# Обзор существующих методов оптимизации технологии трассировки лучей

И.С. Черемисенов
cheremisenov96@gmail.com
Государственное бюджетное общеобразовательное учреждение города Москвы «Лицей «Вторая школа»

*Проблема вычислений затратных по производительности алгоритмов в режиме реального времени вынуждает разработчиков программного обеспечения задумываться о поисках механизмов по оптимизации данных вычислений. В данной статье проведен обзор некоторых существующих методов оптимизации технологии трассировки лучей.*

***Ключевые слова:*** *алгоритм, модель, оптимизация, трассировка лучей.*

# Review of existing methods of optimization of ray tracing technology

I.S. Cheremisenov
State budgetary educational institution of the city of Moscow «Lyceum «School No. 2»

*The problem of computing performance-intensive algorithms in real time forces software developers to think about finding mechanisms to optimize these calculations. This article reviews some of the existing techniques for optimizing ray tracing technology.*

***Keywords:*** *algorithm, model, optimization, ray tracing.*

## 1. Введение

С момента появления алгоритмов расчета отражения и преломления света (ray tracing, трассировка лучей) их вычисление обычно рассматривалось как статический процесс и представляло собой весьма трудоемкий способ получения красивых изображений. В последнее время появились возможности расчета лучей в интерактивном режиме, даже при использовании довольно доступного аппаратного обеспечения. С учетом повышения эффективности таких расчетов и благодаря преимуществам современных разработок в области микропроцессоров, в частности, таких как технология Hyper-Threading и многоядерных процессоров, можно ожидать, что в ближайшем будущем эти технологии будут применяться как в интерактивных приложениях, так и в приложениях реального времени.

## 2. Метод «Трассировки лучей»

Одним из методов визуализации (рендеринга) в компьютерной графике является метод «бросания лучей» (ray casting), при котором растровое изображение строится на основе замеров пересечения лучей с визуализируемой поверхностью в пространстве. Этот термин впервые использовался в компьютерной графике в 1982 году в работе Скотта Рота [1], который применил его для описания метода рендеринга CSG-моделей.

Первый алгоритм рейкастинга, используемый для рендеринга, был представлен Артуром Аппелем в 1968 году [1]. В основе рейкастинга лежит идея генерировать лучи из точки наблюдения сцены, один луч на пиксель, и находить самый близкий объект, который блокирует путь распространения этого луча. Используя свойства материала и эффект света в сцене, алгоритм рейкастинга может определить затенение данного объекта [1].

В реальной природе источник света испускает луч света, который, проходя через пространство, пересекает какую-либо преграду, которая прерывает распространение этого светового луча. В любой точке пути с лучом света может случиться любая комбинация трех процессов: поглощение, отражение и преломление. Некоторые из лучей, сгенерированных источником света, распространяются по пространству и в конечном счете попадают на область просмотра [1].

Алгоритм трассировки лучей (ray tracing) применяет рейкастинг для расчета первичных пересечений луча с объектами сцены и дополняет его генерацией дополнительных лучей для формирования световых бликов, теней, отражений, тем самым повышая уровень фотореалистичности изображения [1].

Для построения первичного луча и определения первых пересечений с объектами сцены вводятся понятия источника лучей и плоской области обзора. В основе модели лучевого эмиттера лежит механизм упрощенной камеры-обскуры (рис. 1) с бесконечно малым отверстием, через которое свет проникает на область обзора [1].

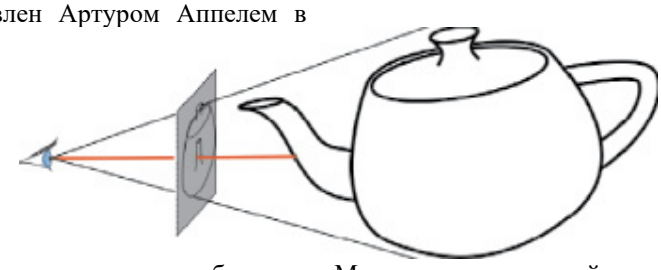

**Рис. 1.** Принцип построения изображения. Механизм упрощенной камеры-обскуры [1]

Достоинством данного алгоритма является слабая зависимость вычислительной сложности метода от

сложности сцены, а также есть возможность распараллеливания вычисления. Например, можно параллельно вычислять несколько лучей или разделять экран на части [2].

