Содержание

ВВЕДЕНИЕ

В небесной механике – известна задача n тел, Которая заключается в описании относительного движения n материальных объектов, связанных друг с другом законом всемирного тяготения Ньютона. До сих пор общее решение задачи трёх тел не получено. Проблема не имеет решения в виде однозначных аналитических функций в общем случае, как, например, для двух тел [2]. При этом в различных конфигурациях тела могут образовывать интересные, красивые зацикленные или не зацикленные траектории.

Целью данной работы является разработка программного обеспечения для визуализации задачи n тел в 3-х мерном пространстве в виде анимации.

Для достижения этой цели необходимо выполнить следующие задачи:

1) Анализ физической задачи n тел и описание метода её решения;
2) Анализ существующих методов и алгоритмов для создания динамического изображения трёхмерной сцены;
3) Выбор наиболее подходящих алгоритмов для построения трёхмерной сцены;
4) Проектирование архитектуры ПО;
5) Выбор средств реализации ПО;
6) Реализация ПО и выбранных алгоритмов;
7) Проведение сравнения точного решения задачи для 2-х тел и решения, полученного программой.

1 Аналитическая часть

В данной части будет проведён анализ поставленной задачи, объектов трёхмерной сцены, а также существующих методов построения 3-х мерных сцен.

1.1 Задача n тел

1.1.1 Постановка задачи

Пусть существует N частиц, представляемых материальными точками с массами $m_1, ..., m_N$, радиус векторами $\vec{r_1}, ..., \vec{r_N}$ и векторами скоростей $\vec{v_1}, ..., \vec{v_N}$ в момент времени $t = 0$. Попарное взаимодействие точек подчинено закону тяготения ньютона:

$$\vec{F_{jk}} = G\hat{\vec{r_{jk}}}\frac{m_k m_j}{\vec{r_{jk}}^2}, \tag{1.1}$$

где

— $\vec{F_{jk}}$ - Сила тяготения, действующая j-ой частицей на k-ю;

— $G = 6.67430 * 10^{-11} m^3 s^{-2} kg^{-1}$ - гравитационная постоянная, константа;

— $\vec{r_{jk}} = \vec{r_j} - \vec{r_k}$ - вектор из точки k-ой частицы в точку j-ой;

— $\hat{\vec{r_{jk}}}$ - единичный вектор из точки k-ой частицы в точку j-ой, задающий направление силы;

— m_k, m_j - массы k-ой и j-ой частиц соответственно;

Так как на k-ю частицу действую все остальные частицы системы с силами (1.1), а также с учётом того факта, что силы аддитивны, можно получить силу, с которой все частицы действую на k-ю:

$$\vec{F_k} = Gm_k \sum_{j=1, j\neq k}^{N} \frac{m_j}{\vec{r_{jk}}^2}\hat{\vec{r_{jk}}} \tag{1.2}$$

По второму закону Ньютона можно связать силу действующую на частицу с её ускорением:

$$m\frac{d^2\vec{r}}{dt^2} = \vec{F}, \tag{1.3}$$

где

— m - масса частицы;

— $\vec{r}$ - радиус-вектор частицы;

— $\vec{F}$ - полная сила, действующая на частицу;

— t - интервал времени.

Тогда объединив уравнения (1.2) и (1.3) получим уравнение движения k-ой частицы:

$$m_k\frac{d^2\vec{r_k}}{dt^2} = Gm_k \sum_{j=1, j\neq k}^{N} \frac{m_j}{\vec{r_{jk}}^2}\hat{\vec{r_{jk}}} \tag{1.4}$$

или

$$\frac{d^2\vec{r_k}}{dt^2} = G \sum_{j=1, j\neq k}^{N} \frac{m_j}{\vec{r_{jk}}^2} \hat{\vec{r_{jk}}} \tag{1.5}$$

Дифференциальные уравнения второго порядка вида (1.5) всех частиц объединяются в систему уравнений:

$$\begin{cases} \frac{d^2\vec{r_1}}{dt^2} = G \sum_{j=1, j\neq 1}^{N} \frac{m_j}{\vec{r_{j1}}^2} \hat{\vec{r_{j1}}} \\ \dots \\ \frac{d^2\vec{r_N}}{dt^2} = G \sum_{j=1, j\neq N}^{N} \frac{m_j}{\vec{r_{jN}}^2} \hat{\vec{r_{jN}}} \end{cases} \tag{1.6}$$

Задачей N тел является решение системы уравнений (1.6) движения частиц с течением времени.

