



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 и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

**«Создание модели детали на основе объектов с
использованием операций объединения и вычитания»**

Студент группы ИУ7-56Б

П. А. Калашков

(Подпись, дата)

(И.О. Фамилия)

Руководитель курсовой работы

Н. В. Новик

(Подпись, дата)

(И.О. Фамилия)

2022 г.

РЕФЕРАТ

Расчетно-пояснительная записка 48 с., 18 рис., 4 табл., 20 ист.

КОМПЬЮТЕРНАЯ ГРАФИКА, КОНСТРУКТИВНАЯ БЛОЧНАЯ ГЕОМЕТРИЯ, МАРШИРОВАНИЕ ЛУЧЕЙ, ТВЕРДОТЕЛЬНЫЕ МОДЕЛИ

Цель работы — разработка программы, позволяющей создавать твердотельные модели на основе примитивов и логических операций.

Методом представления моделия является конструктивная блочная геометрия (CSG), рендера — Ray Marching. Для преобразования модели используются матрицы преобразований, для придания им трёхмерного вида — шейдеры.

Результаты: разработка программа, предназначенная для создания моделей на основе объектов (куб, цилиндр, сфера и тор) с использованием операций пересечения, объединения и вычитания. Проанализированы различные алгоритмы, методы представления, преобразования и отображения модели, выбраны наиболее подходящие для поставленной задачи технологии, а также разработаны алгоритмы для их программной реализации.

Проведено исследование быстродействия программы при запуске на встроенной и дискретной видеокартах. Из результатов следует, что дискретная видеокарта позволяет получить большее количество кадров в секунду, является приемлемой для использования программы в режиме реального времени (более 30 кадров в секунду), в то время как встроенной видеокарты недостаточно для выполнения программы в режиме реального времени.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	9
1 Аналитическая часть	10
1.1 Анализ методов представления твердотельных моделей	10
1.1.1 Граничное представление	10
1.1.2 Нумерация пространственного заполнения (воксельный метод)	11
1.1.3 Октаантное дерево	13
1.1.4 Выметание (Sweeping)	14
1.1.5 Конструктивная блочная геометрия (CSG)	15
1.1.6 Сравнение методов	16
1.2 Анализ методов рендера модели	17
1.2.1 Растеризация	17
1.2.2 Трассировка лучей	19
1.2.3 Бросание лучей	21
1.2.4 Маркирование лучей	23
1.2.5 Сравнение методов	25
1.3 Методы преобразования и визуализации	26
1.3.1 Матрицы преобразования	26
1.3.2 Шейдеры	28
2 Конструкторская часть	30
2.1 Конструктивная блочная геометрия	30
2.2 Маркирование лучей	31
2.3 Схема приложения и диаграмма классов	32
3 Технологическая часть	35
3.1 Средства реализации	35
3.2 Реализация алгоритмов	35
4 Экспериментальная часть	40
4.1 Результаты разработки	40
4.1.1 Описание интерфейса	40
4.1.2 Демонстрация работы программы	41
4.2 Постановка эксперимента	42

4.3 Сравнение производительности	43
--	----

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

46

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В работе используются следующие термины с соответствующими определениями:

Ray Marching — алгоритм марширования лучей

ПО — программное обеспечение

CSG — Constructive Solid Geometry — конструктивная блочная геометрия

SDF — Signed Distance Field — поле расстояний со знаком

ВВЕДЕНИЕ

Моделирование твёрдых тел является неотъемлемой частью проектирования и разработки различных изделий, например, инженерных деталей. Все тела можно разделить на базовые и составные. К базовым относят примитивы: параллелепипед, цилиндр, шар, конус и др., в то время как составные можно сформировать в результате операций над базовыми (используя булевы функции сложения, пересечения и вычитания).

Целью работы: разработка программы, позволяющей создавать твердотельные модели на основе примитивов (куб, цилиндр, сфера, тор) с использованием логических операций (объединение, пересечение, вычитание). Для достижения поставленной цели необходимо выполнить следующие задачи:

- 1) изучить методы представления твёрдых тел;
- 2) изучить методы рендера смоделированного тела;
- 3) изучить методы преобразования и визуализации объектов;
- 4) разработать программу, соответствующую поставленной цели, с использованием выбранных методов представления, рендера, преобразования и визуализации объектов;
- 5) исследовать быстродействие программы при запуске на встроенной и дискретной видеокартах.

1 Аналитическая часть

В данном разделе проводится анализ и выбор методов представления, рендера, преобразования и визуализации твердотельных моделей.

1.1 Анализ методов представления твердотельных моделей

Моделирование твёрдого тела — это последовательный и непротиворечивый набор принципов математического и компьютерного моделирования трёхмерных твёрдых тел. Рассмотрим существующие методы представления твёрдых тел.

1.1.1 Границочное представление

В границочном представлении (*англ. Brep — Boundary Representation*) [1] твердотельные модели представляются как совокупность двумерных границ, которые описывают трёхмерную модель. Твёрдое тело описывается как замкнутая пространственная область, ограниченная набором элементарных поверхностей (граней), имеющих образующие контуры (ребра) на границе и признак внешней или внутренней стороны поверхности (пример представлен на рисунке 1).

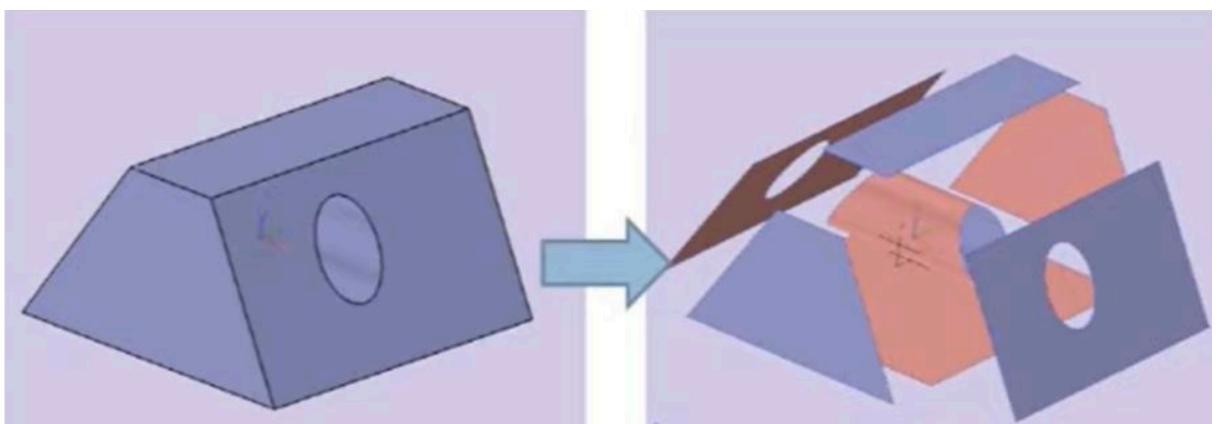


Рисунок 1 – Пример представления твёрдого тела как замкнутой области и набора поверхностей

Такая схема проектирования крайне распространена в приложениях САПР, но, несмотря на это, она также имеет ряд недостатков, которые могут привести к проблемам визуализации результата.

Недостатки:

- большие затраты памяти;
- проблемы получения описывающих формул в случае сложных объектов;
- потребность в большой вычислительной мощности в момент рендера.

Преимущества:

- метод подходит не только для твёрдых тел с плоскими гранями, но и для тел с криволинейными гранями или краями;
- благодаря хранению информации о всех составляющих модели, метод обеспечивает высокую точность;
- метод позволяет эффективно хранить информацию о свойствах материалов получаемого тела.

Таким образом, данный метод не подходит для решения поставленной задачи, так как он требует значительное количество памяти для хранения необходимой информации, а также вычислительных мощностей в момент рендера.

1.1.2 Нумерация пространственного заполнения (воксельный метод)

Данный метод получил своё название благодаря работе с пространственными ячейками (вокселями), который заполняют моделируемое тело [2]. Данные ячейки представляют собой кубы фиксированного размера и расположены в заданной пространственной сетке. Полученный в результате 3D объект является совокупностью закрашенных вокселей.

Каждая ячейка при этом должна быть представлена основной характеристикой — например, координатами центра. При сканировании обычно устанавливается определённый порядок обхода, а соответствующий упорядоченный набор координат называется пространственным массивом.

Такие пространственные массивы помогают однозначно определить модель, т.е. построенная таким образом модель может быть получена только одним способом. Пример изображения, полученного в результате использования воксельного метода, представлен на рисунке 2.

