На правах рукописи

Санжаров Вадим Владимирович

Разработка расширяемой системы фотореалистичного рендеринга на GPU

Специальность 2.3.5 – математическое и программное обеспечение
вычислительных систем, комплексов и компьютерных сетей

АВТОРЕФЕРАТ

диссертации на соискание учёной степени
кандидата физико-математических наук

Москва 2023

Работа выполнена в Федеральном государственном учреждении «Федеральный исследовательский центр Институт прикладной математики им. М.В. Келдыша Российской академии наук»

Научный руководитель:	**Фролов Владимир Александрович** кандидат физико-математических наук, старший научный сотрудник (по совместительству) отдела №2 ИПМ им. М.В. Келдыша РАН, научный сотрудник факультета ВМК МГУ им. М.В.Ломоносова
Официальные оппоненты:	**Семенов Виталий Адольфович** доктор физико-математических наук, профессор, заведующий отделом «Системная интеграция и прикладные программные комплексы» ИСП РАН
	Турлапов Вадим Евгеньевич доктор технических наук, доцент, профессор кафедры математического обеспечения и суперкомпьютерных технологий (МОСТ) института информационных технологий, математики и механики (ИИТММ), руководитель лаборатории компьютерной графики и мультимедиа МОСТ ИИТММ, соруководитель магистерской программы «Искусственный интеллект» ИИТММ, Нижегородский государственный университет им. Н.И. Лобачевского
Ведущая организация:	Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

Защита диссертации состоится 31 октября 2023 г. в __ часов на заседании диссертационного совета 24.1.237.02, созданного на базе ИПМ им. М.В. Келдыша РАН, расположенного по адресу: 125047, г. Москва, Миусская пл., д.4

С диссертацией можно ознакомиться в библиотеке и на сайте ИПМ им. М.В. Келдыша РАН: www.keldysh.ru

Автореферат разослан «____ » _____ 202_ г.

Ученый секретарь диссертационного совета 24.1.237.02
кандидат физико-математических наук М.Г. Широбоков

Общая характеристика работы

Актуальность работы. Фотореалистичный рендеринг подразумевает возможность синтеза изображений с высоким уровнем реализма, которые визуально практически неотличимы от фотографий. Области применения программных систем фотореалистичной визуализации включают в себя: промышленный дизайн, архитектурную визуализацию (рис. 1), светодизайн, создание кино- и анимационных фильмов, различных видеопоследовательностей, тренажерные комплексы, генерацию данных для задач машинного обучения. Требуемая функциональность рендер-системы в перечисленных областях варьируется как между отдельными областями, так и между различными проектами внутри одной области. Поэтому одной из наиболее критичных характеристик рендер-систем является расширяемость, т.е. возможность «безболезненного» (т.е. не нарушающего работоспособность программы) добавления или изменения функциональности (включая новые математические модели) для адаптации к новым практическим задачам. Т.к. невозможно заранее предусмотреть всё, что потребуется пользователям в будущем, расширяемая рендер-система должна позволять добавлять новую функциональность и конечным пользователям.

Другой важной характеристикой является скорость расчета. В настоящее время реализация рендер-систем на GPU является одним из наиболее действенных и практичных способов ускорения расчёта.

Серьезным препятствием использования GPU для фотореалистичного рендеринга является высокая трудоемкость добавления новой функциональности и интеграции с пользовательскими CPU приложениями. Одна из причин в том, что GPU рендер-система и пользовательское приложение работают в разных адресных пространствах, из-за чего невозможно напрямую передавать или вызывать функции одного приложения из другого. Другая проблема связана со сложностью и ограничениями разработки на GPU.

В индустриальных приложениях фотореалистичного рендеринга часто возникает потребность в расширении доступной функциональности рендер-системы пользовательским кодом, а также в интеграции рендер-систем с многочисленными клиентскими приложениями (3D моделирования, текстурирования, композирования и пр.). Пользовательский код часто является специфичным для конкретного проекта в связи с творческим характером областей применения. Это является одной из причин, по которой CPU рендер-системы, за счет высокой степени гибкости и относительной простоты интеграции с другими приложениями, продолжают использоваться, несмотря на значительные преимущества GPU рендер-систем по скорости.

Таким образом, разработка методов и подходов к построению систем фотореалистичного рендеринга на GPU, которые бы позволили обеспечить расширяемость, сохранив преимущества в производительности, является актуальной задачей.

Рис. 1 Примеры изображений, созданных пользователями разработанных решений в ПО Autodesk 3ds Max.

Целью работы является разработка подходов и архитектурных решений для систем фотореалистичного рендеринга на GPU, допускающих расширение функциональности системы (включая добавление новых математических моделей) путем интеграции с готовыми инструментами приложений-клиентов (таких как приложения для 3D моделирования) и использования пользовательского кода расширений на GPU, без необходимости внесения изменений в существующий программный код рендер-системы и его полной перекомпиляции.

Основные задачи исследований

- Анализ требований к современным системам фотореалистичного рендеринга.
- Разработка архитектуры GPU рендер-системы, позволяющей обеспечить её интеграцию с клиентскими приложениями с возможностью использовать реализованную в них функциональность по созданию программируемых компонентов рендер-системы (таких как процедурные текстуры).

- Разработка подхода к исполнению кода пользовательских расширений для GPU рендер-системы.
- Реализация и апробация разработанных решений в составе GPU рендер-системы.