Главным недостатком метода является требование к высокой производительности системы. Для рендеринга нового изображения требуется выполнить вычисления для каждого луча заново [2].

## 3. Технология CUDA

Технология CUDA появилась в 2006 году и представляет из себя программно-аппаратный комплекс производства компании Nvidia, позволяющий эффективно писать программы под графические адаптеры [3]. С 2006 года компания Nvidia обещает, что все графические адаптеры их производства независимо от серии будут иметь сходную архитектуру, которая полностью поддерживает программную часть технологии CUDA. Программная часть, в свою очередь, содержит в себе всё необходимое для разработки программы: расширения языка C, компилятор, API для работы с графическими адаптерами и набор библиотек [3].

Общее устройство графического адаптера схематически представлено на рис. 2 [3].

### 3.1. Графический адаптер на «аппаратном» уровне

Рассмотрим устройство графических адаптеров, основанных на различных микросхемах компании NVIDIA, более подробно и сравним, чем они отличаются, а в чем они схожи. Общее устройство графического адаптера схематически представлено на рис. 2 [3].

Как показано на рисунках 2-4, произошли как количественные изменения (увеличение количества потоковых мультипроцессоров SM в блоке выборки текстур TPC), так и качественные (в семействе G200 в потоковом мультипроцессоре SM появляется блок для расчета с двойной точностью). Одинаковыми же остаются [3]:

1. 8 потоковых процессоров SP (Streaming Processor) в мультипроцессоре SM (Streaming Multiprocessor).
2. блок SFU (Super Function Unit) для расчета сложных математических функций (экспонента, корень и т. д.).
3. 32 КБ регистрового файла на SM.
4. 16 КБ разделяемой (shared) памяти на SM.
5. Кэш инструкций.
6. Кэш констант.
7. Блок выборки инструкций.