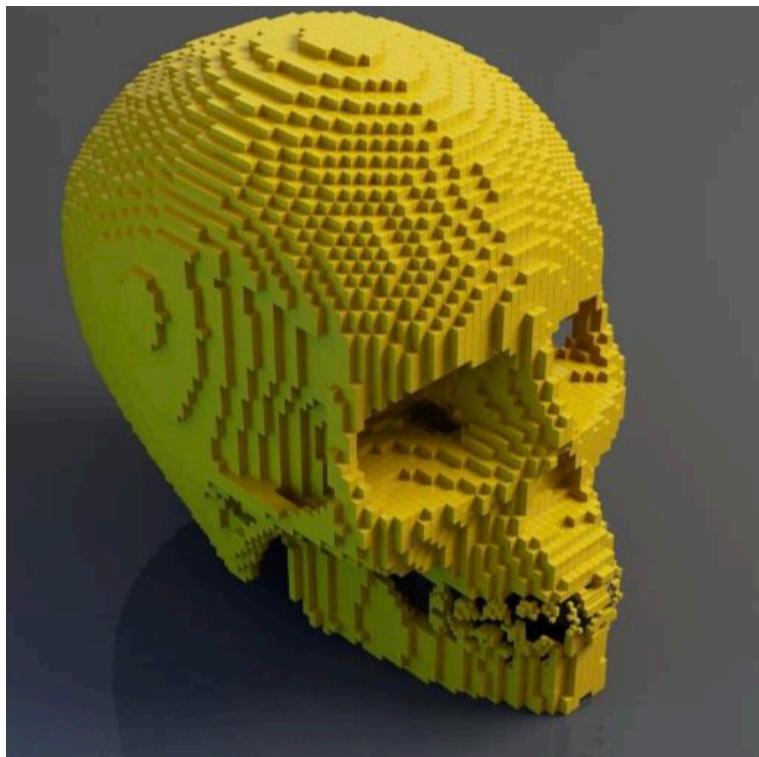


Рисунок 2 – Пример изображения, полученного в результате использования
воксельного метода

Недостатки:

- большие затраты памяти;
- каждая ячейка хранит информацию не только о своей координате, но и о цвете, плотности, оптических характеристиках и т. д.;
- разрешение итогового изображения зависит от размера и формы вокселей.

Преимущества:

- простота метода представления;
- однозначность представления.

Простота и однозначность построения способствуют использованию данного метода при построении представлений, однако структурное использование кубических вокселей приводит к тому, что для рендеринга, например, сферы, будет необходимо проводить сглаживание краёв, что является дополнительной вычислительной нагрузкой.

В нашем случае недостатком является и избыточная информация о каждой ячейке. Таким образом, самостоятельное использование данного метода является неэффективным, однако в совокупности с другими методами можно использовать преимущества аппроксимации для повышения качества изображения тела.

1.1.3 Октаантное дерево

Данная схема представления является улучшением воксельного метода. Строится октаантное дерево [2], каждый узел которого соответствует некоторому кубу в трёхмерном пространстве, и для каждого куба определяется его принадлежность модели. У каждого корня дерева есть 8 потомков, т.е. куб делится на 8 равных частей (пример подобного разделения представлен на рисунке 3).

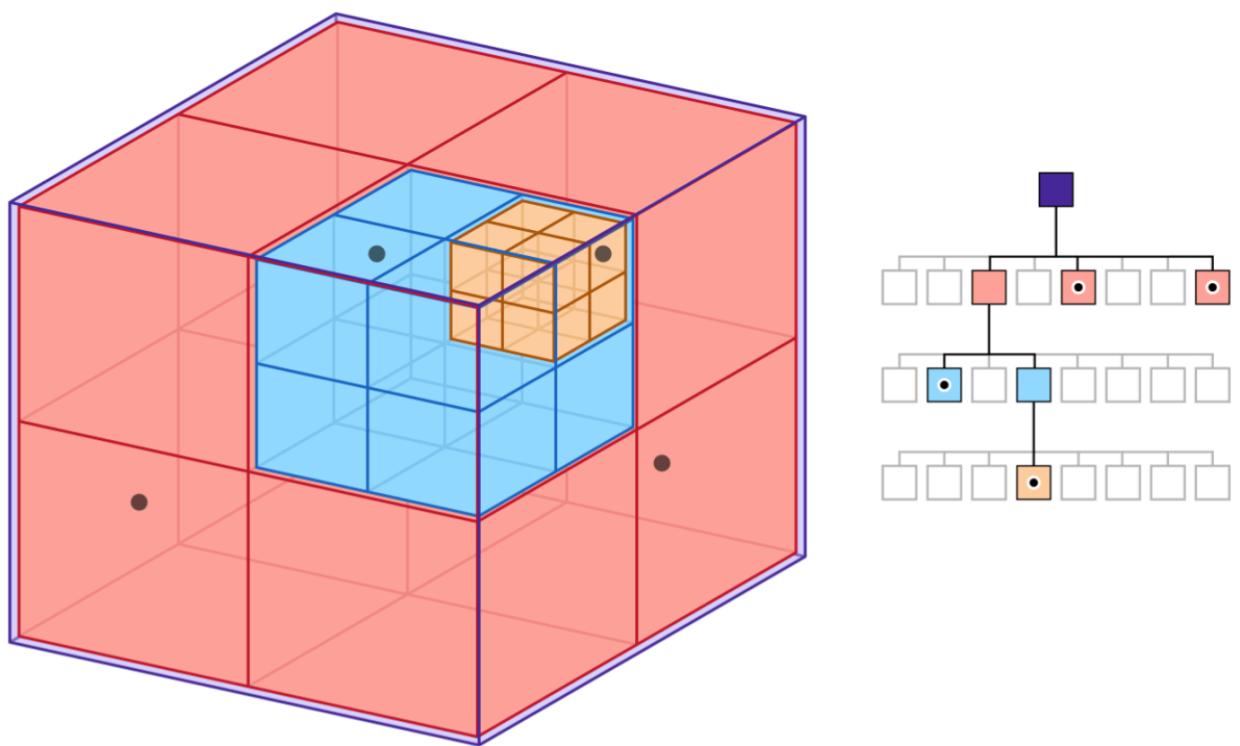


Рисунок 3 – Разделение куба на 8 равных частей и получение октаантного дерева

Метод позволяет устраниТЬ недостаток метода 1.1.2, связанного с затратами памяти из-за хранения большого количества данных. И хотя, сохраняя данные только об используемых частях модели, данный метод экономит память в

сравнении с воксельным, по отношению к другим методам, памяти расходуется много.

Недостатки:

- деление примитива рёбрами кубов дерева может снижать эффективность;
- возможное выведение на экран невидимых объектов.

Преимущества:

- простота метода представления;
- однозначность представления.

1.1.4 Выметание (Sweeping)

Данный метод [3] позволяет получить трёхмерную модель из двумерной посредством движения по заданной траектории (sweep), например, движения вокруг заданной оси или относительно грани (пример использования выметания для получения трёхмерной модели представлен на рисунке 4).

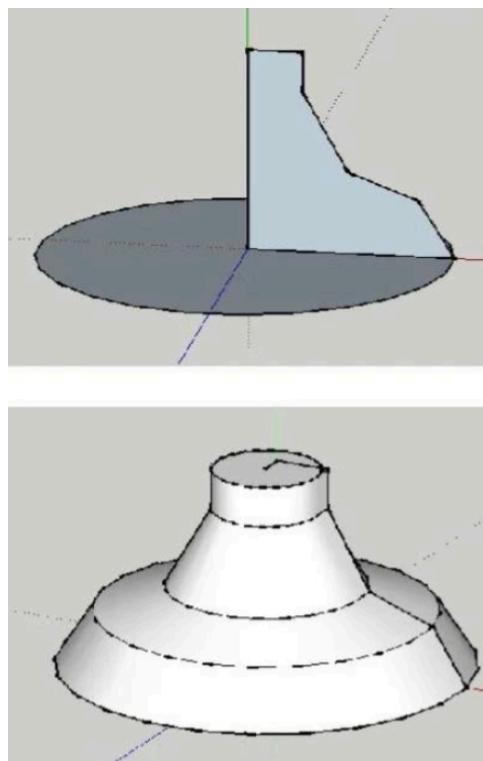


Рисунок 4 – Пример получения трёхмерного изображения посредством вращения фигуры вокруг заданной оси

Недостатком данного метода является необходимость задавать траектории движения 2D объектов, что может быть проблематично для тел, имеющих сложную форму.

К преимуществам выметания можно отнести удобство определения простых форм через простые плоские фигуры, а также возможность использования метода для быстрого удаления материала внутри тела (пример подобного использования выметания представлен на рисунке 5).

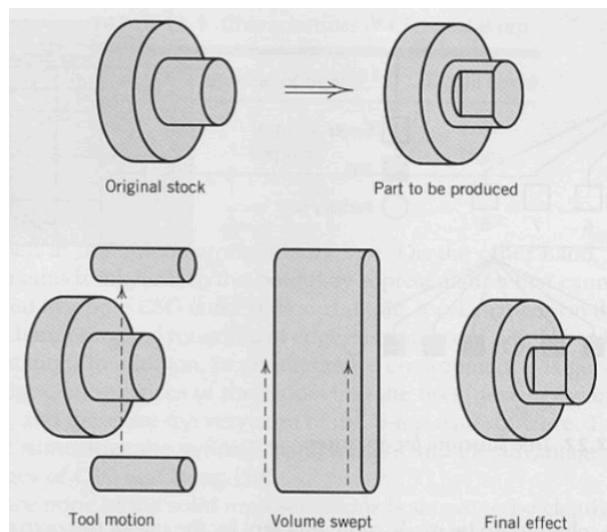


Рисунок 5 – Пример удаления материала внутри тела посредством алгоритма выметания

1.1.5 Конструктивная блочная геометрия (CSG)

Метод конструктивной блочной геометрии [4] основан на комбинировании примитивов посредством логических операций (объединения, пересечения, вычитания). Любое составное тело может быть описано в виде традиционного уравнения из булевых функций, аргументами которого могут быть как примитивы, так и другие составные тела. Такое представление ещё называют деревом построений (пример дерева построений представлен на рисунке 6).

