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

ОТЧЕТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ

Студент Варин Дмитрий Владимирович
(фамилия, имя, отчество)

Группа ИУ7-46Б

Тип практики Технологическая

Название предприятия МГТУ им. Н. Э. Баумана, каф. ИУ7

Студент _____ Варин Д. В.
(подпись, дата) (фамилия, и.о.)

Руководитель практики _____ Куров А.В.
(подпись, дата) (фамилия, и.о.)

Оценка _____

2021 г.

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)

УТВЕРЖДАЮ
Заведующий кафедрой **ИУ7**,
_____ **И.В. Рудаков**
«_____» _____ **2021** г.

З А Д А Н И Е
на выполнение курсовой работы

по дисциплине Компьютерная графика

Студент группы ИУ7-56Б

Варин Дмитрий Владимирович

Тема курсовой работы Разработка программного обеспечения для моделирования твёрдых тел на основе примитивов и логических операций.

Направленность КР (учебная, исследовательская, практическая, производственная, др.)

учебная

Источник тематики (кафедра, предприятие, НИР) кафедра

График выполнения работы: 25% к 4 нед., 50% к 7 нед., 75% к 11 нед., 100% к 14 нед.

Задание Разработать программу для моделирования твердых тел с помощью примитивов (куб, сфера) и логических операций (пересечение, объединение, разность). Предоставить пользователю возможность выбирать примитивы, из которых моделируется тело. Предоставить возможность пользователю выбирать операции композиции тел и трансформации.

Оформление курсовой работы:

Расчетно-пояснительная записка на 25-30 листах формата А4.

Расчетно-пояснительная записка должна содержать постановку введение, аналитическую часть, конструкторскую часть, технологическую часть, экспериментально-исследовательский раздел, заключение, список литературы, приложения.

Перечень графического материала (плакаты, схемы, чертежи и т.п.)

На защиту проекта должна быть представлена презентация, состоящая из 15-20 слайдов. На слайдах должны быть отражены: постановка задачи, использованные методы и алгоритмы, расчетные соотношения, структура комплекса программ, диаграмма классов, интерфейс, характеристики разработанного ПО, результаты проведенных исследований.

Дата выдачи задания «___» _____ 2021 г.

Студент группы ИУ7-56Б _____ Д. В. Варин
(Подпись, дата) (И.О. Фамилия)

Руководитель курсовой работы _____ А.А. Волкова
(Подпись, дата) (И.О. Фамилия)

Дополнительные указания по проектированию

Пользователь должен иметь возможность выбирать примитивы (куб, сфера), операции композиции (объединение, пересечение, разность). Модель должна состоять только из заданных в программе примитивов. На сцене должна присутствовать 1 сконструированная модель. На сцене должна присутствовать одна камера, положение которой зафиксировано. Пользователь должен иметь возможность выполнять следующие действия: использовать операции композиции для создания модели, использовать операции трансформации модели, выбирать примитивы, используемые при моделировании. Модель располагается в центре сцены, пользователь может изменять положение посредством операций трансофрмации.

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)

Кафедра « Программное обеспечение ЭВМ и информационные технологии » (ИУ7)

З А Д А Н И Е

на прохождение производственной практики

на предприятии МГТУ им. Н. Э. Баумана, каф. ИУ7

Студент Варин Дмитрий Владимирович ИУ7-46Б

Во время прохождения производственной практики студент должен:

1. Спроектировать программу для моделирования твердых тел с помощью примитивов (куб, сфера) и логических операций (пересечение, объединение, разность).

2. Проанализировать методы и алгоритмы. Выбрать необходимые для решения поставленной задачи.

3. Разработать архитектуру приложения.

Дата выдачи задания «___» _____ 2021 г.

Руководитель практики от кафедры _____ Куров А.В.
(Подпись, дата) (Фамилия И.О.)

Студент _____ Варин Д. В.
(Подпись, дата) (Фамилия И.О.)

Дополнительные указания по проектированию

Пользователь должен иметь возможность выбирать примитивы (куб, сфера), операции композиции (объединение, пересечение, разность). Модель должна состоять только из заданных в программе примитивов и их комбинаций. На сцене должна присутствовать 1 сконструированная модель. На сцене должна присутствовать одна камера, положение которой зафиксировано. Пользователь должен иметь возможность выполнять следующие действия: использовать операции композиции для создания модели, использовать операции трансформации модели, выбирать примитивы, используемые при моделировании. Модель располагается в центре сцены, пользователь может изменять положение модели посредством операций трансформации (перемещение, масштабирование, поворот).

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ

В настоящем отчете о практике применяют следующие термины с соответствующими определениями.

Raymarching – алгоритм трассировки лучей.

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ПО – программное обеспечение.

CSG – Constructive solid geometry — конструктивная сплошная геометрия.

SDF – Signed distance functions — поле расстояния с знаком.

CPU – Central processing unit — центральный процессор.

GPU – Graphics processing unit — графический процессор.

ВВЕДЕНИЕ

Сегодня для увеличения эффективности труда и повышения качества разрабатываемой продукции двухмерное проекционное черчение заменяется трехмерным (твердотельным) моделированием, которое работает с объектами, состоящими из замкнутого контура. Данный подход обеспечивает полное описание трехмерной геометрической формы [1].

Моделирование твердого тела является неотъемлемой частью проектирования и разработки изделий [2]. Все тела можно разделить на базовые и составные. К базовым относятся примитивы: параллелепипед, цилиндр, шар, конус и др.. Однако в жизни редко можно встретить объекты, состоящие из одного базового тела. Как правило, изделия сложны по своей структуре, что приводит к появлению составных. Такие тела формируются в результате операций над базовыми (булевы функции сложения, вычитания, пересечения) [1]. Существует несколько схем представления таких тел, из которых нужно выбрать наиболее подходящую.

Смоделировав тело, предстаёт новая задача: нужно его визуализировать, а также предусмотреть возможность просмотра модели с разных ракурсов. Для этого есть несколько способов, каждый из которых имеет свои преимущества и недостатки [3]. Необходимо выбрать наиболее подходящий под выделенную задачу.

После создания тела нужно его отрисовать. Это требует больших вычислительных мощностей. Данную задачу можно возложить на центральный процессор - CPU или графический - GPU [4]. Следует выбрать наиболее подходящий вычислительный ресурс и отрендерить созданную модель. Все эти действия приводят к идее создания программного обеспечения, которое объединит в себе решение всех озвученных выше задач и приведёт к конечному результату - твердотельной модели.

Цель работы на время практики - проектирование программного обеспечения для моделирования твердых тел на основе примитивов и логических операций. Таким образом, необходимо выбрать оптимальные алгоритмы представления твердотельной модели, её преобразования, визуализации

и аппаратной обработки. Спроектировать процесс моделирования и предоставить схему для его реализации.

1 Аналитический раздел

В данном разделе проводится анализ и выбор методов создания, рендера сложных моделей.

1.1 Анализ методов создания сложных моделей

Моделирование твердого тела - это последовательный набор принципов математического и компьютерного моделирования трехмерных твердых тел [5]. На данном этапе стоит рассмотреть существующие схемы представления твердотельной модели.

1.1.1 Клонирование примитивов

Схема основана на понятии семейств объектов: выделяются определённые члена семейства, отличающиеся несколькими параметрами друг от друга. С помощью операций поворота, масштабирования (см. рис. 1.1) можно изменять объект.

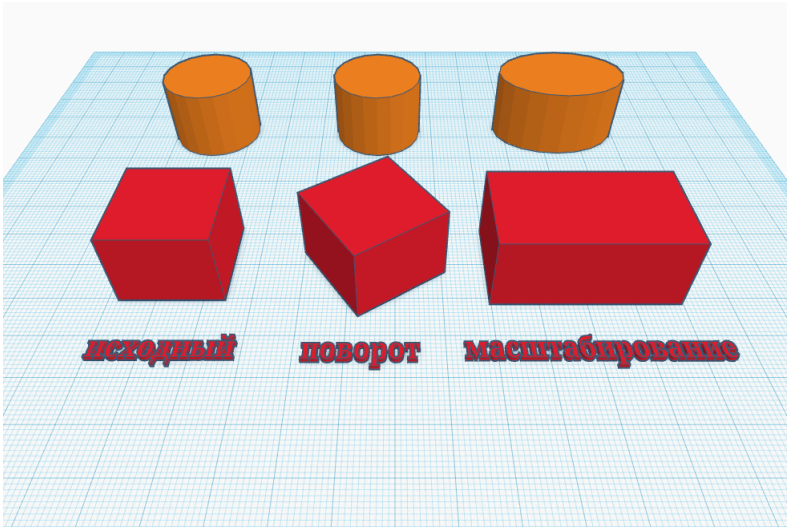

Рисунок 1.1 — Пример клонирования примитивов на примере куба и цилиндра

Каждое семейство объектов называется общим примитивом, а отдельные объекты называются примитивными экземплярами. Например, семейство болтов является общим примитивом, а отдельный болт, заданный определенным набором параметров (см. рис. 1.2), является примитивным экземпляром.

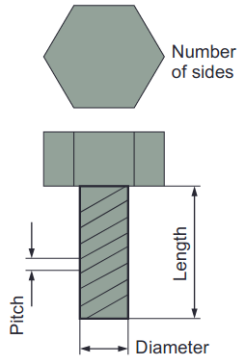

Рисунок 1.2 – Болт задан параметрами: количество сторон головы, шаг резьбы, длина, диаметр

Особенности:

– невозможно создать сложный объект, с помощью объединения экземпляров.

