Министерство общего и профессионального образования
Российской Федерации
УФИМСКИЙ ГОСУДАРСТВЕННЫЙ АВИАЦИОННЫЙ ТЕХНИЧЕСКИИ УНИВЕРСИТЕТ

УДК № 378.14
№ госрегистрации 01970006723
Инв. №_____

УТВЕРЖДАЮ

Проректор университета
по научной работе

_____ Н.С. Жернаков

«_____» _____ 1999 г.

ОТЧЁТ
О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

Социально-экономические проблемы подготовки военных специалистов в гражданских вузах России

по теме:

ФЕМИНИЗАЦИЯ АРМИИ КАК СОЦИАЛЬНЫЙ ПРОЦЕСС

(промежуточный)

Зам. проректора по научной работе _____ Р.А. Бадамшин

Руководитель темы _____ Г.А. Кабакович

Уфа 1999

СПИСОК ИСПОЛНИТЕЛЕЙ

Первый исполнитель _____ ФИО

Второй исполнитель _____ ФИО

РЕФЕРАТ

Отчет содержит 47 стр., 18 рис., 1 табл., 16 источн., 2 прил.

Это пример каркаса расчётно-пояснительной записки, желательный к использованию в РПЗ проекта по курсу РСОИ .

Данный опус, как и более новые версии этого документа, можно взять по адресу (`https://github.com/latex-g7-32/latex-g7-32`).

Текст в документе носит совершенно абстрактный характер.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ

В настоящем отчете о НИР 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.

Но смоделировав тело, предстаёт новая задача: нужно его визуализировать, а также предусмотреть возможность просмотра модели с разных ракурсов. На первый взгляд, решение кажется очевидным, но углубившись в тему, появляется несколько методов реализации, каждый из которых имеет свои преимущества и недостатки [3]. Следовательно, необходимо выбрать наиболее подходящий под выделенную задачу.

На данном этапе, удалось создать и визуализировать тело, но рендер сложных моделей 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 мысли создания программного обеспечения, которое объединит в себе решение всех озвученных выше задач и приведёт к конечному результату - твердотельной модели.

Целью работы является разработка программного обеспечения для моделирования сложных поверхностей на основе примитивов: кубов и сфер. Для достижения поставленной цели необходимо решить следующие задачи:

– проанализировать методы создания сложных моделей;

– проанализировать методы обработки и рендера моделей;

– реализовать алгоритмы для создания поверхностей и их рендера с возможностью изменения ракурса обзора посредством камеры;

– проверить работоспособность ПО.

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.

1.1.1 Клонирование примитивов

Схема основана на понятии семейств объектов: выделяются определённые члена семейства, отличающиеся несколькими параметрами друг от друга. Каждое семейство объектов называется общим примитивом, а отдельные объекты в семействе называются примитивными экземплярами. Например, семейство болтов является общим примитивом, а отдельный болт, заданный определенным набором параметров(количество сторон головы, шаг резьбы), является примитивным экземпляром. На рисунке ниже дано графическое пояснение примера [5].

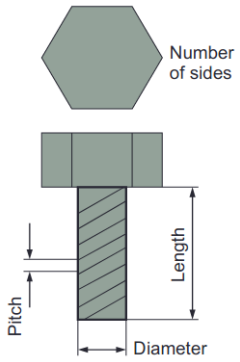

Рисунок 1.1 – Болт задан параметрами: количество сторон головы, шаг резьбы, длина, диаметр

Особенности:

– невозможно создать сложный объект, с помощью объединения экземпляров.

Минусы:

– сложность написания алгоритмов для вычисления свойств представленных твердых тел. Каждый примитив уникален. 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 зависить от типа модели: его формы, параметров. Рассмотрев схему можно заключить, что она не подходит для решения задачи. На рисунке 1.2 изображен пример, иллюстрирующий работу с схемой клонирования.

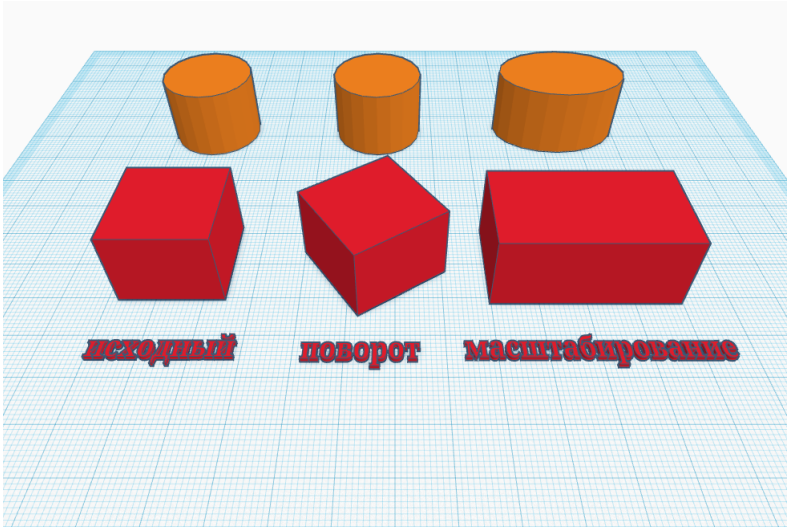

Рисунок 1.2 – Пример клонирования примитивов на примере куба и цилиндра

1.1.2 Нумерация пространственного заполнения

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

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.

На рисунке 1.3 изображен пример представления тора данной схемой.

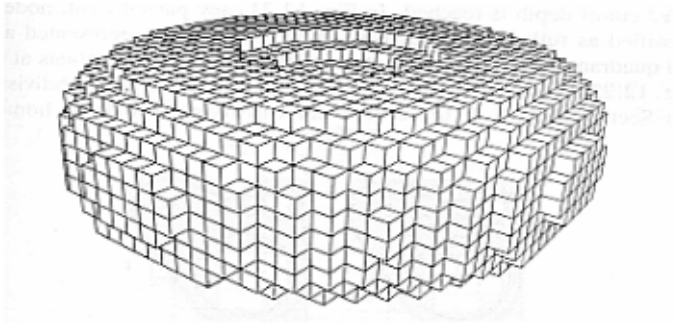

Рисунок 1.3 – Пример представления тора методом нумерации пространственного заполнения

Данная схема удобна с точки зрения простоты и точности представления, однако, структурно состоит из кубических форм, значит, 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 [6]. На рисунке 1.5 иллюстрируется создание модели путём вращения вокруг оси.

