроизведения видеоматериала. Команда содержит два параметра:

– название устройства, с которого происходит воспроизведение, или путь к файлу;

– трехмерный объект в виртуальной трехмерной сцене, на котором видео будет отображено.

В случае успешного выполнения команды возвращается идентификатор воспроизводимого видео в системе, а также начинается воспроизведение. Команда имеет следующий синтаксис: Media.PlayVideo (file name, object name). При вводе первых символов пользователю предлагается список возможных категорий команд, в данном случае – мультимедиа. После выбора категории введенные символы заменяются на название категории (Media). При вводе точки, являющейся символом-разделителем, пользователю предлагается список команд данной категории. Если команда выбрана и у нее присутствуют параметры, которые должны вводиться пользователем, то также отображаются предлагаемые варианты. Для рассматриваемой функции, при вводе первого параметра пользователю предлагаются определенные системой внешние устройства и видеофайлы, а при вводе второго параметра отображаются имена всех объектов в виртуальной трехмерной сцене.

Применение программируемых сценариев позволило сократить время на подготовку курса за счет оперативного изменения виртуальной трехмерной сцены при внесении коррекций в сценарий, а также методов оптимизации ввода правил в систему.

5.3. Реализация результатов работы при создании мультимедийного курса по обследованию и диагностике щеточно-контактных аппаратов турбогенераторов ГРЭС

Созданная автоматизированная система обучения предназначена для разработки курсов теоретической подготовки промышленно-производственного персонала по обследованию и диагностике щеточно-контактных аппаратов (ЩКА) турбогенераторов (ТГ) ТВВ-160 государственных районных электростанций (ГРЭС) с использованием разнородной мультимедийной

информации (Рис. 5.2.). Во время разработки курса теоретической подготовки использовались разнородные материалы: трехмерные модели с высокой детализацией, чертежи, фотографии и видеоматериалы для изучения конструктивного и технологического состава ЩКА ТГ. В состав материалов курса были включены термограммы, гистограммы, схемы охлаждения для демонстрации методов обследования ЩКА (Рис. 5.3.). Ниже приведены основные вопросы, освещаемые данным курсом.

Определение токораспределения на электрощетках. Анализируя гистограммы, построенные по результатам замера токов, проходящих через каждую щетку, можно получить информацию о токораспределении по электрощеткам. Выявив перечень электрощеток, несущих повышенную нагрузку или, наоборот, разгруженных по току, можно оперативно предотвратить аварийные ситуации путем проведения регулировочных операций и операций по замене электрощеток.

Определение теплового состояния ЩКА. С помощью определения теплового состояния ЩКА (термограммы) удается получить информацию по функционированию системы охлаждения, температуре контактных колец, температуре нагрева электрощеток, что позволяет выявить аварийно-опасные участки ЩКА и устранить сбои функционирования системы охлаждения.

В курсе используется анимированная трехмерная модель ЩКА для отображения процесса охлаждения. Движение охлаждающего воздуха в тангенциальном направлении обусловлено увлечением его контактными кольцами. Движение охлаждающего воздуха в осевом направлении происходит спонтанно по путям наименьшего пневмосопротивления. Горячий воздух выбрасывается концентрированно вниз по воздуховоду. На всасывающих холодный воздух частях со стороны ТГ усредненные значения скоростей потоков воздуха ниже, чем с правой стороны, на 0,22-0,95 м/с. Это приводит к неравномерному нагреву элементов ЩКА [102].

Для подготовки теоретического курса применяется персональный компьютер с операционной системой Windows или Linux с разработанным программным комплексом.