Минусы:

– сложность написания алгоритмов для вычисления свойств представленных твердых тел из - за уникальности каждого примитива, следовательно, обобщить обработку невозможно.

Плюсы:

– способ хорош, если требуется только представление определённого семейства моделей.

Чтобы решить поставленную задачу, нужна схема, с помощью которой можно не зависеть от типа модели: его формы, параметров.

Рассмотрев схему можно заключить, что она не подходит для решения задачи, так как возникает потребность в подробном описании всех свойств определённого семейства, а учитывая потребность в составных телах, сделать для них это будет проблематично.

1.1.2 Нумерация пространственного заполнения

Схема из себя представляет список-сетку пространственных ячеек, занятых твердым телом. Ячейки (воксели) представляют собой кубы фиксированного размера (см. рис. 1.3) и расположены в заданной пространственной сетке. 3D объект представляет собой список закрашенных вокселей.

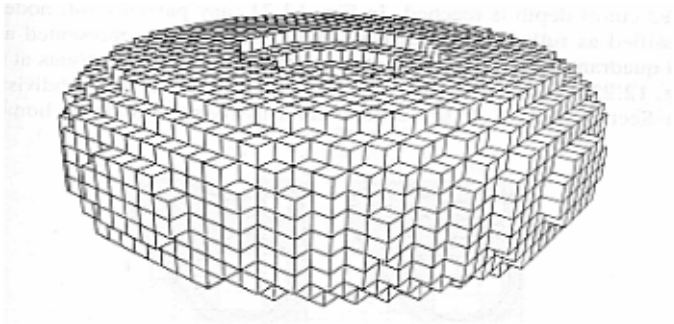

Рисунок 1.3 – Пример представления тора методом нумерации пространственного заполнения

Каждая ячейка может быть представлена координатами одной точки, например центроида ячейки.

Обычно устанавливается определенный порядок сканирования, и соответствующий упорядоченный набор координат называется пространственным массивом.

Пространственные массивы являются однозначными и уникальными твердыми представлениями, но слишком подробны для использования в качестве «основных» представлений [6].

Минусы:

– затраты памяти высоки;

– каждая ячейка хранит информацию о занятости, цвете, плотности и др.;

– разрешение ограничено размером и формой вокселя.

Плюсы:

– простая структура данных;

– однозначное представление.

Данная схема удобна с точки зрения простоты и точности представления, однако, структурно состоит из кубических форм, для рендера, например, сферы, потребуется дополнительно сглаживать края, что накладывает дополнительную вычислительную нагрузку.

Также для нас избыточно хранения в каждой ячейке информации о её состоянии. Самостоятельное использование метода нецелесообразно, однако совместно с другими методами, можно использовать преимущество грубой апроксимации, для повышения точности изображаемого тела.

1.1.3 Sweeping

Эта схема позволяет создавать 3D модели из 2D с помощью движения по заданной траектории: вокруг оси (см. рис. 1.4), относительно граней и др. [6].

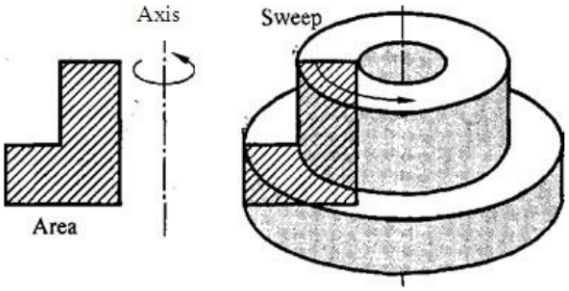

Рисунок 1.4 – Тело образовано вращением вокруг оси

Плюсы:

– простые формы удобно задавать через плоские фигуры;

– может использоваться для быстрого удаления материала внутри тела (см. рис. 1.5).

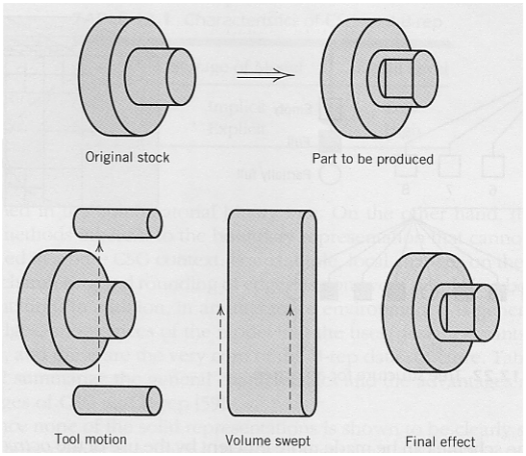

Рисунок 1.5 – Удаление материала с помощью sweeping

Минусы:

– необходимо задать траекторию движения 2D объекта, что проблематично для тел, имеющих сложную форму.

1.1.4 Октантное дерево

Схема является улучшением воксельного представления. Каждый узел октантного дерева соответствует некоторому кубу в трехмерном пространстве, для которого определяется принадлежность модели. У каждого корня дерева есть 8 потомков, т.е куб делится на 8 равных частей (см. рис. 1.6).

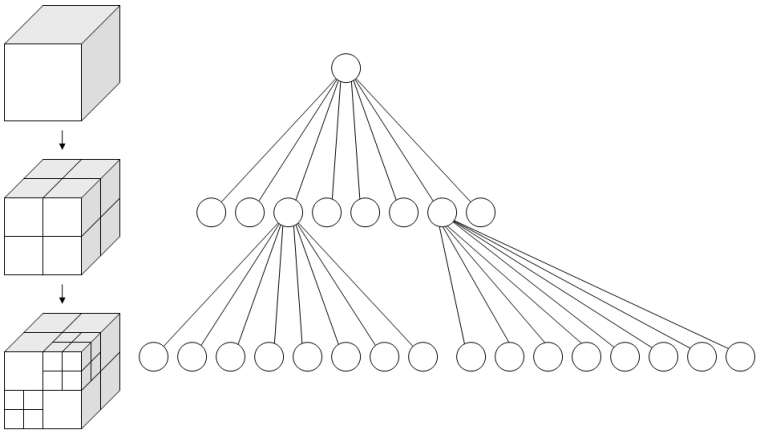

Рисунок 1.6 – Слева: Рекурсивное разделение куба на октанты. Справа: Соответствующее октодерево.

Метод позволяет устранить недостаток метода 1.1.2, связанный с хранением большого количества данных, сохраняя информацию только об используемых частях модели. Однако в сравнении с другими методами, памяти расходуется всё так же много.

Обычно, используется в сферах, требующих точное представление, например, в медицине [7].

Минусы:

– возможное деления примитива ребрами кубов дерева, что снижает эффективность;

– вывод всех объектов, находящихся в поле камеры, но на самом деле в конечном счёте невидимых.

1.1.5 Граничное представление(BREP)

Способ представления модели с помощью границ. Замкнутая 2D-поверхность определяет 3D-объект.

Суть BRep-представления заключается в том, что твердое тело описывает замкнутая пространственная область, ограниченная набором элементарных поверхностей (граней) с общими образующими контурами (ребрами) на границе поверхностей (см. рис. 1.7) и признаком внешней или внутренней стороны поверхности.

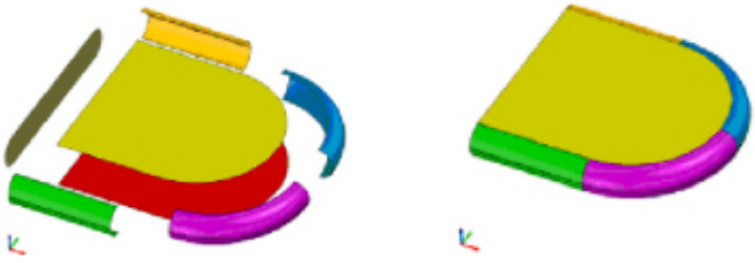

Рисунок 1.7 – BRep-представление простых твердых тел

Данный способ проектирования в приложениях САПР является наиболее распространенным, однако имеет недостатки, из-за которых возникает проблема визуализации результата [8].

Минусы:

– высокие затраты памяти;

– когда объект необходимо отрендерить объект требует слишком много вычислительной мощности;

– для сложных объектов возникают трудности получения математических формул их описывающих.

Плюсы:

– подходит не только для твердых тел с плоскими гранями, но и для криволинейных объектов с замкнутыми криволинейными гранями или краями.

– позволяет проводить различные вычисления, требующие точность, благодаря хранению информации о всех составляющих модели.

1.1.6 Конструктивная сплошная геометрия(CSG)

Данный способ представления основан комбинирования примитивов посредством логических операций, что позволяет создавать сложные модели на основе простых с помощью:

– объединения;

– пересечения;

– разности.

Любое составное тело может быть описано в виде традиционного уравнения из булевых функций, в котором аргументами являются либо элементарные тела, либо другие составные тела. Это представление называют деревом построений (см. рис. 1.8) [8].

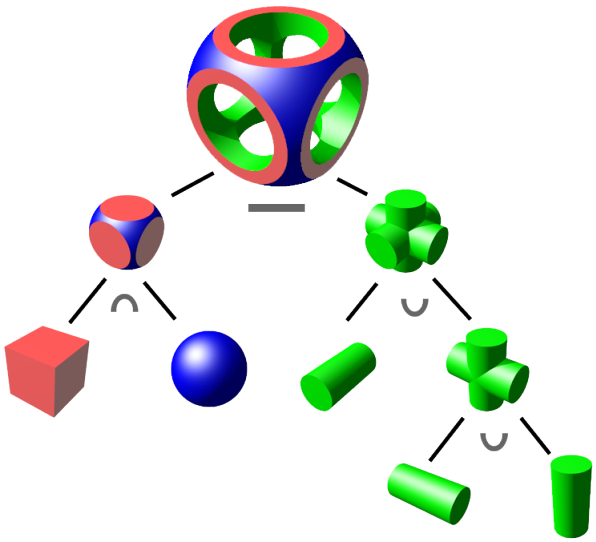

Рисунок 1.8 – Сложный объект может быть представлен двоичным деревом, где «листья» — это объекты, а узлы — операции. (пересечение, объединение, – - разность)

Особенности:

– При использовании логических операций нужно иметь ввиду, что комбинирование может привести к вырождению твердого тела в плоское.