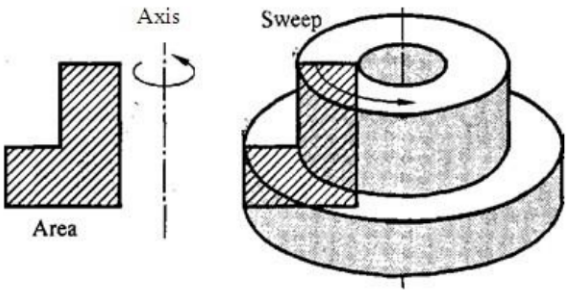

Рисунок 1.4 – Тело образовано вращением вокруг оси

Плюсы:

– простые формы удобно задавать через плоские фигуры;

– может использоваться для быстрого удаления материала внутри тела.

На рисунке 1.5 изображено удаление материала внутри тела.

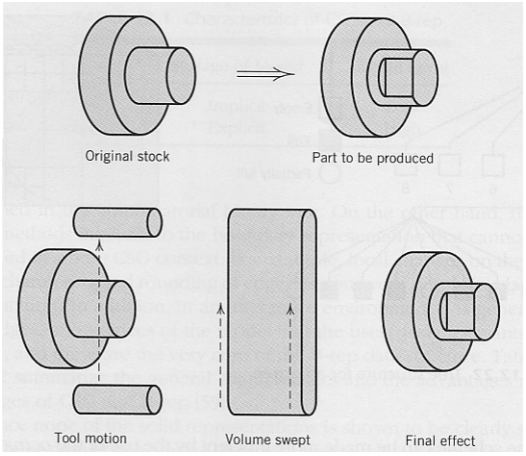

Рисунок 1.5 – Удаление материала с помощью sweeping

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.

Метод позволяет устранить недостаток предыдущего метода, так как хранит 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.

На рисунке 1.6 изображен пример разделения куба на октанты по схеме.

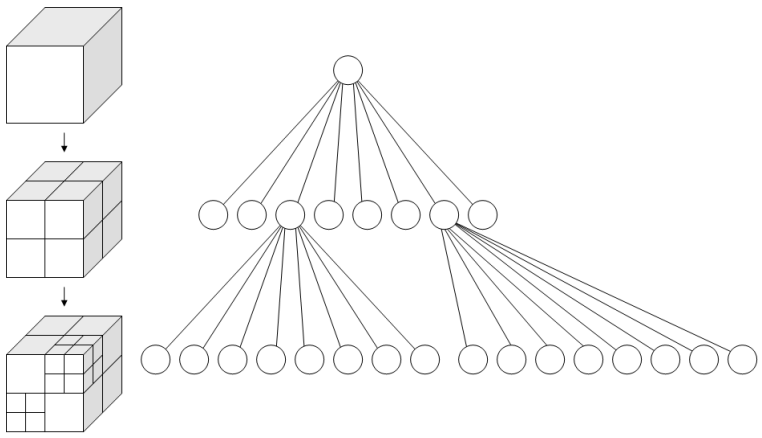

Рисунок 1.6 – Слева: Рекурсивное разделение куба на октанты. Справа: Соответствующее октодерево.

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 и признаком внешней или внутренней стороны поверхности.

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

Ниже, на рисунке 1.7 представлен пример создания модели по схеме BREP.

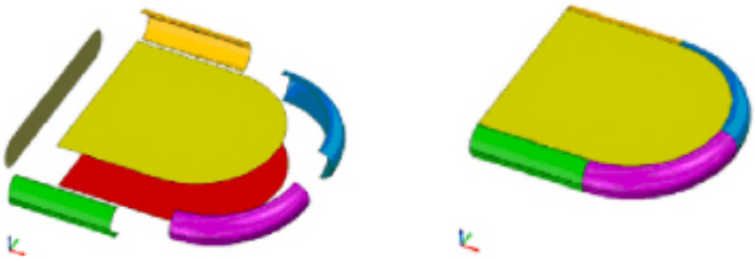

Рисунок 1.7 – BRep-представление простых твердых тел

Минусы:

– файлы тяжелые и хранят много данных, которые занимают место на диске;

– 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:

– объединение

– пересечение

– разность

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 [8]. Его вид представлен на рисунке 1.8.

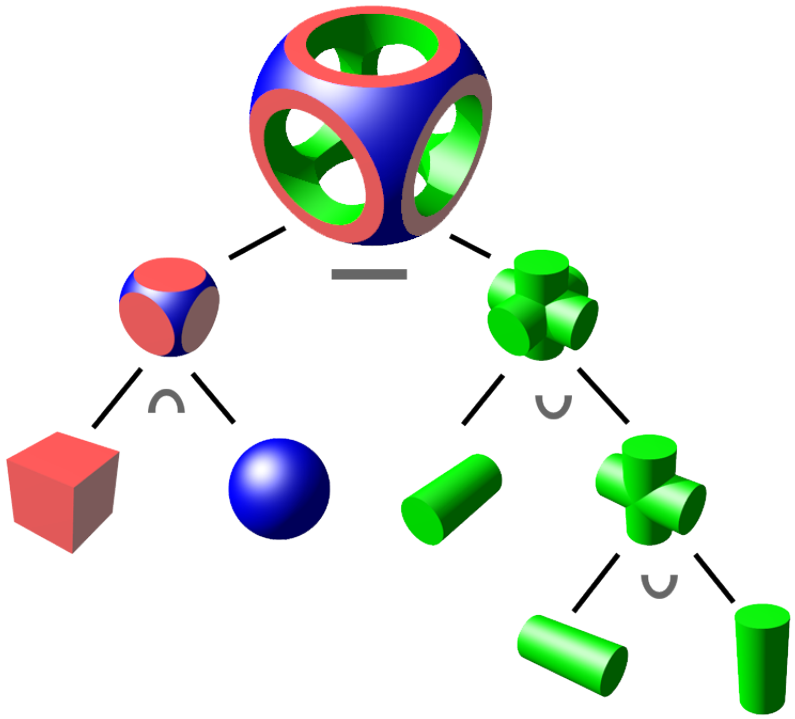

Рисунок 1.8 – Сложный объект может быть представлен двоичным деревом, где «листья» — это объекты, а узлы — операции. (пересечение, объединение, – - разность)

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.

Ниже представлена сравнительная таблица методов 1.1, в которой собраны плюсы и минусы рассмотренных схем.

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, помимо этого, 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, что имеет смысл, но для решения задачи 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 пускании лучей через вершины треугольника, который остаётся самим собой даже после попадания из трехмерного пространства в двухмерное.