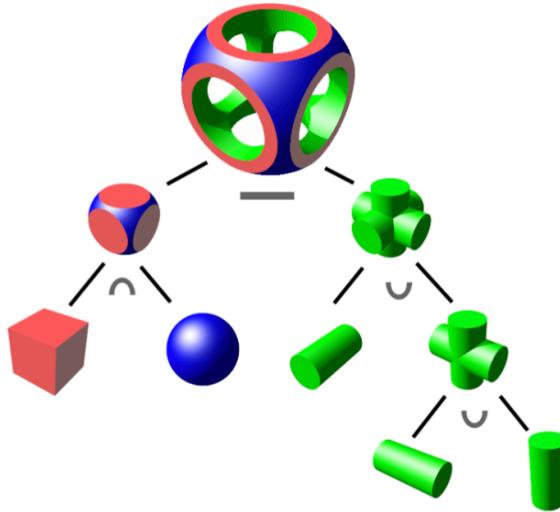


Рисунок 6 – Дерево построений при использовании CSG

Особенностью данного метода является то, что необходимо учитывать возможное вырождение тела в плоское в результате логических операций.

1.1.6 Сравнение методов

Сравним разобранные выше методы, разобрав их по некоторым критериям.

1. Эффективность — насколько много времени необходимо для создания, исследования и изменения формы модели.
2. Однозначность — вместе с моделью хранятся данные, которых достаточно для осуществления геометрических расчётов.
3. Практичность — создание модели без введения дополнительных параметров, поддерживая удобство использования.
4. Лаконичность — сколько памяти компьютера занимает модель.
5. Сохранность — хранение истории преобразования модели, чтобы у пользователя была возможность вернуться к предыдущему шагу моделирования без использования преобразовательных вычислений.
6. Уникальность — возможность получить модель единственным способом.

Разбор соответствующих критериев представлен в таблице 1.

Метод	Эф-ть	Од-ть	Прак-ть	Лак-ть	Сох-ть	Ун-ть
Brep	+	+	—	—	—	+
Нумерация пространственного заполнения	—	+	—	—	—	+
Октаантное дерево	—	+	—	+	—	+
Выметание	—	+	—	+	—	—
CSG	+	+	+	+	+	—

Таблица 1 – Сравнительная таблица методов представления

После рассмотрения методов представления предлагается использовать CSG, как наиболее подходящий метод представления моделей. Дерево построений является удобным с точки зрения модификации объекта и организации пользовательского интерфейса, обеспечивающего наглядный и быстрый доступ к любому элементу, входящему в описание геометрии тела. Остальные схемы требуют дополнительную информацию, которая, хоть и является необходимой для представления модели, не требуется для решения поставленной задачи.

Недостатками выбранного метода является отсутствие высокой точности представляющей модели и возможность получать одну и ту же модель несколькими способами, что не входит в требования, соответствующие поставленной задаче.

1.2 Анализ методов рендера модели

Рендеринг (англ. *rendering* — визуализация) — термин, обозначающий процесс получения изображения по модели при помощи компьютерной программы. Рассмотрим различные методы рендера.

1.2.1 Растеризация

Растеризацией [5] называют процесс получения растрового изображения — изображения, представляющего собой сетку пикселей — цветных точек (обыч-

но прямоугольных) на мониторе, бумаге или других отображающих устройствах.

Технология растеризации основана на обходе вершин треугольника, который после переноса из трёхмерного пространства в двумерное остаётся самим собой. Каждая точка каждого обрабатываемого объекта в трёхмерном пространстве переводится в точку на экране, затем эти точки соединяются и получается изображение объекта. На рисунке 7 представлен пример переноса треугольника из трёхмерного пространства в двумерное.

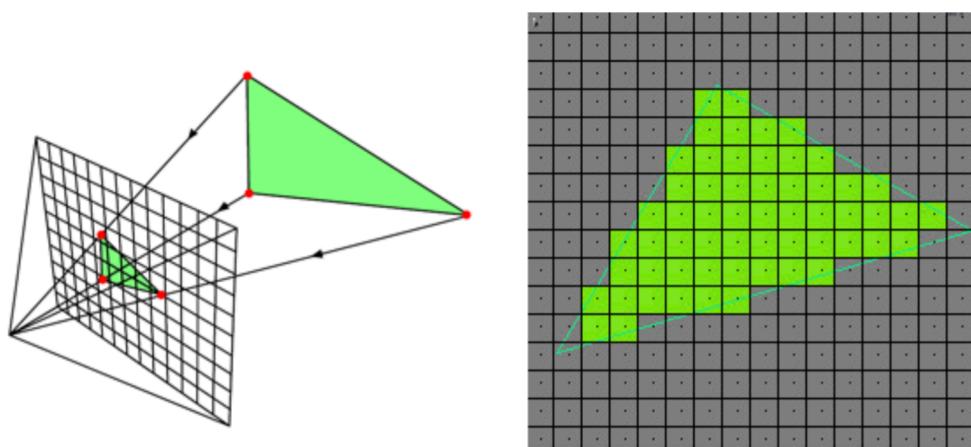


Рисунок 7 – Пример переноса треугольника из трёхмерного пространства в двумерное

Недостатки:

- изображение может получиться «угловатым» (ступенчатым), что требует использования алгоритмов сглаживания;
- алгоритм требует больших вычислительных ресурсов — на дисплее 4К примерно 8 миллионов пикселей, могут использоваться миллионы полигонов, частота обновления кадров может быть высокой;
- каждый пиксель обрабатывается много раз;
- составные модели будут ресурсозатратно обрабатывать, т. к. 3D модель должна быть описана как набор примитивов.

Преимущества:

- сравнительная простота алгоритма;
- современные компьютеры оптимизированы для отрисовки растровых изображений, что позволяет делать это довольно эффективно.

1.2.2 Трассировка лучей

Трассировка лучей (англ. *Ray Tracing*) [6] — технология отрисовки трёхмерной графики, симулирующая физическое поведение света.

Одним из принципов трассировки лучей является поддержка совместимости с существующими трёхмерными моделями, большинство которых собрано из треугольников. Для этого проверяется случай столкновения луча не со стеной, а с треугольником. Рассмотрим этот случай.

Пусть вершины треугольника обозначаются как V_0, V_1, V_2 . Векторы двух его рёбер будут обозначены как \vec{a}, \vec{b} и заданы формулами 1 и 2

$$\vec{a} = V_1 - V_0 \quad (1)$$

$$\vec{b} = V_2 - V_0 \quad (2)$$

Определим луч P с помощью параметрической формулы 3:

$$P = R_0 + t \cdot R_d \quad (3)$$

где R_0 — начальная точка луча, R_d — направление луча, t — расстояние вдоль луча до точки.

В плоскости треугольника точка пересечения будет иметь координаты (u, v) . Приравняв уравнение луча P и плоскости в точке (u, v) , можно найти пересечение:

$$P = V_0 + u \cdot \vec{a} + v \cdot \vec{b} \quad (4)$$

Таким образом, составив систему из трёх уравнений 5 для координат x, y, z и решив её для t, u, v , необходимо проанализировать определитель. Если он ненулевой (луч не параллелен плоскости), а $t \geq 0$ и $u, v, u + v$ лежат в

диапазоне от 0 до 1, то P находится внутри треугольника и поиск столкновения завершается. Система 5:

$$\begin{cases} R_{0x} + t \cdot R_{dx} = V_{0x} + u \cdot A_x + v \cdot B_x \\ R_{0y} + t \cdot R_{dy} = V_{0y} + u \cdot A_y + v \cdot B_y \\ R_{0z} + t \cdot R_{dz} = V_{0z} + u \cdot A_z + v \cdot B_z \end{cases} \quad (5)$$

Стоит учесть, что вычисления, связанные с конкретным лучом, могут не закончиться после одного треугольника: например, в случае отражения или преломления необходимо проследить дальнейшее поведение луча до тех пор, пока не произойдёт поглощение луча, возврат в начальную точку или превышение максимального числа отражений.

Несмотря на это, сложность алгоритма трассировки лучей позволяет получить более качественное изображение по сравнению с результатом, который может дать растеризация, за счёт отслеживания траектории лучей после отражения от объектов, а также учёта общей освещённости сцены (пример изображений, полученных в результате растеризации и трассировки лучей, представлен на рисунке 8).