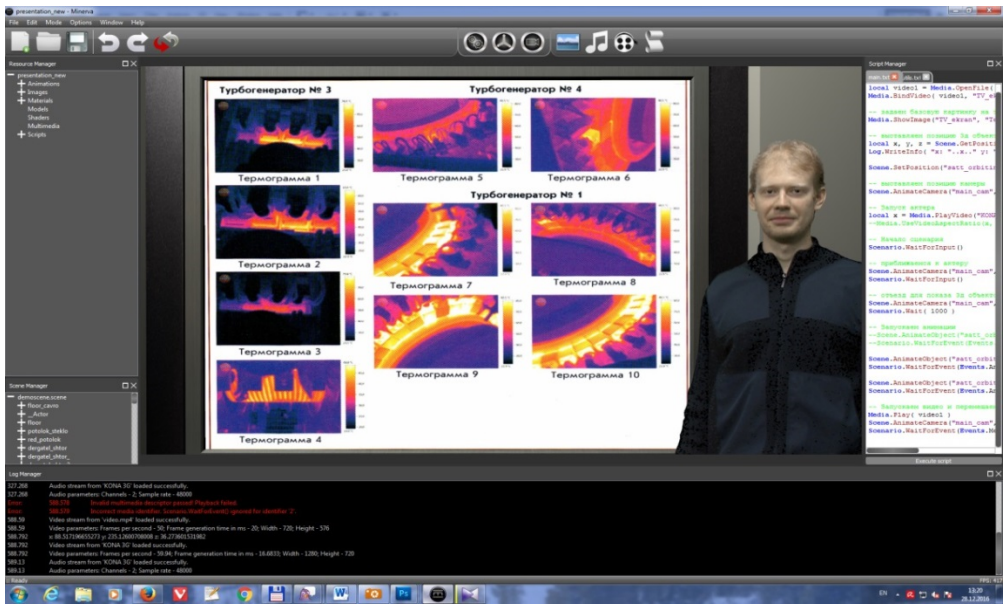

Рис. 5.2. Графический интерфейс разработанной автоматизированной системы обучения

Захват изображения инструктора производится при помощи видеокамеры, сигнал с которой передается на плату видеозахвата. Инструктор располагается перед однородным зеленым или синим фоном, который должен равномерно освещаться для уменьшения вероятности образования тени на однородном фоне. Для внедрения динамического изображения инструктора в виртуальную трехмерную сцену применяется созданная архитектура подсистемы воспроизведения мультимедийных материалов, обеспечивающая отображение видеоматериалов высокой четкости на гранях объектов виртуальной трехмерной сцены, а также методы воспроизведения потоковых мультимедиа материалов в виртуальном трехмерном окружении, обеспечивающие получение и отображение данных с внешних источников с минимальной задержкой. Выделение изображения инструктора с однородного фона и его внедрение в виртуальную трехмерную сцену производится при помощи разработанного метода хромакеинга.

Порядок действий, выполняемых автоматизированной системой обучения, задается при помощи разработанного метода управления с использованием программируемых сценариев.

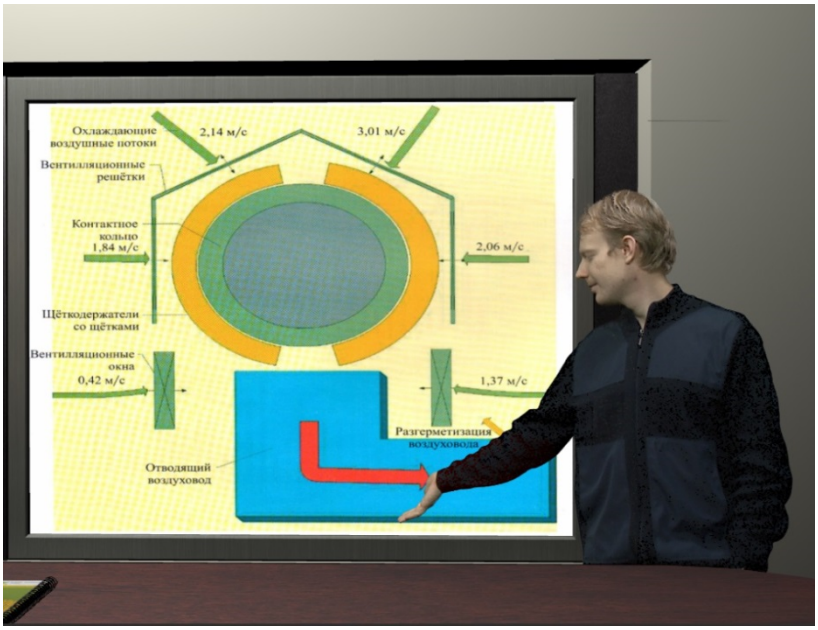

Рис. 5.3. Отображение схемы движения воздуха в ЩКА ТГ

Например: для прорисовки изображений на гранях объектов применяется команда ShowImage, задающая конкретный объект трехмерной сцены. Воспроизведение видеоматериалов осуществляется при помощи команды PlayVideo, позволяющей воспроизводить видео как из потоковых источников, таких как видеокамеры, так и из файлов. Анимация виртуальной камеры производиться командой AnimateCamera, задаются параметры анимируемой камеры, диапазон воспроизводимых кадров, а также скорость воспроизводимой анимации. Также предусмотрены команды для отображения трехмерных объектов, как статических, так и анимированных. Реализован режим ожидания событий, таких как команда от пользователя для продолжения сценария, завершение анимации трехмерного объекта. Указанные команды наиболее часто применялись при создании мультимедийного курса.

Разработанные новые методы, алгоритмы и программные модули были использованы ООО «ЭФ-КОНТЭЛ» в период 2015/16 г.г. при проведении обучения персонала предприятий электроэнергетики по обследованию и диагностике щеточно-контактных аппаратов турбогенераторов ТВВ-160 ГРЭС.