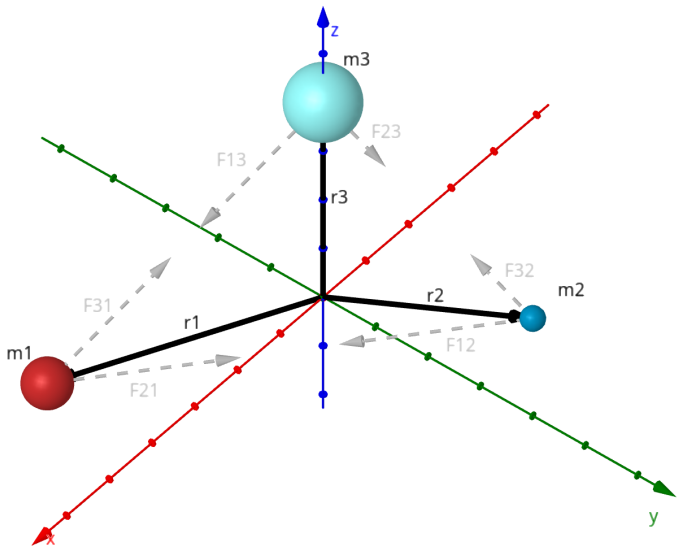

Рисунок 1.1 — "Пример задачи n тел для трёх тел"

1.1.2 Аналитические решения

Для случаев $N = 1$ и $N = 2$ существуют точные аналитические решения, при этом для $N = 1$ по законам инерции частица будет двигаться прямолинейно с постоянной скоростью, а в случае $N = 2$ тела будут двигать по одной из кривых второго порядка: эллипсу, гиперболе или параболе.

Для случая $N = 3$ Г.Э. Брунс и А. Пуанкаре показали, что решение общее решение задачи невозможно выразить через алгебраические или через однозначные трансцендентные функции координат и скоростей тел [1]. Однако в 1912 году Зульдман нашёл точное решение

задачи, но выраженный в виде бесконечных рядов, использовать которые, однако, невозможно в следствие слишком высокой скорости роста кол-ва вычислений [1].

Для случаев $N \geq 3$ на текущий момент времени не было найдено точного аналитического решения ни в какой форме, однако существуют решения для отдельных конфигураций [2].

1.1.3 Метод Эйлера

Поскольку основная задача данной работы визуализация, а не поиск точного решения задачи n тел, то для решения системы уравнений (1.6) будут использованы численные методы интегрирования. Одним из таких является метод эйлера, который и будет использоваться в данной работе [3] [4].

Метод эйлера: для решения задачи Коши вида:

$$\frac{dy}{dt} = f(t, y(t)), 0 \leq t \leq T, y(0) = y_0 \tag{1.7}$$

Введём на отрезке интегрирования сетку $\omega_\tau = t_n = n\tau, n = 1, 2, ...$, при этом будем обозначать $y_n = y(t_n)$ - сеточную функцию. Тогда разностная схема Эйлера:

$$\frac{y_{n+1} - y_n}{\tau} = f(t_n, y_n) \tag{1.8}$$

Все значения y_i, кроме y_0 ищутся итерационно по формуле:

$$y_{n+1} = y_n + \tau f(t_n, y_n) \tag{1.9}$$

Для применения метода эйлера для решения системы уравнений (1.6) приведём к системе дифференциальных уравнений первого порядка с помощью введения новых переменных - скоростей v_i

$$\begin{cases} \frac{d\vec{r_1}}{dt} = \vec{v_1} \\ \dots \\ \frac{d\vec{r_N}}{dt} = \vec{v_1} \\ \frac{d\vec{v_1}}{dt} = G \sum_{j=1, j\neq 1}^{N} \frac{m_j}{\vec{r_{j1}}^2} \hat{\vec{r_{j1}}} \\ \dots \\ \frac{d\vec{v_N}}{dt} = G \sum_{j=1, j\neq N}^{N} \frac{m_j}{\vec{r_{jN}}^2} \hat{\vec{r_{jN}}} \end{cases} \tag{1.10}$$

Такую систему можно решить методом эйлера используя формулу:

$$\begin{cases} \vec{r}_{1(n+1)} = \vec{r}_{1(n)} + \tau\vec{v}_{1(n)} \\ \dots \\ \vec{r}_{N(n+1)} = \vec{r}_{N(n)} + \tau\vec{v}_{N(n)} \\ \vec{v}_{1(n+1)} = \vec{v}_{1(n)} + \tau G \sum_{j=1,j\neq 1}^{N} \frac{m_j}{\vec{r}^2_{j1(n)}} \hat{\vec{r}}_{j1(n)} \\ \dots \\ \vec{v}_{N(n+1)} = \vec{v}_{N(n)} + \tau G \sum_{j=1,j\neq N}^{N} \frac{m_j}{\vec{r}^2_{jN(n)}} \hat{\vec{r}}_{jN(n)} \end{cases}, \tag{1.11}$$

где $\vec{r}_k(n)$ - радиус-вектор k-ой частицы в n-ый шаг сетки, а $\vec{v}_k(n)$ - скорость k-ой частицы в n-ый шаг сетки.