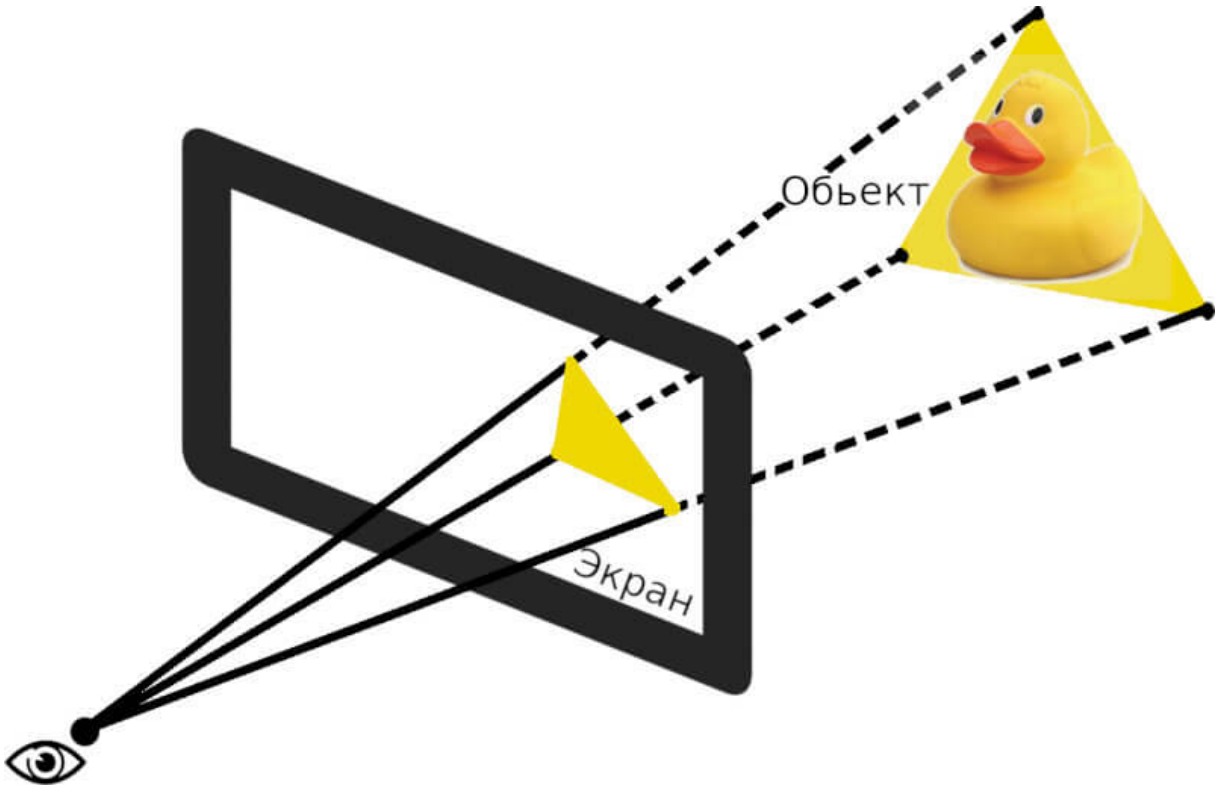

Рисунок 1.9 – Схематическое отображение предмета на экран

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

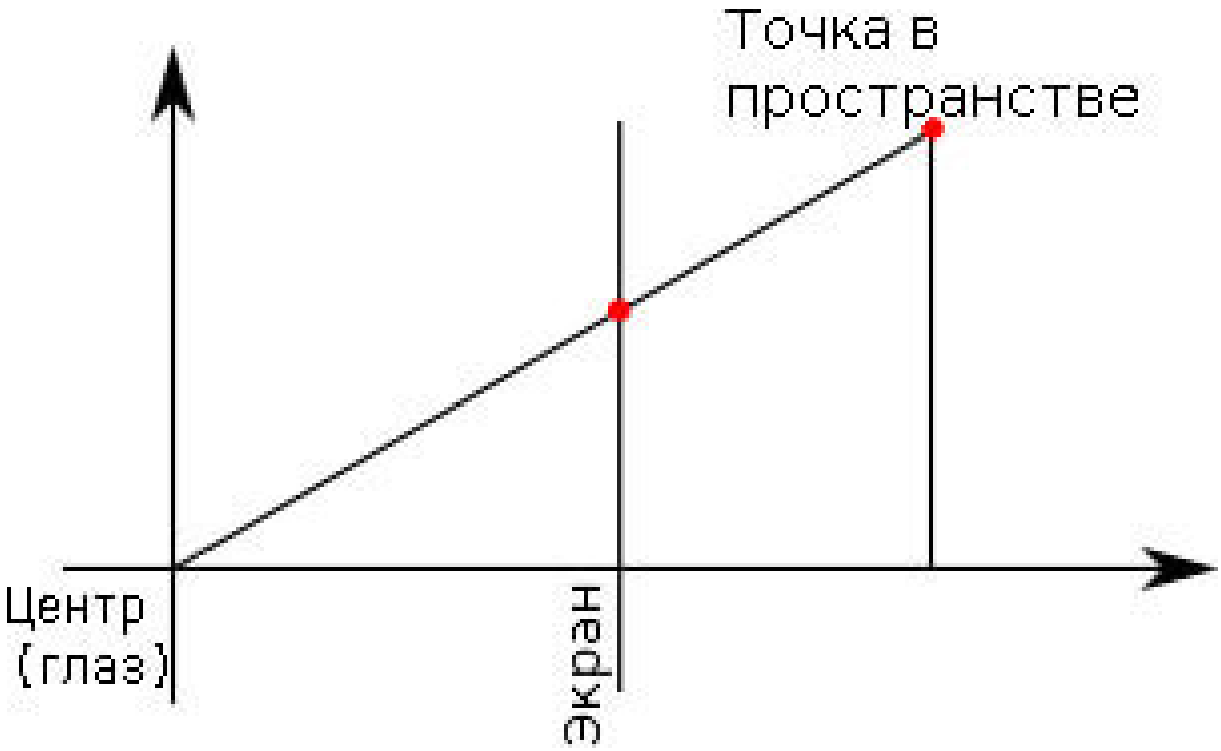

Рисунок 1.10 – Проецирование точки из пространства на плоскость экрана, вид сбоку

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;

— 3д модель должна быть описана из примитивов -> составные модели обрабатывать ресурсозатратно, т.к нужно разбить его на примитивы, обычно, треугольники.