Заключение

В диссертационной работе содержится решение важной научной задачи разработки методов и алгоритмов визуализации, трансформации и анализа информации в режиме реального времени в тренажерно-обучающих системах промышленного применения. В ходе решения данной задачи автором лично получены следующие результаты:

- разработана архитектура подсистемы визуализации, обеспечивающая генерацию изображений в реальном времени (не менее 25 кадров в секунду), работу на нескольких платформах (Windows, Linux, Macintosh), имеющая небольшое число связей с внешними модулями, что позволяет повысить надежность системы;
- создана архитектура подсистемы воспроизведения мультимедийных материалов, позволяющая отображать разнородные материалы на гранях трехмерных объектах синтезированного виртуального окружения;
- разработаны методы и алгоритмы, позволяющие одновременно воспроизводить несколько видеоматериалов высокой четкости с разрешением 1920*1080 в виртуальном трёхмерном окружении, при поддержании частоты смены кадров подсистемы визуализации не ниже 25;
- представлены методы воспроизведения потоковых мультимедиа материалов в виртуальном трехмерном окружении, обеспечивающие прием данных с различных устройств, таких как видеокамеры, платы захвата, звуковые карты и др.;
- создан алгоритм хромакеинга, позволяющий выделять объекты переднего плана из однородного фона. Алгоритм использует цветокоррекцию для устранения засветки на краях объектов переднего плана и оптимизирован для использования мощностей современных видеокарт;
- разработана модель языковой структуры программируемых сценариев, обеспечивающая взаимодействие пользователя с системой предтренажерной подготовки. Данные из модели языковой структуры также используются для

оптимизации процесса ввода команд пользователем - подсветки текста и выведения подсказок в процессе ввода команд.

Разработанные новые методы, алгоритмы и программные модули были использованы ООО «ЭФ-КОНТЭЛ» в период 2015/16 г.г. при проведении обучения персонала предприятий электроэнергетики по обследованию и диагностике щеточно-контактных аппаратов турбогенераторов ТВВ-160 ГРЭС.

Обеспечена возможность использования в курсах подготовки промышленно-производственного персонала разнородной мультимедийной информации в т.ч. данных интерактивных моделирующих комплексов, поступающих в режиме реального времени. Применение программируемых сценариев позволило сократить время на подготовку курса за счет оперативного изменения виртуальной трехмерной сцены при внесении коррекций в сценарий, а также методов оптимизации ввода правил в систему. С учетом того, что обработка мультимедийных материалов происходит в режиме реального времени, обеспечена возможность коммуникации инструктора и обучаемого персонала промышленного предприятия с территориально-распределённой структурой.

Список терминов

видеокадр: Массив данных, хранящий декодированное изображение из видео файла в оперативной памяти, в цветовой модели BGRA;

кадр подсистемы визуализации: Изображение, сгенерированное графическим модулем, выводимое в подсистему отображения информации;

видеотекстура: Текстура в цветовой модели BGRA, используемая для представления видеокадра в подсистеме визуализации;

цикл генерации кадров подсистемы визуализации: Процесс обновления генерируемого графическим модулем изображения;

кэширование: Процесс сохранения данных в оперативной памяти компьютера для их дальнейшего использования; применяется во избежание задержек получения данных перед их обработкой.

видеопакет: Пакет, содержащий видеоинформацию, представленный в формате, используемом подсистемой декодирования FFmpeg.

аудиопакет: Пакет, содержащий звуковую информацию, представленный в формате, используемом подсистемой декодирования FFmpeg.

декодированный видеокадр: изображение без сжатия, которое может быть использовано подсистемой визуализации для обновления текстур в генерируемом виртуальном трёхмерном окружении.