1.1.7 Сравнение схем

Сравнивая схемы, предлагается разобрать следующие критерии:

1) Лаконичность, т.е сколько памяти компьютера занимает модель (+ значит мало, - иначе: модель хранит не всегда используемую информацию), для решения задачи требуется минимальный расход;

2) Эффективность, т.е насколько много времени необходимо для создания, исследования или изменения формы модели;

3) Уникальность, т.е получить модель можно только одним способом;

4) Однозначность, т.е вместе с моделью хранятся данные, которых достаточно для осуществления геометрических расчётов;

5) Хранение истории преобразования модели, чтобы у пользователя была возможность вернуться к предыдущему шагу моделирования тела без применения афинных преобразований и дополнительных вычислений;

6) Практичность, т.е создание модели без введения дополнительных параметров, поддерживая удобство использования.

Разбор рассмотренных по критериям методов представлен в таблице 1.1.

Таблица 1.1 — Сравнительная таблица схем представления твёрдого тела

Методы	Лак-ть	Эф-ть	Ун-ть	Одноз-ть	История	Прак-ть
Клонирование примитивов	-	+	-	+	-	-
Нумер-я простр-го заполн.	-	-	+	+	-	-
Октантное дерево	+	-	+	+	-	-
BREP	-	+	+	+	-	-
Sweeping	+	-	-	+	-	-
CSG	+	+	-	+	+	+

После рассмотрения схем представления предлагается использовать CSG как наиболее подходящий метод создания моделей. Представление твердых тел в виде дерева построений (листья - примитивы, узлы - результат операции) удобно также 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 необходима для обработки и создания моделей, но для решения задачи она не является необходимостью.

1.2 Анализ методов рендера модели

После того, как был выбран способ создания твердотельной модели, следует изучить методы рендера.

Рендеринг (англ. rendering — «визуализация») – термин, обозначающий процесс получения изображения по модели с помощью компьютерной программы [9].
Рассмотрим возможные варианты:

1.2.1 Растеризация

Растеризация – это процесс получения растрового изображения [10]. Растровое изображение - это изображение, представляющее собой сетку пикселей — цветных точек (обычно прямоугольных) на мониторе, бумаге и других отображающих устройствах [11].
Технология основана на обходе лучем вершин треугольника (см. рис. 1.9), который остаётся самим собой даже после попадания из трехмерного пространства в двухмерное.

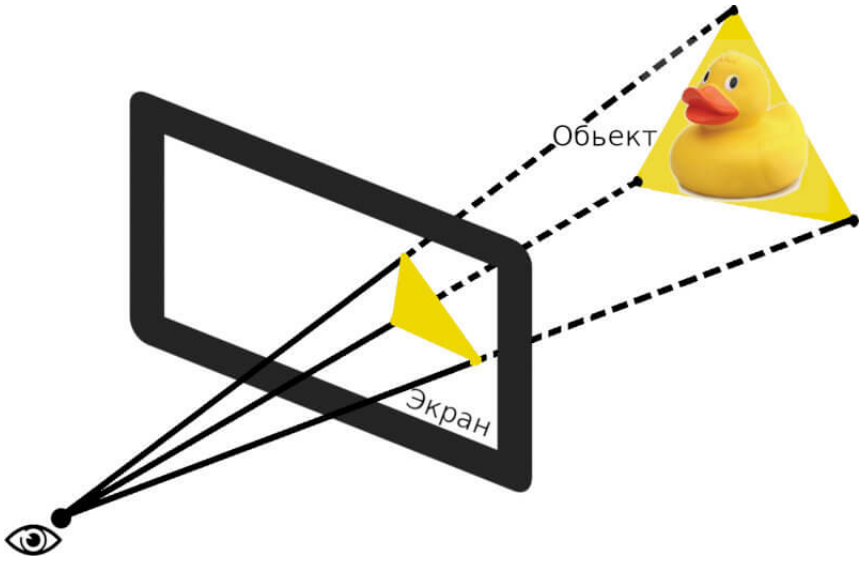

Рисунок 1.9 – Схематическое отображение предмета на экран

Каждая точка каждого объекта в трехмерном пространстве переводится в точку на экране (см. рис. 1.10), а затем точки — заданные в модели треугольники — соединяются и получается изображение исходного объекта.

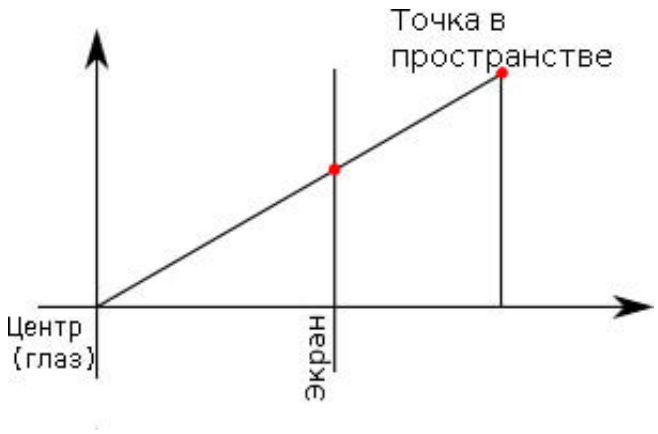

Рисунок 1.10 – Проецирование точки из пространства на плоскость экрана, вид сбоку

Преимущества и недостатки:

– изображение может быть ”угловатым”(ступенчатым) - нуждается в сглаживании;

– требует больших вычислительных ресурсов - могут использоваться миллионы полигонов для всех моделей объектов сцены и примерно 8 миллионов пикселей на дисплее 4K, и каждый кадр или изображение, отображаемое на экране, обычно обновляется на дисплее от 30 до 90 раз в секунду [12];

– каждый пиксель обрабатывается много раз;

– 3D модель должна быть описана из примитивов, следовательно, составные модели обрабатывать ресурсозатратно, т.к нужно разбить его на примитивы, обычно, треугольники.

– современные компьютеры оптимизированы для рендеринга растровых изображений, благодаря чему этот процесс достаточно быстрый, однако, как было сказано выше, с ростом сложности изображения, возрастают время рендера и потребности к ресурсам.

1.2.2 Трассировка лучей

1.2.2.1 RayCasting

Ray-casting (рус. — бросание лучей) — это технология, которая преобразует набор данных в 3D проекцию путем «бросания лучей» из точки обзора по всей области видимости.

Основная идея - испускать лучи из «глаз» наблюдателя, один луч на пиксель, и находить самый близкий объект, который блокирует путь распространения этого луча (см. рис. 1.11). Последующая обработка преломленных от объекта лучей в этом методе отсутствует. Используя свойства материала и эффект света в сцене, алгоритм бросания лучей может определить затенение данного объекта [13].

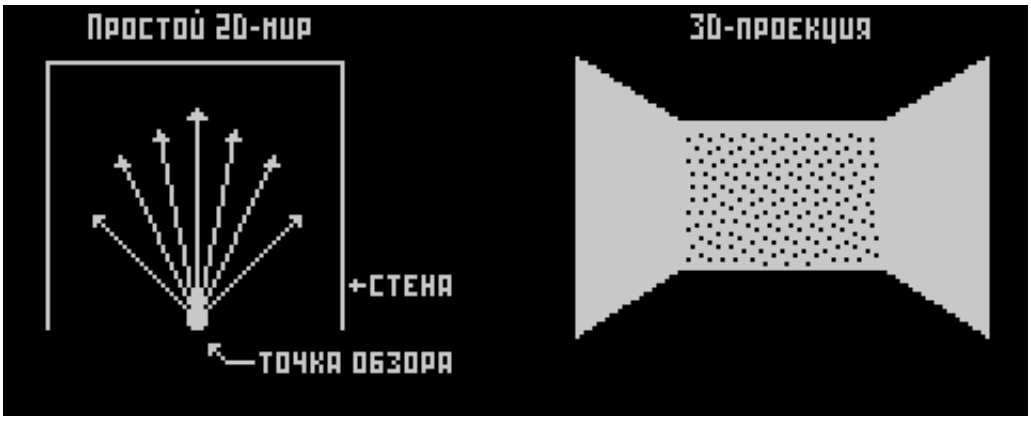

Рисунок 1.11 — На рисунке показано, как бросание преобразует что-то двумерное во что-то почти что трехмерное.

Метод использовался при создании игр в конце прошлого века, такую графику иногда называют ”псевдо 3D”или ”2.5D”.

Из простоты алгоритма вытекает ряд ограничений, сказывающихся на качестве результата (см. рис. 1.12), ведь на деле это всего лишь трёхмерная проекция плоского изображения. Ограничения алгоритма на примере применения в игровом мире [14]:

1) все доступное в игре пространство — это комната с прямоугольными (чаще квадратными) стенами;

2) нет лестниц, лифтов, любого вида спусков и подъемов;

3) потолок везде имеет одинаковую высоту;

4) нет других трехмерных объектов, кроме стен, пола и потолка, все остальные объекты - двумерные изображения, расположенные в трехмерном пространстве.