Научная новизна

- Предложена архитектура рендер-системы с использованием дополнительного программного слоя на основе объектной базы данных, решающего задачи организации интеграции и инфраструктуры. Предложенная новая программная архитектура позволяет:
    - работать с 3D сценами, не помещающимися в оперативную память вычислительной машины пользовательского приложения;
    - изменять и добавлять новые математические модели материалов, источников света и других компонентов 3D сцен без внесения изменений в код инфраструктурного слоя;
    - выполнять сериализацию, импорт и экспорт объектов 3D сцен между разными сценами и различными приложениями;
    - отслеживать изменения, производимые в сцене пользователем, что дает возможность эффективной передачи данных, в том числе и при распределенной работе со сценой – между разными приложениями передаются только изменения, а не вся сцена целиком;
    - организовать эффективную отладку и поиск ошибок за счет механизма отслеживания изменений, который позволяет локализовать действия пользователя, повлекшие за собой сбой в рендер-системе или клиентском приложении.
- Предложен метод для разработки и исполнения пользовательского кода расширений GPU рендер-системы, позволяющий модифицировать отдельные этапы процесса синтеза изображений с использованием кода расширений без необходимости внесения изменений в существующий программный код рендер-системы и его полной перекомпиляции. Разработанный метод позволил реализовать ряд процедурных текстур, а также расширить функциональность рендер-системы такими возможностями, как проективное текстурирование (рис. 2).
- Разработан алгоритм оценки уровня детализации для предрассчитанных процедурных текстур. Алгоритм позволяет до начала рендеринга вычислить разрешение процедурных текстур и снизить затраты памяти без потерь качества изображения. Алгоритм применен для синтеза процедурных текстур средствами клиентских приложений (таких как 3D-редакторы). Тем самым была обеспечена прямая поддержка инструментов клиентских приложений. Разработанный алгоритм позволяет определять разрешение текстур-изображений, что позволяет достичь экономии памяти в 1.6-4 раза без видимых потерь качества.

Теоретическая значимость

Разработанная программная архитектура обеспечивает решение проблем интеграции, инфраструктуры, распределенного рендеринга и отладки, и применима при разработке GPU рендер-систем фотореалистичного рендеринга. Разработанный алгоритм оценки разрешения процедурных текстур позволяет обеспечить экономию памяти GPU и интегрировать существующие сторонние инструменты создания процедурных текстур, а также применим для обычных текстур-изображений. Предложенный метод разработки и исполнения пользовательских расширений может быть использован для поддержки новых математических моделей в рендер-системах на GPU, в том числе использующих возможности аппаратного ускорения трассировки лучей.

Практическая значимость

- Разработанная программная архитектура и алгоритмы внедрены в открытую GPU систему фотореалистичного рендеринга Hydra Render, разработанную в Институте прикладной математики имени М. В. Келдыша РАН.
- Возможности предложенной архитектуры дополнительного программного слоя позволили обеспечить интеграцию рендер-системы с рядом программных продуктов:
    - пакет 3D моделирования Autodesk 3ds max (рис. 1), рендер-система используется конечными пользователями (3D художниками) через разработанный дополнительный программный слой;
    - программным продуктом для светодизайна LightCAD в качестве фотореалистичной рендер-системы по умолчанию (рис. 4);
    - программный продукт 3d моделирования с открытым исходным кодом Blender.
- Разработанный метод исполнения пользовательских расширений апробирован для создания процедурных текстур в составе проекта для синтеза обучающих данных для нейросетей (рис. 3). Интеграция компонентов программного комплекса была осуществлена с использованием предложенной архитектуры дополнительного программного слоя.
- Проведено сравнение разработанного подхода с конкурирующими решениями создания расширений для GPU рендер-систем, показавшее состоятельность предложенных подходов.

Методы исследования

При создании программной архитектуры рендер-системы использовались методы и алгоритмы реализации систем управления версиями и объектно-ориентированных баз данных. Для оценки разрешения предрассчитанных процедурных текстур использовались методы математического анализа. При создании метода разработки и исполнения пользовательских расширений использовалась теория синтаксического анализа и компиляции.

Рис. 2 Пример работы разработанного механизма создания пользовательских процедурных текстур. Слева – реализация проективного текстурирования (проекция изображений полос и надписи на модель), справа – процедурная текстура имитирующая ржавчину.

Рис. 3 Слева – фотографии с наложенными изображениями автомобилей, справа – с наложенными изображениями дорожных знаков. Наложенные изображения синтезированы помощью разработанных в работе средств. Применение – повышение качества распознавания объектов, которые редко встречаются в реальных условиях, а также имеющих различного рода изменения внешнего вида, например, загрязнения.

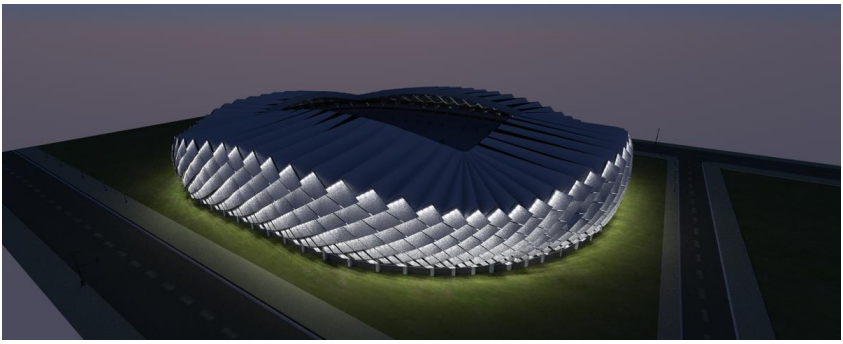

Рис. 4 Проект освещения нового стадиона в г. Батуми, выполненный компанией «Интилед» с использованием разработанной системы.

Достоверность и обоснованность результатов обеспечивается с помощью проведенной экспериментальной оценки результатов использования предложенных методов и алгоритмов при их интеграции в GPU рендер-систему с открытым исходным кодом, в которой был осуществлен рендеринг набора тестовых 3D сцен. Для предложенного алгоритма оценки разрешения предрассчитанных процедурных текстур была доказана теорема, что вычисленное им разрешение позволяет достичь соотношения минимум 1 тексель текстуры на 1 пиксель синтезированного рендер-системой изображения. Для предложенной программной архитектуры и метода разработки пользовательских расширений также было проведено сравнение результатов с аналогами.