Список литературы

1. Тренажер ПГУ-39 | АО «Тренажеры электрических станций и сетей» [Электронный ресурс]. URL: http://testenergo.ru/tren-pgu39/ (дата обращения: 28.12.2016).
2. Тренажер - Энциклопедический Фонд [Электронный ресурс]. URL: http://www.russika.ru/ef.php?s=3215 (дата обращения: 19.05.2014).
3. Васильев А.В. Дистанционное обучение как возможность расширения образовательного пространства в системе подготовки космонавтов. Звездный городок: РИО ФГБУ НИИЦПК им. Ю.А.Гагарина, 2009. С. 44–46.
4. А.Г.Бюшгенс. Современные тренажерные технологии в России // Аэрокосмический Курьер. 2010. № 5.
5. В.М.Шибаев. Современные нормы годности авиационных тренажеров – залог безопасности полетов // IV Международной конференции «Авиатренажеры, учебные центры и авиаперсонал – 2012. Москва: ЗАО ЦНТУ «Динамика», 2012.
6. Шибаев В.М., Аполлонов Д.В., Еркин И.Н. Методика определения запаздывания ответной реакции систем авиационного тренажера. Жуковский: ЦЭСАТ ФГУП “ЦАГИ”, 2011. 12 С.
7. Дьяков А .Ф. Гибридные тренажеры в энергетике. Теория и методы построения. Москва: МЭИ, 1994. 213 С.
8. Дьяков А.Ф., Венда В.Ф., Магазаник Я.М. Методические рекомендации по созданию систем обучения и тренажера для подготовки, переподготовки и поддержания натренированности операторов энергоблоков ТЭС и АЭС // 1985.
9. Дьяков А.Ф. Методы и технические средства подготовки персонала. Москва: МЭИ, 1996. 283 С.
10. Кубланов М.С., Ципенко В.Г., Бехтина Н.Б. Математическое моделирование движения тяжелых транспортных средств по ИВПП // Сборник научных трудов «Вопросы строительной механики и надёжности машин и конструкций». Москва: РИО МАДИ, 2010. С. 179–185.
11. Кубланов М.С., Ципенко В.Г., Васин И.С. Математические модели аэродинамики самолетов ОКБ АК «ИЛ» в научно-практических исследованиях и учебном процессе МГТУ ГА // Материалы XX школы-семинара «Аэродинамика летательных аппаратов». : Изд. ЦАГИ, 2009. С. 75–80.
12. Кубланов М.С., Ципенко В.Г., Васин И.С. Математическое моделирование поведения самолета Ил-96Т на больших углах атаки // Материалы XXI научно-технической конференции по аэродинамике. : Изд. ЦАГИ, 2010. С. 40–41.
13. Мамросенко К.А., Решетников В.Н. Моделирование подстилающей поверхности в имитационных системах // Программные Продукты И Системы. 2015. № 4. С. 70–74.
14. Решетников В. Н., Мамросенко К.А. Развитие тренажерных систем сложных технических комплексов // Материалы научно-практической конференции Полеты в космос. История, люди, техника. Москва: ФГБУ «Научно-исследовательский испытательный центр подготовки космонавтов имени Ю.А.Гагарина», 2014. С. 19–20.

15. Решетников В. Н., Мамросенко К.А. Принципы построения инновационных тренажерно-обучающих систем // Всероссийская научно-практическая конференция Проблемы и перспективы экономического развития ракетно-космической отрасли промышленности на период до 2030 г. и ее ресурсное обеспечение. Москва: , 2013.
16. Семенов Н.А., Мутовкина Н.Ю., Клюшин А.Ю. Программное обеспечение информационной технологии решения конфликтных ситуаций в многоагентной среде // Программные Продукты И Системы. 2016. № 2. С. 70–76.
17. Семенов Н.А. и др. Человекоразумные программные системы интеллектуальной поддержки решений креативных проблем // Программные Продукты И Системы. 2015. № 3. С. 12–18.
18. Семенов Н.А., Бурдо Г.Б., Палюх Б.В. Основы построения интеллектуальных систем управления технологическими процессами в многономенклатурном машиностроительном производстве // 2015.
19. Семенов Н.А., Бурдо Г.Б., Сорокин А.Ю. Модель автоматизированной системы управления качеством в многономенклатурном машиностроительном производстве // Программные Продукты И Системы. 2013. № 4. С. 46–51.
20. Бондарь Е.М., Сединко А.М. Применение технологий виртуальной реальности в учебном процессе // Пилотируемые полеты в космос: Сборник тезисов. М.О., Звездный городок: Редакционно-издательский отдел РГНИИЦПК им. Ю.А.Гагарина, 2005.
21. Радченко В.М., Бондарь Е.М., Чуланов А.О. Применение систем виртуальной реальности при подготовке персонала к борьбе за живучесть // Программные Продукты И Системы. 2015. № 4. С. 34–38.
22. Фомченко В.Н., Кошкин В.В. Учебно-тренировочный комплекс.
23. В.Н. Решетников. Космические телекоммуникации (начала). Тверь: ЗАО Научно-исследовательский институт «ЦентрПрограммСистем», 2009. 128 С.
24. В.Н. Решетников. Космические телекоммуникации. Системы спутниковой связи и навигации. Санкт-Петербург: ИПК ООО «Ленинградское издательство», 2010. 132 С.
25. Бобков А.Е., Леонов А.В., Ерёмченко Е.Н. 3D-документирование территории для систем виртуальной реальности // Вестник Компьютерных И Информационных Технологий. 2012. № 9. С. 13–17.
26. Бобков А.Е. и др. Особенности формирования изображений статических и динамических 3d-сцен в системах виртуального окружения // Приволжский Научный Журнал. 2015. № 1 (33). С. 84–92.
27. Бобков А.Е. и др. Программно-аппаратный комплекс для визуализации геофизических данных на сферическом экране // Научная Визуализация. 2015. Т. 7. № 2. С. 38–49.
28. Бобков А.Е., Леонов А.В. Процедурная реконструкция территорий на виртуальном ГЛОБУСЕ // Вестник Компьютерных И Информационных Технологий. 2015. № 11. С. 10–17.