1.2 Виды 3-х мерных объектов

В машинной графике существует 3 основные модели, которыми описываются стереометрические тела [10]: каркасная, поверхностная и твердотельная модели.

1) Каркасная модель - простейшая из трёх моделей. Объекты в такой модели задаются с помощью множество точек и рёбер, связывающих эти точки.

2) Поверхностная модель - модель, задаётся информация о поверхности, ограничивающей объект. Сама поверхность может задаваться аналитически или с помощью полигональной аппроксимации.

3) Твердотельная модель - модель, в которой задана поверхность и информация о том, с какой стороны находится материал.

В данной работе не требуется информация о том, с какой стороны расположен материал, однако внутренняя нормаль к поверхности требуется для некоторых алгоритмов, поэтому в данной работе будет использована твердотельная модель.

1.3 Способы представления твердотельных моделей

Существует два подхода к представлению поверхностных моделей в памяти:

— **Аналитический** - задание поверхностей с помощью описания функций и уравнений. Используются для качественного отображения поверхностей вращения и гладких функций;

— **Полигональная сетка** - задание поверхности с помощью набора многоугольников. Если с помощью многоугольников нельзя точно задать поверхность, то с помощью них она аппроксимируются.

1.4 Формализация объектов 3-х мерной сцены

Сцена состоит из объектов следующих типов:

— Объект сцены - 3-х мерное тело, передающее относительное расположение материальной точки. Задаётся с помощью полигональной сетки и представляет собой одно из платоновых тел [5]. Тело обладает как визуальными характеристиками: размер, цвет, так и характеристиками материальной точки для физической задачи: масса, положение, линейная скорость.

— Точечный источник света - точка создающий свет определённого цвета распространяющийся во все стороны равномерно. Задаётся положением и интенсивностью света.

— Камера - объект, который задаёт плоскость, на которую будет спроецирована сцена при её отображении. Задаётся положением, направлениями взгляда и вверх, а также расположение и размерами окна.

1.4.1 Представление полигональных сеток

Существуют различные способы хранения полигональных сеток в памяти [9].

Представление точка-точка

Представление "точка-точка"описывает поверхность с помощью списка точек, у каждой из которых хранится список точек, с которыми она соединена. Информация о рёбрах и гранях поверхности не хранятся явно. Представление продемонстрировано на рисунке 1.2.

Список вершин		
v0	0,0,0	v1 v5 v4 v3 v9
v1	1,0,0	v2 v6 v5 v0 v9
v2	1,1,0	v3 v7 v6 v1 v9
v3	0,1,0	v2 v6 v7 v4 v9
v4	0,0,1	v5 v0 v3 v7 v8
v5	1,0,1	v6 v1 v0 v4 v8
v6	1,1,1	v7 v2 v1 v5 v8
v7	0,1,1	v4 v3 v2 v6 v8
v8	.5,.5,1	v4 v5 v6 v7
v9	.5,.5,0	v0 v1 v2 v3

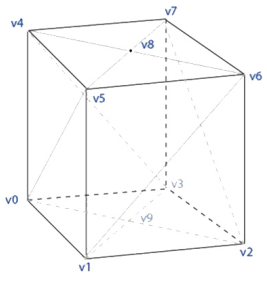

Рисунок 1.2 — "Представление точка-точка"

Представление грань-точка

Представление "грань-точка"описывает поверхность с помощью списка точек и граней. При этом для каждой грани хранятся образующие точки. Представление продемонстрировано на рисунке 1.3.

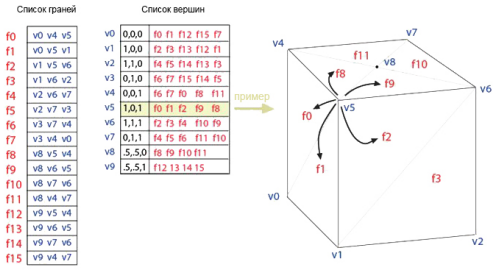

Рисунок 1.3 — "Представление грань-точка"

"Крылатое"представление

При крылатом представлении каждого ребра хранятся две точки, образующие его, две грани, которые он образует и 4 ребра, которые соединены с ним – по 2 с каждой точкой. Для каждого ребра хранятся ближайшие ребра по часовой и против часовой стрелок. Представление продемонстрировано на рисунке 1.4.