— современные компьютеры оптимизированы для рендренга 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. Идея raycasting-а - испускать лучи из «глаз» наблюдателя, один луч на пиксель, и находить самый близкий объект, который блокирует путь распространения этого луча. Последующая обработка преломленных от объекта лучей в этом методе отсутствует. Используя свойства материала и эффект света в сцене, алгоритм рейкастинга может определить затенение данного объекта. [13]

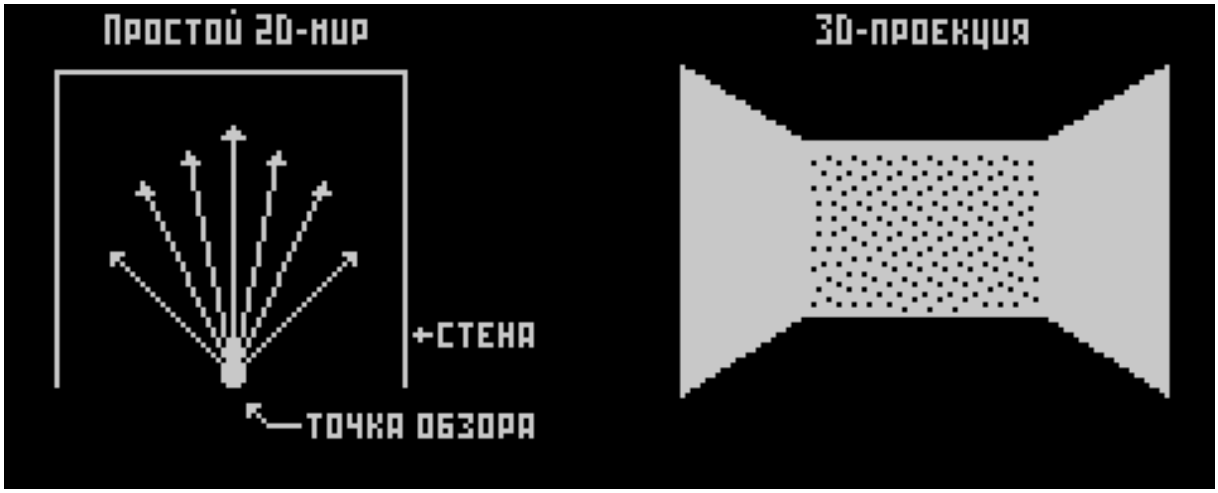

Рисунок 1.11 — На рисунке показано, как бросание преобразует что-то двумерное во что-то почти что трехмерное.

Метод использовался при создании игр в конце прошлого века, такую графику иногда называют ”псевдо 3D”или ”2.5D”. Почему? Потому что из простоты алгоритмы вытекает ряд ограничений, ведь на деле это всего лишь трёхмерная проекция плоского изображения. Ограничения алгоритмы 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 пространтсве.

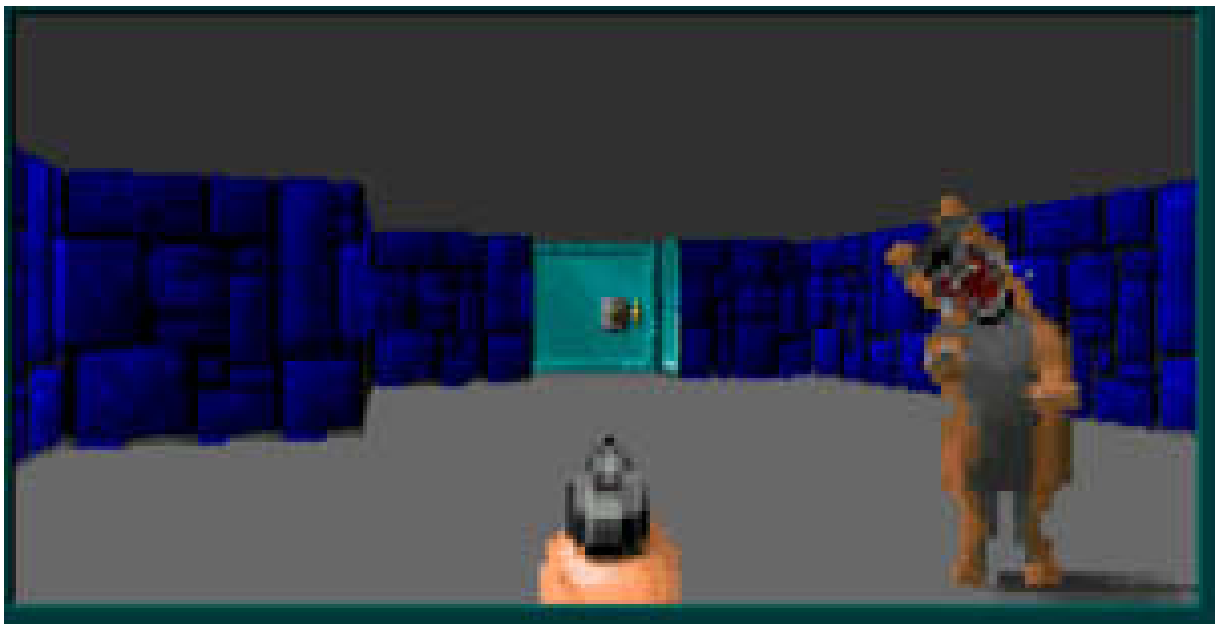

Рисунок 1.12 – Сцена из Wolfenstein 3D. Картинка делится на прямоугольные блоки. Объекты (оружие) и враги (собака) — это просто прозрачные растровые изображения, которые масштабированы и отрисованы поверх заднего фона.

Минусы метода:

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

– геомерические 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. Принцип работы трассировки лучей: Поскольку все существующие трехмерные модели собраны из треугольников, нужно было обязательно сохранить обратную совместимость. Для этого надо проверять случай столкновения луча не со стеной, а с треугольником. [14]

Пусть вершины треугольника равны $V0$, $V1$, $V2$
Векторы рёбер треугольника:

$$A = V1 - V0 \tag{1.1}$$

$$B = V2 - V0 \tag{1.2}$$

Луч задан параметрическим уравнением

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

Equation 1.1 — Параметрическое уравнение прямой

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:

$$P = V0 + u \cdot A + v \cdot B \tag{1.4}$$

Таким образом, можно написать систему из трех уравнений для координат x, y, z и решить её для t, u, v.
Если определитель 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.

Главная черта трассировки лучей в том, что одном треугольнике эта 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

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

Equation 1.2 — Система уравнений для определения пересечения прямой и плоскости

вернётся в начальную точку, или не превысит максимальное число отражений. Вся сложность алгоритма позволяет получать качественное изображения, в сравнении с полученным в результате растеризации.