29. А. Игнатенко, В. Фролов. Интерактивная трассировка лучей и фотонные карты на GPU // 9-я международная конференция по компьютерной графике и ее приложениям ГрафиКон’2009. Москва: , 2009. С. 255–262.
30. А. Игнатенко и др. Моделирование отражательных свойств материалов плоских объектов по фотоизображениям // 19-я международная конференция по компьютерной графике и ее приложениям ГрафиКон’2009. Москва: , 2009. С. 198–201.
31. А. Игнатенко, Е. Чепурнова. Трёхмерная визуализация архитектуры программных систем // Материалы докладов XV Международной конференции студентов, аспирантов и молодых ученых «Ломоносов». Москва: , 2008.
32. А. Игнатенко, В. Гаганов. Ускоренная визуализация эффекта дифракции Френеля для произвольного объектива // атериалы докладов XV Международной конференции студентов, аспирантов и молодых ученых «Ломоносов». Москва: , 2008.
33. Jorge Jimenez, Etienne Danvoye, Javier von der Pahlen. Next Generation Character Rendering // GDC 2013. , 2013.
34. Jorge Jimenez и др. Real-Time Realistic Skin Translucency // IEEE Comput. Graph. Appl. 2010. № 30. С. 32–41.
35. Jorge Jimenez и др. SMAA: Enhanced Subpixel Morphological Antialiasing // Comput. Graph. Forum. 2012. № 31.
36. Белышев Д.В., Куликов Д.Е., Хаткевич М.И. Визуализация данных в автоматизированном рабочем месте руководителя лечебно-профилактического учреждения // Программные Системы Теория И Приложения. 2010. № 4. С. 23–32.
37. Вязилов Е.Д. и др. Подходы по визуализации данных ЕСИМО // Российский Научный Электронный Журнал Электронные Библиотеки. 2014. № 3 (17).
38. ГОСТ 21659-76 Тренажеры авиационные. Термины и определения // 1976.
39. Трухин А.В. Анализ существующих в РФ тренажёрно-обучающих систем // Открытое И Дистанционное Образование. 2008. № 1 (29). С. 32–40.
40. Копытенкова О.И., Алиев О.Т. Анализ современных автоматизированных тренажерно-обучающих комплексов для подготовки локомотивных бригад // Известия Петербургского Университета Путей Сообщения. 2014. № 3 (40). С. 143–150.
41. Литвиненко А.А. Анализ состояния российского рынка авиационных технических средств обучения // IV Международной конференции «Авиатренажеры, учебные центры и авиаперсонал – 2012. Москва: ЗАО ЦНТУ «Динамика», 2012.
42. Мамросенко К.А., Решетников В.Н., Торгашев М.А. Архитектура программно-методического обеспечения на основе мультимедийных технологий // Пилотируемые полеты в космос. М.О., Звездный городок: РИО РГНИИЦПК им. Ю.А.Гагарина, 2007. С. 135–138.
43. Решетников В.Н., Мамросенко К.А. Базовые аспекты построения тренажерно-обучающих систем // Сборник материалов V Всероссийской научно-практической конференции «Компьютерная интеграция производства и ИПИ-технологии». Оренбург: ИП Осиночкин Я.В., 2011. С. 41–48.

44. В.Н. Решетников. Космические телекоммуникации (начала). М.: ИТЦ МАТИ, 2013. Вып. 2–е, дополненное. 184 С.
45. Решетников В.Н., Мамросенко К.А. Методика и алгоритмы визуализации для обучающих модулей компьютерных тренажерно-обучающих систем // Труды международной научной конференции, посвященной 80-летию со дня рождения академика В.А.Мельникова. Москва: Научный фонд «Первая Исследовательская Лаборатория имени академика В.А.Мельникова», 2009. С. 76–80.
46. Решетников В.Н., Мамросенко К.А. Основы построения тренажерно-обучающих систем сложных технических комплексов // Программные Продукты И Системы. 2011. № 3. С. 86–90.
47. Мамросенко К.А., Решетников В.Н. Подсистема управления образовательными ресурсами в тренажерно-обучающих системах // Пилотируемые полеты в космос. М.О., Звездный городок: ФГБУ НИИ Испытательный центр подготовки космонавтов им. Ю.А.Гагарина, 2011. С. 100 – 101.
48. Манаков Д.В. и др. Развитие подходов к разработке специализированных систем компьютерной визуализации // GraphiCon2015 : Междунар. науч. конф., Протвино, 22-25 сентября 2015: труды. Протвино: Ин-т физ.-техн. информатики, 2015. С. 17–21.
49. Манаков Д.В. и др. Задачи визуализации программного обеспечения параллельных и распределенных вычислений // Международная научная конференция Параллельные вычислительные технологии 2014. Ростов-на-Дону: Южный федеральный университет, 2014. С. 7–18.
50. Манаков Д.В. и др. Системные и визуализационные предпосылки создания виртуального испытательного стенда // Вопросы Оборонной Техники. 2012. № 2. С. 20–26.
51. Фомин В.И., Федоров А.В. Автоматизированная система обучения.
52. Graphic engine definition by The Linux Information Project (LINFO) [Электронный ресурс]. URL: http://www.linfo.org/graphic_engine.html (дата обращения: 08.12.2014).
53. 3ds Max | Программа для 3D-моделирования и визуализации | Autodesk [Электронный ресурс]. URL: http://www.autodesk.ru/products/3ds-max/overview (дата обращения: 08.12.2014).
54. Maya | Программа для компьютерной анимации и моделирования [Электронный ресурс]. URL: http://www.autodesk.ru/products/maya/overview (дата обращения: 08.12.2014).
55. Cinema4D Overview [Электронный ресурс]. URL: https://www.maxon.net/en/products/cinema-4d/overview/ (дата обращения: 08.12.2016).
56. Blender features [Электронный ресурс]. URL: https://www.blender.org/features/ (дата обращения: 08.12.2014).
57. 3D редакторы, плюсы и минусы [Электронный ресурс]. URL: http://www.interface.ru/home.asp?artId=27860 (дата обращения: 08.12.2016).