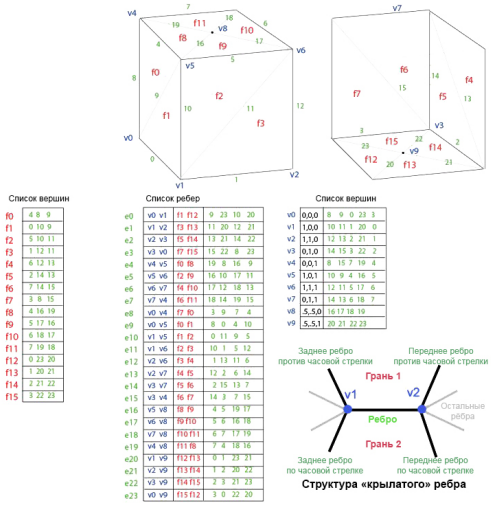

Рисунок 1.4 — "Крылатое представление"

Таблица 1.1 — Сравнение представлений полигональных сеток

Критерий	**"Точка-точка"**	**"Грань-точка"**	**"Крылатое"**
Кол-во указателей для хранения куба [9]	24	**30**	192
Получение всех граней	неявное	**явное**	явное
Получение всех рёбер	неявное	**неявное**	явное

В данной работе было выбрано представление "грань-точка так как оно позволяет получать список граней без дополнительных операций, а также не требует для хранения дополнительной информации.

1.5 Алгоритмы удаления невидимых поверхностей

Алгоритм удаления невидимых поверхностей является наиболее вычислительно сложной задачей в компьютерной графике [10]. На данном этапе определяются объекты или их части, которые буду отрисованы на экране.

Классифицировать алгоритмы можно по системе координат, в которой они работают:

— **В пространстве объектов** - имеют дело с физической системой координат, в которой заданы объекты. Точность ограничена точностью вычислений.

— **В пространстве экрана** - имеют дела с системой координат, связанной с экраном. Точность ограничена разрешением экрана.

При выборе алгоритма удаления невидимых поверхностей, важно учесть, что изображение будет строиться динамически в реальном времени и, что объекты будут иметь небольшой размер и отдалены от камеры, так как они созданы для визуализации положения материальной точки, размеры которой малы.

Будут рассмотрены 4 алгоритма удаления невидимых поверхностей:

1) алгоритм Робертса;
2) алгоритм обратной трассировки лучей;
3) алгоритм Варнока;
4) алгоритм использующий z-буфер.

1.5.1 Алгоритмы в объектном пространстве

Алгоритм Робертса

Алгоритм Робертса - первое известное решение задачи удаления невидимых поверхностей [10]. Для работы алгоритма требуются выпуклые тела. Алгоритм состоит из 4-х этапов.

Первый этап - подготовка матрицы тела. Матрица тела - матрица, хранящая информацию об уравнениях, задающих плоскости всех граней. Имеет размерность $4 * n$, где n - число граней тела.

$$V = \begin{pmatrix} a_1 & a_2 & \dots & a_n \\ b_1 & b_2 & \dots & b_n \\ c_1 & c_2 & \dots & c_n \\ d_1 & d_2 & \dots & d_n \end{pmatrix}, \tag{1.12}$$

где $a_i x + b_i y + c_i z + d_i = 0$ - уравнение i-ой грани. Матрица должна быть сформирована так, чтобы при умножению на неё точку внутри тела получался вектор положительных чисел.

Второй этап - удаление нелицевых граней. На данном этапе удаляются грани, которые экранируются самими объектами.

Третий этап - удаление рёбер, экранируемых другими телами. Для этого для каждого оставшегося ребра нужно пустить луч из точки взгляда к точке ребра. Если луч проходит через другие грани, значит эта точка отрезка экранируется.

Четвёртый этап - создание рёбер соединения граней. Для этого запоминаются все точки протыкания в предыдущем пункте и попарно соединяются отрезками. Далее эти отрезки также проверяются на экранирование, как и в 3-ем пункте.

Алгоритм обеспечивает высокую точность вычислений, но сложный вычислительно и налагает условие выпуклости на тела.

Алгоритм обратной трассировки лучей

В данном алгоритме удаление невидимых граней происходит за счёт построения луча из точки обзора через каждый пиксель экрана [7]. Таким образом на экран попадают лишь объекты, пересечение которых с лучом происходит ближе к точке обзора. Трассировку луча можно продолжить отражением и преломления по законам оптики для моделирования зеркального отражения.

1.5.2 Алгоритмы в пространстве экрана

Алгоритм Варнока

Алгоритм Варнока основан на идее, что для обработки областей с малой плотностью информации мозг человека тратит малое количество времени, а с большой плотностью — больше.

Идея алгоритма в том, чтобы разбивать экран на окна и подокна до тех пор, пока его содержимое не будет элементарным для визуализации, или пока не будет достигнут предел разрешения. На рисунке 1.5 представлен пример разбиения экрана алгоритмом Варнока.