Основные положения выносимые на защиту:

1. Программная архитектура рендер-системы с использованием дополнительного программного слоя, решающего задачи организации интеграции и инфраструктуры.
2. Алгоритм оценки уровня детализации для предрассчитанных процедурных текстур, основанный на выполнении специализированного предварительного рендеринга сцены. Алгоритм интегрирован в предложенную программную архитектуру.
3. Метод разработки и исполнения пользовательских расширений для GPU рендер-системы. Метод интегрирован в предложенную программную архитектуру.

Апробация работы

Основные положения работы были доложены на:

1. Международная конференция по компьютерной графике, обработке изображений и машинному зрению, системам визуализации и виртуального окружения Графикон, Томск, Россия, 24-27 сентября, 2018.
2. Международная конференция по компьютерной графике и машинному зрению Графикон, Брянск, Россия, 23-26 сентября, 2019

3. 14th International Conference on Computer Graphics, Visualization, Computer Vision and Image Processing (CGVCVIP), Zagreb, Croatia, 23-25 July, 2020
4. Научный семинар им. М.Р. Шура-Бура в ИПМ им. М.В. Келдыша РАН, 10 декабря 2020.

Публикации

По теме диссертации имеется 8 публикаций [1-8] в журналах из перечня ВАК.

Личный вклад автора

Содержание диссертации и основные положения, выносимые на защиту, отражают персональный вклад автора в опубликованных работах.

Диссертационная работа соответствует паспорту специальности (ПС) 2.3.5 - математическое и программное обеспечение вычислительных систем, комплексов и компьютерных сетей. Работа посвящена разработке программной архитектуры GPU рендер-системы решающей задачи организации интеграции и инфраструктуры (направления 3 и 7 ПС). Был предложен и реализован метод разработки и исполнения кода пользовательских расширений в GPU рендер-системе (направления 1 и 2 ПС). Предложен и реализован алгоритм оценки разрешения предрассчитанных процедурных текстур (направление 7 ПС).

Структура и объем диссертации

Диссертация состоит из введения, обзора литературы, пяти глав, заключения, библиографии. Общий объем диссертации 156 страниц, из них 156 страниц основного текста, включая 39 рисунков и 5 таблиц. Библиография включает 167 наименований.

Содержание работы

Во введении приводится обоснование актуальности диссертационной работы, приведена постановка целей и задач исследования, показана научная новизна и практическая значимость работы, сформулированы положения, выносимые на защиту.

В первой главе проводится обзор предметной области, рассматривается схема работы конечного пользователя (3D художника), сформулированы и описаны требования к приложениям создания 3D сцен и рендер-системам, включающие:

1. Интерактивность процесса работы, позволяющая пользователю видеть результаты своих действий с минимальной задержкой.
2. Отсутствие ограничений на память – 3D сцены могут обладать высокой сложностью и большим объемом.
3. Расширяемость, обеспечивающая возможность добавлять новую функциональность в систему, в том числе и конечным пользователям.
4. Организация импорта и экспорта данных между специализированными приложениями, выполняющими разные задачи по построению финальной 3D сцены.

5. Наличие средств отладки и тестирования, позволяющих выявлять и исправлять ранее неизвестные ошибки, возникающие при определенной последовательности действий пользователя в 3D редакторе или рендер-системе.
6. Возможность использования распределенного рендеринга в связи с большими временами фотореалистичного синтеза изображений в индустриальных приложениях.
7. Скорость расчета.

Далее описана задача фотореалистичного синтеза изображения, решаемая рендер-системой, задаваемая в виде т.н. уравнения рендеринга:

$$L_o(x,\omega_0) = L_e(x,\omega_0) + \int_{\Omega} L_i(x,\omega_i) f(x,\omega_0,\omega_i)(\omega_i \cdot n) d\omega_i \qquad (1)$$

где x – точка в сцене, n – нормаль в этой точке, ω_0 – единичное направление исходящего света, ω_i – единичное направление приходящего света, $(\omega_i \cdot n)$ – величина скалярного произведения между направлением приходящего света и нормалью, $\boldsymbol{\Omega}$ - область определения всех возможных направлений приходящего света, $L_e(x,\omega_0)$ – свет излучаемый в точке x в направлении ω_0, $L_i(x,\omega_i)$ – свет, приходящий в точку x вдоль направления ω_i, $f(x,\omega_0,\omega_i)$ – двунаправленная функция отражательной способности (ДФОС, англ. bidirectional reflectance distribution function, BRDF), определяющая долю излучения, пришедшего в точку x по направлению ω_i, которая будет отражена по направлению ω_0. Более общий вариант ДФОС – двунаправленная функция рассеивания (ДФР), помимо отражения света также учитывает пропускание света.

Затем рассматриваются технологии программирования и подходы к реализации систем фотореалистичного рендеринга на GPU и технологии аппаратного ускорения трассировки лучей как фактор, влияющий на архитектуру рендер-систем. В работе [7] были опубликованы программные эксперименты с реализаций трассировки путей с использованием RTX и её сравнение с открытой реализацией трассировки путей на OpenCL, показавшие значительный прирост производительности (2-5 раз) в том числе на сложных сценах, что говорит об актуальности использования этой технологии в рендер-системах.

Во второй главе представлена архитектура рендер-системы, основанная на дополнительном программном слое между рендер-системой и клиентскими приложениями, решающим задачи их интеграции [2].

Сначала рассматриваются и анализируются на предмет соответствия требованиям (глава 1) существующие подходы к построению рендер-систем и их интеграции с клиентскими приложениями. Эти решения в той или иной мере решают проблему разного представления данных, которую в некотором смысле можно назвать фундаментальной, – большинство операций экспорта из редактора в рендер-систему долгие и ресурсоёмкие по определению, ускорить

или упростить их затруднительно. Например, загрузка большой текстуры с диска будет долгой практически при любых возможных оптимизациях. Решение заключается в том, чтобы не экспортировать повторно то, что уже было экспортировано хотя бы раз.