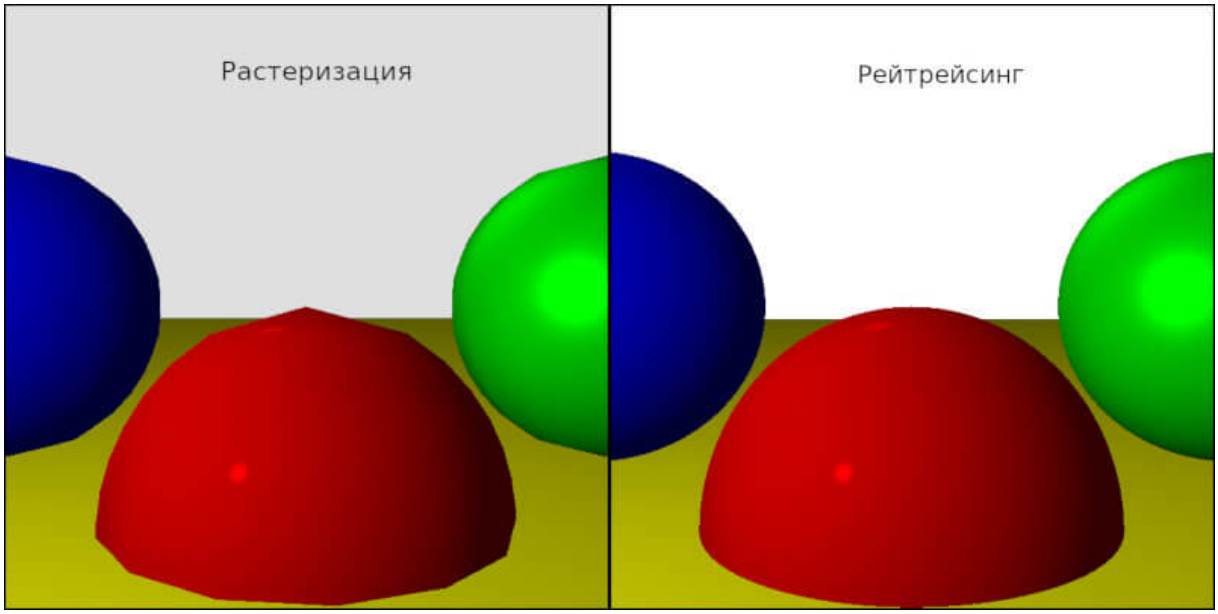

Рисунок 1.13 — Сравнение растеризации и трассировки лучей

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 у нас есть набор уравнений, определяющих пересечение луча и рендерящихся объектов. Благодаря этому можно найти объекты, которые пересекает луч (то есть объекты, которые видит камера). Также таким образом можно рендерить неполигональные объекты, например, сферы, потому что достаточно только знать формулу сферы и луча. Однако raytracing очень затратен, особенно когда в сцене есть множество объектов и сложное освещение. Кроме того, невозможно выполнять трассировку лучей через объёмные материалы, например, облака и воду. Поэтому трассировка лучей редко соответствует требованиям приложений реального времени. [15]

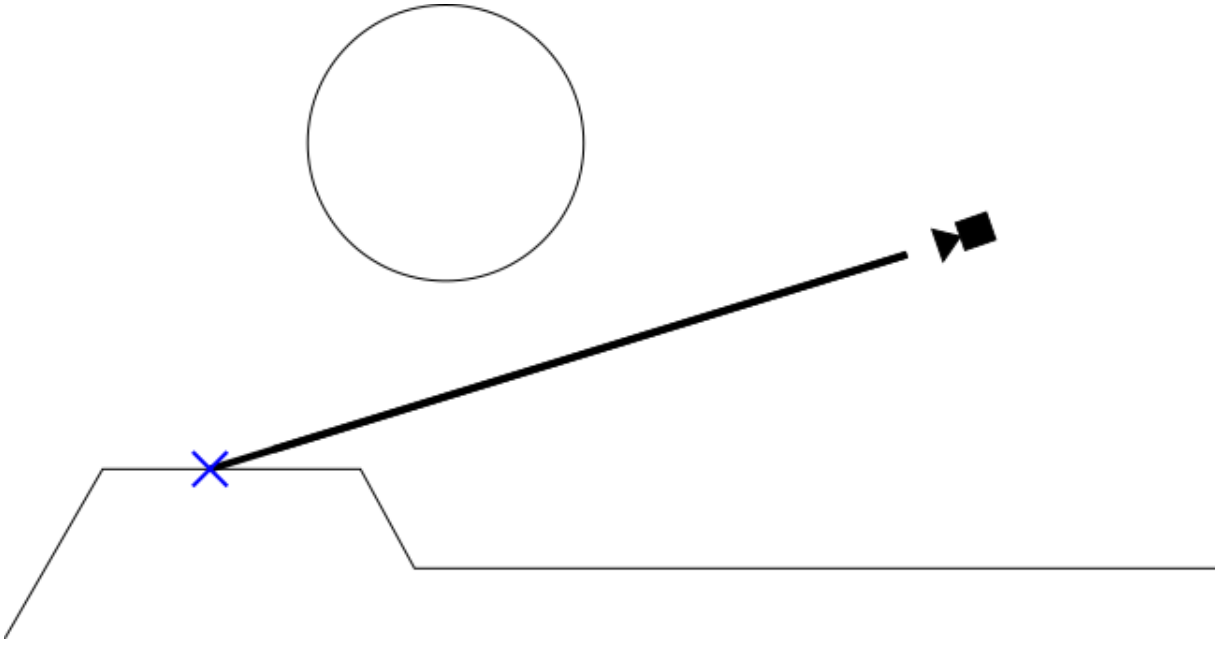

Рисунок 1.14 — упрощённое представление трассировщика лучей. Жирная чёрная линия — это пример испущенного из камеры луча для рендеринга пикселя.

Raymarching предлагает другой метод решения задачи пересечения луча и объекта, не пытается вычислить персечение аналитически. При нём мы «шагаем» точкой вдоль луча, пока не найдём, где точка пересекает объект. Эта операция является относительно простой и малозатратной, гораздо более практичной в реальном времени. Правда, на рисунке можно заметить, что этот способ менее точен, чем raytracing.