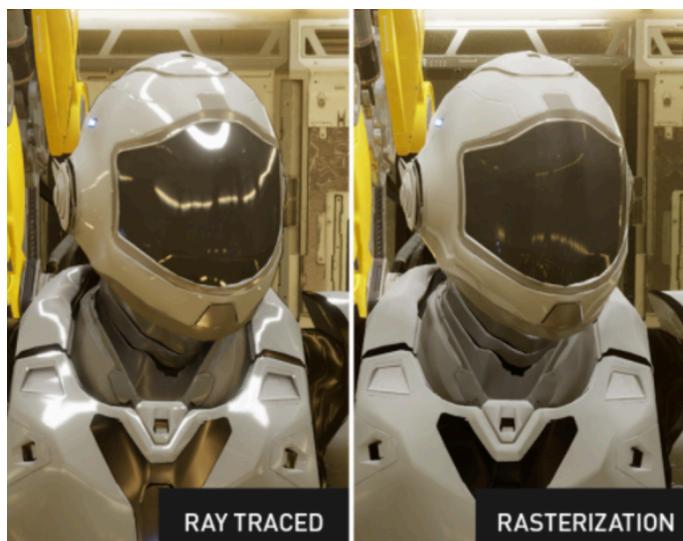


Рисунок 8 – Сравнение результатов работы алгоритмов растеризации и трассировки лучей

Недостатком данного метода является производительность (требуется большая вычислительная способность).

Преимущества:

- возможность получать гладкие изображения без использования дополнительных алгоритмов аппроксимации;
- вычислительная сложность практически не зависит от сложности сцены;
- отсечение невидимых поверхностей, изменение поля зрения и перспективы являются частью алгоритма.

Несмотря на то, что данный алгоритм позволяет получать изображения хорошего качества, его использование требует больших мощностей. Метод трасировки лучей каждый раз начинает процесс определения цвета пикселя заново, рассматривая каждый луч наблюдения отдельно. Это позволяет придать изображению реалистичность и решает задачи отражения и преломления, однако требует много ресурсов.

1.2.3 Бросание лучей

Бросание лучей (англ. — *Ray Casting*) [7] — это технология, позволяющая преобразовать набор данных в 3D проекцию посредством «бросания лучей» из точки обзора по всем областям видимости.

Идея алгоритма заключается в том, чтобы испускать лучи из камеры (глаз наблюдателя), по одному лучу на пиксель, после чего находить ближайший объект, который блокирует путь распространения данного луча. Используя информацию о характеристиках материалов, алгоритм бросания лучей также может определять затенение объектов.

Данная технология была достаточно распространена при создании игр в конце прошлого века, а из-за возможности преобразования чего-то двумерного в что-то почти трёхмерное графические изображения, полученные в результате бросания лучей, часто называли «псевдо 3D» или «2.5D». Пример работы алгоритма бросания лучей приведён на рисунке 9.

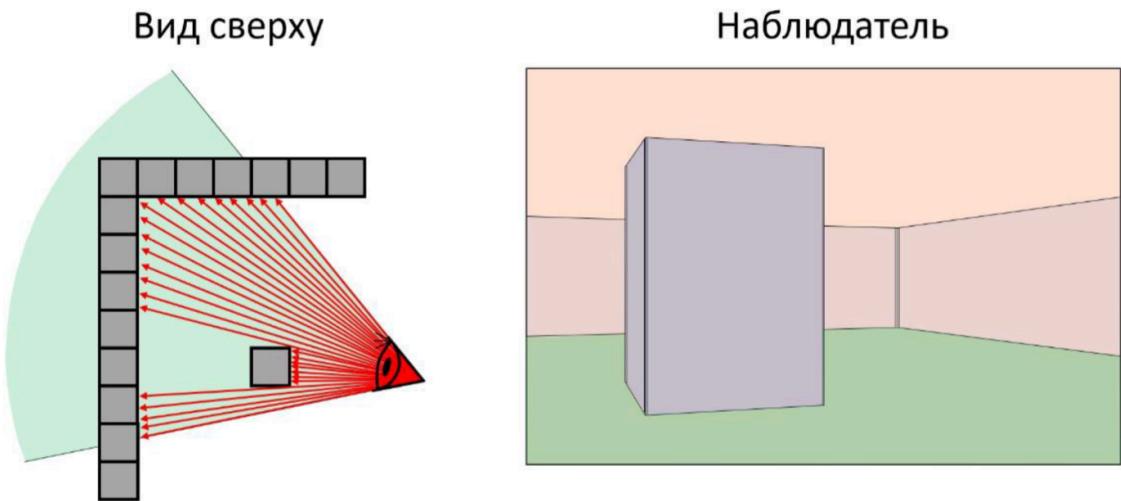


Рисунок 9 – Пример работы алгоритма бросания лучей

Недостатки:

- в результате рендеринга получается изображение средне-плохого качества по сравнению с другими методами;
- есть геометрические ограничения, накладываемые на обрабатываемую поверхность (только простые фигуры).

Преимущества:

- простота реализации алгоритма;
- низкие требования к вычислениям;
- возможность получать изображение с течением времени («на лету»).

Из-за указанных геометрических ограничений у игр, созданных с использованием технологии бросания лучей, было множество недостатков: потолок был одной высоты, не было никаких трёхмерных объектов, кроме потолка, стен и пола, всё остальное — двумерные изображения в трёхмерном пространстве (пример подобной игры представлен на рисунке 10).



Рисунок 10 – Скриншот из игры Doom 1: Объекты (оружие и противники) — просто растровые изображения, перенесённые и отрисованные поверх фона

1.2.4 Марширивание лучей

Марширивание лучей (англ. *Ray Marching*) [8] — разновидность алгоритмов трассировки лучей, является изменённой версией технологии Ray Tracing.

Ray Marching похож на традиционную технологию трассировки лучей тем, что лучи в сцену испускаются для каждого пикселя. Однако в то время, как в трассировке лучей имеются системы уравнений, позволяющие получить точку пересечения луча и объектов, в технологии марширивания лучей предлагается другое решение.

Происходит смещение текущего положения вдоль луча до тех пор, пока не будет найдена точка, пересекающая объект. Данная операция является простой по сравнению с решением системы уравнений, однако находит точку пересечения не слишком точно. На рисунке 11 продемонстрирован пример простейшей реализации марширивания лучей с фиксированным интервалом шага (красными точками отмечены все обрабатываемые точки).

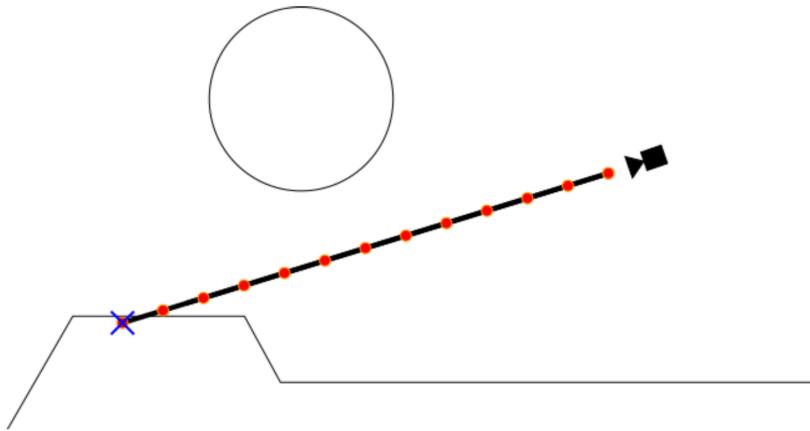


Рисунок 11 – Простейшая реализация метода Ray Marching с фиксированным интервалом шага

Данная реализация вполне достаточна для использования во множестве областей, например, в случаях построения объёмных и прозрачных поверхностей. Для непрозрачных объектов можно ввести ещё одну оптимизацию, для которой будет использовано поле расстояний со знаком.

Поле расстояний со знаком (англ. *Signed Distance Field*) — это функция, получающая на входе координаты точки и возвращающая кратчайшее расстояние от этой точки до поверхности каждого объекта в сцене. При этом возвращается отрицательное число, если точка находится внутри объекта. Таким образом, мы можем ограничить количество шагов при движении вдоль луча.

Описание алгоритма: для каждого пикселя на экране определяется расстояние до ближайшего объекта, которое позволяет получить радиус, на который можно пустить луч. Для конечной точки луча определяем радиус до тех пор, пока он не станет достаточно маленьким — это будет означать столкновение с объектом. Если же радиус стабильно увеличивается, то это показывает, что луч прошёл мимо объектов на сцене. На рисунке 12 продемонстрирована визуализация метода марширования лучей с использованием SDF. Красным цветом отвачены обрабатываемые точки, синим цветом — области, которые гарантированно не содержат объектов, пунктирными зелёными линиями — истинные кратчайшие векторы между каждой обрабатываемой точкой и сценой.