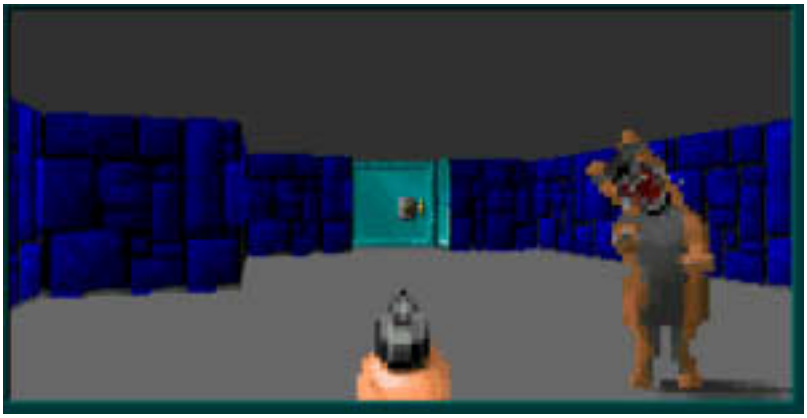

Рисунок 1.12 – Сцена из Wolfenstein 3D. Картинка делится на прямоугольные блоки. Объекты (оружие) и враги (собака) — это просто прозрачные растровые изображения, которые масштабированы и отрисованы поверх заднего фона.

Минусы:

– в результате рендера получается изображение среднего-плохого качества, в сравнении с остальным методами;

– геометрические ограничение на обрабатываемую поверхность(только простые фигуры);

– эффективен для объектов, у которых нет больших затрат на вычисление пересечения луча.

Плюсы:

– прост в реализации;

– нетребователен;

– изображение генерируется ”на лету”.

1.2.2.2 RayTracing

Трассировка лучей (англ. Ray Tracing) – это технология отрисовки трехмерной графики, симулирующая физическое поведение света [14].
Принцип работы трассировки лучей:
поскольку все существующие трехмерные модели собраны из треугольников, нужно было обязательно сохранить обратную совместимость - для этого проверяется случай столкновения луча не со стеной, а с треугольником. Рассмотрим его.

Пусть вершины треугольника обозначены через V_0, V_1, V_2. Векторы двух его рёбер обозначены как A, B и заданы формулами 1.1 и 1.2:

$$A = V_1 - V_0 \tag{1.1}$$

$$B = V_2 - V_0 \tag{1.2}$$

Определим луч P с помощью параметрической формы уравнения 1.3:

$$P = R_0 + t \cdot R_d \tag{1.3}$$

где R_0 – начальная точка луча, R_d – направление луча, t – расстояние вдоль луча, на которое попала точка
С другой стороны, точка пересечения имеет координаты u, v в плоскости треугольника. Приравняв уравнение луча P и плоскости в точке u, v (см. формулу 1.4), можно найти пересечение:

$$P = V_0 + u \cdot A + v \cdot B \tag{1.4}$$

Таким образом, составив систему из трех уравнений 1.5 для координат x, y, z и решив её для t, u, v, необходимо проанализировать определитель. Если он ненулевой(луч не параллелен плоскости), а $t >= 0$ и $u, v, u + v$ лежат в диапазоне от $0 \dots 1$, то P находится внутри треугольника и поиск столкновения завершается.

$$\begin{cases} R_0.x + t \cdot R_d.x = V_0.x + u \cdot A.x + v \cdot B.x \\ R_0.y + t \cdot R_d.y = V_0.y + u \cdot A.y + v \cdot B.y \\ R_0.z + t \cdot R_d.z = V_0.z + u \cdot A.x + v \cdot B.z \end{cases} \tag{1.5}$$

Особенность же трассировки лучей состоит в том, что на одном треугольнике серия вычислений не заканчивается, ведь некоторые поверхности могут быть зеркальными или просто блестеть. В таком случае луч не останавливается, а отражается от этого треугольника и снова ищет себе цель до тех пор, пока не вернётся в начальную точку, или не превысит максимальное число отражений. Вся сложность алгоритма позволяет получить качественное изображения, в сравнении с полученным в результате растеризации (см. рис. 1.13).

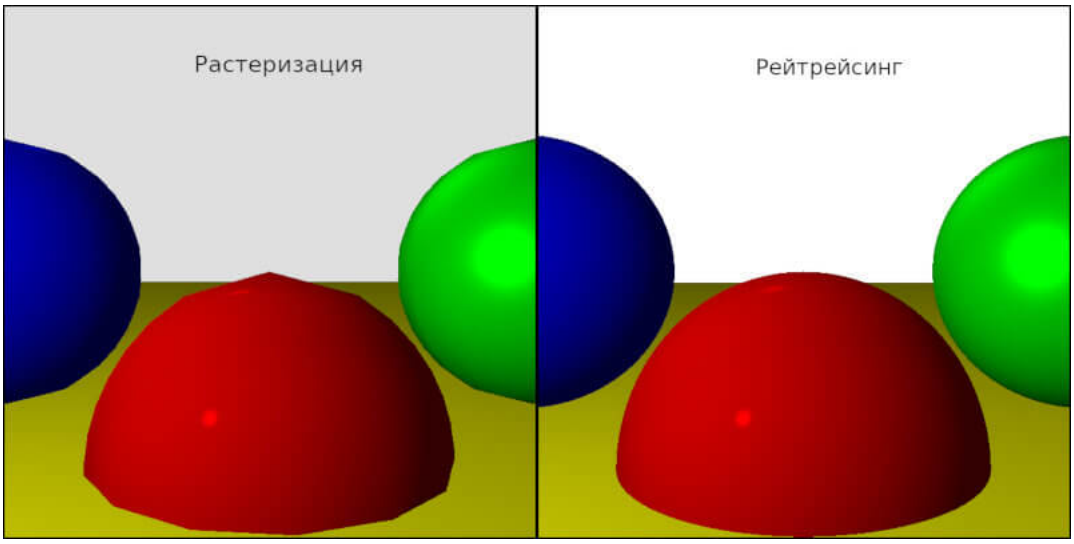

Рисунок 1.13 — Сравнение растеризации и трассировки лучей

Плюсы:

– возможность рендеринга гладких объектов без аппроксимации их полигональными поверхностями (например, треугольниками);

– вычислительная сложность метода слабо зависит от сложности сцены;

– отсечение невидимых поверхностей, перспектива и корректное изменения поля зрения являются логическим следствием алгоритма.

Однако, главным недостатком метода является производительность. Метод трассирования лучей каждый раз начинает процесс определения цвета пикселя заново, рассматривая каждый луч наблюдения в отдельности. Что даёт изображению реалистичность, решает задачу отражения, но требует много ресурсов.

Следует рассмотреть модификацию данного алгоритма.

1.2.2.3 RayMarching

Марширование лучей (англ. RayMarching) - разновидность алгоритма трассировки лучей.

Raymarching похож на традиционную трассировку лучей (raytracing) тем, что луч в сцену испускается для каждого пикселя. В трассировщике лучей есть набор уравнений, определяющих пересечение луча и рендерящихся объектов. Raymarching предлагает другой метод решения задачи пересечения луча и объекта, не пытается вычислить персечение аналитически. При нём происходит смещение текущего положения вдоль луча, пока не будет найдена точка, пересекающая объект. Эта операция является относительно простой и малозатратной, гораздо более практичной в реальном времени.
Однако, этот способ не точно находит пересечение (см. рис. 1.14).

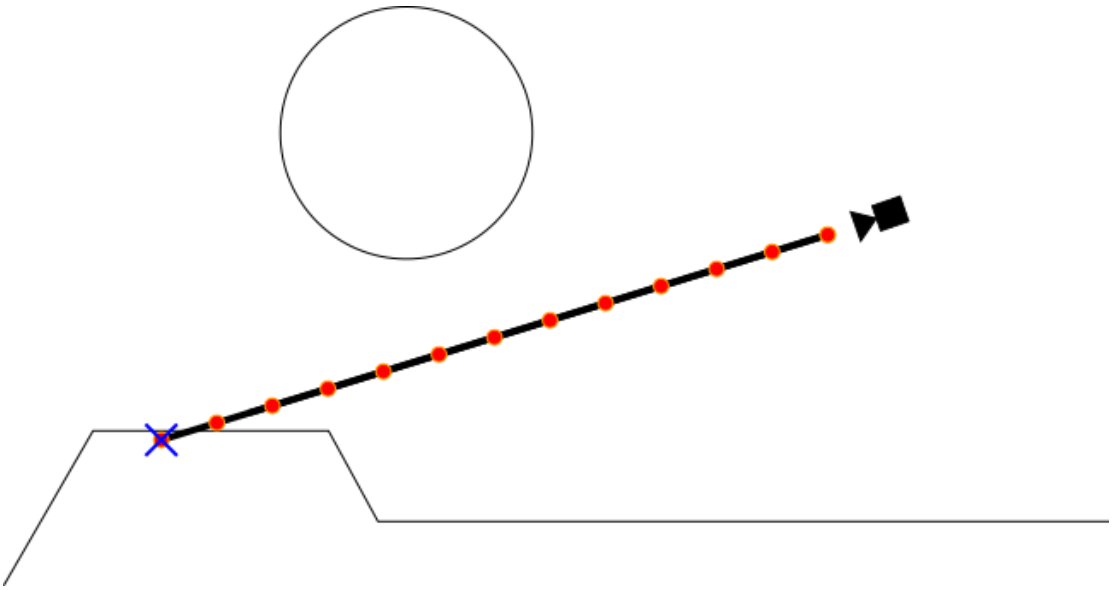

Рисунок 1.14 – Простейшая реализация метода с фиксированным интервалом шага. Красными точками обозначены все сэмплируемые точки.