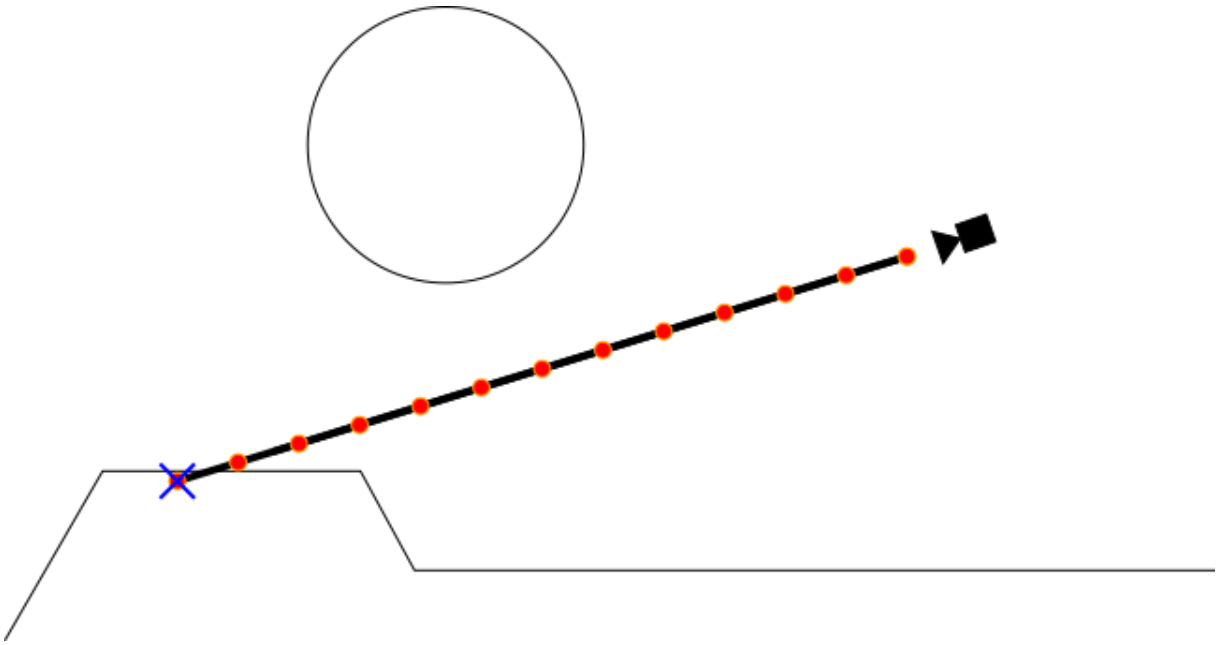

Рисунок 1.15 – простейшая реализация функции raymarching-а с фиксированным интервалом шага. Красными точками обозначены все сэмплируемые точки.

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 мы можем ввести ещё одну оптимизацию. Для этой оптимизации требуется использование полей расстояний со знаком (signed distance fields).

Поле расстояний — 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 сэмплов при движении raymarching-ом вдоль луча.

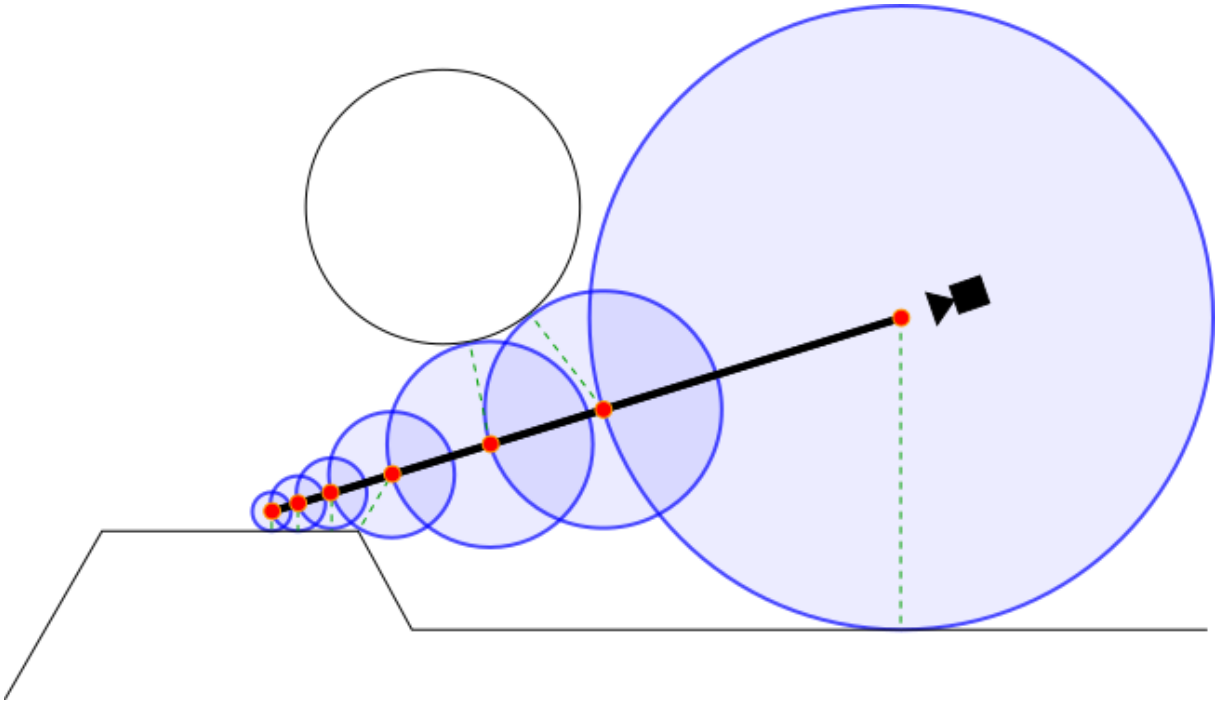

Рисунок 1.16 — визуализация raymarcher-а, использующего поля расстояний со знаком. Красными точками показаны все сэмплируемые точки. Синими кругами показаны области, которые гарантированно не содержат объектов (потому что они находятся в результатах функции поля расстояний). Пунктирными зелёными линиями показаны истинные кратчайшие векторы между каждой сэмплируемой точкой и сценой.

Алгоритм можно описать так: [16]

– Для пикселя на экране определяется расстояние до ближайшего объекта.

– Это расстояние определяет радиус на который мы можем пустить наш 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 Вывод

В данном разделе были рассмотрены методы рендера модели. Соединяя знания, полученные в предыдущем разделе, 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 Аппаратная обработка

В данном разделе предлагается рассмотреть аппаратную составляющую рендеринга. Есть 2 основных варианта: рендеринг на центральном и графичеком процессорах. Хотя движки рендеринга, основанные на CPU и GPU, имеют много общего, они обладают существенными различиями, которые оказывают огромное влияние на скорость и качество рендера. Рендеринг на базе ЦП является традиционным способом и широко используется. Напротив, рендеринг с помощью графических процессоров с годами становится все более популярным в сообществе разработчиков благодаря быстроразвивающемуся миру технологий. Теперь, зная о двух вариантах, следует понять, какой из них лучше для рендеринга?

1.3.1 CPU

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

1.3.2 GPU

GPU — графический процессор. Разработан для параллельной обработки. Графический процессор используется в широком спектре приложений, ускоряющих рендеринг 3D-графики. Хотя они наиболее известны своими игровыми возможностями, графические процессоры становятся все более популярными для использования в творческом производстве и искусственном интеллекте (ИИ). Кроме того, этот микропроцессор также используется для разгрузки некоторых задач с центрального процессора, что заставляет компьютер работать быстрее.

1.3.3 Различия

1) главное различие между микропроцессорами CPU и GPU заключается в том, как каждый из них выполняет разные задачи. В то время как ЦП выполняет различные вычисления для обработки задач, графический процессор, концентрирует все вычислительные возможности на конкретной задаче. ЦП используется для последовательной последовательной обработки в отличие от параллельной обработки или многозадачности.

2) архитектурно ЦП 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.