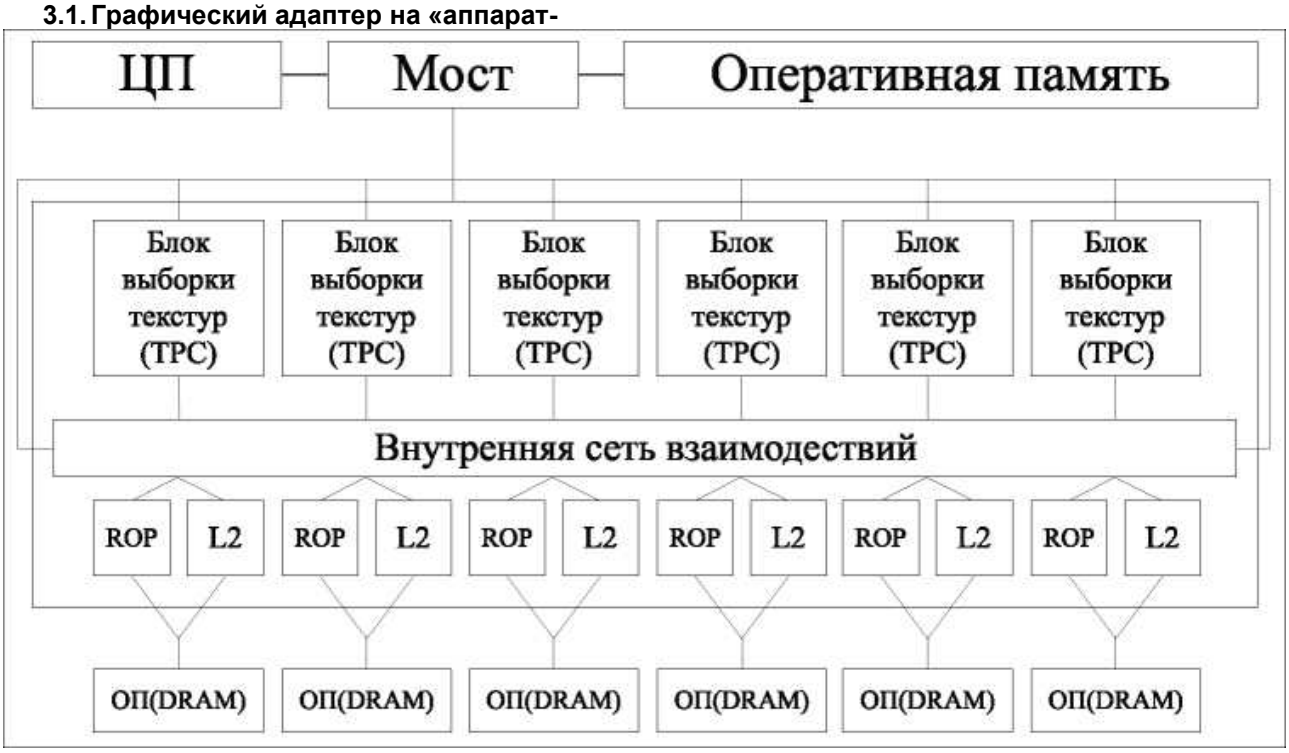


**Рис. 2.** Схематическое изображение устройства графического адаптера [3]

Все вычислительные ядра на видеокарте объединены в независимые блоки TPC (Texture process cluster), количество которых определяется как принадлежностью к тому или иному семейству процессоров (G80 - максимум 8, G200 - максимум 10), так и конкретной моделью (GeForce 220GT - 2, GeForce 275 - 10). Как от видеокарты к видеокарте могут меняться количество TPC, так же может меняться тип и количество микросхем памяти DRAM и, соответственно, общий объем оперативной памяти. Микросхемы памяти имеют кэш второго уровня (L2) и конвейер растровых операций (ROP - Raster Operations Pipeline). Они объединены между собой коммуникационной сетью, в которую также подключены все TPC. Любая дискретная видеокарта взаимодействует с центральным процессором через мост, который может быть как интегрированным в CPU (Intel Core i7), так и вынесенным в чипсет (Intel Core 2 Duo) [3].