Предлагаемая архитектура дополнительного программного слоя (и его интерфейса программирования, API) построена исходя из требований, сформулированных в первой главе, и может быть в общем охарактеризована как объектная база данных с возможностями системы контроля версий и стратегией отсутствия перезаписи при внесении изменений.

Описание некоторого состояния сцены выглядит как один XML файл, описывающий параметры объектов, плюс набор бинарных или текстовых файлов в произвольном формате (рис. 5). Подобный подход позволяет обеспечить расширяемость, – добавление новых моделей или параметров в существующие модели требует лишь создания их XML-описания, которое будет обработано существующими механизмами API и будет автоматически доступно рендер-системе (которая должна поддержать эти изменения на своей стороне).

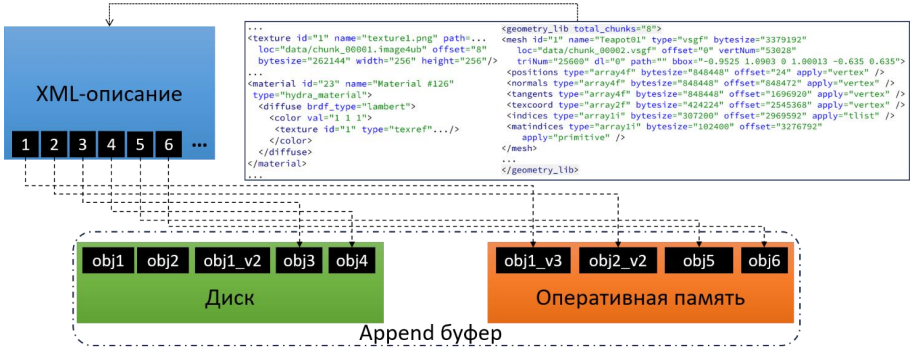

Рис. 5 Схема описания сцены и хранения объектов. Часть объектов хранится на диске, а другая – в оперативной памяти.

Для работы с геометрией и текстурами используется подход на основе т.н. append-буфера с бесконечным размером, реализующего операции добавления объекта в конец, поиска и копирования объектов. При этом осуществляется стратегия отсутствия перезаписи, – для измененных объектов создаются их копии и добавляются в конец буфера (рис. 5). Благодаря такой стратегии, объекты с которыми работает пользователь почти всегда находятся в оперативной памяти.

Стратегия отсутствия перезаписи также гарантирует идентичность состояний сцены на разных вычислительных узлах в сети, т.к. невозможно появление нескольких версий одного и того же файла. Эффективность реализации сетевого фотореалистичного рендеринга была подтверждена

моделированием [1]. Одним из факторов здесь является объем данных 3D сцены, передаваемых по сети, снижаемый в предлагаемом решении за счет механизма отслеживания изменений.

Интеграция рендер-системы с клиентскими приложениями и работа с 3D сценой в предлагаемом решении реализована через интерфейс из 3-х основных функций для каждого типа объектов сцены – *Create, Open* и *Close*. Вызов *Create* создаёт новый пустой объект, вызов *Open* открывает объект для редактирования, а вызов *Close* сохраняет внесенные изменения.

Работа с 3D сценой с точки зрения разработанного API выглядит подобно системе управления версиями git (рис. 6).

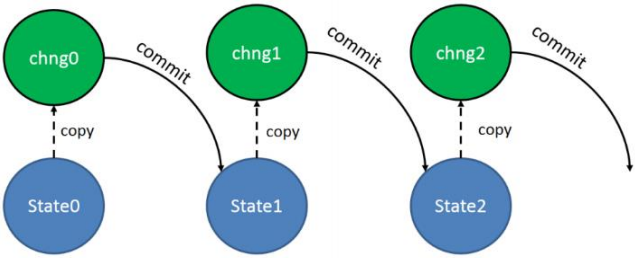

Рис. 6 Процесс работы со сценой со стороны промежуточного слоя.

Для внесения изменений в описание объекта пользователь открывает его на редактирование. После завершения работы с объектом, его измененное состояние хранится как отдельная копия в файле изменений (верхняя строчка на рис. 6), при этом файлы не должны быть обязательно сохранены на диск. Подобными действиями пользователь может изменить любое количество любых объектов, все изменения накапливаются в одном файле изменений.

Отличительной и важной чертой предлагаемого подхода является явное отслеживание и явная запись изменений. Если текущее состояние объекта хранится в файле описания сцены «state1.xml», то новое состояние объекта будет храниться в отдельном файле изменений «chng1.xml» (рис. 6). Операция *Commit* формирует новое состояние в «state2.xml» по алгоритму 1, содержащее:

1. объекты из предыдущего состояния «state1.xml», которые не подверглись изменениям,
2. новые объекты из «chng1.xml»,
3. изменённые объекты из «chng1.xml», которые заместят свои оригиналы из «state1.xml».

Ещё раз отметим, что рендер-системе будут переданы только новые и измененные объекты, которым соответствуют файл «chng1.xml» на рис. 6 и множества $M_{new}, B_{new}, L_{new}, T_{new}$ в алг. 1.

Алгоритм 1. Формирование нового состояния сцены

Входные данные:

$\tilde{M}, \tilde{B}, \tilde{L}, \tilde{T}$ – множества идентификаторов объектов сцены (мешей, материалов, источников света, текстур соответственно), переданных рендер-системе ранее, если это первое состояние, то $\tilde{M} = \emptyset, \tilde{B} = \emptyset, \tilde{L} = \emptyset, \tilde{T} = \emptyset$;
$\bar{M}, \bar{B}, \bar{L}, \bar{T}$ – множества идентификаторов измененных объектов сцены, полученные в результате вызовов *Open*/*Close*;
I_M – множество всех экземпляров мешей в 3D сцене, каждый элемент – четверка $\{id,\ \mathcal{M}, mat\_ids\}$, где id – уникальный идентификатор, $\mathcal{M}$ – матрица трансформации, mat_ids – массив идентификаторов материалов для экземпляра;
I_L – множество всех экземпляров источников света в 3D сцене, каждый элемент – пара $\{id,\ \mathcal{M}\}$, где id – уникальный идентификатор, $\mathcal{M}$ – матрица трансформации;

Выходные данные:

M, B, L, T – множества идентификаторов объектов сцены в новом состоянии;
$drawSeq$ – отображение идентификатора меша на список его экземпляров в сформированном состоянии.

