Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика, искусственный интеллект и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

# РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

## *К КУРСОВОЙ РАБОТЕ*

## *НА ТЕМУ:*

## *«Визуализация воды с использованием вокселей»*

*2023 г.*

# СОДЕРЖАНИЕ

# Введение

Термин "Компьютерная графика"обозначает любое использование компьютера для создания и манипуляции изображениями. Задачи и области применения компьютерной графики сложно охарактеризовать в силу их большого колчичества, однако в общем случае можно выделить следующие направления:

— **Моделирование** – занимается математическим описанием формы и внешнего вида в формате, пригодным для использования в компьютере;

— **Рендеринг** или **Отрисовка** – термин, заимствонный из изобразительного искусства, обозначающий создание изображений из компьютерных моделей;

— **Анимация** – это техника создания иллюзии движения используя последовательность изображений [1].

Рендеринг – это основная составляющая компьютерной графики. На самом высоком уровне абстракции рендеринг является процессом преобразования описания трехмерной сцены в изображение. Алгоритмы для создания анимации, геометрического моделирования, нанесения текстур и других областей компьютерной графики должны проходить через некий процесс рендеринга, чтобы их результаты могли быть видны на изображении. Рендеринг стал повсеместным: от кино до игр и далее, он открыл новые горизонты для творческого выражения, развлечения и визуализации.

В первые годы развития этой области исследования в графическом рендеринге сосредоточивались на решении фундаментальных проблем, таких как определение, какие объекты видны из определенной точки обзора. По мере нахождения эффективных решений для этих проблем и доступности более богатых и реалистичных описаний сцен благодаря продолжающемуся прогрессу в других областях графики, современный рендеринг начал включать идеи из широкого спектра дисциплин, включая физику и астрофизику, астрономию, биологию, психологию и изучение восприятия, а также чистую и прикладную математику. Междисциплинарный характер рендеринга является одной из причин, почему это такая увлекательная область исследований.

В последние годы активно стала развиваться фотореалистичная компьютерная графика. Это дисциплина находится на стыке физики и классической

компьютерной графики. Высокий реализим изображений требует больших вычислительных мощностей для их получения, что заставляет разработчиков постоянно искать новые, более эффективне способы рендеринга.

Фотореалистичная компьютерная графика теперь повсеместно используется, включая такие области как развлечения, в частности, кино и видеоигры, дизайн продуктов и архитектура. За последнее десятилетие широкое распространение получили физически ориентированные методы визуализации, где точное моделирование физики рассеяния света является основой синтеза изображений. Эти подходы обеспечивают как визуальный реализм, так и предсказуемость [2].

Цель данной работы – реализовать программу для построения реалистичных изображений трехмерных воксельных сцен в реальном времени, с возможностями физически-корректной визуализации воды.

Чтобы достигнуть поставленой цели, требуется решить следующие задачи.

— Описать структуру трехмерной сцены, включая объекты, из которых состоит сцена и их материалы, и определить способ задания исходных данных;

— Выбрать или разработать алгоритмы компьютерной графики, позволяющие визуализировать трехмерную воксельную сцену в реальном времени;

— Реализовать выбранные и адаптированные алгоритмы построения трехмерной сцены;

— Исследовать возможности улучшения производительности программы и повышения сложности задаваемых сцен.

# 1 Аналитический раздел

В данном разделе будут рассмотрены возможножсти отрисовки трехмерных изображений в реальном времени, проанализированы особенности трехмерной воксельной графики; будет выбран метод решения поставленной задачи.

## 1.1 Выбор оборудования

Построение трехмерных изображений – вычислительно сложная, и в то же время, чрезвычайно параллельная задача [3]. Такие особенности стали причиной появляния специального оборудования для выполнения популярных задач компьютерной графики. Такие вычислительные устройства называются Графическими процессорами (GPU), хотя их применение не ограничивается вычислениями графических данных. Появление таких устройств вызвано ограничениями в дизайне центральных процессоров. Они используются для последовательного выполнения кода с большим числом условных переходов, на небольшом числе ядер при SMP. И несмотря на развитие SIMD парадигмы, позволяющей обрабатывать большее число данных, чем позволяет последовательное исполнения, ЦПУ не смогли стать универсальными устройствами для рендеринга. Особенностью графических сопроцессоров ялвяется большое число ядер, способных эффективно выполнять тысячи вычислений параллельно [4].

Несмотря на большие возможности параллельного выполнения вычислительных задач, графические сопроцессоры имеют органиченные возможности условного выполнения программ, в частности, рекурсивных, малую скорость памяти, и возможности взаимодействия с ней.

Такие особенности породили целый ряд алгоритмов, оптимизированных для выполняения на графических сопроцессорах, которые отличаются от аналогичных для процессоров общего назначения. В частности, графические сопроцессоры могут иметь специальные аппаратные блоки для ускоренного выполнения алгоритмов трассировки лучей, и все из них имеют аппартное ускорение алгоритма Z-буфера [5].

Из-за развития графических сопроцессоров, использование ЦПУ для реализации графических вычислений устарело. Графические сопроцессоры предоставляют более широкие возможности для проведения операций, типич-

ных для компьютерной графики.

## 1.2 Выбор алгоритма отрисовки

При выборе алгоритма отрисовки должны быть учтены особенности поставленной задачи. Отрисовка будет выполняться в режиме реального времени. Этот факт предъявляет к алгоритму требование по скорости работы, время отрисовки кадра не должно занимать больше небольшого количества миллисекунд. Также для визуализации воды следует построить некоторые физические эффекты: отражение и преломление света. Выбираемый алгоритм отрисовки должен поддерживать визуализацию или приближение физических явлений.

Будут рассмотрены алгоритмы, эффекивная реализация которых может быть получена при использовании графических ускорителей.