58. Основы оптики [Электронный ресурс]. URL: http://aco.ifmo.ru/el_books/basics_optics/glava-2/glava-2-3-s.html (дата обращения: 08.12.2014).
59. Eric Lengyel. Mathematics for 3D Game Programming and Computer Graphics Third Edition. : Course Technology, 2012. 545 С.
60. Naty Hoffman. Physically-Based Shading // Practical Physically-Based Shading in Film and Game Production. , 2010.
61. Yoshiharu Gotanda. Practical Implementation of Physically-Based Shading Models at tri-Ace // Practical Physically-Based Shading in Film and Game Production. , 2010.
62. Unreal Engine 4 [Электронный ресурс]. URL: https://www.unrealengine.com/unreal-engine-4 (дата обращения: 08.06.2016).
63. Alireza Tavakkoli. Game Development and Simulation with Unreal Technology. : A K Peters/CRC Press, 2015. 741 С.
64. Unreal Engine 4 Documentation [Электронный ресурс]. URL: https://docs.unrealengine.com/latest/INT/ (дата обращения: 15.12.2016).
65. Unreal Engine — Википедия [Электронный ресурс]. URL: http://ru.wikipedia.org/wiki/Unreal_Engine (дата обращения: 16.01.2014).
66. Richard Gerard Marcoux III и др. CRYENGINE Game Development Blueprints. : Packt Publishing Ltd, 2015. 322 С.
67. CryEngine 2 — Википедия [Электронный ресурс]. URL: http://ru.wikipedia.org/wiki/CryEngine_2 (дата обращения: 16.01.2014).
68. CryENGINE® 3 | Crytek [Электронный ресурс]. URL: http://www.crytek.com/cryengine/cryengine3/overview (дата обращения: 20.01.2014).
69. Valve Developer Community [Электронный ресурс]. URL: https://developer.valvesoftware.com/wiki/Main_Page (дата обращения: 10.12.2016).
70. История технологий - Source Engine | History | Тест GPU [Электронный ресурс]. URL: http://gamegpu.com/history/istoriya-tehnologii-source-valve.html (дата обращения: 10.12.2016).
71. Source — Википедия [Электронный ресурс]. URL: http://ru.wikipedia.org/wiki/Source_engine#.D0.9C.D0.BD.D0.BE.D0.B3.D0.BE.D1.8F.D0.B4.D0.B5.D1.80.D0.BD.D1.8B.D0.B9_.D1.80.D0.B5.D0.BD.D0.B4.D0.B5.D1.80.D0.B8.D0.BD.D0.B3 (дата обращения: 16.01.2014).
72. Fraser Harrison - Post-Production: The Difference Between Chroma Key and Luma Key [Электронный ресурс]. URL: http://fraser-harrison-postproduction.blogspot.ru/2013/03/the-difference-between-chroma-key-and.html (дата обращения: 30.09.2014).
73. Bill Schnarr. How Chroma Keying Works [Электронный ресурс]. URL: http://www.signvideo.com/chrom-ky-wks.htm (дата обращения: 30.09.2014).
74. Thermo-Key [Электронный ресурс]. URL: http://nae-lab.org/project/thermo-key/ (дата обращения: 30.09.2014).
75. CryEngine 3 обойдется покупателям в большую сумму [Электронный ресурс]. URL: http://gotps3.ru/article/cryengine_3_obojdetsja_pokupateljam_v_boljshuju_summu-180312/ (дата обращения: 13.05.2014).