3) некоторое программное обеспечение имеет тенденцию перегружать процессор, особенно приложения с ”сложной”графикой, которые замедляют производительность компьютера. В данном случае GPU помогает разгрузить ЦП. Современные графические процессоры предлагают более высокую вычислительную мощность, чем традиционные процессоры. 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) между рендерингом CPU и GPU первый считается традиционным рендерером. Однако, рендеринг CPU занимает много часов, в то время как GPU делает несколько кадров в течение минут. Поскольку графические процессоры могут иметь тысячи ядер на одной карте, выдающимся преимуществом рендеринга с помощью графического процессора над рендерингом с помощью процессора является скорость.

5) GPU позволяет в реальном времени просматривать и манипулировать вашими 3d моделями, источниками света и проекциями в трех измерениях. Некоторое программное обеспечение для рендеринга, предназначенное только для графического процессора, может даже позволить вам 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.

Источник: [4]

1.3.3.1 Вывод

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

1.4 Анализ возможных решений

В данном подразделе представлен сравнительный анализ возможных решений поставленной задачи.

1.4.1 Растеризация и CSG

Самый простой способ получения результата является использования метода растеризации. Однако, это только на первый взгляд, ведь в таком случае возникает ряд проблем:

1) необходимо решить проблему с сглаживанием, чтобы получить реалистичную картинку;

2) модель нужно разбить на примитивы, т. к метод работает с объектом, составленным из треугольников.

3) дополнительная проверка на рендер теней, отражений, текстурирование.

Рассмотренные проблемы можно исправить с использованием дополнительных оптимизаций: Z-буфер, алгоритмы сглаживания, затенения, но главную проблему - производительность - исправить не получится.

1.4.2 Raycasting и CSG

Самый простой в теории и практике алгоритм трассировки лучей тоже можно использовать для решения задачи. Но возникает проблема с качеством изображения и наличием ограничений на использование, описанные в соответствующем разделе.

1.4.3 Raytracing и CSG

Улучшение предыдущего алгоритма за счёт обработки луча до возврата к камере или ограничения количества пересечений. Качество самое лучшее из всех рассмотренных методов, но для обработки в режиме реального времени, что мы хотим достичь, не подходит. Следовательно, использовать в качестве решения нецелесообразно.

1.4.4 Raymarching и CSG

Метод RayMarching берёт всё лучшее из трассировки лучей:

- качество на высоком уровне;
- изображение реалистично.

Также благодаря использованию SDF возможен рендеринг в режиме реального времени, что нам и нужно. Самое главное, потребность алгоритма в вычислительной мощности не такая большая, как в остальных вариантах, так как используется не аналитическое вычисление пересечения луча с поверхностью, а через итеративный проход до пересечения с поверхностью, что позволяет обрабатывать поверхности любой сложности без повышения сложности.

1.4.5 GPU и CPU

Ранее обсуждались преимущества и недостатки рендера с помощью имеющхся процессоров и выбор был сделан в пользу GPU, как минимум из потребности получения изображения в режиме реального времени с приемлемым количеством кадров (FPS).

1.4.6 Вывод

В данном разделе был проведен анализ возможных решений поставленной задачи. В качестве метода создания модели, при помощи которой будет решаться задача, была выбрана модель CSG, метода рендера – RayMarching, аппаратного обработчика - GPU.

2 Конструкторский раздел

В данном разделе проектируется новая всячина.

2.1 Архитектура программного обеспечения

Проверка параграфа. Вроде работает.

Вторая проверка параграфа. Опять работает.

Вот.

– Это список с «палочками».

– Хотя он и по ГОСТ, но…

1) Для списка, начинающегося с заглавной буквы, лучше список с цифрами.

Формула (2.1) совершено бессмысленна.

$$a = cb \tag{2.1}$$

А формула (2.2) имеет некоторый смысл. Кроме этого она пытается иллюстрировать применение окружения **eqndesc** которое размещает формулу совместно с её описанием. Однако обратите внимание на нумерацию формул (2.2) и (2.3), попробуйте добавить **[H]** к такой формуле.

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{+\infty} A_k \cos\left(k\frac{2\pi}{\tau}x + \theta_k\right) \tag{2.2}$$

где A_k – амплитуда k-го гармонического колебания,
A_k – амплитуда k-го гармонического колебания,
$k\frac{2\pi}{\tau} = k\omega$ – круговая частота гармонического колебания,
θ_k – начальная фаза k-го колебания.

Окружение `cases` опять работает (см. (2.3)), спасибо И. Короткову за исправления..

$$a = \begin{cases} 3x + 5y + z, \text{если хорошо} \\ 7x - 2y + 4z, \text{если плохо} \\ -6x + 3y + 2z, \text{если совсем плохо} \end{cases} \tag{2.3}$$

2.2 Разработка алгоритмов

Культурная вставка dot-файлов через утилиту dot2tex (рис. 2.1).

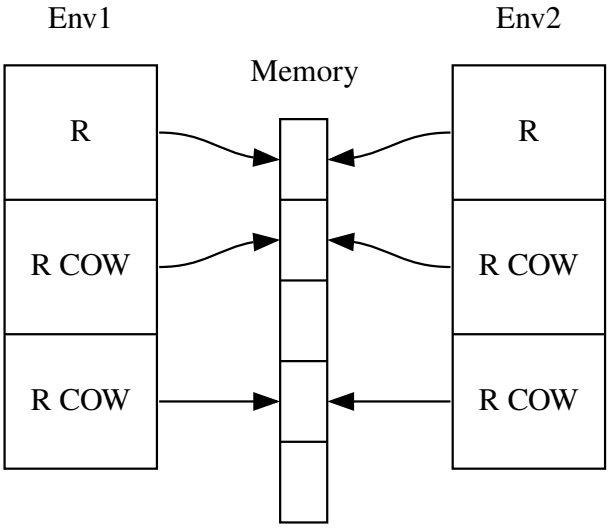

Рисунок 2.1 — Рисунок

Кстати о заголовках

У нас есть и **subsubsection**. Только лучше её не нумеровать.

3 Технологический раздел

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

3.1 Средства реализации

3.2 Детали реализации

Для вставки кода есть пакет `minted`. Он хорош всем кроме: необходимости Python (есть во всех нормальных (нет, Windows, я не про тебя) ОС) и Pygments и того, что нормально работает лишь в XeLaTeX.