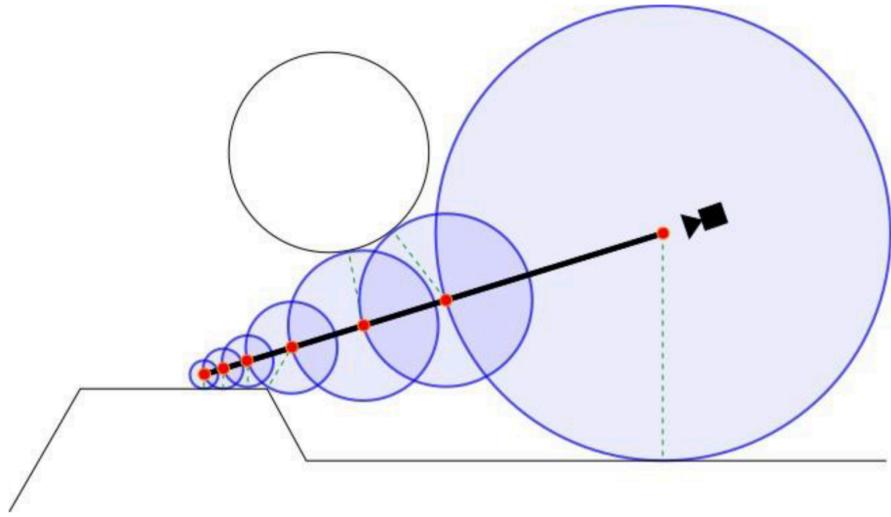


Рисунок 12 – Визуализация метода RayMarching с использованием поля расстояний со знаком

Недостатком данного алгоритма является неточность найденного значения координат пересечения луча и объекта.

Преимущества:

- неплохая производительность по сравнению с технологией трассировки лучей;
- метод подходит для рендеринга сложных поверхностей, для которых сложно определить пересечение аналитическими методами;
- используя поля расстояний со знаком можно произвести ускорение рендеринга до реального времени;
- хорошее качество изображения (сопоставимое с результатами алгоритма трассировки лучей).

1.2.5 Сравнение методов

Сравним разобранные выше методы, разобрав их по некоторым критериям.

1. Быстродействие, т.е. как много времени нужно для рендеринга модели.
2. Точность, т.е. обработка истинных границ рассматриваемой модели.
3. Качество получаемого изображения.
4. Реальность — возможность работы в режиме реального времени.

Разбор соответствующих критериев представлен в таблице 2

Метод	Быстродействие	Точность	Качество	Реальность
Растеризация	+	+	-	+
Ray Tracing	-	+	+	-
Ray Casting	+	+	-	+
Ray Marching	+	-	+	+

Таблица 2 – Сравнительная таблица методов рендера

Таким образом, были рассмотрены методы рендера модели. Учитывая поставленную задачу, а также метод представления модели, выбранный в предыдущем разделе, можно сказать, что алгоритм марширования лучей (Ray Marching) является оптимальным при использовании конструктивной блочной геометрии (CSG). В такой связке можно получать качественное изображение при отрисовке в режиме реального времени. Есть и недостаток такого выбора: будет присутствовать некоторая погрешность при вычислении границ объекта.

1.3 Методы преобразования и визуализации

В данном разделе рассматриваются способы преобразования модели и придания ей реалистичного вида.

1.3.1 Матрицы преобразования

Для преобразования тела в пространстве обычно используются операции перемещения, поворота и масштабирования. Осуществление этих преобразований может быть реализовано при помощи матриц преобразования [9] (таблица 3).

Таблица 3 – Таблица преобразований

Преобразование	Матрица	Формула
Перемещение	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ dx & dy & dz & 1 \end{pmatrix}$	$\begin{cases} x_1 = x + dx \\ y_1 = y + dy \\ z_1 = z + dz \end{cases}$
Масштаб - е	$\begin{pmatrix} k_x & 0 & 0 & 0 \\ 0 & k_y & 0 & 0 \\ 0 & 0 & k_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{cases} x_1 = k_x + (1 - k_x)x_m \\ y_1 = k_y + (1 - k_y)y_m \\ z_1 = k_z + (1 - k_z)z_m \end{cases}$
Поворот OX	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & \sin \theta & 0 \\ 0 & -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{cases} x_1 = x \\ y_1 = y_c + (y - y_c) \cos \theta - (z - z_c) \sin \theta \\ z_1 = z_c + (y - y_c) \sin \theta + (z - z_c) \cos \theta \end{cases}$
Поворот OY	$\begin{pmatrix} \cos \theta & 0 & -\sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{cases} x_1 = x_c + (x - x_c) \cos \theta + (z - z_c) \sin \theta \\ y_1 = y \\ z_1 = z_c - (x - x_c) \sin \theta + (z - z_c) \cos \theta \end{cases}$
Поворот OZ	$\begin{pmatrix} \cos \theta & \sin \theta & 0 & 0 \\ -\sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{cases} x_1 = x_c + (x - x_c) \cos \theta - (y - y_c) \sin \theta \\ y_1 = y_c + (x - x_c) \sin \theta + (y - y_c) \cos \theta \\ z_1 = z \end{cases}$

Происходит умножение исходных координат объекта на одну или несколько матриц, в результате чего происходит перемещение, поворот или масштабирование объекта в зависимости от применённых преобразований. Матрицы позволяют перевести координаты модели в мировые координаты, настроить перспективу, а также переместить сцену относительно камеры.

1.3.2 Шейдеры

Шейдером называют компьютерную программу, предназначенную для исполнения процессорами видеокарты (GPU).

В разрабатываемом приложении использование шейдеров необходимо для придания изображению трёхмерного вида (используя тени и освещение). Для этого существуют два вида шейдеров: фрагментные и вершинные [10].

Фрагментные шейдеры работают с конкретными пикселями объекта и управляют их цветом. Работа фрагментного шейдера в графической обработке происходит после выполнения работы вершинным шейдером и влияет только на цветовую составляющую, преобразуя вершины в пиксели определённого цвета.

Вершинные шейдеры работают с вершинами (из списка вершин) и отображают их в пространстве. Шейдеру передаются три матрицы: матрица модели, матрица вида и матрица проекции. Матрица модели необходима для перехода от пространства модели, в котором координаты вершин определены относительно центра модели, в мировое пространство (в котором все вершины определены относительно центра мира).

Матрица вида используется для перемещения сцены относительно камеры. В реальном мире происходит перемещение камеры относительно сцены, однако в компьютерной графике проще и удобнее переместить сцену относительно камеры.

Матрица проекции позволяет представить перспективу камеры. При умножении координат на данную матрицу происходит сопоставление вершины с перспективой камеры, её соотношению сторон и полю обзора.

Итак, в данном разделе были рассмотрены методы преобразования модели — матрицы преобразования, а также методы преобразования плоского изображения в трёхмерный вид — шейдеры (вершинные и фрагментные).

Вывод

В данной главе был проведён анализ возможных методов для решения поставленной задачи. Как метод представления модели, при помощи которой будет решаться поставленная задача, была выбрана конструктивная блочная геометрия (CSG), рендера — Ray Marching. Для преобразования модели будут использоваться матрицы преобразований, для придания им трёхмерного вида — шейдеры.

2 Конструкторская часть

В данном разделе рассматриваются реализуемые алгоритмы и методы, приводится схема приложения и диаграмма классов.

2.1 Конструктивная блочная геометрия

Рассмотрим функцию поля расстояния со знаком (SDF). Она может быть использована для получения кратчайшего расстояния от точки до поверхности тела, причём если точка находится внутри тела, то на это должен указывать отрицательный знак результата. Рассмотрим функцию поля расстояния со знаком для сферы и куба.

Чтобы вычислить SDF сферы, нужно знать длину радиуса сферы и расстояние от центра сферы до заданной точки (см. формулу 6):

$$L(x, y, z) = \sqrt{(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2} \quad (6)$$

где (x_0, y_0, z_0) — координаты центра сферы, $p(x, y, z)$ — рассматриваемая точка.

Получим SDF сферы:

$$dist = L(x, y, z) - r \quad (7)$$

В случае куба необходимо учесть случаи расположения точки относительно грани.

$$q = |p| - r \quad (8)$$

$$dist = L(max(q, 0)) + min(max(q_x, max(q_y, q_z)), 0) \quad (9)$$

где $p(x, y, z)$ — рассматриваемая точка, $q(x, y, z)$ — координаты точки за вычетом радиуса, $dist$ — искомое расстояние с учётом двух расположений точки относительно граней и угла.

Функция поля расстояния со знаком позволяет эффективно использовать алгоритм маркирования лучей, поскольку позволяет двигаться к рассматриваемому объекту с максимальным шагом так, что не придётся пропускать лицевую грань объекта.

Стоит отметить, что SDF можно использовать в случае с CSG для выполнения композиции объектов.