### 1.2.1 Алгоритм, использующий Z-буфер

Z-буфер, или буфер глубины, является одним из простейших алгоритмов отрисовки. Z-буфер – это простое расширение идеи буфера кадра. Буфер кадра используется для хранения атрибутов (интенсивности или оттенка) каждого пикселя в пространстве изображения. Z-буфер является отдельным буфером глубины, используемым для хранения координат или глубины каждого видимого пикселя в пространстве изображения. При использовании глубина или значение $z$ нового пикселя, который записывается в буфер кадра, сравнивается с глубиной этого пикселя, хранящейся в z-буфере. Если сравнение показывает, что новый пиксель находится перед пикселем, хранящимся в буфере кадра, то новый пиксель записывается в буфер кадра, а $z$-буфер обновляется новым значением $z$. В противном случае никаких действий не предпринимается. Концептуально алгоритм является поиском по $x$, $y$ для нахождения наибольшего значения $z(x, y)$.

Простота алгоритма – его главное преимущество. Кроме того, он легко справляется с проблемой видимой поверхности и отображением сложных пересечений поверхностей. Хотя алгоритм $z$-буфера часто реализуется для многогранно представленных сцен, он применим для любого объекта, для которого можно рассчитать глубину и характеристики тени [6].

Аппартные графические ускорители изначально были спроектированы

для использования в мультимедийных приложениях, использующих алгоритм $z$-буффера. В силу этого почти все их конвееры заточены специально под эффективную реализацию этого алгоритма с некоторыми расширениями. Таким образом, использование алгоритма $z$-буффера при отрисовке на графических ускорителях обеспечивает хорошую производительность.

Однако, алгоритм $z$-буффера плохо подходит для реализации физически-корректного рендеринга. В частности, симуляция отражения или преломления требует множественной отрисовки сцены из разных точек, что замедляет общий процесс отрисовки. Дополнительно, получаемые при отрисовке результаты оказываются менее физически точными, чем при использовании алгоритма трассировки лучей. В силу этого, в последние годы традиционные конвееры отрисовки на графических ускорителях расширяются добавлением трассировки лучей для получения лучшего качества изображения[1].

Характеристики алгоритма:

— Поддержка физически-корректной отрисовки требует больших модификаций алгоритма, дополнительных затрат памяти и времени;

— Отличная аппаратная поддержка;

— Малые вычислительные затраты за счет аппаратной поддержки.

### 1.2.2 Трассировка лучей

Почти все системы фотореалистичной рендеринга основаны на алгоритме трассировки лучей. Трассировка лучей на деле очень простой алгоритм; он основан на отслеживании пути луча света через сцену, по мере его взаимодействия и отражения от объектов в окружающей среде. Несмотря на то, что существует множество способов реализовать алгоритм трассировки лучей, все такие системы должны включать в себя и симулировать следующие объекты и феномены:

— Камеры: Модель камеры определяет, как и откуда просматривается сцена, включая то, как изображение сцены записывается на сенсоре. Многие системы рендеринга генерируют оптические лучи, начинающиеся в камере, которые затем прослеживаются в сцене.

— Пересечение лучей и объектов: Нам необходимо точно определить, где заданный луч пересекает заданный геометрический объект. Кроме того, нам нужно определить некоторые свойства объекта в точке пересечения, такие как нормаль поверхности или его материал. Большинство трассировщиков лучей также имеют некоторую возможность проверки пересечения луча с несколькими объектами, обычно возвращают ближайшее пересечение по лучу.

— Источники света: Без освещения бессмысленно визуализировать сцену. Трассировщик лучей должен моделировать распределение света по всей сцене, включая не только местоположение самих источников света, но и способ, которым они распространяют свою энергию в пространстве.

— Видимость: Чтобы знать, может ли заданный источник света передавать энергию в точку поверхности, нам нужно знать, есть ли непрерывный путь от точки до источника света. К счастью, на этот вопрос легко ответить в трассировщике лучей, так как мы просто можем построить луч от поверхности до источника света, найти ближайшее пересечение луча с объектом и сравнить расстояние пересечения с расстоянием до источника света.

— Рассеяние на поверхности: Каждый объект должен предоставить описание своего вида, включая информацию о том, как свет взаимодействует с поверхностью объекта, а также о характере перераспределенного (или рассеянного) света. Модели рассеивания на поверхности обычно параметризуются таким образом, чтобы можно было смоделировать разнообразные внешние виды.

— Непрямое распространение света: Поскольку свет может достигать поверхности после отражения от других поверхностей или прохождения через них, обычно необходимо проследить дополнительные лучи, исходящие из поверхности, чтобы полностью учесть этот эффект.

— Распространение лучей: Нам нужно знать, что происходит с светом, распространяющимся вдоль луча по мере его прохождения через пространство. Если мы отображаем сцену в вакууме, энергия света остается постоянной вдоль луча. Хотя истинные вакуумы необычны на Земле,

для многих сред они являются разумным приближением. Для отслеживания лучей через туман, дым, атмосферу Земли и т. д. доступны более сложные модели [2].

Трассировка лучей, являясь крайне простым алгоритмом, предоставляет несчислимые возможности для расширения. Алгоритм трассировки лучей может использоваться для создания фотореалистичных изображений [2]. Такие трассировщики используются при создании спецэффектов к фильмам, анимационных картин и др.

Характеристики алгоритма:

— Поддержка физически-корректной отрисовки не требует больших модификаций алгоритма, дополнительных затрат памяти;

— Физически-корректная отрисовка работает для большинства физических эффектов, исключая некоторые специфичные (к примеру, каустики);

— Слабая аппаратная поддержка (только некоторые новые графические ускорители, и только некоторые графические API);

— Большие вычислительные затраты.

## 1.3 Выбор алгоритма обхода воксельной сцены

Воксель – способ представления геометрии, альтернативный типичному полигональному. Воксель представляет собой некоторое значение на регулярной решетке в трехмерном пространстве. Воксели часто испольуются при анализе медицинских и научных данных.