Можно пользоваться расширенным BFN:

```
 letter = "A" | "B" | "C" | "D" | "E" | "F" | "G"
        | "H" | "I" | "J" | "K" | "L" | "M" | "N"
        | "O" | "P" | "Q" | "R" | "S" | "T" | "U"
        | "V" | "W" | "X" | "Y" | "Z" ;
digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;
symbol = "[" | "]" | "{" | "}" | "(" | ")" | "<" | ">"
        | "'" | '"' | "=" | "|" | "." | "," | ";" ;
character = letter | digit | symbol | "_" ;

identifier = letter , { letter | digit | "_" } ;
terminal = "'" , character , { character } , "'"
         | '"' , character , { character } , '"' ;

lhs = identifier ;
rhs = identifier
     | terminal
     | "[" , rhs , "]"
     | "{" , rhs , "}"
     | "(" , rhs , ")"
     | rhs , "|" , rhs
     | rhs , "," , rhs ;

rule = lhs , "=" , rhs , ";" ;
grammar = { rule } ;
```

Листинг 1 — EBNF определённый через EBNF

А вот в листинге 3.2 на языке C работают русские комменты. Спасибо Pygments и Minted за это.

```c
#include <stdio.h>

int main()
{
        /* Комментарий на русском с пробелами
        обратный апостроф ` */
        printf("Это строчка с пробелами и русскими буквами");

        return 0;
}
```

Листинг 2 — Пример — test.c

Можно также использовать окружение **verbatim**, если **listings** чем-то не устраивает. Только следует помнить, что табы в нём «съедаются». Существует так же команда **verbatiminput** для вставки файла.

```
a_b = a + b; // русский комментарий
if (a_b > 0)
    a_b = 0;
```

4 Экспериментальный раздел

В данном разделе проводятся вычислительные эксперименты. А на рис. 4.1 показана схема мыслительного процесса автора...

4.1 Пример работы программного обеспечения

4.2 Постановка эксперимента

4.2.1 Цель эксперимента

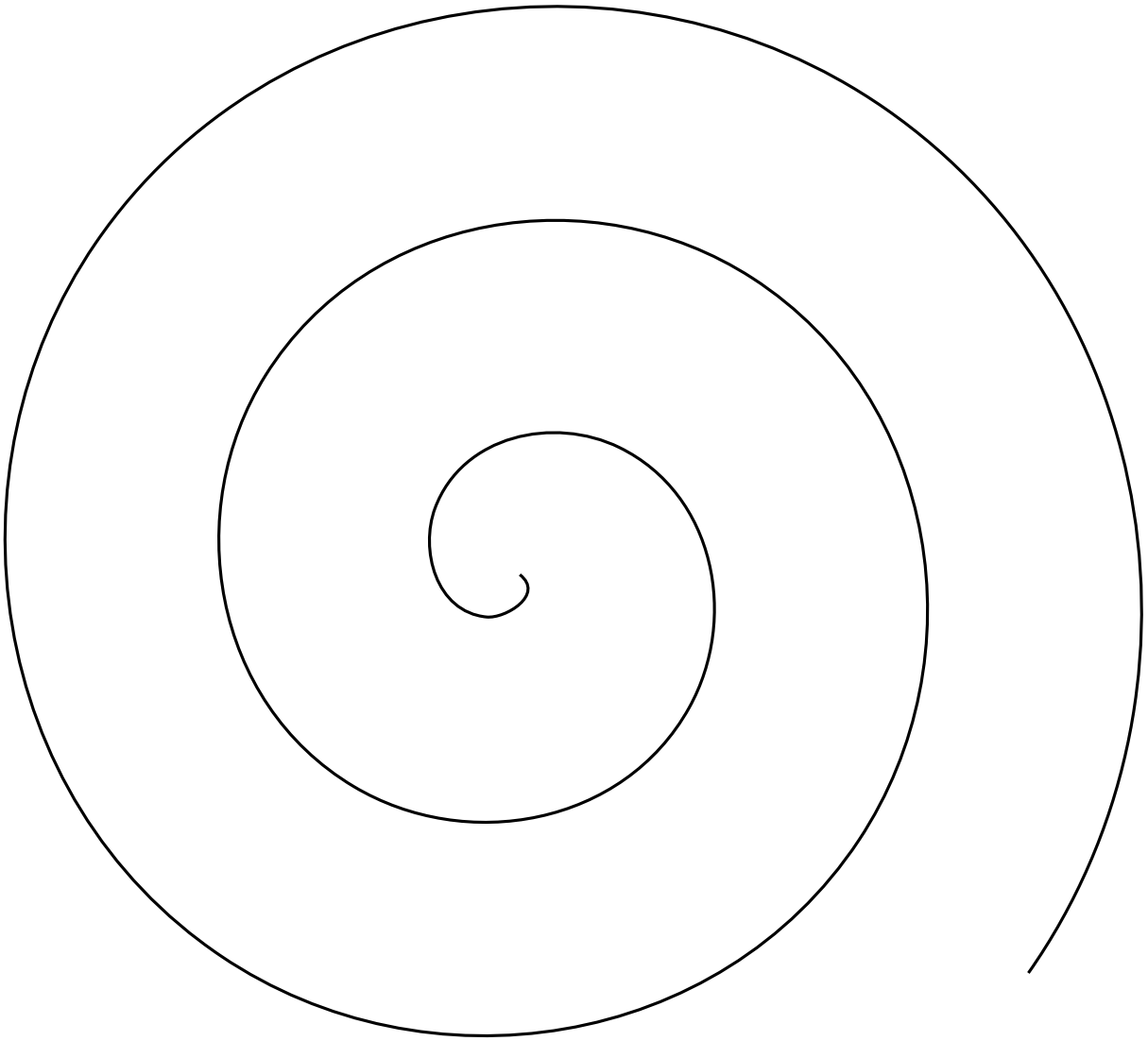

Рисунок 4.1 — Как страшно жить

ЗАКЛЮЧЕНИЕ

В результате проделанной работы стало ясно, что ничего не ясно...

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

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

2. Wikipedia. Solid-modeling. — Режим доступа: https://en.wikipedia.org/wiki/Solid_modeling (дата обращения: 07.07.2021).

3. Wikipedia. Трёхмерная графика. — Режим доступа: https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D1%91%D1%85%D0%BC%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0#%D0%A0%D0%B5%D0%BD%D0%B4%D0%B5%D1%80%D0%B8%D0%BD%D0%B3 (дата обращения: 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. — Режим доступа: https://gamedev.ru/articles/?id=30114 (дата обращения: 10.07.2021).

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

9. Wikipedia. Рендеринг. — Режим доступа: https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BD%D0%B4%D0%B5%D1%80%D0%B8%D0%BD%D0%B3 (дата обращения: 11.07.2021).

10. Wikipedia. Растеризация. — Режим доступа: https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D1%82%D0%B5%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F (дата обращения: 11.07.2021).

11. Wikipedia. Растровая графика. — Режим доступа: https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0 (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. — Режим доступа: https://habr.com/ru/post/515256/ (дата обращения: 12.07.2021).

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

15. Biagioli Adrian. Raymarching Distance Fields: Concepts and Implementation in Unity. — Access mode: http://adrianb.io/2016/10/01/raymarching.html (online; accessed: 12.07.2021).

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

ПРИЛОЖЕНИЕ А

КАРТИНКИ

ПРИЛОЖЕНИЕ Б

ЕЩЕ КАРТИНКИ