1.2.2.4 Функция поля расстояний

Реализация, разобранная выше, вполне достаточна для множества областей применения, например, объёмных и прозрачных поверхностей. Однако для непрозрачных объектов можно ввести ещё одну оптимизацию. Для этой оптимизации требуется использование полей расстояний со знаком.

Поле расстояний (signed distance fields) — это функция, получающая на входе точку и возвращающая кратчайшее расстояние от этой точки до поверхности каждого объекта в сцене. SDF возвращает отрицательное число, если точка находится внутри объекта.

Этот инструмент позволяет нам ограничить количество шагов при движении вдоль луча (см. рис. 1.15), что увеличивает эффективность метода.

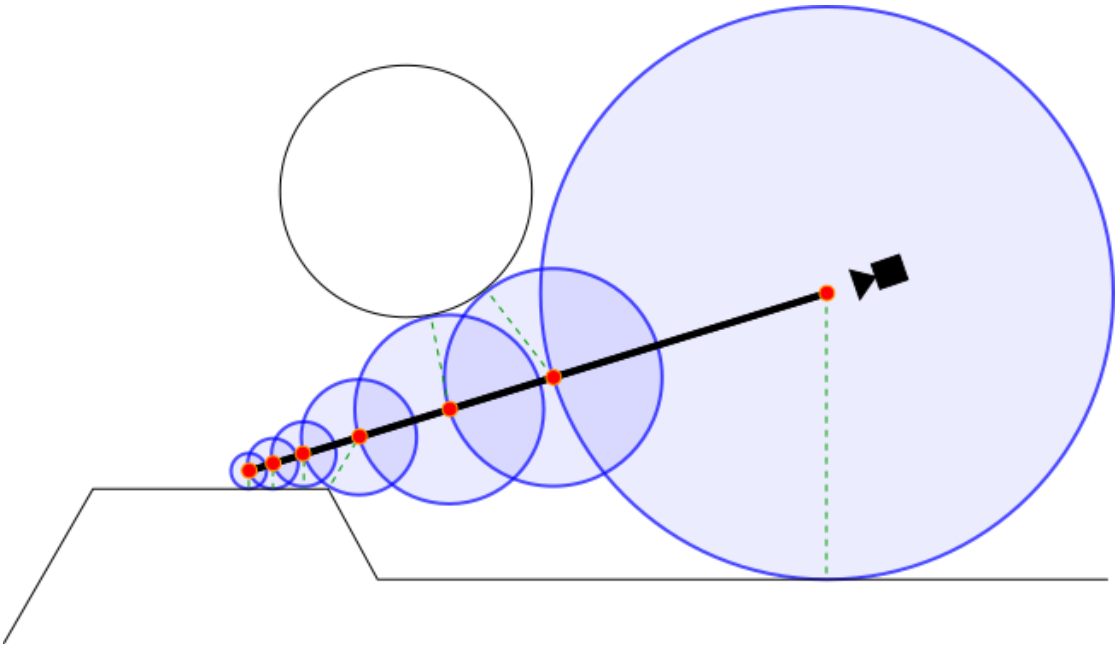

Рисунок 1.15 – Визуализация метода, с использованием SDF. Красными точками показаны все сэмплируемые точки. Синими кругами показаны области, которые гарантированно не содержат объектов (потому что они находятся в результатах функции поля расстояний). Пунктирными зелёными линиями показаны истинные кратчайшие векторы между каждой сэмплируемой точкой и сценой.

Описание алгоритма [15]:

– для пикселя на экране определяется расстояние до ближайшего объекта;

– это расстояние определяет радиус на который можно пустить луч;

– пускаем луч;

– для конечной точки луча предыдущие пункты повторяются до тех пор пока следующий радиус не станет достаточно маленьким, что будет означать столкновение с объектом. Или же радиус может начать стабильно увеличиваться, если луч прошел мимо всех объектов.

Плюсы:

– решает проблему производительность трассировки лучей, работает быстро;

– подходит для рендера поверхностей, для которых сложно или невозможно вычислить пересечение аналитически;

– используя SDF можно ускорить рендеринг до реального времени;

– подходит для графического процессора, так как каждый пиксель можно рассчитать независимо;

– качество изображения сопоставимо с полученным при трассировке лучей.

Минусы:

– пересечение вычисляется неточно, с некоторой погрешностью;

1.2.2.5 Сравнение методов

Сравнивая рассмотренные методы рендеринга, предлагается разобрать следующие критерии:

1) качество;

2) эффективность, т.е насколько много времени необходимо для рендера модели;

3) выполнение в режиме реального времени;

4) точность, т.е обработка истинных границ обрабатываемой модели;

Разбор рассмотренных по критериям методов представлен в таблице 1.2.

Таблица 1.2 — Сравнительная таблица методов рендеринга

Методы	Качество	Эффективность	Реал.время	Точность
Растеризация	-	+	+	+
RayCasting	-	+	+	+
RayTracing	+	-	-	+
RayMarching	+	+	+	-

В данном разделе были рассмотрены методы рендера модели. Учитывая метод, выбранный в предыдущем разделе, а также анализ в текущем, можно сказать, что алгоритм raymarching наиболее привлекателен для рендера конструктивной сплошной геометрии, в такой связке можно получить качественное изображение при его отрисовке в реальном времени. Один недостаток, это вычисление границ объекта с некоторой погрешностью, но остальные плюсы метода затмевают его.

1.3 Преобразования и визуализация

В данном пункте рассматриваются средства для преобразования модели и придания ей трехмерного вида.

1.3.1 Шейдеры

Шейдер - компьютерная программа, предназначенная для исполнения процессорами видеокарты (GPU) [16].

В разрабатываемом приложении использование шейдеров необходимо для придания изображению трехмерного вида, используя тени, освещение. Для этого предназначены фрагментный и вершинный шейдеры.

1.3.1.1 Вершинные шейдеры

Вершинный шейдер получает вершину из списка вершин и отображает ее в пространстве. Вершинному шейдеру передаются следующие данные:

- матрица модели;
- матрица вида;
- матрица проекции.

Матрица модели:

Необходима для перехода от пространства модели (все вершины определены относительно центра модели) в мировое пространство (все вершины определены относительно центра мира).

Матрица вида:

Необходима для перемещения сцены относительно камеры. В реальном мире происходит перенос самой камеры, точки обзора. В компьютерной графике более просто и удобно переместить сцену относительно камеры, тем самым в результате получить перемещение камеры относительно сцены.

Матрица проекции:

Эта матрица используется для представления перспективы камеры. Умножение координат на данную матрицу позволяет сопоставить вершину с перспективой камеры, учитывая ее соотношения сторон, поля обзора и самого дальнего и ближайшего объекта, который виден.

Для преобразования координат выполняются следующие действия:

1) положение вершины умножается на матрицу модели, чтобы определить, где эта вершина находится относительно центра модели в сцене;

2) результат умножается на матрицу вида, чтобы определить, где расположена вершина относительно камеры;

3) результат второй операции умножается на матрицу проекции, чтобы определить, где находится вершина в перспективе камеры.

1.3.1.2 Фрагментные шейдеры

Фрагментный шейдер работает с пикселями объекта и управляет цветом пикселя. Выполнение фрагментного шейдера в графическом конвейере происходит после вершинного, следовательно, никак не влияет на координаты вершины, он влияет только на цветовую составляющую, преобразуя вершины уже в пиксели или фрагменты.

Помимо придания трехмерного вида, к моделям следует применять операции, которые позволяют изменять их положение и ориентацию.

1.3.2 Матрицы преобразования

Для преобразования тела в пространстве используются операции:

- перемещения;
- масштабирования;
- поворота.

Осуществляются преобразования с помощью матриц 1.3.

Таблица 1.3 — 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	$\begin{pmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$	$\begin{cases} x_1 = x_c + (x - x_c)\cos\theta - (y - y_c)\sin\theta \\ y_1 = y_c + (x - x_c)\sin\theta + (y - y_c)\cos\theta \\ z_1 = z \end{cases}$

Исходные координаты объекта умножаются на одну или несколько матриц, в результате чего объект перемещается/масштабируется/поворачивается, в зависимости от примененных преобразований. Помимио этого, матрицы позволяют перевести координаты модели в мировые, переместить сцену относительно камеры, настроить перспективу.

В данном пункте были рассмтрены средства для преобразования модели - матрицы преобразования, а также для преобразования плоского изображения в трехмерный вид - шейдеры.

1.4 Аппаратная обработка

В данном разделе предлагается рассмотреть аппаратную составляющую рендеринга. Есть 2 основных варианта: рендеринг на центральном и графическом процессорах. Они имеет много общего, но существуют различия, которые оказывают огромное влияние на скорость и качество изображения. Рендеринг на базе ЦП является традиционным способом и широко используется. Напротив, рендеринг с помощью графических процессоров с годами становится все более популярным в сообществе благодаря быстроразвивающемуся миру технологий. Рассмотрим подробнее каждый из процессоров.

1.4.1 CPU

CPU — центральный процессор, это основной компонент компьютера, обрабатывающий инструкции. Он выполняет вычисления, действия, запускает программы, включая рендеринг. Центральный процессор постоянно принимает ввод от пользователя или активных программ, затем обрабатывает данные и выдает вывод, который может быть сохранен приложением или отображен на экране [17].