При пересечении двух тел луч должен пересечься с тем телом, которое дальше от камеры, следовательно, необходимо выбрать SDF с максимальным значением:

$$\text{intersection}(\text{sdf1}, \text{sdf2}) = \max(\text{sdf1}, \text{sdf2}) \quad (10)$$

При объединении двух тел луч должен пересечься с ближайшим телом, поэтому нужно выбрать SDF с минимальным значение:

$$\text{union}(\text{sdf1}, \text{sdf2}) = \min(\text{sdf1}, \text{sdf2}) \quad (11)$$

Поскольку в CSG используется операция разности, то имеет значение и порядок параметров (поскольку операция разности некоммутативна). В случае операции разности необходимо определить порядок operandов и найти минимальное расстояние между первым и вторым телом. Расстояние до второго тела возьмётся с отрицательным знаком. Будем считать, что луч должен пересечься с первым телом и не пересечься со вторым телом. В таком случае второе тело можно представить как всё пространство за пределами второго тела посредством операции инвертирования:

$$\text{invert}(\text{sdf}, x, y, z) = -\text{sdf}(x, y, z) \quad (12)$$

После этого можно получить результат разности, применив пересечение первого тела и инвертированного второго:

$$\begin{aligned} \text{diff}(\text{sdf1}, \text{sdf2}, x, y, z) &= \text{union}(\text{sdf1}(x, y, z), \text{invert}(\text{sdf2}(x, y, z))) \\ &= \min(\text{sdf1}(x, y, z), -\text{sdf2}(x, y, z)) \end{aligned} \quad (13)$$

2.2 Марширование лучей

Алгоритм бросания лучей отрисовывает объекты, полагаясь на SDF (см. предыдущий пункт). Маршировка лучей состоит в итерационном перемеще-

нии точки вдоль луча обзора и проверки результата: в случае отрицательного результата произошло столкновение с объектом.

Также алгоритм марсирования лучей должен использовать границы сцены, которые должны быть определены и переданы в качестве входных данных.

Алгоритм марсирования лучей представлен на псевдокоде 1.

Псевдокод 1 Алгоритм марсировки лучей

```
1: Функция rayMarch(start, end)
2:   depth  $\leftarrow$  start
3:   i  $\leftarrow$  0
4:   До тех пока i  $<$  MAX_STEPS выполнять
5:     dist  $\leftarrow$  расстояние до объекта
6:     Если внутри объекта тогда
7:       Возвратить depth
8:     Конец условия
9:     depth  $+ =$  dist
10:    i  $+ =$  1
11:    Если луч вышел за пределы сцены тогда
12:      Возвратить end
13:    Конец условия
14:  Конец цикла
15:  Возвратить end
16: Конец функции
```

2.3 Схема приложения и диаграмма классов

Приложение строится из частей, приведённых на рисунке 13.



Рисунок 13 – Схема приложения

На данной схеме представлена функциональная структура приложения. Используя графический интерфейс, пользователь сможет задать примитивы, положение камеры, а также используемые операции композиции. Этап моделирования объектов представлен на рисунке 14. После него следует рендер итогового изображения с использованием камеры и освещения, полученных благодаря вершинному и фрагментному шейдерам.

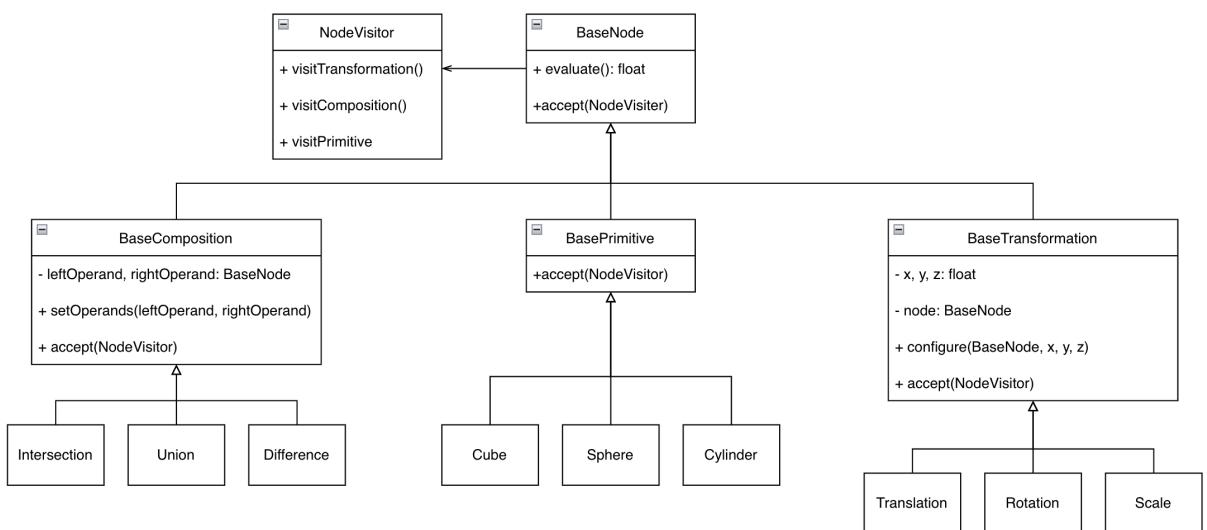


Рисунок 14 – Диаграмма классов этапа моделирования объектов

Поскольку в технологии конструктивной блочной геометрии используется дерево построений, то в приложении данное дерево представлено как двоичное дерево, элементами которого являются объекты класса *BaseNode*, которыми могут быть элементы трансформации, композиции или примитивы — соответственно, классы *BaseTransformation*, *BaseComposition* и *BasePrimitive*.

Класс *BaseTransformation* используется для преобразования объекта модели и содержит в себе необходимые для выполнения операции (перемещения, поворота или масштабирования) координаты. У него есть метод *configure*, который производит настройку экземпляра класса в соответствии с переданным объектом.

Класс *BaseComposition* используется для выполнения операций композиции (пересечения, объединения и вычитания). Он представляет собой дерево (построений), результатом обхода которого является итоговая модель.

Класс *BasePrimitive* предназначен для создания объектов модели, построение и обработка которых будет осуществляться.

Также имеется класс *NodeVisitor*, реализующий паттерн поведения «Посетитель», который необходим для добавления нового функционала классам. Применение данного паттерна помогает абстрагироваться от реализации класса *BaseNode* и предоставить обход узлов дерева посетителю, который будет обходить узлы при помощи методов *visitTransformation*, *visitComposition* и *visitPrimitive*.

Вывод

В данном разделе были рассмотрены методы и алгоритмы, необходимые для понимания и создания приложения, а также приведены схема приложения и диаграмма классов для этапа моделирования объектов.

3 Технологическая часть

В данном разделе рассмотрены средства разработки программного обеспечения, приведены детали реализации и листинги исходных кодов программы, а также приведён пример результата работы программы.

3.1 Средства реализации

Как основное средство реализации и разработки ПО был выбран язык программирования JavaScript [11]. Причиной выбора данного языка является тот факт, что он представляет из себя браузерный язык программирования и позволяет запускать приложение в браузере, что делает его также кроссплатформенным приложением и помогает избавиться от лишних зависимостей. Также была использована библиотека ThreeJS [12] для языка JavaScript, которая предоставляет холст (англ. *canvas*) для отрисовки сцены, а также позволяет подключить написанные шейдеры. Также был подключён модуль Stats [13], который позволяет отслеживать количество кадров в секунду (FPS), что обеспечивает возможность оценки производительности получившегося ПО. Для создания пользовательского интерфейса был использован модуль dat-gui [14], который поможет пользователю самостоятельно изменять параметры моделей. Функциональное тестирование ПО проводиться не будет по причине специфиности приложения — оно является GUI-приложением, что усложняет процесс тестирования. Средой разработки послужил графический редактор Visual Studio Code [15], который известен содержанием большого количества плагинов, ускоряющих процесс разработки программы. Для запуска приложения используется *Python* [16] сервер, позволяющий запускать программу в браузере.

3.2 Реализация алгоритмов

В расположенных ниже листингах 3.1 – 3.3 приведён исходный код реализации алгоритмов, выбранных в аналитическом разделе. Была проведена декомпозиция алгоритма отрисовки модели на подпрограммы: функции получения расстояния до каждого объекта, пускание луча, получение расстояния до

композиции объектов.

Листинг 1 – Реализация операций композиции

```
1  /*
2   * Логическое пересечение
3   * distA, distB: расстояние от текущей точки
4   * до объектов. Возвращает: расстояние до объекта, полученного
5   * в результате логического пересечения объектов
6   */
7   float intersect(float distA, float distB) {
8       return max(distA, distB);
9   }
10
11 /*
12  * Логическое объединение
13  * distA, distB: расстояние от текущей точки
14  * до объектов. Возвращает: расстояние до объекта, полученного
15  * в результате логического объединения объектов
16  */
17  float union(float distA, float distB) {
18      return min(distA, distB);
19  }
20
21 /*
22  * Логическое вычитание
23  * distA, distB: расстояние от текущей точки
24  * до объектов. Возвращает: расстояние до объекта, полученного
25  * в результате логического вычитания объектов
26  */
27  float difference(float distA, float distB) {
28      return max(distA, -distB);
29 }
```

Листинг 2 – Реализация операций преобразования

```
1  /*
2   * Операция переноса
3   */
4  vec3 translate(vec3 p, vec3 v) {
5      return p - v;
6  }
7
8  /*
9   * Операция поворота
10  */
11 vec3 rotate(vec3 p, vec3 rad) {
12     float x = rad.x;
13     float y = rad.y;
14     float z = rad.z;
15     mat3 m = mat3(
16         cos(y)*cos(z),
17         sin(x)*sin(y)*cos(z) - cos(x)*sin(z),
18         cos(x)*sin(y)*cos(z) + sin(x)*sin(z),
19
20         cos(y)*sin(z),
21         sin(x)*sin(y)*sin(z) + cos(x)*cos(z),
22         cos(x)*sin(y)*sin(z) - sin(x)*cos(z),
23
24         -sin(y),
25         sin(x)*cos(y),
26         cos(x)*cos(y)
27     );
28     return m * p;
29 }
```