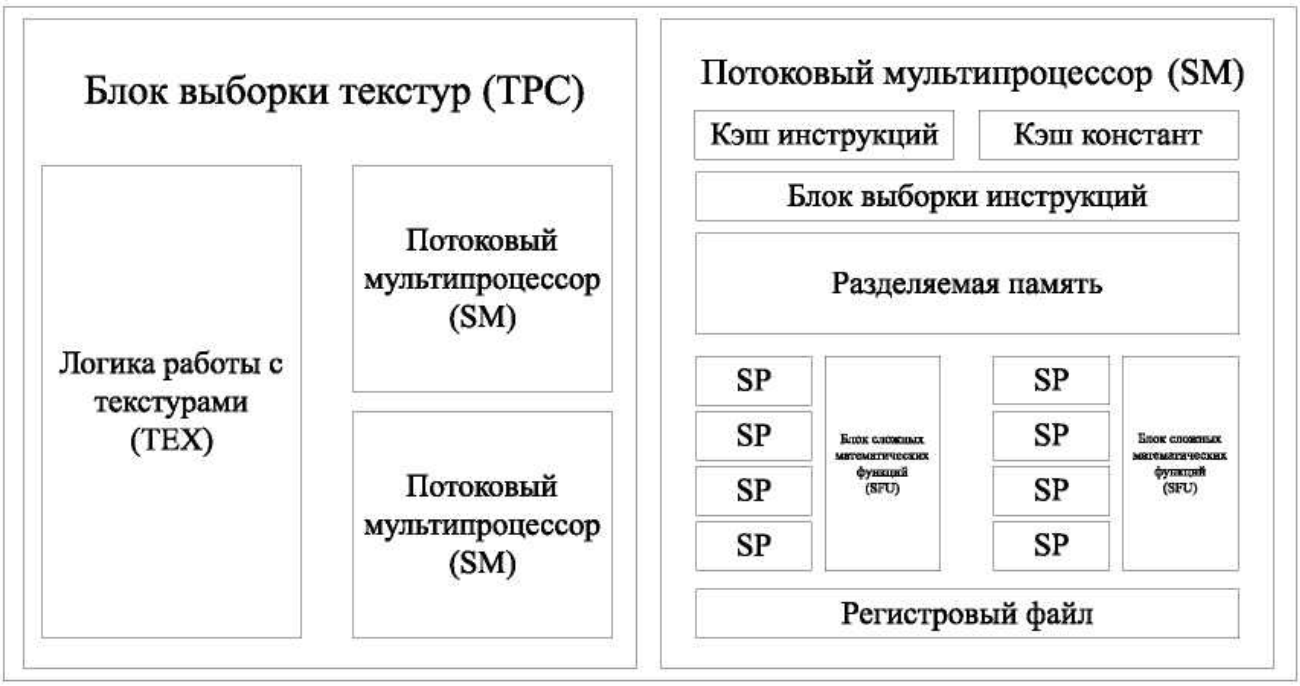


**Рис. 3.** Схематическое изображение устройства TPC и SM чипа G80

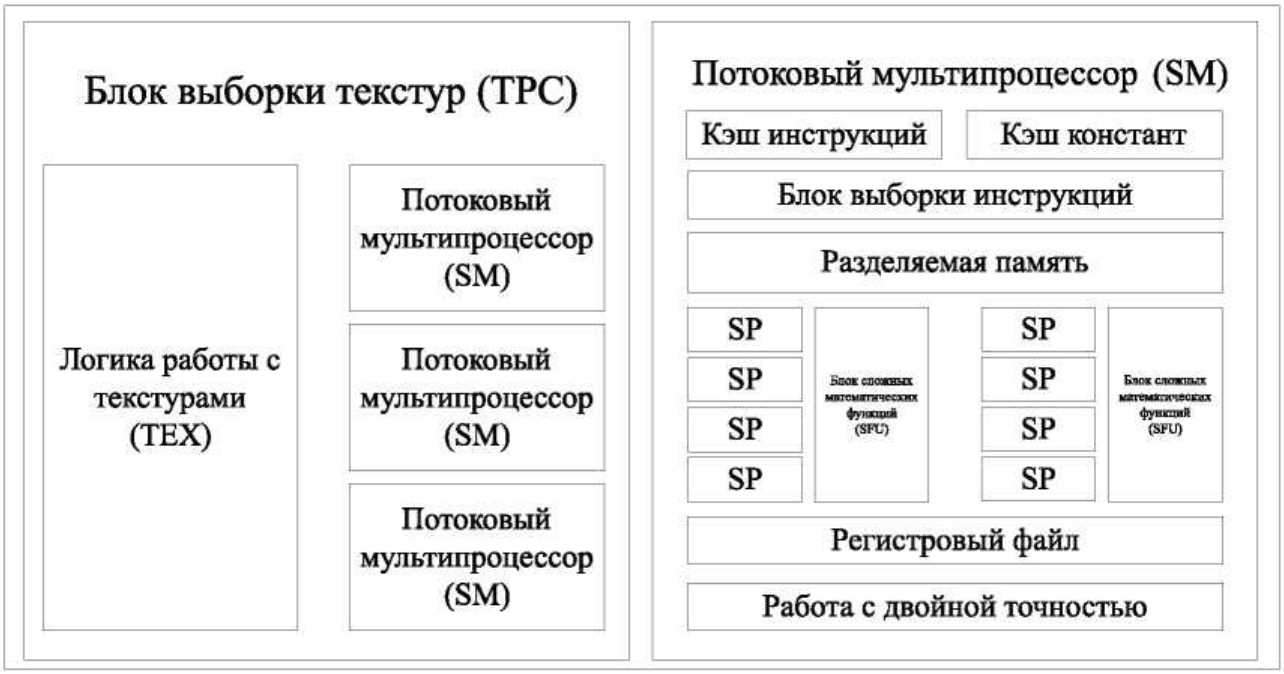


**Рис. 4.** Схематическое изображение устройства TPC и SM чипа G200

### 3.2. Семейство процессоров NVIDIA Fermi

Ключевыми архитектурными особенностями Fermi (рис. 5) являются:

1. Третье поколение потокового мультипроцессора (SM):

- потоковых процессора на SM, что в четыре раза больше, чем у G200;

- 8-кратный прирост производительности в операциях с плавающей точкой (FP) с двойной точностью по сравнению с графическими адаптерами предыдущего поколения;

- 2 блока выборки инструкций (WS - Warp Scheduler) на SM вместо одного;

- 64 КБ ОЗУ с конфигурируемым разделением на общую память и L 1-кэш.