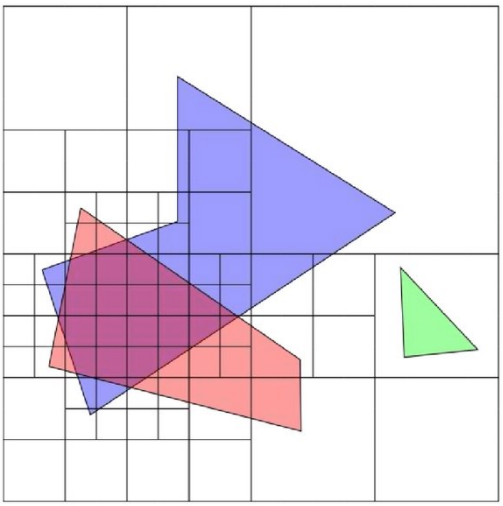

Рисунок 1.5 — "Пример работы алгоритма Варнока"

Алгоритм использующий z-буфер

Алгоритм z-буфера — один из широко используемых алгоритмов отрисовки удаления невидимых поверхностей.

Основой алгоритма является два буфера, которые имеют размеры экрана в пикселях:

— **Буфер кадра** — используется для хранения значения интенсивности(цвета) каждого пикселя, который будет отрисован на экране. По сути хранит в себе то, как экран будет выглядеть в следующем кадре.

— **Буфер глубины(z-буфер)** — Используется для хранения глубины каждого пикселя, описанного в буфере кадра. Нужен для определения видимых граней.

Предварительно алгоритм требует проецирование объекта в пространство экрана. Алгоритм растеризует объект и сохраняет его в буфере кадра и глубины, при этом, если глубина очередного пикселя больше, чем уже сохранённого в буфере, то этот пиксель пропускается.

Но при этом алгоритм требует большего объёма памяти, для хранения буферов [10], чем другие алгоритмы, но он обрабатывает только те пиксели, в которых есть объекты.

1.5.3 Выбор алгоритма удаления невидимых поверхностей

Таблица 1.2 — Сравнение алгоритмов удаления невидимых поверхностей

Критерий	**Трассировка лучей**	**Робертс**	**Варнок**	**Z-буфер**
Обрабатываются ли пустые пиксели?	Да	Нет	Да	Нет

Таблица 1.2 — Сравнение алгоритмов удаления невидимых поверхностей

Критерий	**Трассировка лучей**	**Робертс**	**Варнок**	**z-буфер**
Дополнительная память	Нет	Нет	Да, 1 буфер размером с количество пикселей	Да, 2 буфера размером с количество пикселей
Требует информацию о рёбрах тела	Нет	Да	Нет	Нет
Вычислительная сложность	$O(n_{\text{объекты}} * n_{\text{пиксели}} * n_{\text{лучи}})$	$O(n^2_{\text{объекты}})$	$O(n_{\text{объекты}} * n_{\text{подокна}})$	$O(n_{\text{объекты}} * n_{\text{пиксели проекции}})$

Для данной работы оптимальным является алгоритм z-буфера, так как он является достаточно производительным для динамических сцен, не обрабатывает пустые пиксели, которых будет больше непустых, так как объекты небольшие, а дополнительные затраты памяти не являются критическими.

1.6 Модель освещения

Модели освещения в компьютерной графике делятся на 2 больших класса: глобальные и локальные.

Глобальные модели освещения учитывают не только свет от прямых источников, но и отражённый или преломлённый от других объектов. Расчёт такой модели реализуется с помощью алгоритмов трассировок лучей, поэтому не может быть реализован в этой работе.

Локальные модели освещения учитывают только прямой свет от источника до рассматриваемого объекта. В данной работе будет использовать модель Ламберта [6], для расчёта освещения.

Отражённый свет зависит от характеристик и положения источника, относительно тела. При этом свет может быть отражён диффузно и зеркально. Зеркальное отражение не учитывается в модели ламберта. Диффузное отражённый свет испускается во все стороны равномерно. Рассчитать интенсивность отражаемого света можно по закону Ламберта [6]:

$$I = I_l k_d cos(\phi),\ 0 \leq \phi \leq \frac{\pi}{2}, \tag{1.13}$$

где

— I - интенсивность отражённого света;

— I_s - интенсивность источника света

— k_d - коэффициент диффузного отражения тела

— ϕ - угол между нормалью к поверхности в точке падения и направлением света.

В реальности на объекты падает свет от других объектов, но его точный просчёт требует

глобальной модели, поэтому в локальных моделях он заменяется на рассеянный свет:

$$I = I_a k_a + I_l k_d cos(\phi),\ 0 \leq \phi \leq \frac{\pi}{2}, \tag{1.14}$$

где I_a - интенсивность рассеянного света, а k_a - коэффициент диффузного отражения рассеянного света.

1.7 Метод учёта теней

Тени делят на проекционные и собственные. Собственные тени образуются телами на себя, проекционные - образованные другими телами.

Для учёта как проекционных, так и собственных теней, будет использован метод теневых карт [7]. Данный метод является модификацией z-буфера. Для каждого источника света строится буфер глубины по алгоритму z-буфера. Значения буфера кадра при этом отбрасываются. Затем при построении изображения из наблюдателя для проверки, что точка освещена источником, она преобразуется в его пространство и глубина точки сравнивается с соответствующим значением в буфере глубины источника. Если точка ближе, то она освещена, иначе находится в тени.