$M_{new} = \bar{M}, B_{new} = \bar{B}, L_{new} = \bar{L}, T_{new} = \bar{T}$
$drawSeq = \emptyset$
foreach $i \in I_M$ **do**:
    if $i.id \notin \tilde{M}$ **then**:
        $M_{new} = M_{new} \cup \{i.id\}$
    end if
    $drawSeq = drawSeq \cup \{i.id \to i\}$
    $B_{new} = B_{new} \cup \{j \mid j \in i.mat\_ids\}$
end for
foreach $l \in I_L$ **do**:
    if $l.id \notin \tilde{L}$ **then**:
        $L_{new} = L_{new} \cup \{l.id\}$
    end if
end for
foreach $m \in M_{new}$ **do**:
    $Q = \text{GetMaterialsUsedByMesh}(m)$
    $B_{new} = B_{new} \cup \{j \mid j \in Q\}$
    foreach $j \in Q$ **do**:

$B_{new} = B_{new} \cup \{k \mid k \in \text{GetAllChildMaterialsRecursive}(j)\}$
end for
end for
foreach $b \in B_{new}$ **do**:
$T_{new} = T_{new} \cup \{j \mid j \in \text{GetTexturesUsedByMaterial}(b)\}$
end for
foreach $l \in L_{new}$ **do**:
$T_{new} = T_{new} \cup \{j \mid j \in \text{GetTexturesUsedByLight}(l)\}$
end for
$M = \tilde{M} \cup M_{new}, B = \tilde{B} \cup B_{new}, L = \tilde{L} \cup L_{new}, T = \tilde{T} \cup T_{new}$
return $M, B, L, T, drawSeq$

Для интеграции рендер-системы с предлагаемой архитектурой реализован специальный интерфейс, через который все изменения передаются в рендер всегда в одном и том же строго определенном порядке, независимо от того, в каком порядке они были сделаны пользователем до операции *Commit*. Это позволяет в значительной мере упростить интеграцию рендер-системы с предлагаемым решением, т.к. она может полагаться на заранее известную и предопределенную последовательность вызовов независимо от того, в каком порядке производятся вызовы API в прикладном приложении.

Благодаря упорядочиванию вызовов появляются возможности для отладки и тестирования, в частности, разработчики могут создавать раздельные тесты для интеграционного слоя и рендер-системы. Например, для тестирования интеграционного слоя возможно проверять корректность полученных XML описаний состояний и списков изменений для 3D сцены, а для тестирования рендер-системы – синтезированные изображения. Упрощенная интеграция с промежуточным слоем позволяет использовать отладочные рендер-системы, например, для визуализации отдельных компонентов геометрических моделей.

В конце главы 2 рассмотрены механизмы интеграции в распространенных рендер-системах с открытым исходным кодом и приведены результаты сравнения предлагаемого решения с существующими (табл. 1) по требованиям, сформулированным в первой главе. Предлагаемый подход в полной мере обеспечивает выполнение всех требований. При этом лучшие из существующих подходов (Pixar USD и Multiverse) являются специализированными решениями, ориентированными на задачи создания кино- и анимационных фильмов и потому их интеграция с решениями для других задач является трудоемкой.

Требование	Обмен файлами	Динамические плагины	Pixar USD/ Multiverse	Предлагаемое решение
Скорость / GPU	+	-	+	+
Гибкость/расширяемость	+	+	+	+
Интерактивность	-	+	+	+
Отсутствие ограничений на память	-	-	+	+
Сериализация, импорт/экспорт	+	-	+	+
Механизмы отладки	+/-	-	+	+
Распределенный рендеринг	-	-	+/-	+
Простота интеграции	+	-	-	+

Таблица 1. Сравнение с существующими решениями.
Обозначения: + соответствие требованию, +/- частичное соответствие требованию, - несоответствие требованию.

В третьей главе представлен алгоритм оценки разрешения предрассчитанных процедурных текстур [3]. При интеграции рендер-системы в клиентские приложения возникает задача поддержки средств клиентского приложения для создания компонентов 3D сцены, включая процедурные текстуры. Для CPU-рендера это обычно требует лишь вызова виртуальной функции клиентского приложения по необходимости в процессе вычислений, что невозможно для GPU-рендера. Возможное решение – синтезировать все процедурные текстуры заранее, т.е. до начала процесса рендеринга. Такой подход влечет за собой необходимость определить такое разрешение текстур, которое позволит получить максимальное визуальное качество при минимальном объеме вычислений на синтез текстуры и затрат памяти на её хранение.

В данной работе предлагается решение, основанное на введении предварительного этапа быстрого рендеринга сцены, который может быть выполнен как с помощью растеризации, так и трассировки лучей, после которого следует синтез текстур. Цель предварительного этапа – определить разрешение текстуры в сцене с помощью предложенного в данной работе алгоритма (алгоритм 2), основанного на вычислении частных производных текстурных координат и являющегося модификацией подхода, реализуемого графическими API (OpenGL, Vulkan и др.). Модификации связаны с тем, что разрешение рендеринга для предварительного этапа и для финального изображения могут не

совпадать, и с тем, что разрешение текстуры является искомым, а в базовом методе выбирается mip-уровень для заранее известного разрешения.