В интерактиных графических приложениях, выполняющих отрисовку в реальном времени, воксели редко применялись в качестве основного геометрического примитива. Это связано с тем, что графические сопроцессоры были специально оптимизированы для работы с полигонылными данными, и они были более простым способом достичь требуемого качества изображения.

В последние годы замечается тенденция повышения популярности вокселей в интерактивных приложениях. Это связано с повышением мощности вычислительной техники, позволяющей выполнять ранее невозможные вычисления на пользовательских компьютерах. Воксельная графика позволяет достигать большей детализации, чем возможно при использовании полигонов.

Главная проблема воксельной графики – большое число затрачиваемой памяти и медленный доступ к ней. Поэтому все воксельные алгоритмы фокусируются на ускорении доступа к индивидуальным вокселям, для использования в алгоритмах отрисовки. Рассмотрим и выберем алгоритм обхода воксельной сцены и структуру данных хранения вокселей.

### 1.3.1 Алгоритм быстрого обхода вокселей для трассировки лучей

Рассматриваемый алгоритм описан в классической статье задолго до массового распространения графических ускорителей и повсеместного использования трассировки лучей. В ней формализован и описан несложный алгоритм обхода воксельной трехмерной сетки с константным масштабом (см. рисунок 1.1).

Переход от одного вокселя к его соседу требует только двух сравнений чисел с плавающей запятой одного сложения чисел с плавающей запятой. Кроме того, исключаются множественные пересечения лучей с объектами, находящимися в более чем одном вокселе [7].

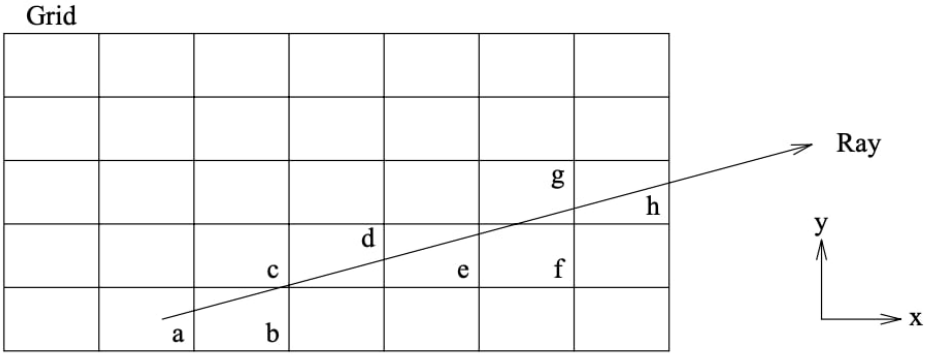


Рисунок 1.1 – Алгоритм быстрого обхода вокселей для трассировки лучей

Авторами отмечается один существенный недостаток: высокие вычислительные затраты. Для решения этой пробремы они предлагают вводить оптимизационные структуры, к примеру, BVH. Такие структуры должны разделять пространство мира и проверять луч на столкновение только с его частью, вместо всего пространства. Проблема такого подхода заключается в том, что BVH требует больших вычислительных затрат на построение, и само построение – NP-полная задача. Это делает рассматриваемый алгоритм малоприменимым во всех нетривиальных случаях.

### 1.3.2 k-d дерево

k-d дерево (сокращение от k-мерного дерева) - это структура данных для разделения пространства, предназначенная для организации точек в k-мерном пространстве. k-d деревья являются полезной структурой данных для нескольких приложений, таких как поиск с использованием многомерного ключа поиска (например, поиск по диапазону и поиск ближайшего соседа) и создание облаков точек. k-d деревья являются особым случаем деревьев бинарного разделения пространства. k-d дерево – это бинарное дерево, где каждая вершина – это точка в k-мерном пространстве. Каждая вершина, не являющаяся листом может быть представлена как неявная гиперплоскость, разделяющая пространство на две части. Точки в каждой из частей представлены соответсвующии поддеревьями (см. пример на рисунке 1.2).

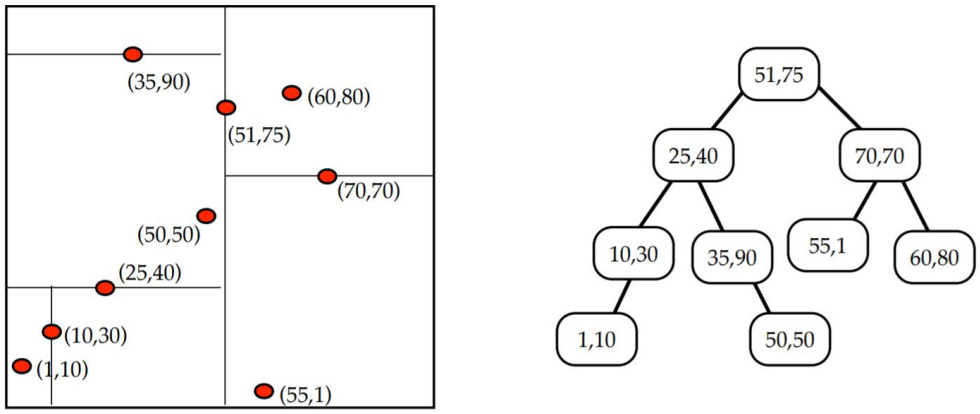


Рисунок 1.2 – Пример двумерного k-d дерева

k-d дерево стало популярным как основная оптимизационная структура в трассировщиках лучей. Были предложены алгоритмы оптимизации структуры для графических ускорителей. Результаты замеров показывают, что скорость отрисовки сцены с использованием k-d дерева как минимум на порядок больше, чем скорость отрисовки в сетке с константным размером сетки.

k-d дерево можно использовать для отрисовки сцен с геометрией, хранимой в произвольном формате. Вариант k-d дерева, используемый при воксельной отрисовке – Октодерево. Это k-d дерево в трехмерном пространстве, где у каждой родительской вершины 8 детей [8].