2. Второе поколение набора инструкций параллельного выполнения потоков (Parallel Thread Execution, PTX 2.0):

- унифицированное адресное пространство с полной поддержкой C++;

- полная поддержка вычислений с плавающей точкой с 32- и 64-битной точностью в соответствии со стандартом IEEE 754-2008 [4];

- поддержка 64-битной адресации;

- улучшенная производительность предсказаний

обращений к памяти.

3. Улучшенная подсистема памяти:

- иерархия NVIDIA ParallelDataCache с конфигурируемым L1-кэшем и общим L2-кэшем;
- поддержка памяти с кодом коррекции ошибок ECC (впервые на GPU);
- существенно увеличенная производительность операций чтения и записи в память.

4. Система управления вычислительными потоками NVIDIA GigaThread версии 3.0:

- 10-кратное по сравнению с G200 ускорение процедуры контекстного переключения;
- одновременное выполнение нескольких потоков вычислений.

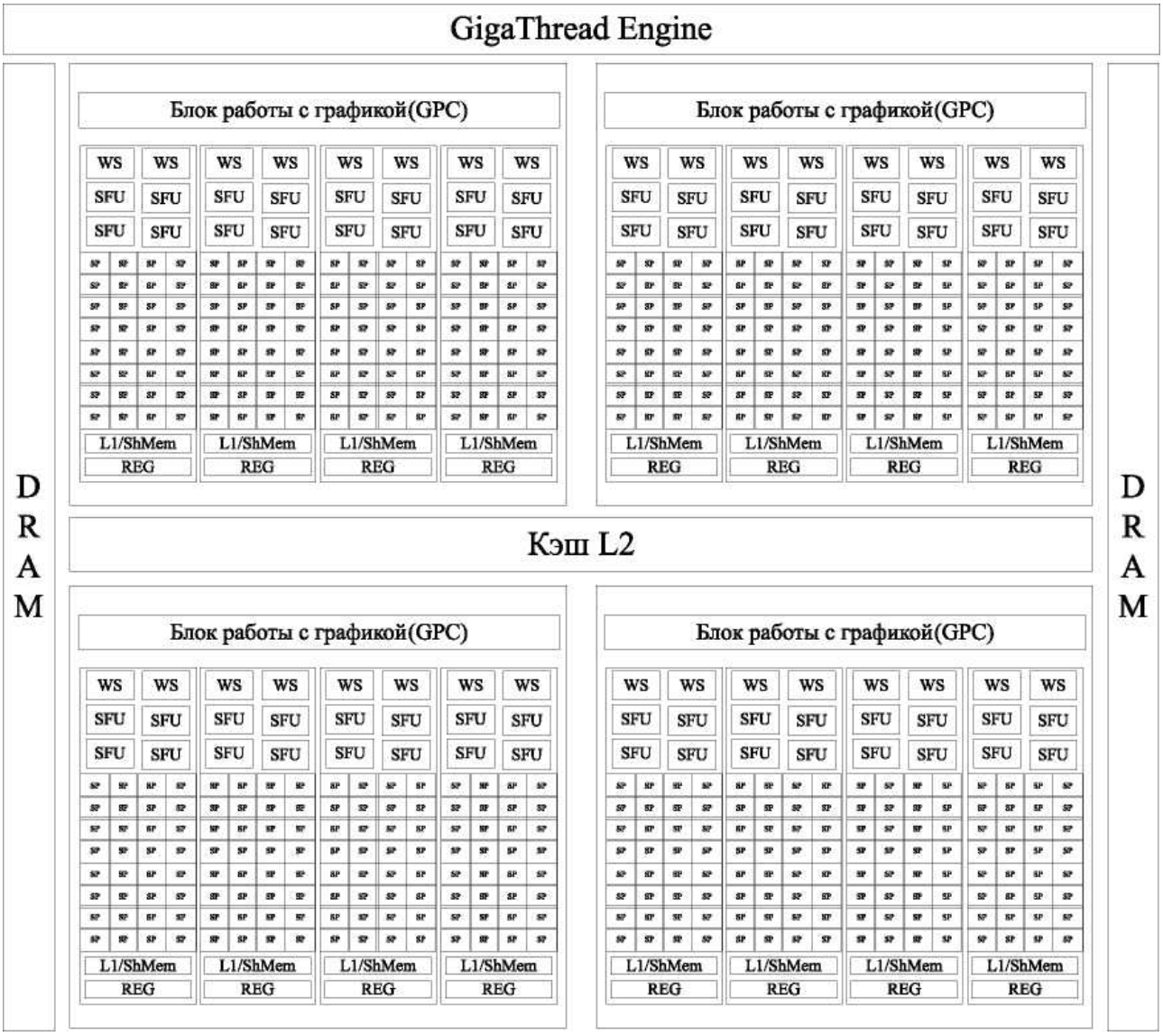


**Рис. 5.** Схематическое изображение устройства графического адаптера на чипе Fermi

В таблице 1 представлено сравнение различных чипов графических адаптеров от Nvidia

**Таблица 1.** Сравнение различных чипов графических адаптеров от NVidia

| Архитектура | G80 | GT200 | Fermi |
|---|---|---|---|
| Год вывода на рынок | 2006 | 2008 | 2009 |
| Число транзисторов, млн. | 681 | 1400 | 3000 |
| Количество потоковых процессоров (CUDA-ядер) | 128 | 240 | 512 |
| Объем разделяемой памяти в расчете на SM, КБ | 16 | 16 | 48 или 16 (конфигурируется) |
| Объем кэш-памяти первого уровня в расчете на SM, КБ | 0 | 0 | 16 или 48 (конфигурируется) |
| Объем кэш-памяти второго уровня, КБ | 0 | 0 | 768 |
| Контроль ошибок ECC | Нет | нет | есть |

Возможности GPU обозначаются при помощи количественных характеристик скорости выполнения определенных операций на графическом процессоре (ComputeCapability), старшая цифра которой соответствует версии архитектуры, а младшая - небольшим изменениям внутри архитектуры. На данный момент