Алгоритм 2. Определение разрешения текстур

Входные данные:
I– изображение полученное на этапе предварительной отрисовки с разрешением R_{px}, R_{py}, каждый пиксель изображения содержит кортеж (x, y, uv, mat_id), где x и y – координаты пикселя в изображении I, uv - текстурные координаты объекта, видимого из виртуальной камеры, и mat_id - идентификатор его материала;
$materials$ – массив материалов, каждый из которых содержит массив пар (tex_id, M), где tex_id - идентификатор текстуры и M - матрица трансформации текстурных координат; R_{rx}, R_{ry}– разрешение финального (фотореалистичного) рендеринга;
w_{max}, h_{max} – максимальное разрешение текстуры; mip_{max} – максимальный mip-уровень.
Выходные данные:
$out\_resolution$ – массив рассчитанных разрешений текстур.

$mipbuf[\dots] \leftarrow 0$
foreach $pix \in I$ ***do***:
    $uv \leftarrow pix.uv$
    foreach $(tex\_id, M) \in materials[pix.mat\_id]$ ***do***:
        $\widehat{uv} \leftarrow M * uv$
        $dx \leftarrow \frac{R_{px}}{R_{rx}} * w_{max} * \frac{\partial \widehat{uv}}{\partial pix.x}$
        $dy \leftarrow \frac{R_{py}}{R_{ry}} * h_{max} * \frac{\partial \widehat{uv}}{\partial pix.y}$
        $\Delta \leftarrow \max(dx \cdot dx, dy \cdot dy)$
        $\Delta \leftarrow \min\left(\max(\Delta, 1), 2^{mip_{max}*2}\right)$
        $mip \leftarrow floor(\log_2 \sqrt{\Delta})$
        if $mip < mipbuf[tex_{id}]$ ***then***:
            $mipbuf[tex\_id] \leftarrow mip$
        end if
    end for
end for
foreach $mip_i \in mipbuf$ ***do***:
    $width \leftarrow \frac{w_{max}}{2^{mip_i}}$

$height \leftarrow \frac{h_{max}}{2^{mip_i}}$

$out\_resolution[i] \leftarrow (width, height)$

end for

return *out_resolution*

Теорема. *Алгоритм 2 вычисляет разрешение текстуры* (w_{tex}, h_{tex}) *такое, что хотя бы для одной координаты достигается соотношение 1 тексель текстуры на 1 пиксель синтезированного рендер-системой изображения. Т.е. для любой пары соседних пикселей синтезированного изображения будет выполнена выборка из пары соседних текселей текстуры, если в этих пикселях виден один и тот же объект 3D сцены.*

Результаты экспериментальной оценки предложенного алгоритма, показывают экономию памяти от 1.6 до 4 раз по сравнению с наивным подходом, при котором процедурные текстуры синтезируются в некотором фиксированном разрешении (например, в разрешении рендеринга), а текстуры-изображения используются в своем исходном разрешении (рис. 7, 8). Численная оценка с помощью среднеквадратической ошибки между изображениями, полученными базовым методом, и изображениями, полученными с использованием предложенного подхода, для широкого динамического диапазона (HDR) показывает малые значения отличий (табл. 2). Значения пикового отношения сигнала к шуму (PSNR) в узком динамическом диапазоне (LDR) превышают величину в 30 дБ, которая обычно принимается пороговой для хорошего качества восстановления изображений.

Предложенный алгоритм интегрирован с архитектурой промежуточного программного слоя, описанной в главе 2.

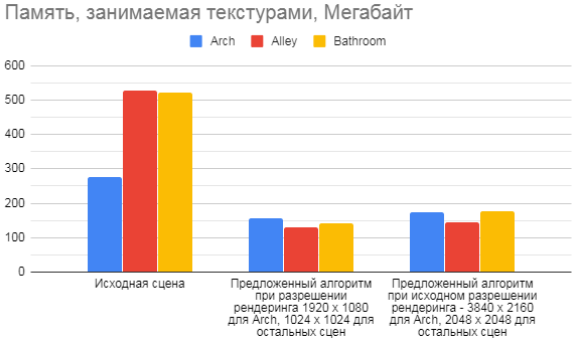

Рис. 7 Память, занимаемая текстурами, для тестовых сцен при наивном подходе («Исходная сцена») и с использованием разработанного алгоритма. Измерения на Nvidia GTX 1070.

Сцена	MSE (HDR)	MSE (LDR)	PSNR (LDR), дБ
Alley	1.67	5.929	40.401
Bathroom	3.59	21.563	34.794
Arch	3.02	20.626	34.987

Таблица 2. Среднеквадратические ошибки (MSE) и пиковое отношение сигнала к шуму (PSNR) между изображением-эталоном и изображением, для которого размер текстур был изменен по предложенному алгоритму, для изображений с узким (LDR) и широким (HDR) динамическим диапазоном. Рендеринг на Nvidia GTX 1070, 2048 выборок на пиксель.

Рис. 8 Тестовые сцены. Верхняя (Arch) и нижняя правая сцены (Alley) – преимущественно процедурные текстуры, нижняя левая (Bathroom) – текстуры-изображения большого разрешения.

В четвертой главе предложен метод разработки пользовательских расширений GPU рендер-системы [4]. Обзор общих подходов к построению расширяемого программного обеспечения показал сложность их применения для разработки на GPU. Среди существующих решений для обеспечения расширяемости рендер-систем в качестве базового был выбран подход

предметно-ориентированных («шейдерных») языков. Основными отличиями предлагаемого решения являются:

- использование подмножества языка C99 в качестве языка создания расширений (а не некоторого нового языка);
- ввод ограничений на разрабатываемые расширения.

Выбор C99 в качестве языка расширений позволяет обеспечить кроссплатформенность и избежать необходимости реализовывать и поддерживать новые инструменты программирования (англ. «toolchain»). Вводимые ограничения включают отсутствие возможности прямого программирования ДФР и моделей источников света, т.к. эти задачи сложны для рядового пользователя, а неправильное их решение приведет к некорректности работы всей рендер-системы.