Так как в работе будут использоваться точечные источники света, то для каждого источника необходимо будет построить 6 теневых карт, которые образуют куб, описывающий источник.

Полученные тени будут выглядеть ступенчато, так как их точность ограничена разрешением теневого буфера. Для более точного учёта собственных теней будет использован метод, основанный на первых 2-х этапах алгоритма Робертса: если грань относительно источника света является нелицевой, то она находится в тени относительно этого источника света [10].

Вывод

В результате аналитической части была описана математическая модель рассматриваемой задачи, а также описан метод её решения. Были рассмотрены теоретические сведения о способах представления объектов программно и методы преобразования этих объектов для отображения на экране. Также были выбраны методы и алгоритмы, которые будут использоваться в данной работе.

2 Конструкторская часть

В данной части будут разработаны функциональные схемы разрабатываемого ПО, а также разработаны структуры и алгоритмы, которые будут использованы в ПО.

2.1 Функциональные схемы

На рисунках 2.1 – 2.3 представлено формальное описании разрабатываемого ПО в виде idef0-диаграммы.

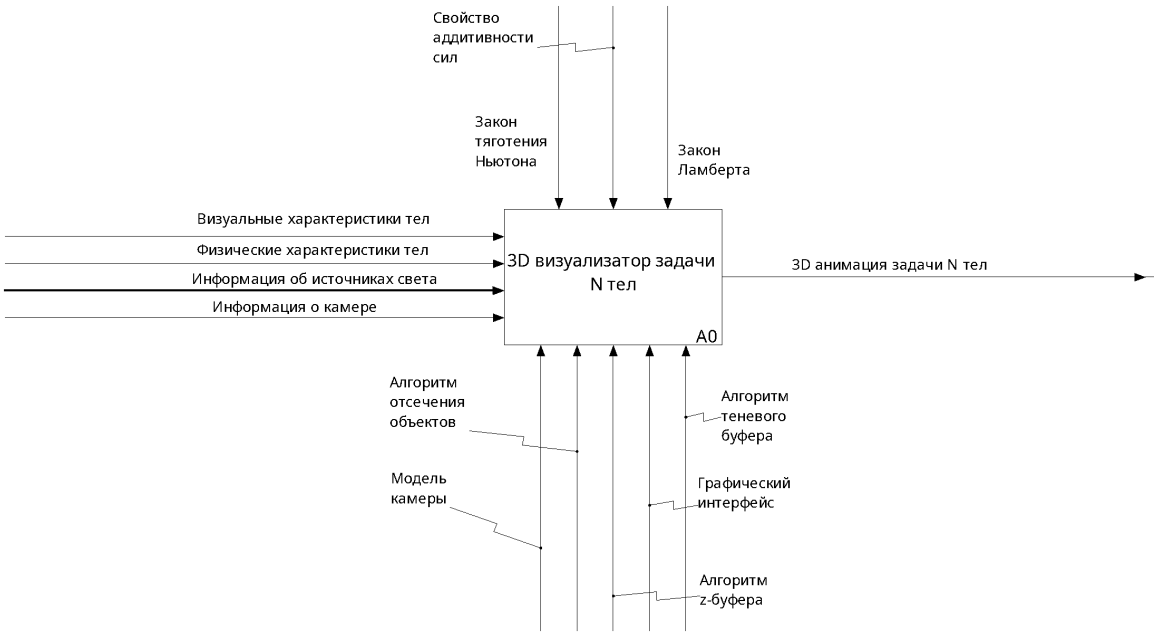

Рисунок 2.1 — "Контекстная диаграмма верхнего уровня"