Листинг 3 – Реализация шейдерных алгоритмов маркирования луча

```
1  float getDistance(vec3 rayOrigin, vec3 rayDirection,
2      out vec3 rayPosition, out vec3 normal, out bool hit) {
3
4      float dist;
5
6      float depth = 0.0;
7
8      rayPosition = rayOrigin;
9
10     for (int i = 0; i < 64; i++) {
11
12         dist = sceneDist(rayPosition);
13
14         if (abs(dist) < EPS) {
15
16             hit = true;
17
18             break;
19
20         }
21
22         depth += dist;
23
24         rayPosition = rayOrigin + depth * rayDirection;
25
26     }
27
28     return depth;
29
30 }
```



```
17
18
19     float distance(vec3 p) {
20
21         float cube = cubeDist(rotate(translate(p, cubePosition),
22             cubeRotation),
23             vec3(cubeScale * 2., cubeScale * 2., cubeScale * 2.));
24
25         float cylinder = cylinderDist(rotate(translate(p,
26             cylinderPosition), cylinderRotation), cylinderScale * 0.5,
27             cylinderScale * 4.0);
28
29         float sphere = sphereDist(translate(p, spherePosition),
30             sphereScale * 1.);
31
32         float torus = torusDist(rotate(translate(p, torusPosition),
33             torusRotation), vec2(1.0 * torusScale, 0.25 * torusScale));
34
35         return union(cube, difference(sphere, cylinder));
36
37 }
```

Вывод

В данном разделе были рассмотрены средства реализации ПО, приведены листинги исходного кода программы, разработанной на основе алгоритмов, выбранных в аналитическом разделе и изложенных в конструкторском разделе.

4 Экспериментальная часть

В данном разделе будут приведены примеры работы разработанной программы и поставлен эксперимент по сравнению производительности в зависимости от сложности моделируемой сцены. Сложность оценивается количеством используемых на сцене поверхностей.

4.1 Результаты разработки

4.1.1 Описание интерфейса

В верхнем правом углу программы располагается панель управления, позволяющая контролировать созданные объекты, например, вращать их, масштабировать или перемещать (пример такой панели для куба, сферы и цилиндра приведён на рисунке 15).



Рисунок 15 – Пример панели управления для куба, сферы и цилиндра

В нижнем правом углу располагаются кнопки, позволяющие пользователю перейти к созданию нового объекта или к удалению последнего созданного (рисунок 16).

Разработанная программа позволяет пользователю создавать твердотельные модели на основе примитивов (куб, цилиндр, сфера и тор) и использовать логические операции (объединение, пересечение, вычитание). Присутствует воз-

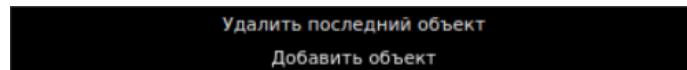


Рисунок 16 – Кнопки добавления нового объекта и удаления последнего созданного

можность добавлять на сцену объекты, удалять их и преобразовывать, а также менять позицию камеры.

4.1.2 Демонстрация работы программы

На рисунке 17 продемонстрирована работа программы на примере вычитания куба из сферы и объединения с цилиндром.

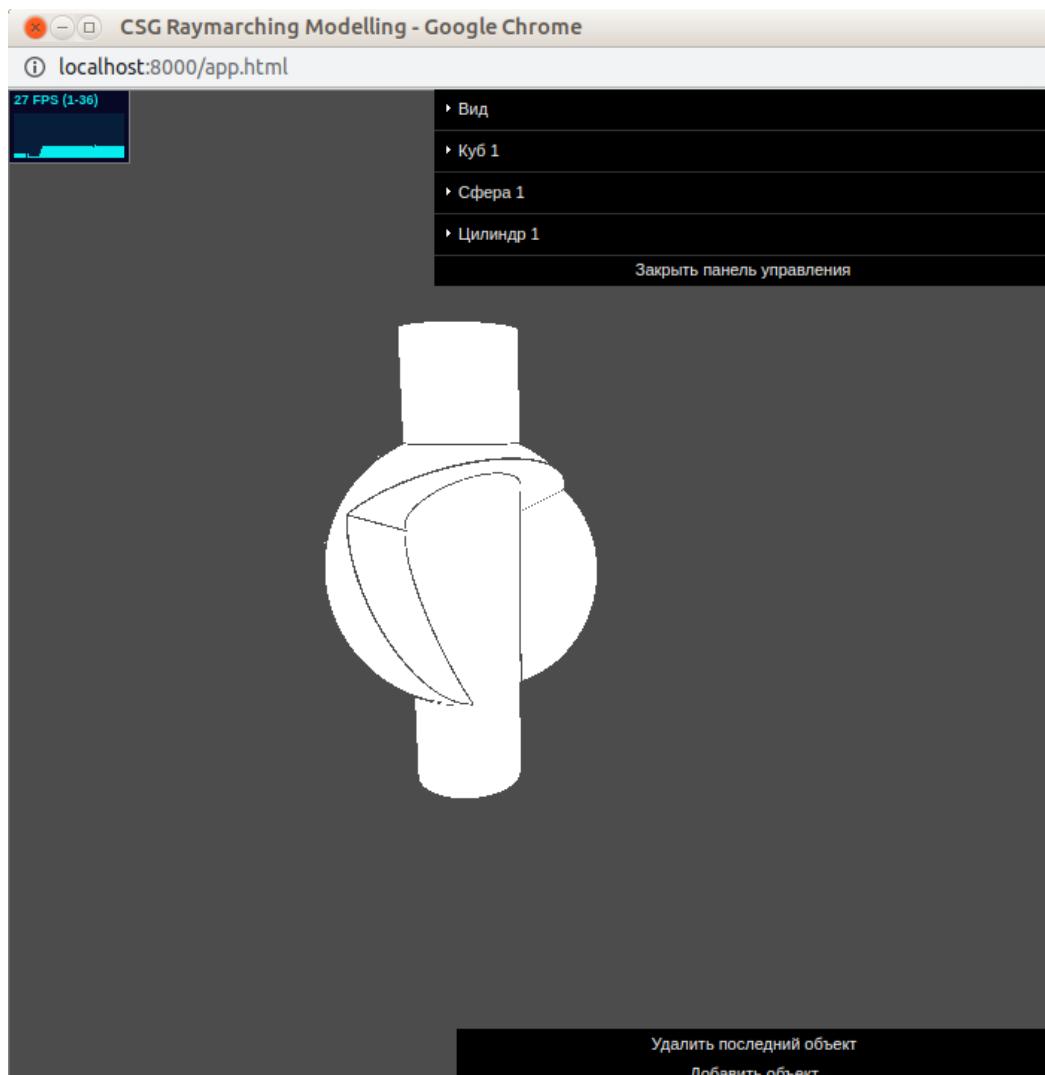


Рисунок 17 – Демонстрация работы программы (композиция моделей куба, сферы и цилиндра)

На рисунке 18 продемонстрирована работа программы на примере пересечения куба с тором и объединения с цилиндром.