### 1.3.3 Разреженное воксельное октодерево

Проблема обычного Октодерева – это то, что оно хранит данные для каждого из элементов пространства. Обычно сцены являются разреженными, что делает плотное хранение данных избыточным по памяти, и менее производительным. Поэтому чаще всего при отрисоввке воксельных сцен в качестве оптимизационной структуры данных применяется разреженное воксельное октодерево (см. рисунок 1.3).

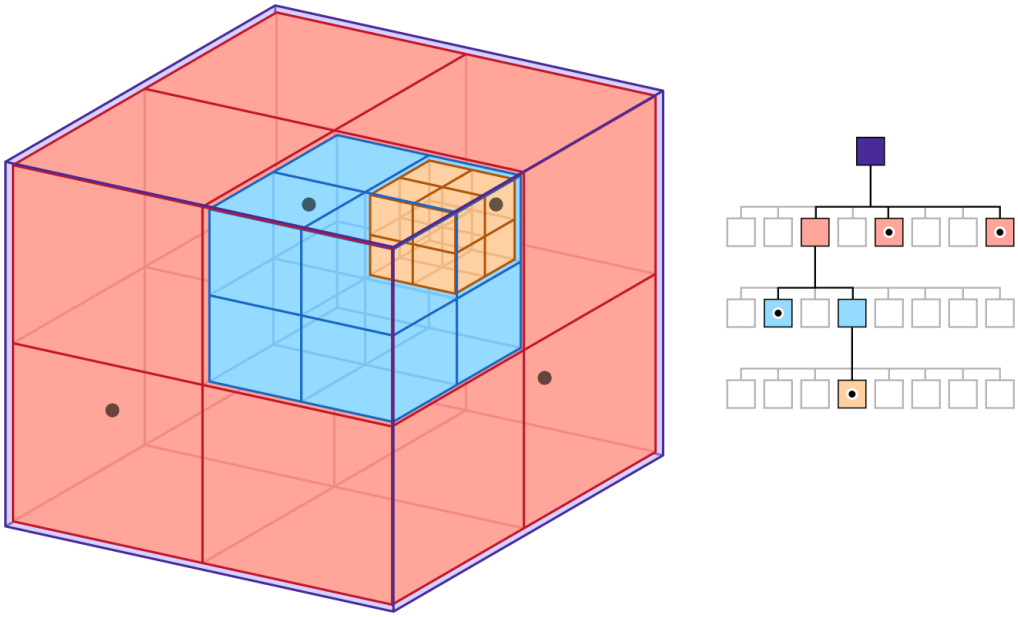

Рисунок 1.3 – Октодерево

Такая структура данных имеет как отличные временные характеристики, так и малые затраты памяти. Основная сложность заключается в организации данных в оперативной памяти, на диске, и в памяти графического ускорителя [9].

## 1.4 Выбор модели освещения

Модель освещения в случае трассировки лучей – это фундаментальная характеристика программы-отрисовщика. Это связано с тем, что трассировка лучей полагается на выбранную модель освещения при построении изображения, и от выбранной модели зависят не только детали картинки, как в случае с $z$-буффером, но качества результата в целом.

### 1.4.1 Модель освещения Ламберта

Самая простая модель затенения основана на наблюдении, сделанном Ламбертом в 18 веке: количество энергии от источника света, которое падает на поверхность, зависит от угла поверхности к свету. Поверхность, направленная прямо на свет, получает максимальное освещение; поверхность, касательная к направлению света (или обращенная к свету), не получает освещение; и между ними освещение пропорционально косинусу угла $\theta$ между нормалью поверхности $n$ и источником света $l$. Это приводит к модели освещения Ламберта:

$$L = k_d I max(0, n \cdot l) \tag{1.1}$$

где $I$ – интенсивность источника света. Поскольку $n$ и $l$ – единичные векторы, мы можем использовать $n \cdot l$ как удобное сокращение для $cos(\theta)$. Вектор $l$ вычисляется путем вычитания точки пересечения луча и поверхности из положения источника света [1].

### 1.4.2 Модель освещения Фонга

Модель освещения Ламберта не зависит от точки обзора: цвет поверхности не зависит от направления, с которого на нее смотреть. Многие реальные поверхности обладают определенной степенью блеска, создающего мерцание или зеркальные отражения, которые кажутся перемещающимися при изменении точки обзора. Ламбертово освещение не создает мерцания и создает очень матовый, меловидный вид, и множество моделей освещения добавляют зеркальную составляющую к ламбертовому освещению; ламбертова часть в таком случае является диффузной составляющей.

Очень простая и широко используемая модель для зеркальных мерцаний была предложена Фонгом, а позже обновлена Блинном до формы, которая наиболее распространена сегодня. Идея заключается в создании отражения, которое ярче всего, когда векторы v и l симметрично расположены относительно нормали поверхности, то есть, когда происходит зеркальное отражение; отражение затем постепенно уменьшается, по мере того, как векторы отдаляются от зеркальной конфигурации. Таким образом, формула модели освещения Фонга:

$$h = \frac{v+1}{||v+1||}, \tag{1.2}$$

$$L = k_d Imax(0, n \cdot l) + k_s Imax(0, n \cdot h)^2 \tag{1.3}$$

где $h$ – бисскетриса угла между $v$ и $l$; $p$ – экспонента Фонга; $k_s$ – коэффицент зеркальности [10] [1].

### 1.4.3 Физически-корректная модель

Модели Ламберта и Фонга просты, и поэтому их использование не требует больших вычислительных затрат. Но в то же время, они лишь пытаются построить упрощенную модель освещения пространства, в то время как может быть построена более общая симуляция физических явлений.

Подход, в котором для построение изображений используется симуляция физических явлений, а не ее приближение, называется физически-корректной моделью освещения. Реализация алгоритмов такого подхода построена на Методе Монте-Карло. Производится отрисовка сцены с множеством переменных параметров, к примеру, направлению лучей, или распределению света по пространству, которые имеют заранее известные характеристики вероятностного распределения.