1.4.2 GPU

GPU — графический процессор. Разработан для параллельной обработки трудоёмких вычислительных задач. Графический процессор исполь-

зуется в широком спектре приложений, ускоряющих рендеринг 3D-графики. Кроме того, этот микропроцессор также используется для разгрузки некоторых задач с центрального процессора, что заставляет компьютер работать быстрее [17].

1.4.3 Сравнение процессоров

Рассмотрим различия процессоров приминительно к рендеру [4]:

1) главное различие между процессорами CPU и GPU заключается в том, как каждый из них выполняет разные задачи;

2) архитектурно CPU состоит всего из нескольких ядер с большим количеством кэш-памяти, которая может обрабатывать несколько программных потоков одновременно, Напротив, графический процессор состоит из сотен меньших и более эффективных ядер, которые могут одновременно выполнять несколько задач и быстрее обрабатывать изображения;

3) рендеринг с помощью графического процессора более эффективен с точки зрения задач обработки, требующих нескольких параллельных процессов, фактически, рендеринг GPU примерно в 10-100 раз быстрее, чем рендеринг CPU;

4) GPU позволяет в реальном времени просматривать и манипулировать 3D моделями, источниками света и проекциями в трех измерениях. Некоторое программное обеспечение для рендеринга, предназначенное только для графического процессора, может даже позволить полностью работать в окне просмотра с включенным Real Time рендерингом, увеличивая результат и минимизируя возможные ошибки, которые могут возникнуть при рендеринге в другой программе. CPU же не позволяет рендерить в реальном времени качественные изображения.

Изначальная цель заключалась в создании приложения для рендера 3D модели в режиме реального времени. Такой сценарий позволяет осуществить рендеринг на видеокарте, следовательно, для поставленной задачи следует выбрать её, а не CPU.

1.5 Вывод

В данной главе был проведен анализ возможных методов для решения поставленной задачи. В качестве метода создания модели, при помощи которого будет решаться задача, была выбрана CSG, рендера – RayMarching. Для преобразования модели будут использоваться матрицы преобразований, для придания изображению трехмерного вида - шейдеры. На GPU ложится графическая составляющая задачи, а на CPU - взаимодействие с пользователем.

2 Конструкторский раздел

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

2.1 Метод конструктивной сплошной геометрии

Данный метод делится на несколько алгоритмов, рассматриваемых ниже.

Знаковая функция расстояния (англ. SDF) может быть использована для получения кратчайшего расстояния от точки до поверхности тела, причём отрицательный знак соответствует положению точки внутри тела. Ниже рассматриваются SDF для сферы и куба.

Для получения SDF сферы необходимо знать расстояние до её центра от заданной точки (см. формулу 2.1) и длину радиуса.

$$length(x,y,z) = \sqrt{(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2} \tag{2.1}$$

где $C(x_0, y_0, z_0)$ - центр координат, $p(x, y, z)$ - рассматриваемая точка.

Подставляя формулу 2.1 в формулу расстояния 2.2 можно получить SDF сферы.

$$dist = length(x, y, z) - r \tag{2.2}$$

Для куба необходимо учесть 3 расположения точки относительно граней и получить формулу 2.3 SDF куба.

$$\begin{cases} q = abs(p) - r \\ dist = length(max(q, 0.0)) + min(max(q.x, max(q.y,q.z)), 0.0) \end{cases} \tag{2.3}$$

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.

Данные SDF позволяют эффективно использовать алгоритм маршировки лучей, так как луч двигается к рассматриваемому объекту с максимальным шагом, который не приведёт к пропуску лицевой грани объекта.

SDF используется в моделировании тел CSG. Алгоритмы композиции необходимо использовать, чтобы связать методы создания тел и визуализации

При пересечении двух тел (см. формулу 2.4) луч должен пересечься с тем, которое дальше от камеры, следовательно, необходимо выбрать SDF с наибольшим значением.

$$intersection(sdf1, sdf2, x, y, z) = max(sdf1(x, y, z), sdf2(x, y, z)) \quad (2.4)$$

где $sdf1$, $sdf2$ - функции, возвращающие расстояние до точки с координатами x, y, z.

Для объединения (см. формулу 2.5) тел луч должен пересечься с ближайшим телом, следовательно, нужно выбрать SDF с минимальным значением.

$$union(sdf1, sdf2, x, y, z) = min(sdf1(x, y, z), sdf2(x, y, z)) \quad (2.5)$$

В методе CSG имеет важность порядок параметров при поиске разности, которая является некоммутативной операцией. Необходимо определить порядок тел - операндов и найти максимальной расстояние между первым и вторым телом. Расстояние до второго тела берётся с отрицательным знаком. Луч должен пересечься с первым телом и не пересечься с вторым, следовательно, второе тело можно представить в виде всего пространства за пределами второго тела (см. формулу 2.6) с помощью инвертирования.

$$invert(sdf, x, y, z) = -sdf(x, y, z) \quad (2.6)$$

Теперь можно найти пересечние (см. формулу 2.7) первого тела и инвертированного второго:

$$diff(sdf1, sdf2, x, y, z) = union(sdf1(x, y, z), invert(sdf2, x, y, z)) = \\ min(sdf1(x, y, z), -sdf2(x, y, z))) \quad (2.7)$$

2.2 Алгоритм raymarching

Выбранный алгоритм raymarching отрисовывает объекты с помощью SDF. Маршировка лучей предполагает иттеративное перемещение точки

вдоль луча обзора (от камеры к объекту) и проверку результата: отрицательный знак возвращенного значения показывает столкновение с объектом.

$Raymarching$ использует границы сцены, которые должны передаваться в качестве входных данных алгоритму. Алгоритм $raymarching$ представлен на псевдокоде 1.

Псевдокод 1 Алгоритм маршировки лучей

```
1:  Функция rayMarch(start, end)
2:      depth ← start
3:      i ← 0
4:      До тех пока i < MAX_STEPS выполнить
5:          dist ← 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

В данном пункте приведена схема частей приложения, а также подробно рассмотрено моделирование тела.

Всё приложение строится из частей, приведённых на рисунке 2.1.

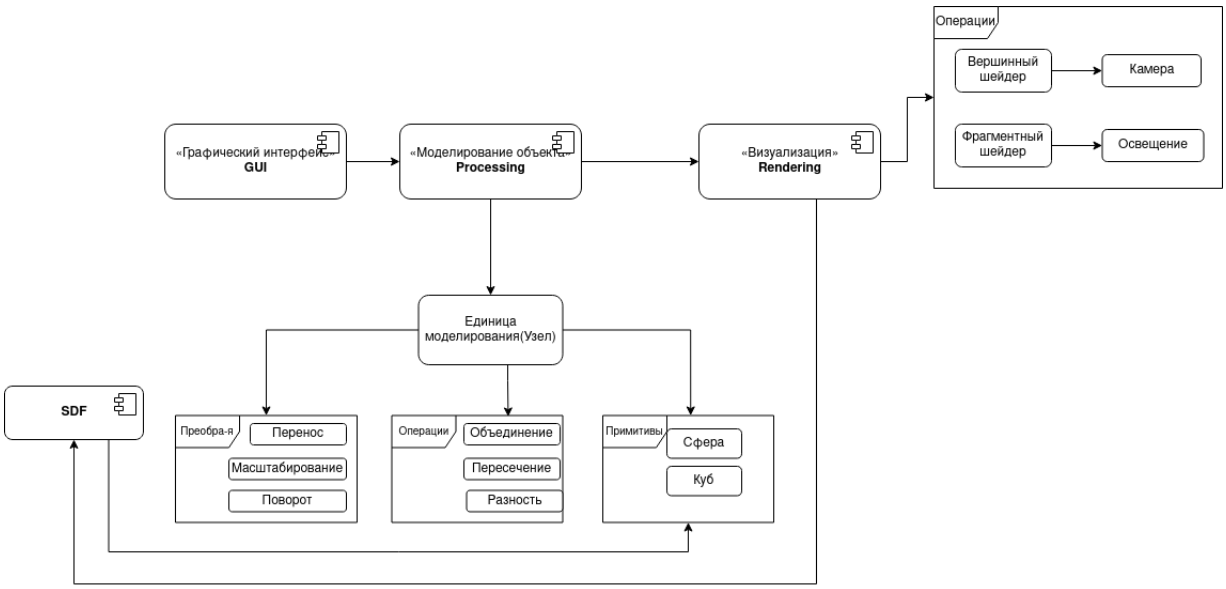

Рисунок 2.1 — Схема приложения

На схеме представлено приложение, разделенное на функциональные части. Пользователь может задать входные данные в графическом интерфейсе (GUI): примитивы, типы операций над ними, положение камеры. Выполнение отрисовки требует предварительного моделирования тела ($Processing$), которое описано рисунком 2.2, после которого следует этап отрисовки, использующий шейдеры для установки положения камеры и выставления теней ($Rendering$). Для этапа моделирования были разработаны классы, которые представлены на рисунке 2.2.

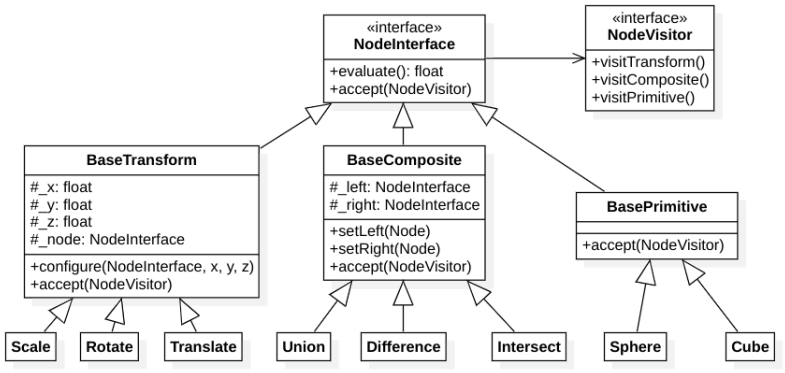

Рисунок 2.2 – Диаграмма классов Processing

Для создания объектов в CSG используется двоичное дерево. Данное дерево состоит из узлов, которые представлены в виде классов $Node$, которые являются реализацией интерфейса - $BaseTransform$, $BaseComposite$, $BasePrimitive$. Класс $NodeInterface$ задаёт интерфейс для операций преобразования тел. Все операции над моделью разделены на три группы: операции преобразования, операции композиции, операции создания примитивов.

Класс $BaseTransform$ предназначен для преобразования тела. Он содержит координаты для перемещения/масштабирования/поворота модели. Метод $configure$ конструирует экземпляр класса, принимая на вход модель - узел и координаты. Класс $BaseComposite$ предназначен для конструирования модели с помощью CSG. Он представляет собой дерево. Результатом обхода является модель. Класс $BasePrimitive$ предназначен для создания примитивов, из которых происходит построение модели. Класс $NodeVisitor$ реализует паттерн проектирования "Посетитель"и необходим для добавления нового функционала классам. В разрабатываемом ПО этот паттерн позволяет определить, какой класс представлен в $NodeInterface$, т.к его использование интерфейса подразумевает абстрагирование от наследуемых реализаций. Обход происходит с помощью методов $visitTransform$, $visitComposite$, $visitPrimitive$.

2.4 Вывод

В данном разделе были рассмотренны методы и алгоритмы, необходимые для создания приложения. Была приведена схема приложения и спроектирована диаграмма классов моделирования тела.

3 Технологический раздел

В данном разделе представленны средства разработки программного обеспечения, детали реализации и тестирование функций.

3.1 Средства реализации

В качестве языка программирования, на котором будет реализовано программное обеспечение, выбран язык программирования JavaScript [18]. Выбор языка обусловлен тем, что данный язык является языком программирования для бразуера, что позволяет запускать приложение в браузере и делает его кроссплатформенным решением, а также позволяет запускать приложение без установки дополнительных зависимостей. Помимо этого, для JavaScript существует бибиотека ThreeJS [19], которая предоставляет canvas (холст), на котором происходит отрисовка сцены, модуль для работы с камерой, а также позволяет подключить шейдеры, используя небольшое количество подготовительных этапов. Для создания пользовательского интерфейса программного обеспечения будет использоваться модуль dat-gui [20]. Этот модуль позволяет предоставить пользователю возможность изменять параметры модели. Для мониторинга производительности будет использоваться модуль Stats [21]. Этот модуль позволяет отслеживать FPS (количество кадров в секунду). Данная информация поволит оценить производительность ПО. Функциональное тестирование ПО проводиться не будет из-за своей специфики – разработанное ПО является GUI-приложением, что усложняет процесс тестирования. В качестве среды разработки выбран текстовый редактор Visual Studio Code [22], содержащий большое количеством плагинов и инструментов для различных языков программирования, в том числе JavaScript. Такие инструменты облегчают и ускоряют процесс разработки программного обеспеченияd. Для запуска приложения используется python сервер [23], позволяющий использовать ПО в браузере.

3.2 Детали реализации

В листингах 3.1 – 3.3 приведен исходный код реализации алгоритмов для преобразования тел и отрисовки на сцене модели. Алгоритмы отрисовки модели были разделены на подпрограммы: получение расстояния до каждого из объектов (куб, цилиндр, сфера), получение расстояния до композии объектов, пускание луча.

Листинг 3.1 — Реализация алгоритмов композиции тел