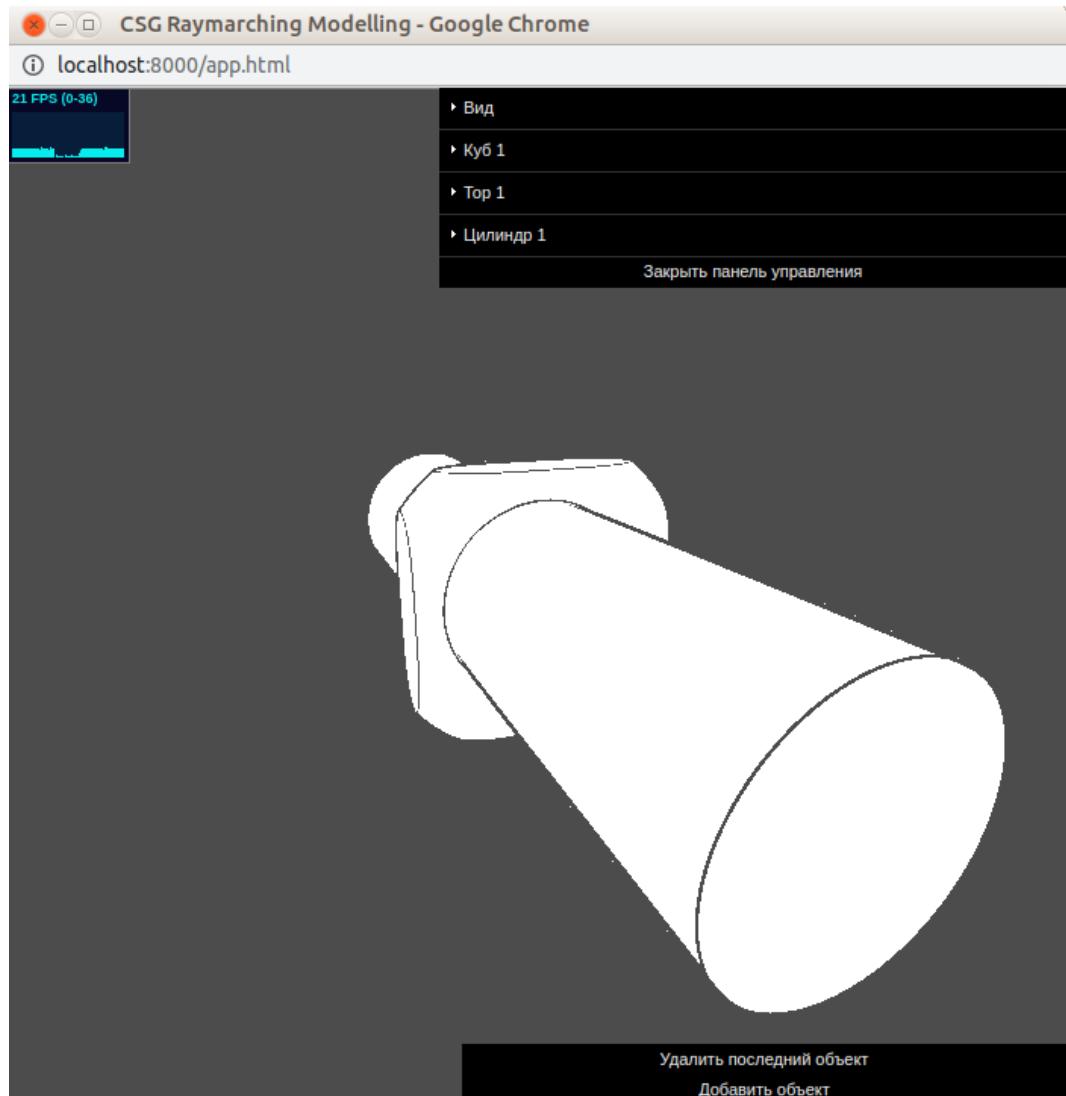


Рисунок 18 – Демонстрация работы программы (композиция моделей куба, тора и цилиндра)

4.2 Постановка эксперимента

Целью эксперимента является сравнение производительности приложения при использовании встроенной и дискретной видеокарт. Производительность будет оцениваться с помощью измерения количества кадров в секунду (FPS), с которыми работает приложение. Нагрузка будет меняться в зависимости от количества объектов, расположенных на сцене.

Технические характеристики устройства, на котором выполнялось тестирование:

- Операционная система: Ubuntu 21.04 [17] 64-bit;
- Память: 16 Гб;
- Процессор: Intel Core™ i5-8300H [18] с тактовой частотой 2.30 ГГц;
- Видеокарты:
 - AMD Radeon (TM) VEGA 8 Graphics (встроенная) [19];
 - NVIDIA GeForce GTX 1080 (дискретная) [20].

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только системой тестирования (работающим приложением) и системным окружением операционной системы.

4.3 Сравнение производительности

Результаты сравнения производительности сцен разной нагруженности приведены в таблице 4.

Таблица 4 – Сравнение FPS при запуске приложения на встроенной и дискретной видеокартах.

Количество объектов	FPS-NVIDIA	FPS-AMD
1	60	24
2	55	18
3	48	10
4	36	8
5	32	6

Результаты тестирования приводят к следующим выводам:

- 1) с ростом количества объектов, количество FPS уменьшается (в лучшем случае 60 кадров (24 на встроенной), в худшем 32 кадра (6 на встроенной));
- 2) дискретная видеокарта позволяет получить большее количество кадров, которое приемлемо для использования приложения в режиме реального времени (30+ FPS), это объясняется наличием встроенной памяти (2 ГБ) и большей мощностью в сравнении с встроенной, которая менее производительна, а также использует разделяемую память (часть оперативной);
- 3) встроенной видеокарты недостаточно для выполнения программы в режиме реального времени - приемлемое FPS должно быть около 30 и более, встроенная (в лучшем случае - 1 объект на сцене) позволяет получить лишь 24 FPS — для режима реального времени недостаточно;
- 4) в среднем дискретная видеокарта производительнее встроенной в 4 раза.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы было разработано программное обеспечение, предназначенное для создания моделей на основе объектов (куб, цилиндр, сфера, тор) с использованием операций логического пересечения, объединения и вычитания. Были проанализированы различные алгоритмы, методы представления, преобразования и отображения модели, выбраны наиболее удовлетворяющие поставленной задачи технологии, а также разработаны алгоритмы для их программной реализации. Разработанная программа позволяет пользователю создавать твердотельные модели на основе примитивов (куб, цилиндр, сфера и тор) с использованием логических операций (объединение, пересечение, вычитание). Присутствует возможность добавлять на сцену объекты, удалять их и преобразовывать, а также менять позицию камеры. Проведено исследование быстродействия программы при запуске на встроенной и дискретной видеокартах. Из результатов следует, что дискретная видеокарта позволяет получить большее количество кадров в секунду, является приемлемой для использования программы в режиме реального времени (более 30 кадров в секунду), в то время как встроенной видеокарты недостаточно для выполнения программы в режиме реального времени.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Вельтмандер П. В. Машинная графика. Книга 2 Основные алгоритмы компьютерной графики. — Новосибирск: Новосибирский государственный университет, 1997.
2. Витиска Н. И., Гуляев Н. А. Метод визуализации трёхмерных сцен и объектов воксельной графики для систем имитационного моделирования. — Таганрог: Таганрогский институт им. А. П. Чехова Ростовского государственного экономического университета, 1962.
3. Новокщенов С. Л., Черных Д. М. Компьютерная графика: учебное пособие. — Воронеж: ФГБОУ ВО «Воронежский государственный технический университет», 2017.
4. ГОСТ Р ИСО 10303-515-2007. Системы автоматизации производства и их интеграция. Представление данных об изделии и обмен этими данными. Часть 515. Прикладные интерпретированные конструкции. Конструктивная блочная геометрия
5. Белов Л. Б., Довгаль В. М., Гордиенко В. В. Растеризация точечных графических объектов на основе продукционной алгоритмической схемы. — Курск: Курский государственный университет, ООО «Конус-Медик», 2012.
6. Шикин Е. В., Боресков А. В. Компьютерная графика. Динамика, реалистичные изображения. — М: ДИАЛОГ-МИФИ, 1995.
7. Евстратов, В. В. Создание программы визуализации псевдотрехмерного изображения с помощью рейкастинга / В. В. Евстратов. — Текст : непосредственный // Молодой ученый. — 2020. — № 50 (340). — С. 12-15. — URL: <https://moluch.ru/archive/340/76504/>, свободный (дата обращения: 01.11.2022).

8. Adrian Biagioli Raymarching Distance Fields: Concepts and Implementation in Unity. — Pittsburgh: Carnegie Mellon University School of Computer Science, 2016.
9. Коротаев А. И., Кузовлев В. И. Моделирование 3D объектов. — Инженерный журнал: наука и инновации, 2013.
10. Газизов В. Р. Программное профилирование работы графического процессора. — Приволжский научный вестник, издательство Индивидуальный предприниматель Самохвалов Антон Витальевич (Ижевск), 2014.
11. JavaScript — official site [Электронный ресурс]. — Режим доступа: <https://www.javascript.com/> (дата обращения: 18.07.2022).
12. Three JS [Электронный ресурс]. — Режим доступа: <https://threejs.org/docs/> (дата обращения: 18.07.2022).
13. JavaScript Performance Monitor [Электронный ресурс]. — Режим доступа: <https://github.com/mrdoob/stats.js>, свободный (дата обращения: 18.07.2022).
14. dat.GUI. A lightweight graphical user interface for changing variables in JavaScript [Электронный ресурс]. — Режим доступа: <https://github.com/dataarts/dat.gui> (дата обращения: 18.07.2022).
15. Visual Studio Code — Code Editing. Redefined [Электронный ресурс]. — Режим доступа: <https://code.visualstudio.com>, свободный (дата обращения: 18.07.2022).
16. Python3 documentation. — Режим доступа: <https://docs.python.org/3/index.html>, свободный (дата обращения: 18.07.2022).
17. Ubuntu — Enterprise Open Source and Linux [Электронный ресурс]. —

Режим доступа: <https://ubuntu.com/>, свободный (дата обращения: 27.07.2022).

18. Процессор Intel Core™ i5-8300H [Электронный ресурс]. — Режим доступа: <https://www.intel.ru/content/www/ru/ru/products/sku/134876/intel-core-i58300h-processor-8m-cache-up-to-4-00-ghz/specifications.html>, свободный (дата обращения: 27.07.2022),
19. AMD Radeon (TM) VEGA 8 Graphics [Электронный ресурс]. — Режим доступа: <https://www.techpowerup.com/gpu-specs/radeon-vega-8.c3042>, свободный (дата обращения: 27.07.2022),
20. NVIDIA GeForce GTX 1080 [Электронный ресурс]. — Режим доступа: <https://www.nvidia.com/ru-ru/geforce/10-series/>, свободный (дата обращения: 27.07.2022).