Задача построения изображения в физически-корректной модели – задача интеграции Уравнения рендеринга [11]. Для интеграции используется статистический анализ распределения значений Двулучевой функции отражательной способности [2].

#### Микрогранные модели

Многие подходы к моделированию отражения и пропускания поверхности, основанные на геометрической оптике, основаны на идее того, что шероховатые поверхности могут быть представлены в виде набора маленьких граней. Микрограни часто моделируются в виде поверхностей с высотой, где распределение ориентации граней описывается статистически (см рис. 1.4).

Модели отражения, основанные на микрофасетках, которые проявляют идеальное зеркальное отражение и пропускание, успешно используются для моделирования рассеяния света от различных глянцевых материалов, включая металлы, пластик и матовое стекло. Одной из важных характеристик

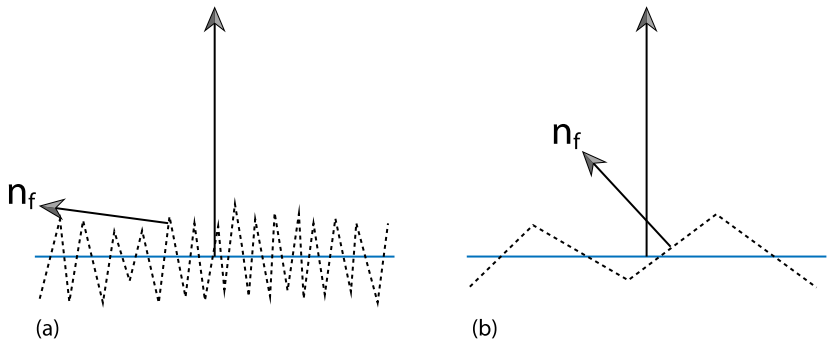


Рисунок 1.4 – Микрогранные модели

поверхности микрофасетки является функция распределения, которая показывает дифференциальную площадь микрофасеток с нормалью поверхности. Две самые популярные модели микрограней – основанная на модели Бекерманна, и основанная на модели GGX. Их разница заключается в различных радиометрических свойствах, которые неважны для непрофессионального разработчика трассировщиков. Однако на практике, модель GGX показывает лучшие результаты при одинаковом количестве вычислений. [2] [12]

## 1.5 Вывод

Было принято решение выполнять разработку для выполнения на графическом сопроцессоре, поскольку это позволяет выполнять отрисовку значительно более эффективно, чем на ЦПУ.

В качестве алгоритма трассировки был выбран алгоритм трассировки лучей, поскольку он имеет лучшие возможности для получния физически-корректных изображений, чем алгоритм, использующий $z$-буфер. Для оптимизации отрисовки был выбран способ кеширования обратной репроекции.

В качестве модели освещения была выбрана физически-корректная модель с использованием микрограней с моделью GGX, поскольку такой вариант предоставляет лучший вариант получения реалистичных изображений, чем другие модели освещения и модель микрограней Бекерманна [12].

# 2 Конструкторский раздел

В данном разделе будут описаны особенности написания программ для графических процессоров, а также алгоритмы и структуры данных, выбранные для решения поставленной задачи.

## 2.1 Особенности написания программ для графических процессоров

Программы, написанные для выполнения на GPU называются шейдерными (шейдерами). Конвеер графического процессора (см. рисунок 2.1) – последовательно запускаемые установленные пользователем шейдеры. Входными данными такого конвеера являются вершины и набор настроек, вроде шейдерных переменных, текстур и др. Конвеер в современных API (Vulkan, DirectX12 и др.) полностью настраивается программистом на этапе создания. В описание конвеера входит определение шейдерных программ, и их переменных. Две основные шейдерные программы – это вершинная, и фрагементная (пиксельная). Особенностью работы GPU являляется возможность параллельного запуска кода на множесвте потоков (более 100) и с встроенной обязательной работой в парадигме SIMD. Это означает, что запускаемый на одном из потоков код на деле выполняет пакет вычислений одновременно. Такой подход дает отличную производительность в случае отсутствия условных переходов. Но появление условных переходов замедляет шейдерную программу в несколько раз. По своему дизайну, трассировщики лучей включают в себя большое число условных переходов (обработка случайных значений, проверка столкновений и т.д.) – этот факт заставляет строить дизайн трассировщика для выполнения на графическом процессоре иначе, чем для исполнения на центральном процессоре. В частности, основная задача написания трассировщика лучей – уменьшить количество условных переходов за счет предварительного вычисления части значений [13].

Из-за того, что современные графические процессоры написаны в первую очередь для реализации алгоритма $z$-буфера, в большинстве из них отсутствует аппаратная поддержка ускорения трассировки лучей. Для большей переносимости программы она должна быть написана без использования таких расширений – алгоритм должен быть реализован с нуля в одной или