```
/*
 * Операция пересечения
 * distA, distB: расстояние до объектов
 *
 * Функция возвращает максимальное расстояние из двух
 */
float intersect(float distA, float distB) {
  return max(distA, distB);
}
/*
* Операция объединения
* distA, distB: расстояние до объектов
*
* Функция возвращает минимальное расстояние из двух
*/
float union(float distA, float distB) {
  return min(distA, distB);
}
/*
* Операция разности
* distA, distB: расстояние до объектов
*
* Функция возвращает максимальное расстояние из двух.
*/
float difference(float distA, float distB) {
  return max(distA, -distB);
}
```

Листинг 3.2 — Реализация алгоритмов преобразования тела

```
/*
* Операция переноса
*/
vec3 translate(vec3 p, vec3 v) {
  return p - v;
}
/*
* Операция поворота
*/
vec3 rotate(vec3 p, vec3 rad) {
  float x, y, z = rad.x, rad.y, rad.z;
  mat3 m = mat3(
    cos(y)*cos(z),
    sin(x)*sin(y)*cos(z) - cos(x)*sin(z),
    cos(x)*sin(y)*cos(z) + sin(x)*sin(z),

    cos(y)*sin(z),
    sin(x)*sin(y)*sin(z) + cos(x)*cos(z),
    cos(x)*sin(y)*sin(z) - sin(x)*cos(z),

    -sin(y),
    sin(x)*cos(y),
    cos(x)*cos(y)
  );
  return m * p;
}
```

Листинг 3.3 — Реализация алгоритмов пускания луча для отрисовки поверхностей

```
/*
* Функция пускания луча
*/
float getDistance(vec3 rayOrigin, vec3 rayDirection, out vec3 rayPosition, out
    vec3 normal, out bool hit) {
    float dist, depth = 0.0, 0.0;
    rayPosition = rayOrigin;

    for (int i = 0; i < 64; i++){
        dist = sceneDist(rayPosition);
        if (abs(dist) < EPS) {
            hit = true;
            break;
        }
        depth += dist;
```