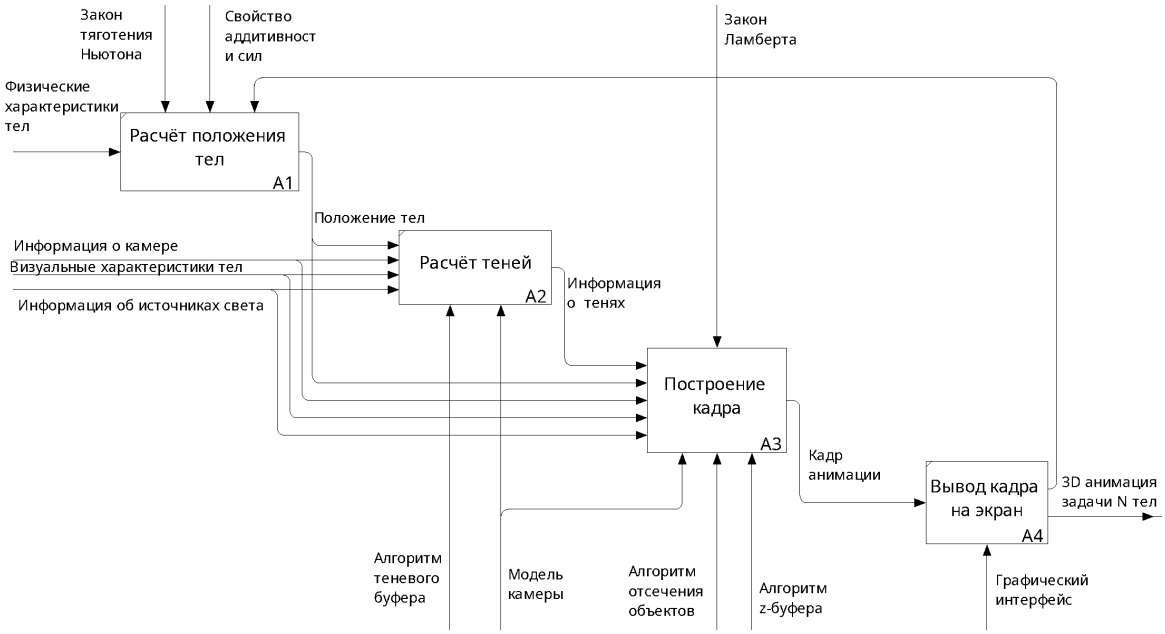

Рисунок 2.2 — "Основной цикл ПО"

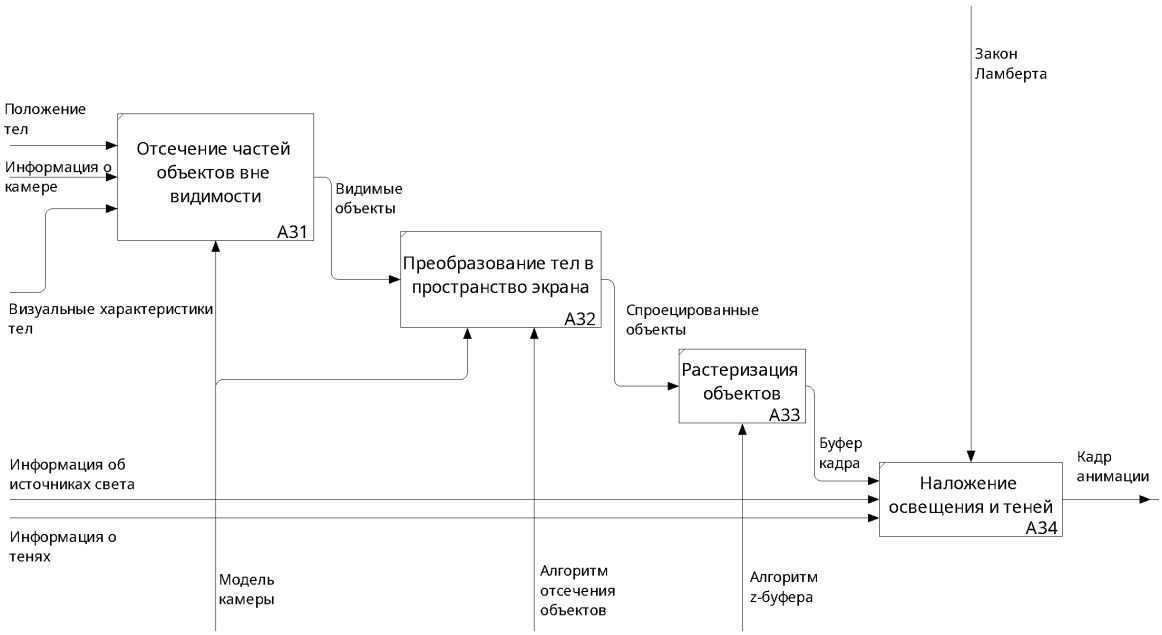

Рисунок 2.3 — "Построение кадра"

2.2 Модель камеры

Камера задаётся своим положением, направлением взгляда и направлением вверх и пирамидой видимости. Вектор в сторону получается векторным произведением вектора взгляда на вектор вверх. В пространстве камеры направление взгляда совпадает с положительным направлением оси z, а направление вверх – с положительным направлением оси y.

Пирамида видимости – усечённая 4-х гранная пирамида, уходящая в бесконечность, основание которой параллельно XoY. Основание пирамиды задаётся 3-мя параметрами:

— d – расстояние от положения камеры до основания;

— rx – полуширина основания;

— ry – полувысота основания.

Пирамида видимости в пространстве камеры продемонстрирована на рисунке 2.4.