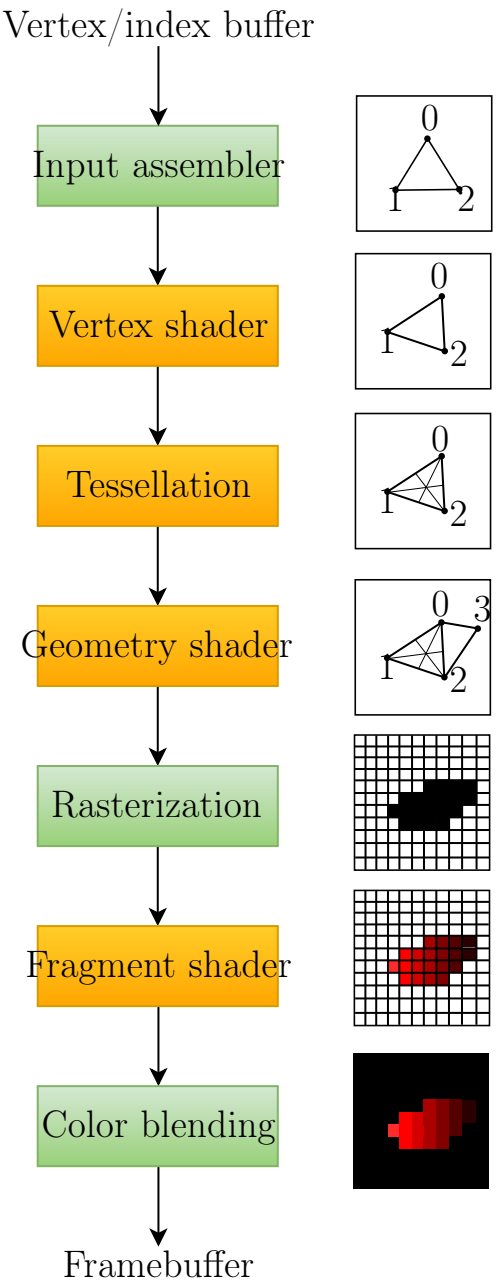


Рисунок 2.1 – Конвеер графического процессора

нескольких шейдерных программах.

Следующая особенность программирования шейдеров – модель доступа к памяти. Шейдеры не имеют прямого доступа к памяти центрального процессора. Они могут получать доступ только к заранее определенным переменным (векторы, матрицы, простые структуры, их массивы), либо к текстурам. Доступ к текстурам производится быстрей, чем доступ к массивам, поэтому сложные данные обычно стараются закодировать в многомерную структуру. Для эффективной реализации алгоритма трассировки лучей требуется составить алгоритм доступа к элементам мирового пространства, переденным через текстуру.

## 2.2 Разработка алгоритмов

### 2.2.1 Алгоритм работы программы

На рисунке 2.2 приведена схема алгоритма работы программы. $MAX\_BOUNCE\_COUNT$ – это целочисленная константа, обозначающая максимальное число отскоков одного луча. Под цветом понимается трехмерный вектор в формате RGB в диапозоне $[0, 1)$. $rgb(0, 0, 0)$ – функция черного цвета.

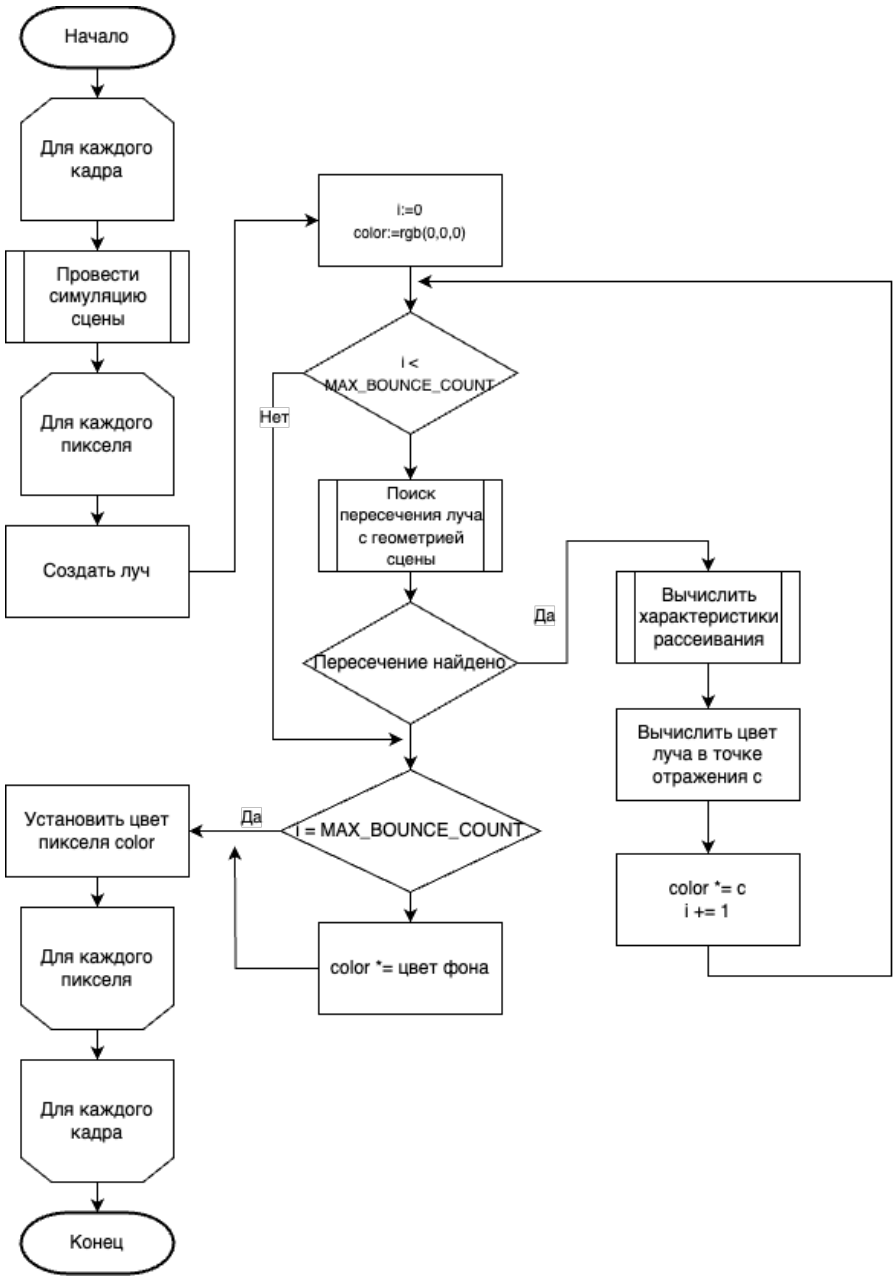


Рисунок 2.2 – Алгоритм работы программы

### 2.2.2 Алгоритм кэширования обратной репроекции

Выполнение пиксельных шейдеров потребляет все большую часть вычислительного бюджета для приложений в реальном времени. Однако, значительная временная согласованность в видимых поверхностных областях, условиях освещения и расположении камеры позволяет повторно использовать вычислительно интенсивные расчеты освещения между кадрами, что позволяет достичь значительного повышения производительности при небольшом снижении визуального качества. Кэширование на основе обратной репроекции позволяет пиксельным шейдерам сохранять и повторно использовать расчеты, выполненные в видимых точках поверхности. Такой подход обеспечивает значительное повышение производительности для многих распространенных эффектов в реальном времени, включая предварительно вычисленные глобальные эффекты освещения, стереоскопическую отрисовку, движущийся размытый фон, глубину резкости и теневую картографию [14].