существуют 1.0, 1.1, 1.2, 1.3, 2.0 ComputeCapability. В таблице 2 приведены количественные характеристики скорости выполнения определенных операций на графическом процессоре (ComputeCapability) для различных графических адаптеров

**Таблица 2.** ComputeCapability для различных графических адаптеров

| GPU | ComputeCapability |
|---|---|
| GeForce 8800GTX | 1.0 |
| GeForce 9800GTX | 1.1 |
| GeForce 210 | 1.2 |
| GeForce 275GTX | 1.3 |
| Tesla C2050 | 2.0 |

## 4. Оптимизация трассировки лучей в октантных деревьях

Вопрос повышения эффективности рендеринга воксельных моделей продолжает сегодня оставаться актуальным, несмотря на достигнутый прогресс в последнее время в этой области. Поиски путей ускорения обработки графических данных ведутся как в направлении совершенствования пространственных структур данных и оптимизации алгоритмических решений, так и в направлении использования существующих аппаратных возможностей и многопроцессорных вычислительных архитектур. Много внимания разными исследователями было уделено вопросам эффективной трассировки октантных деревьев, которые широко применяются в компьютерной графике. В известных алгоритмах этого типа трассировка выполняется сверху вниз (от корневого узла дерева) [5] с реализацией рекурсивности, или снизу вверх [5] с поиском решения в окрестности последнего найденного граничного воксела. При этом наибольший эффект достигается при реализации дискретной трассировки [4]. В алгоритме отслеживания лучей в октантных деревьях, реализованном авторами в рамках системы вексельной графики, алгоритмическая оптимизация обеспечивается, как за счет реализации дискретной трассировки, целочисленной арифметики и использования табличных геометрических решений, так и за счет реализации когерентности октантных траекторий пространственно близких лучей. Оптимизация последнего типа существенно отличается от известного аналога [4], где предлагается использование когерентности траекторий в BSP структуре. В работе получены оценки эффективности предлагаемой оптимизации алгоритма. Также приведены сравнительные оценки алгоритмов трассировки *и* прямого рендеринга октантных сцен.