76. CRYENGINE Limited Licence Agreement [Электронный ресурс]. URL: https://www.cryengine.com/ce-terms (дата обращения: 22.11.2016).
77. Архитектура автоматизированной системы [Электронный ресурс]. URL: http://www.bookasutp.ru/Chapter1_0.aspx (дата обращения: 24.01.2014).
78. ISO/PAS 17506:2012 - Industrial automation systems and integration -- COLLADA digital asset schema specification for 3D visualization of industrial data [Электронный ресурс]. URL: http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?ics1=25&ics2=040&ics3=40&csnumber=59902 (дата обращения: 03.02.2014).
79. Малашин А. DirectX против OpenGL // Хакер Железо. 2004. № 6. С. 68.
80. Михайлюк М.В., Торгашев М.А. Технология работы с модельно-видовыми матрицами в видеотренажерах // Сб. трудов, посвященный академику С.А.Лебедеву. Москва: , 2002.
81. Гиацинтов А.М., Мамросенко К.А. Методы анимации виртуальной камеры и отображения объектов с частичной прозрачностью в тренажерно-обучающих системах // Информационные Ресурсы России. 2011. № №6 (124). С. 31–34.
82. Alpha-канал (Альфа-канал) [Электронный ресурс]. URL: http://www.eskizspb.ru/glos_alph.html (дата обращения: 02.06.2011).
83. Форматы: PNG [Электронный ресурс]. URL: http://netghost.narod.ru/gff2/graphics/summary/png.htm (дата обращения: 26.05.2011).
84. Mehrabian A. Silent messages: Implicit communication of emotions and attitudes (2nd ed.). Belmont, California: Wadsworth, 1981. 196 С.
85. Многопоточность — Википедия [Электронный ресурс]. URL: http://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B3%D0%BE%D0%BF%D0%BE%D1%82%D0%BE%D1%87%D0%BD%D0%BE%D1%81%D1%82%D1%8C (дата обращения: 18.05.2011).
86. Гиацинтов А.М. Отображение разнородных видеоматериалов на гранях трехмерных объектов в подсистеме визуализации тренажерных обучающих систем // Программные Продукты И Системы. 2012. № 3. С. 80–86.
87. Гиацинтов А.М., Мамросенко К.А. Одновременное воспроизведение разнородных видеоматериалов в виртуальной сцене в подсистеме визуализации ТОС // Информационные Ресурсы России. 2012. Т. 4. С. 25–28.
88. Giatsintov Alexander, Mamrosenko Kirill, Reshetnikov Valeriy. Playback of heterogeneous videos in 3d environment in e-learning systems // Proceedings International Conference Southeast Asian In the 21 Century Open and Distance Learning. Danang, Vietnam: In Savina, 2012. С. 29–37.
89. Organization I.C.A. Doc 9625-AN/938 Manual of Criteria for the Qualification of Flight Simulation Training Devices. Montreal, Canada: International Civil Aviation Organization, 2009.
90. Валентин Вовк. Что такое DirectShow? Что такое фильтр? - DirectShow по-русски [Электронный ресурс]. URL: http://directshow.wonderu.com/%D1%81%D1%82%D0%B0%D1%82%D1%8C%D0%B8/%D0%BF%D0%B5%D1%80%D0%B2%D1%8B%D0%B5-