Временная обратная проекция - это процесс отображения ранее сгенерированного кадра на текущий кадр. Это позволяет повторно использовать информацию или, в случае трассировки лучей, накапливать сэмплы (и тем самым сходиться к решению уравнения отображения) даже при движении.

На рисунке 2.3 представлена блок-схема алгоритма Кэширования обратной репроекции.

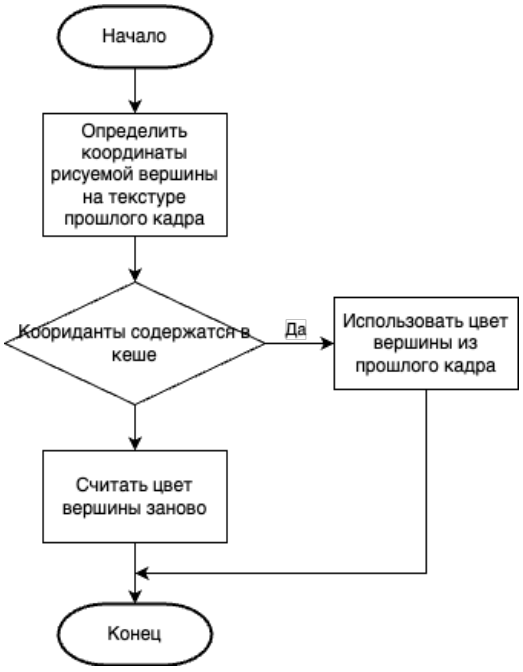


Рисунок 2.3 – Алгоритм Кэширования обратной репроекции

### 2.2.3 Алгоритм Русской Рулетки

Для увелечения производительность вычисления трассировки лучей методом Монте-Карло может применяться т.н. алгоритм Русской рулетки.

Алгоритм заключается в увелечении вероятности того, что каждый пущенный луч будет иметь значимый вклад в итоговую картинку. Русская рулетка отбрасывает вычисление лучей, которые сложны для подсчета и при этом приносят малый радиометрический вклад в итоговую картинку [2].

На рисунке 2.4 приведена блок-схема алгоритма Русской рулетки. Здесь R – минимальное число отскоков для инициализации алгоритма, C – вероятностный параметр успеха работы алгоритма. Под $random[0, 1)$ понимается генерация случайного числа в диапозоне $[0, 1)$.

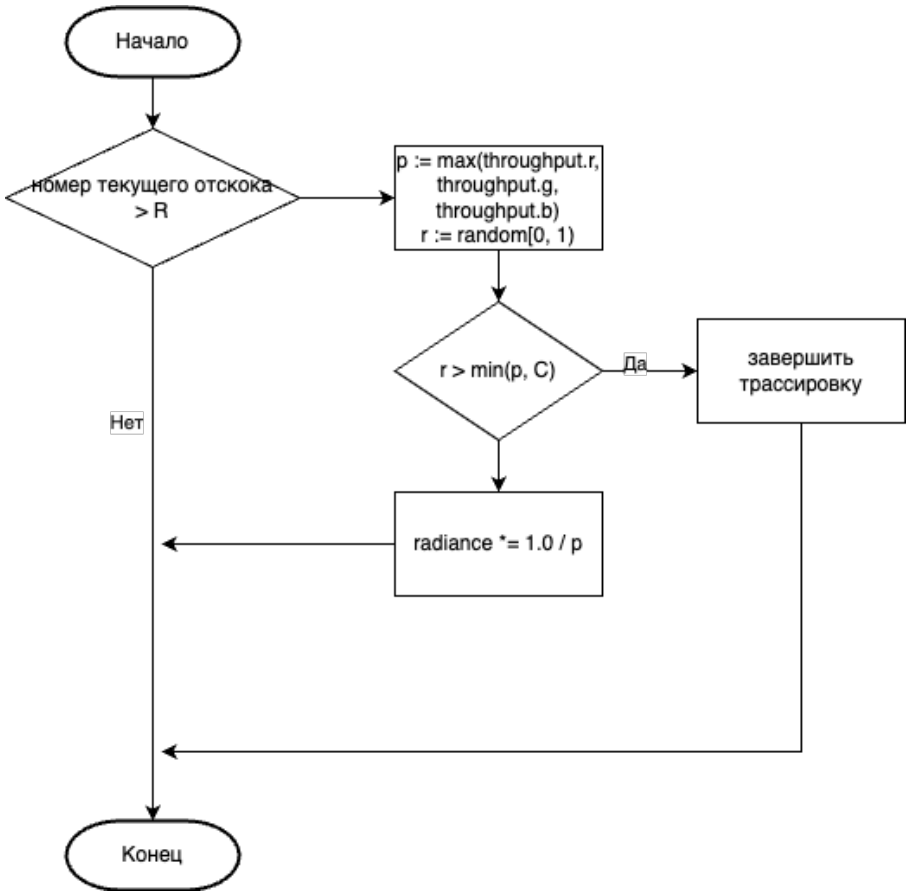


Рисунок 2.4 – Алгоритм Русской рулетки

## 2.3 Вывод

В данном разделе были описаны особенности написания алгоритмов для выполнения на графических процессорах, алгоритмы и структуры данных, выбранные и разработанные для решения поставленной задачи.