### 4.1. Октантная структура данных

Поддерживаются две формы структуры. Основная (назовем ее структура «объем») содержит: промежуточные узлы, внутренние вокселы, внутренние термы, граничные вокселы, граничные термы. Она обеспечивает выполнение булевых операций и визуализацию. Структура «оболочка» - без внутренних термов и внутренних вокселов - только для визуализации. Термы обозначают октанты, полностью принадлежащие объекту. У граничных термов одна грань или несколько принадлежат граням бокса сцены. Структура «оболочка» требует существенно меньше памяти, чем основная. Во внешней памяти данные хранятся в компрессированном виде. Например, для октантного дерева глубиной 8 сцены «изоповерхность температуры» основная структура в оперативной памяти занимает 22 Мб, в сжатом виде во внешней памяти 1.42 Мб, структура «оболочка», соответственно, 12 Мб и 1.16 Мб. А для сцены «город» с глубиной дерева 9 структура «оболочка» в оперативной памяти занимает 22 Мб и в сжатом виде - 3.59 Мб. Значения нормалей для граничных вокселов при этом заданы плавающей арифметикой. Возможно хранение нормалей в целочисленном виде, тогда объем требуемой памяти уменьшается примерно на 30% [4].

Поддерживаются два режима визуализации: с предварительно вычисленной диффузной освещенностью (используется, когда положение источников освещения не меняется при просмотре с разных точек наблюдения) и с вычислением закраски в процессе рендеринга (по хранящимся в узлах-вокселах нормалям к поверхности объекта) [4].

### 4.2. Дискретная трассировка лучей

В алгоритме трассировки лучей в октантных деревьях существенно используются уже известные решения, к которым можно отнести: рекурсивную обработку дерева top-down с горизонтальным движением по дереву (анализ пересекаемых подбоксов в рамках родительского бокса) и вертикальным (спуск от текущего бокса к пересекаемым подбоксам); выбор «главных направлений» при анализе пересечений подбоксов лучом и рекурсивное упрощенное вычисление «средних точек» по точкам входа, выхода луча в боксе; реализация пространственной когерентности с использованием предварительно вычисленных таблиц геометрических решений; переход к целочисленной арифметике для ускорения вычислений. В данном случае рассматривается режим ray casting. Вместе с тем алгоритм поиска упорядоченного списка пересекаемых подбоксов, построенный на декомпозиции пространственной задачи в «плоские», отличается от известных аналогов [4].

## 5. Заключение

В дальнейшем планируется рассмотреть другие методы оптимизации технологии трассировки лучей. Планируется рассмотреть варианты объединения существующих методов и создание общего методического подхода по оптимизации технологии трассировки.

## Благодарность

Работа выполнена при финансовой поддержке РФФИ в рамках научного проекта № 19-07-00455.

## Библиографический список

[1] Ульянов А.Ю., Котюжанский Л.А., Рыжкова Н.Г. Метод трассировки лучей как основная технология фотореалистичного рендеринга // Фундаментальные исследования. – 2015. – № 11-6. – С. 1124-1128; URL: http://www.fundamental-

research.ru/ru/article/view?id=39706 (дата обращения: 25.04.2020).
[2] Городничев М.Г., Гематудинов Р.А., Кухаренко А.М. О некоторых методах визуализации динамических 3D моделей // «Экономика и качество систем связи» 1/2018, с. 18-29
[3] Казённов А.М. Основы технологии CUDA // Компьютерные исследования и моделирование 2010 Т. 2 № 3 С. 295-308.
[4] IEEE Standard for Floating-Point Arithmetic: IEEE Computer Society Sponsored by the Microprocessor Standards Committee. - IEEE Std 754-2008 (Revision of IEEE Std 754-1985). - 70 p. https://irem.univ-reunion.fr/IMG/pdf/ieee-754-2008.pdf
[5] В.А. Бобков, С.В. Мельман, Ю.И. Роньшин. Оптимизация трассировки лучей в октантных деревьях // International Conference Graphicon 2005, Novosibirsk Akademgorodok, Russia, http://www.graphicon.ru/.