%D1%88%D0%B0%D0%B3%D0%B8-%D1%81-directshow/%D1%87%D1%82%D0%BE-%D1%82%D0%B0%D0%BA%D0%BE%D0%B5-directshow-%D1%87%D1%82%D0%BE-%D1%82%D0%B0%D0%BA%D0%BE%D0%B5-%D1%84%D0%B8%D0%BB%D1%8C%D1%82%D1%80 (дата обращения: 13.12.2011).
91. Сжатие видео и декодирование | чем и на чём лучше | Intel Quick Sync, Nvidia CUDA, AMD APP - THG.RU [Электронный ресурс]. URL: http://www.thg.ru/video/video_transcoding_amd_nvidia_intel/print.html (дата обращения: 27.03.2014).
92. Гиацинтов А.М., Мамросенко К.А. Воспроизведение потоковых видеоматериалов в подсистеме визуализации тренажерно-обучающей системы // Программная Инженерия. 2014. № 7. С. 33–39.
93. F. van den Bergh, V. Lalioti. Software Chroma Keying in an Immersive Virtual Environment // South Afr. Comput. J. 1999. № 24. С. 155–162.
94. Гиацинтов А.М., Мамросенко К.А. Метод рир-проекции в подсистеме визуализации тренажерно-обучающей системы // Программные Продукты И Системы. 2014. № 4. С. 31–37.
95. FreePCRF | Policy and Charging Rules Function [Электронный ресурс]. URL: http://freepcrf.com/ (дата обращения: 11.02.2016).
96. Lua - Энциклопедия языков программирования [Электронный ресурс]. URL: http://progopedia.ru/language/lua/ (дата обращения: 13.02.2014).
97. LuaJIT Performance: x86/x64 [Электронный ресурс]. URL: http://luajit.org/performance_x86.html (дата обращения: 11.02.2016).
98. Обобщенный Model-View-Controller [Электронный ресурс]. URL: http://rsdn.ru/article/patterns/generic-mvc.xml (дата обращения: 24.02.2014).
99. Паттерн Model View Controller [Электронный ресурс]. URL: http://serg-dobrinin.narod.ru/tutorial/oop.htm#_Toc182894219 (дата обращения: 10.02.2014).
100. Гиацинтов А.М., Мамросенко К.А. Управление тренажерно-обучающими системами при помощи программируемых сценариев // Вестник Компьютерных И Информационных Технологий. 2016. № 5. С. 52–56.
101. Гиацинтов А.М. Метод формирования управляющих воздействий в системах предтренажерной подготовки // Энергетик. 2015. № 4. С. 27–28.
102. Безчастнов К.К., Прокопенко Н.Н., Старцев А.В. Сравнительный анализ условий функционирования щёточно-контактных аппаратов турбогенераторов ГРЭС // Энергетик. 2012. № 7. С. 2–6.
103. Christopher Schultz. Digital Keying Methods. Germany: University of Bremen, 2006.
104. K.A.Mamrosenko. Training simulation systems for open distance learning // Proceedings Conference Open Distance Learning Towards Building Sustainable Global Learning Communities. Hanoi, Vietnam: The Gioi, 2010. С. 509–515.
105. Гиацинтов А.М., Мамросенко К.А. Описание архитектуры подсистемы визуализации тренажерно-обучающих систем // Труды Международной

молодежной научной конференции Гагаринские чтения. М: ИЦ МАТИ, 2011. С. 85–87.
106. OpenGL 2.1 Reference Pages [Электронный ресурс]. URL: http://www.opengl.org/sdk/docs/man/ (дата обращения: 18.05.2011).
107. Nicolas Schulz. Horde3D Documentation [Электронный ресурс]. URL: http://www.horde3d.org/docs/manual.html (дата обращения: 18.05.2011).
108. MPEG-2 video compression [Электронный ресурс]. URL: http://www.bbc.co.uk/rd/pubs/papers/paper_14/paper_14.shtml (дата обращения: 18.05.2011).
109. YUV pixel formats [Электронный ресурс]. URL: http://www.fourcc.org/yuv.php (дата обращения: 18.05.2011).
110. INTUIT.ru: Курс: Common Intermediate ..: Лекция №11: Основы многозадачности [Электронный ресурс]. URL: http://www.intuit.ru/department/pl/cil/11/1.html (дата обращения: 18.05.2011).
111. Гиацинтов А.М., Мамросенко К.А., Решетников В.Н. Инструментальные средства предтренажерной и тренажерной подготовки операторов сложных технических систем // Программные Продукты Системы И Алгоритмы. 2014. № 1. С. http://swsys-ru/simulator-training-operators.html.
112. Гиацинтов А.М., Мамросенко К.А. Использование изображений с частичной прозрачностью в подсистеме визуализации Horde3D // Труды Международной молодежной научной конференции Гагаринские чтения. М: ИЦ МАТИ, 2011. С. 48–50.
113. Гиацинтов А.М., Мамросенко К.А. Модуль создания структуры курса для этапа предварительной подготовки в тренажерно-обучающих системах (ТОС) // Труды Международной молодежной научной конференции Гагаринские чтения. М: ИЦ МАТИ, 2010.
114. Гиацинтов А.М., Мамросенко К.А. Модуль формирования структуры курса в тренажерно-обучающих системах // Труды Международной молодежной научной конференции Гагаринские чтения. М: ИЦ МАТИ, 2009.
115. Гиацинтов А.М., Мамросенко К.А. Описание графа сцены подсистемы визуализации тренажерно-обучающих систем // Научные труды Международной молодежной научной конференции Гагаринские чтения. М: ИТЦ МАТИ, 2013. С. 88–89.
116. Гиацинтов А.М., Мамросенко К.А., Афанасьев А.П. Основы построения физического модуля для имитационных систем // VI Всероссийская научно-практическая конференция «Компьютерная интеграция производства и ИПИ-технологии». Оренбург: ООО ИПК «Университет», 2013. С. 246–256.
117. Гиацинтов А.М., Мамросенко К.А., Маркианова Н.С. Тренажерно-имитационные и обучающие распределенные системы // Труды Международной молодежной научной конференции «Гагаринские чтения». М: ИЦ МАТИ, 2008. С. 168–170.