# 3 Технологический раздел

## 3.1 Требование к ПО

Программа должна предоставлять следующий функционал:

— Загрузка сцены из файла;

— Вокселизация сцены;

— Получение изображения сцены с учетом материалов объектов сцены и освещения;

— Интерактивное управление камерой;

— Время отрисовки одного кадра не больше 100мс;

— Осуществление поворота и перемещения камеры;

— Симуляция распространения воксельных частиц воды.

Программа должна корректно и оперативно реагировать на все действия пользователя.

## 3.2 Средства реализации разработанного ПО

В качестве графического API для реализации разработанного ПО был выбран WebGPU – современный переносимый графический API, с уровнем контроля над видеокартой, совместимым с графическими API последних поколений (Vulkan, DirectX12, Metal), способный быть использованным при разработке на любой из платформ, поддерживающих классические API (Direct3D, Metal, Vulakn, OpenGLES, WebGL и др.) [15].

В качестве языка программирования для реализации ПО был выбран Rust – статически типизированный компилируемый язык программирования общего назначения [16]. Данный выбор обусловлен наличием возможностей написания производительного кода, компиляции на множественное число платформ, в том числе на WASM, наличием библиотек с поддержкой WebGPU, линейной алгебры.

Для реализации графического интерфейса была выбрана библиотека Dear ImGui. Она реализует графический интерфейс в режиме IMGUI. Библиотека написана для использования с любым графическим API и представляет способ динамического программирования интерфейсов любой сложности [17].

ПО реализовано для платформы WASM и нативных (MacOS, Linux, Windows).

## 3.3 Реализация алгоритма трассировки лучей

Алгоритм трассировки реализована в вершинном и фрагментном шейдерах на языке WGSL [18]. В качестве генератора случайных чисел был выбран алгоритм xorshift32 [19] с временем отрисовываемого кадра в качестве источника энтропии. Это сделано из-за возможности эффективного выполнения алгоритма на GPU в силу параллелазции на уровне инструкций. На листинге 3.1 представлена функции генерации случайных чисел, и инициализация генератора случаных чисел.

Листинг 3.1 – Реализация генератора случайных чисел в шейдере

```
@group(0) @binding(1) var<uniform> random_seed: u32;
var<private> rng_state: u32;

fn xorshift32(state: u32) -> u32 {
    var x = state;
    x ^= x << 13u;
    x ^= x >> 17u;
    x ^= x << 5u;
    return x;
}

fn random_u32() -> u32 {
    let x = xorshift32(rng_state);
    rng_state = x;
    return x;
}

fn random_f32() -> f32 {
    let u = random_u32();
    return f32(u) * bitcast<f32>(0x2F800000u);
}
```

```
@fragment
fn fs_main(in: VertexOutput) -> @location(0) vec4f {
    rng_state = xorshift32(bitcast<u32>(in.uv.x * 123456789.0 +
       in.uv.y) ^ random_seed);
    ...
```

## 3.4 Вывод

В данном разделе были описаны детали реализации разработанной программы. Также был рассмотрен процесс взаимодействия пользователя с программой. Были приведены примеры работы.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ


1. *Shirley P.*, *Marschner S.* Fundamentals of Computer Graphics. — 3rd. — USA : A. K. Peters, Ltd., 2009. — ISBN 1568814690.
2. *Pharr M.*, *Jakob W.*, *Humphreys G.* Physically Based Rendering: From Theory to Implementation (3rd ed.) — 3rd. — San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 10.2016. — С. 1266. — ISBN 9780128006450.
3. *Foster I.* Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering. — 75 Arlington Street, Suite 300 Boston, MAUnited States : Addison-Wesley Longman Publishing Co., 01.1995. — ISBN 9780201575941.
4. *Fatahalian K.*, *Houston M.* A Closer Look at GPUs. — 2008.
5. Ray Tracing In Vulkan / D. Koch [и др.]. — 2020. — Дек.
6. *Rogers D. F.* Procedural Elements for Computer Graphics (2nd Ed.) — USA : McGraw-Hill, Inc., 1997. — ISBN 0070535485.
7. *Amanatides J.*, *Woo A.* A Fast Voxel Traversal Algorithm for Ray Tracing // Proceedings of EuroGraphics. — 1987. — Авг. — Т. 87.
8. *Foley T.*, *Sugerman J.* KD-Tree Acceleration Structures for a GPU Raytracer. — 2005.
9. *Laine S.*, *Karras T.* Efficient Sparse Voxel Octrees – Analysis, Extensions, and Implementation. — 2010.
10. *Phong B. T.* Illumination for Computer Generated Pictures. — 1975.
11. *Kajiya J. T.* The Rendering Equation // SIGGRAPH Comput. Graph. — New York, NY, USA, 1986. — Авг. — Т. 20, № 4. — С. 143—150. — ISSN 0097-8930. — DOI: 10.1145/15886.15902. — URL: https://doi.org/10.1145/15886.15902.
12. Microfacet Models for Refraction through Rough Surfaces / B. Walter [и др.]. — 2007.
13. *Overvoorde A.* Vulkan Tutorial. — URL: https://vulkan-tutorial.com/Introduction.

14. Accelerating Real-Time Shading with Reverse Reprojection Caching / D. Nehab [и др.]. — 2007. — Авг.

15. *W3C*. WebGPU working draft. — 07.2023.

16. *Klabnik S.*, *Nichols C.* The Rust Programming Language. — USA : No Starch Press, 2018. — ISBN 1593278284.

17. *Cornut O.* Dear ImGui. — URL: `https://github.com/ocornut/imgui`.

18. *W3C*. WebGPU Shading Language working draft. — 09.2023.

19. *Marsaglia G.* Xorshift RNGs // Journal of Statistical Software. — 2003. — Т. 8, № 14. — С. 1—6. — DOI: `10.18637/jss.v008.i14`. — URL: `https://www.jstatsoft.org/index.php/jss/article/view/v008i14`.