```
15          rayPosition = rayOrigin + depth * rayDirection;
16      }
17      return depth;
18  }
19  /*
20  * Функция получения расстояния до объекта
21  */
22  float sceneDist(vec3 p) {
23    return distance(p);
24  }
25  /*
26  * Функция получения расстояния до конкретной сцены - разность объединения куба и цилиндра
        с сферой
27  */
28  float distance(vec3 p) {
29    float cube = boxDist(rotate(translate(p, cubePosition), cubeRotation), vec3(
        cubeScale * 2., cubeScale * 2., cubeScale * 2.));
30    float cylinder = cylinderDist(rotate(translate(p, cylinderPosition),
        cylinderRotation), cylinderScale * 0.5, cylinderScale * 4.0);
31    float sphere = sphereDist(translate(p, spherePosition), sphereScale * 1.);
32
33    return difference(union(cube, cylinder), sphere);
34  }
```

3.3 Вывод

В данном разделе были рассмотренны средства реализации программного обеспечения и листинги исходных кодов программного обеспечения, разработанного на основе алгоритмов, изученных в аналитическом разделе и изложенных в конструкторской части.

4 Экспериментальный раздел

В данном разделе будут приведены результаты работы разработанного программного обеспечения и поставлен эксперимент по сравнению производительности программы от сложности моделируемой сцены. Сложность оценивается количеством комбинируемых поверхностей.

4.1 Результаты работы программного обеспечения

На рисунке 4.1 приведено изображение - работа программы.

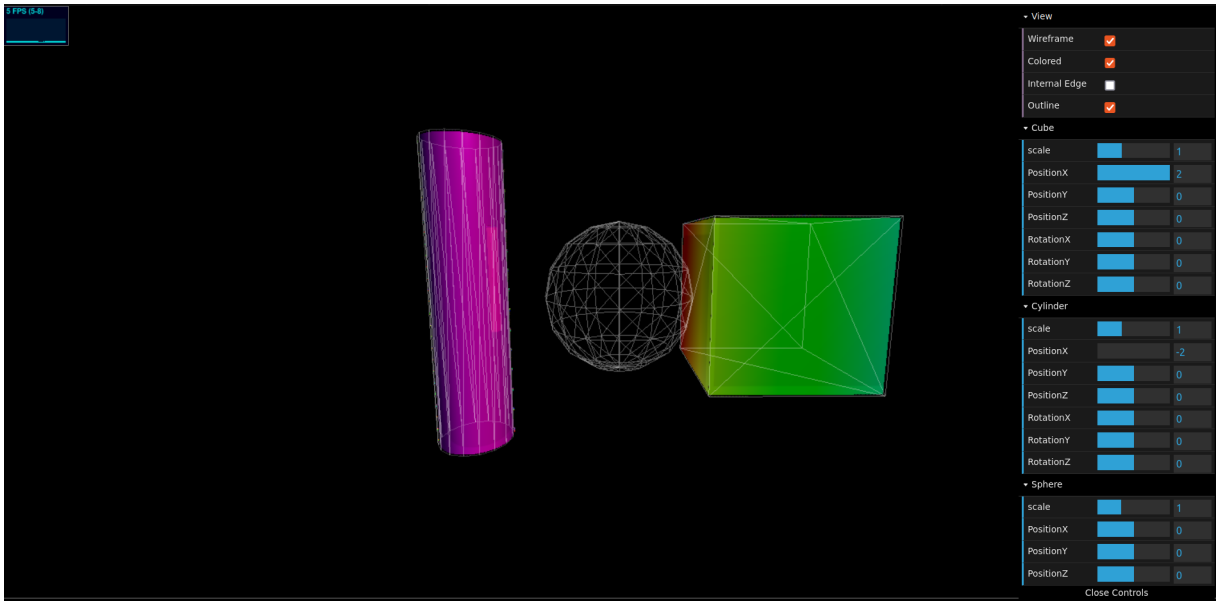

Рисунок 4.1 — Демонстрация работы программы.

На сцене изображено три объекта (куб, сфера, цилиндр). В настройках включен режим отображения каркаса, цветовая окраска.

На рисунке 4.2 приведено изображение - композиция куба, сферы и цилиндра. На этом примере производится вычитание сферы из объединения куба и цилиндра.

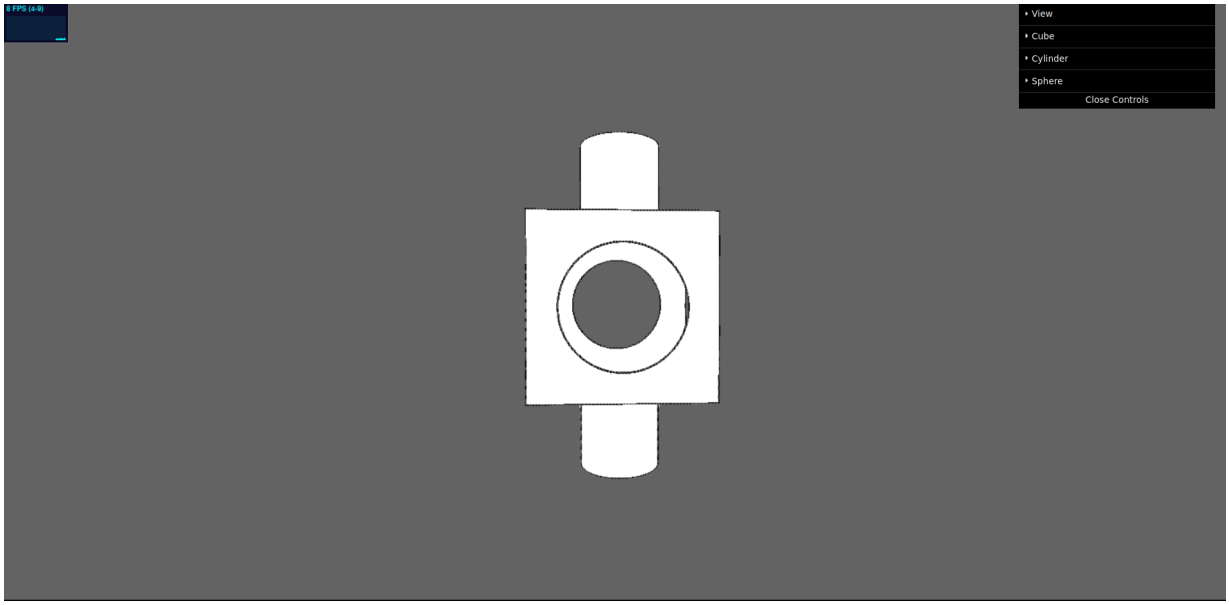

Рисунок 4.2 — Демонстрация работы программы - композиция моделей.

4.2 Постановка эксперимента

4.2.1 Цель эксперимента

Целью эксперимента является проведение тестирование производительности приложения при создании сцен разной нагруженности. Производительность будет оцениваться мерой количеством кадров в секунду (FPS), с которыми приложение работает. Нагрузка будет меняться в зависимости от количества объектов, расположенных на сцене.

4.2.2 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- Операционная система: Ubuntu 21.04 [24] Linux [25] 64-bit.
- Память: 19.4 GiB.

– Процессор: Intel Core™ i5-8300H [26] CPU @ 2.30GHz.

– Видеокарты:

  Intel(R) UHD Graphics 630 (встроенная) [27].

  NVIDIA GeForce GTX 1050 Mobile (дискретная) [28].

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

Таблица 4.1 – Сравнение FPS при выполнении на встроенной и дискретной видеокарте.

N	Количество объектов	FPS-NVIDIA	FPS-INTEL
1	1	60	15
2	2	60	12
3	3	60	12
4	4	58	10
5	5	52	8
6	6	41	8
7	7	35	6
8	8	23	5

ЗАКЛЮЧЕНИЕ

Во время выполнения практики было спроектировано программное обеспечение для моделирования твердых тел на основе примитивов (куб, сфера) с помощью логических операций (объединение, пересечение, разность). Были проанализированы и выбраны методы создания модели, их отрисовки, преобразования и аппаратной обработки, разобраны алгоритмы для программной реализации. Была разработана архитектура приложения, диаграмма классов моделирования твердого тела. При проведении работы были получены знания в области компьютерной графики и закреплены навыки проектирования программного обеспечения, а поиск оптимальных решений для эффективной работы программного обеспечения позволил улучшить навыки анализа информации.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Берлинер Э.М. Таратынов О.В. САПР конструктора машиностроителя. — М. : Форум : ИНФРА-М, 2015.

2. Solid modeling complex machining. — URL: https://www.3dsystems.com/solid-modeling-complex-machining (online; accessed: 07.07.2021).

3. ScienceDaily. 3D computer graphics. — URL: https://www.sciencedaily.com/terms/3d_computer_graphics.htm (online; accessed: 07.07.2021).

4. CPU vs. GPU Rendering: Which Is Best for Your Studio Projects? [Электронный ресурс]. — Режим доступа: https://renderpool.net/blog/cpu-vs-gpu-rendering/ (дата обращения: 07.07.2021).

5. Lecture 8 Solid Modeling.Assembly Modeling. [Электронный ресурс]. — Режим доступа: https://transport.itu.edu.tr/docs/librariesprovider99/dersnotlari/dersnotlarimak537e/notlar/8-solid-and-assembly-modeling.pdf?sfvrsn=4 (дата обращения: 10.07.2021).

6. Solid Modeling. Lectures. [Электронный ресурс]. — Режим доступа: https://www.cs.brandeis.edu/~cs155/ (дата обращения: 10.07.2021).

7. Разбиение объектного пространства сцены путём построения octree-дерева. — URL: https://gamedev.ru/articles/?id=30114 (дата обращения: 10.07.2021).

8. Основы геометрического моделирования в машиностроении. — Издательство Самарского университета, 2017.

9. The science of 3d rendering. — URL: http://digitalarchaeology.org.uk/the-science-of-3d-rendering/ (online; accessed: 11.07.2021).

10. 3D Rasterization–Unifying Rasterization and Ray Casting. — Режим доступа: https://www.researchgate.net/publication/228373400_

3D_Rasterization-Unifying_Rasterization_and_Ray_Casting (дата обращения: 11.07.2021).

11. Raster graphics. — Режим доступа: https://vector-conversions.com/vectorizing/raster_vs_vector.html (дата обращения: 11.07.2021).

12. What's the Difference Between Ray Tracing and Rasterization? [Электронный ресурс]. — Режим доступа: https://blogs.nvidia.com/blog/2018/03/19/whats-difference-between-ray-tracing-rasterization/ (дата обращения: 11.07.2021).

13. Habr. Обучение технологии ray-casting, часть 1. — URL: https://habr.com/ru/post/515256/ (дата обращения: 12.07.2021).

14. TechGeek. Как работает технология Рейтрейсинг. — URL: https://tech-geek.ru/ray-tracing/ (дата обращения: 12.07.2021).

15. Кучеренко Дмитрий. Как на самом деле работает 3D графика. — URL: https://media-xyz.com/ru/articles/1270-kak-na-samom-dele-rabotaet-3d-grafika (дата обращения: 12.07.2021).

16. Shader — это не магия. Написание шейдеров в Unity. Введение [Электронный ресурс]. — Режим доступа: https://habr.com/ru/post/473638/ (дата обращения: 16.08.2021).

17. CPU vs GPU Rendering: Uncover the Best Choice For Your Projects? [Электронный ресурс]. — Режим доступа: https://digitaladvisor.com/cpu/cpu-vs-gpu-rendering/ (дата обращения: 12.07.2021).

18. JavaScript – official site. [Электронный ресурс]. — Режим доступа: https://www.javascript.com/ (дата обращения: 29.11.2021).

19. Three JS. — Режим доступа: https://threejs.org/docs/ (дата обращения: 29.11.2021).

20. dat.GUI. A lightweight graphical user interface for changing variables in JavaScript. [Электронный ресурс]. — Режим доступа: https://github.com/dataarts/dat.gui (дата обращения: 29.11.2021).

21. JavaScript Performance Monitor. [Электронный ресурс]. — Режим доступа: https://github.com/mrdoob/stats.js (дата обращения: 29.11.2021).

22. Visual Studio Code - Code Editing. Redefined. [Электронный ресурс]. — Режим доступа: https://code.visualstudio.com (дата обращения: 29.11.2021).

23. Python 3 documentation. — Режим доступа: https://docs.python.org/3/index.html (дата обращения: 29.11.2021).

24. Ubuntu – Enterprise Open Source and Linux[Электронный ресурс]. — Режим доступа: https://ubuntu.com/ (дата обращения: 07.12.2021).

25. Linux – Википедия [Электронный ресурс]. — Режим доступа: https://ru.wikipedia.org/wiki/Linux (дата обращения: 07.12.2021).

26. 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 (дата обращения: 07.12.2021).

27. Intel(R) UHD Graphics 630. — Режим доступа https://www.techpowerup.com/gpu-specs/uhd-graphics-630.c3105 (дата обращения: 07.12.2021).

28. NVIDIA GeForce GTX 1050 Mobile. — Режим доступа https://www.techpowerup.com/gpu-specs/geforce-gtx-1050-mobile.c2917 (дата обращения: 07.12.2021).