Допускается программирование входных параметров существующих моделей материалов, а также реализация теста пересечения для поддержки новых геометрических примитивов, что покрывает широкий круг задач пользователей.

Математически, возможность создания процедурных текстур означает, что пользователь может параметризовать вычисление ДФР в интеграле освещенности (1). Для процедурных текстур в качестве аргументов может выступать произвольный набор параметров.

Уравнение рендеринга (1) для ДФР, параметризованной k процедурными текстурами может быть записано следующим образом:

$$L_o(x,\omega_0) = L_e(x,\omega_0) + \int_\Omega L_i(x,\omega_i) f(x,\omega_0,\omega_i,T_1,\dots,T_k)(\omega_i \cdot n)d\omega_i$$
$$T_i = \int \hat{f}_i(U_i,x_1,\dots x_n)t_i(U_i,x_1,\dots x_n,a_1,\dots,a_m)dudvdw\,, 1 \le i \le k \quad (2)$$

где T_i – значение, получаемое при вычислении и фильтрации процедурной текстуры, $\hat{f}_i$ – функция фильтрации процедурной текстуры, t_i – процедурная текстура, U_i – отображение из параметров $x_1,\dots x_n$ в текстурные координаты u,v,w и $a_1,\dots a_m$ – параметры процедурной текстуры.

Таким образом, пользователь при создании расширения определяет функцию t_i, которая реализует внутри себя некоторый алгоритм или математическую модель, вычисляющую соответствующий параметр ДФР в зависимости от параметров $a_1,\dots a_m$ и отображение U_i для получения текстурных координат.

Предлагаемое решение в составе рендер-системы реализуется через подход нескольких вычислительных «убер-ядер», – все пользовательские расширения для геометрии в данной сцене собираются в одно вычислительное ядро, а для процедурных текстур – в другое. В процессе работы рендер-системы код расширений при необходимости исполняется в нужном месте алгоритма синтеза изображений (рис. 9). Так код расширений остается обособленным и не

перемешивается с кодом самой рендер-системы, чья основная функциональность остается неизменной. Это позволяет сделать архитектуру рендер-системы модульной, допускающей замену части алгоритма трассировки лучей без влияния на другие части.

Для исполнения в составе рендер-системы код пользовательских расширений подвергается простой обработке (обработка имен функций, замена встроенных вызовов и типов), выдающей на выходе код, который допускает редактирование и отладку как обычная OpenCL или C программа.

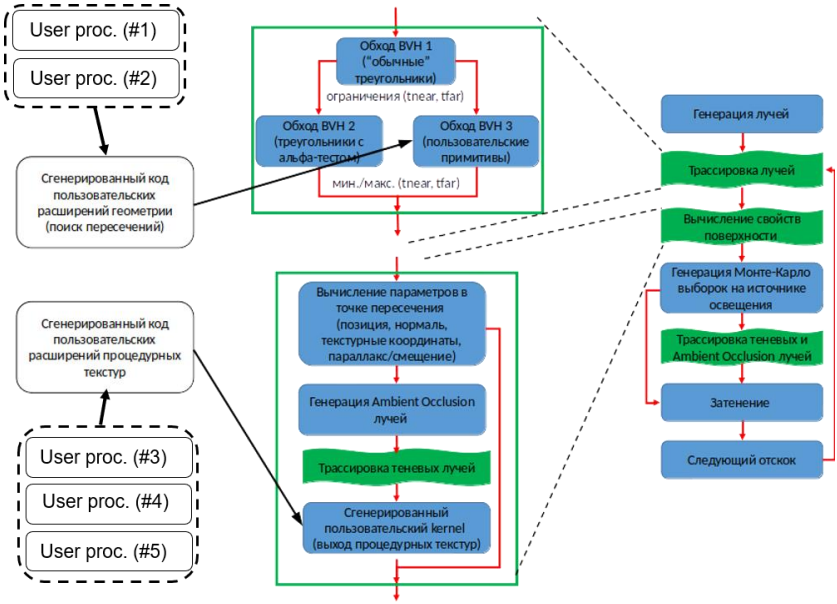

Рис. 9 Схема подключения пользовательских расширений в конвейер синтеза изображений в рендер-системе.

Предложенный подход интегрирован с архитектурой промежуточного программного слоя (глава 2). Объекты процедурных текстур задаются в XML-описании сцены с помощью его API. Для хранения параметров $a_1, \dots a_m$ выделяется отдельная область памяти, «псевдостек». Поскольку на момент генерации кода «убер-ядра» их значения являются константными, «псевдостек» является просто областью глобальной памяти GPU, допускающей только чтение. Это позволяет обновлять её со стороны CPU без необходимости перекомпиляции вычислительных ядер.

В сравнении с подходом на основе технологии RTX и вставкой кода процедурной текстуры в место её использования (табл. 3), разработанный метод демонстрирует наименьшее увеличение времени рендеринга для процедурной

текстуры, реализующей вычислительно затратный алгоритм, моделирующий внешний вид ржавчины (рис. 2, справа). Для простой процедурной текстуры предлагаемое решение теряет в скорости больше, т.к. в этом случае вставка кода оказывается достаточно эффективной и накладные затраты на запуск отдельного вычислительного ядра процедурных текстур оказываются более значительными, чем его выполнение. В обоих случаях предлагаемое решение демонстрирует меньший размер стекового кадра, что является значимым, т.к. регистровое давление – это один из факторов при выборе подхода к реализации рендер-системы на GPU.

Процедурная текстура	Наивная вставка	RTX	Предлагаемое решение
Ржавчина, сложная (прирост времени)	+46%	+33%	+21%
Царапины, простая (прирост времени)	+9%	+4%	+15%
Ржавчина, сложная (размер стекового кадра)	604 байт	-	464 байт
Царапины, простая (размер стекового кадра)	412 байт	-	336 байт

Таблица 3 Результаты сравнения. Прирост времени – изменение времени рендеринга в процентах при добавлении на объект материала с процедурным текстурами по сравнению с серым диффузным материалом. Размер стекового кадра был получен с помощью флага «-cl-nv-verbose» компилятора OpenCL. Рендеринг и измерения проводились на Nvidia RTX 2070.

В пятой главе приводятся примеры практического использования разработанных в работе решений. Все разработки были внедрены в GPU рендер-систему Hydra и позволили выполнить её интеграцию с программными продуктами создания 3D сцен 3ds Max (рис. 1) и Blender и с приложением для светодизайна LightCAD (рис. 4). Предложенные решения использовались для разработки программного комплекса генерации обучающих данных для задач компьютерного зрения [5, 6, 8], – распознавания автомобилей и дорожных знаков (рис. 3). В задаче распознавания дорожных знаков [5] использование генерации синтетических данных с помощью предлагаемых решений позволило поднять точность распознавания редких дорожных знаков от 0% в базовом наборе данных до 71%.

В заключении сформулированы основные результаты работы, состоящие в следующем:

1. Разработана новая программная архитектура расширяемой GPU рендер-системы, позволяющая существенно снизить трудоёмкость интеграции рендер-системы в клиентские приложения и отладки проблем, возникающих в процессе эксплуатации. Разработанная архитектура

обеспечивает расширяемость рендер-системы, позволяя добавлять новые модели и параметры моделей без внесения изменений в архитектуру.

2. Разработан подход создания расширений, позволяющий конечным пользователям реализовывать дополнительную функциональность для GPU рендер-систем с целью их адаптации к решению новых прикладных задач. Разработанный подход позволяет пользователям создавать новые геометрические примитивы, процедурные текстуры и проективное текстурирование без необходимости внесения изменений непосредственно в рендер-систему и её пересборки с сохранением возможности отладки созданных расширений.
3. Разработан алгоритм оценки разрешения для процедурных текстур, также применимый и для текстур-изображений, позволяющий снизить потребление памяти в 1.6-4 раза без видимых потерь качества, а также обеспечить поддержку рендер-системой механизмов генерации процедурных текстур, имеющихся в клиентских приложениях.
4. Разработанные решения внедрены в систему фотореалистичного синтеза изображений на GPU и позволили осуществить её интеграцию и использование в широком круге задач – архитектурной визуализации, светодизайна, генерации синтетических данных для обучения алгоритмов искусственного интеллекта.

Список публикаций

1. *Санжаров В.В., Фролов В.А.* Исследование масштабируемости распределённых рендер-систем на основе алгоритмов адаптивной трассировки путей и Metropolis Light Transport в гетерогенных сетях. // *Препринты ИПМ им. М.В.Келдыша*. — 2016. — № 114. — С. 1–22. — DOI: 10.20948/prepr-2016-114
2. *Санжаров В.В., Фролов В.А.* Современные проблемы интеграции в приложениях компьютерной графики и пути их решения. // *Программирование*. — 2018. — № 4. — С. 36-45. — DOI: 10.31857/S013234740000699-3
   English translation: *Frolov V.A., Sanzharov V.V.* Modern problems of software integration in computer graphics applications and ways to solve them // *Programming and Computer Software*. — 2018. — Vol. 44, no. 4. — P. 233–239. — DOI: 10.1134/S0361768818040060
3. *Санжаров В.В., Фролов В.А.* Уровень детализации для предрасчитанных процедурных текстур // *Программирование*. — 2019. — № 4. — С. 54–63. — DOI: 10.1134/S0132347419040071
   English translation: *Sanzharov V.V., Frolov V.A.* Level of detail for precomputed procedural textures // *Programming and Computer Software*. — 2019. — Vol. 45, no. 4. — P. 187–195. — DOI: 10.1134/s0361768819040078

4. *V. Sanzharov, V. Frolov, I. Pavlov* Restricted Extensions for GPU Photo-realistic Renderer // *CEUR Workshop Proceedings*. — 2019. — Vol. 2485. — P. 37–42. — DOI: 10.30987/graphicon-2019-2-37-42
5. *Фаизов Б.В., Шахуро В.И., Санжаров В.В., Конушин А.С.* Классификация редких дорожных знаков // *Компьютерная оптика*. — 2020. — Т. 44, № 2. — С. 236–243. — DOI: 10.18287/2412-6179-co-601
6. *Санжаров В.В., Фролов В.А., Волобой А.Г., Галактионов В.А., Павлов Д.С.* Система генерации наборов изображений для задач компьютерного зрения на основе фотореалистичного рендеринга // *Препринты ИПМ им. М.В. Келдыша*. — 2020. — № 80. — С. 1–29. — DOI: 10.20948/prepr-2020-80
7. *Санжаров В.В., Фролов В.А., Галактионов В.А*. Исследование технологии RTX // *Программирование*. — 2020. — № 4. — С. 65–72. doi: 10.31857/S0132347420030061
   English translation: Sanzharov V. V., Frolov V. A., Galaktionov V. A. Survey of nvidia RTX technology// *Programming and Computer Software*. — 2020. — Vol. 46, no. 4. — P. 297–304. — DOI: 10.1134/s0361768820030068
8. *Frolov V., Faizov B., Shakhuro V., Sanzharov V., Konushin A., Galaktionov V., Voloboy A.* Image Synthesis Pipeline for CNN-Based Sensing Systems // *Sensors*. — 2022. – Vol. 22, №. 6. – p. 2080. — DOI: 10.3390/s22062080

Подписано в печать 11.08.2023
Формат А5
Бумага офсетная. Печать цифровая.
Тираж 70 Экз. Заказ №1229225-8-23
Типография ООО"МДМпринт"
(Печатный салон МДМ)
119602 , г. Москва, ул. Покрышкина, 4
Тел. 8-495-256-10-00