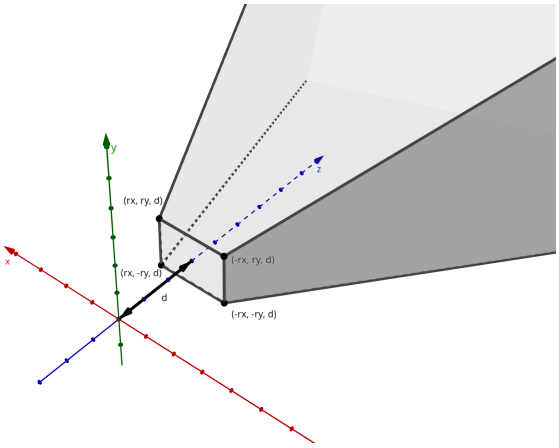

Рисунок 2.4 — "Пирамида видимости камеры"

Перспективная проекция по пирамиде видимости рассчитывается по формуле 2.1 [11].

$$\begin{pmatrix} x_p \\ y_p \\ z_p \\ w_p \end{pmatrix} = \begin{pmatrix} \frac{d}{rx} & 0 & 0 & 0 \\ 0 & \frac{d}{ry} & 0 & 0 \\ 0 & 0 & 1 & 2d \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}. \tag{2.1}$$

Преобразование к пространству камеры выполняется по формуле 2.2 [12].

$$\begin{pmatrix} x_v \\ y_v \\ z_v \\ w_v \end{pmatrix} = \begin{pmatrix} lx & ux & fx & 0 \\ ly & uy & fy & 0 \\ lz & uz & fz & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}, \tag{2.2}$$

где

— $\vec{u} = (ux, uy, uz)^T$ – вектор взгляда;
— $\vec{f} = (fx, fy, fz)^T$ – вектор вверх;
— $\vec{l} = (lx, ly, lz)^T = [\vec{u}, \vec{f}]$ – вектор влево;

2.3 Разработка типов и структур данных

В работе и для разработки алгоритмов необходимо следующие типы и структуры данных.

1) Математические абстракции
   1.1) вектор – задаётся 3-мя вещественными числами – координатами вектора;
   1.2) плоскость:
      — $normal$ – вектор внутренней (если задано направление) нормали плоскости;
      — d – свободный член в уравнение плоскости;
2) Физическое тело:
   — pos – положение тела, заданное вектором;
   — vel – вектор скорости тела;
   — $mass$ – масса;
3) Полигон:
   — $v1$, $v2$, $v3$ – вектора, задающие точки образующие полигон;
   — $normal$ – внутренняя (если задано направление) нормаль плоскости, проходящей через полигон;
4) Полигональный объект:
   — $vertices$ – массив векторов, задающих точки образующие объект;
   — $polygons$ – массив полигонов, из которых состоит объект;
   — pos – вектор, задающий положение объекта;
5) Точечный источник света:

— pos – вектор, задающий положение источника света;

— $intensity$ – интенсивность или цвет света, задаваемый 3-мя беззнаковыми числами: R, G и B, задающие соответственно красный, зелёный и синий цвета;

6) Камера:

— pos – вектор, задающий положение камеры;

— $forward$ – вектор, задающий направление взгляда;

— up – вектор, задающий направление вверх;

— px – полуширина окна камеры;

— py – полувысота окна камеры;

— d – расстояние от камеры, до окна камеры.

2.4 Разработка алгоритма расчёта положения тел

Алгоритм расчёта положения тел

Входные данные:

— Массив тел arr, каждый элемент которого задан:

- pos – точка положения тела;
- vel – вектор скорости тела;
- $mass$ – масса тела;

— Размер массива n;

— Промежуток времени dt

Выходные данные:

— Массив обновлённых тел out_arr.

Алгоритм

1) создать пустой массив тел out_arr размера n;

2) для каждого тела $body$ из массива arr:

2.1) $dp =$ произведение вектора vel на скаляр dt;

2.2) $npos = pos + dp$

2.3) $force =$ нулевой вектор;

2.4) для каждого тела $infl$ из массива arr:

2.4.1) если $infl$ не равен $body$:

2.4.2) прибавить к $force$ силу действующую на $body$ телом $infl$;

2.5) $nvel = vel+$ скалярное произведение $force$ на $\frac{dt}{mass}$;

2.6) $nbody$ – новое тело с параметрами ($npos$, $nvel$, $mass$);

2.7) добавить $nbody$ в out_arr;

3) возврат out_arr.

Алгоритм расчёта силы, действующей одним телом на другое Входные данные:

— $body$ – тело, на которое действует сила;

— $infl$ – тело, которое действует на $body$;

— каждое тело задано:

- pos – точка положения тела;
- vel – вектор скорости тела;
- $mass$ – масса тела;

Выходные данные:

— $force$ – вектор силы, действующий на $body$ со стороны $infl$.

Алгоритм

1) вектор $diff$ – разность векторов $infl.pos$ и $body.pos$;
2) $distance$ – квадрат расстояние между $body$ и $infl$ – $diff^2$;
3) $force$ – нормализованный вектор $diff$;
4) умножить $force$ на скаляр $G = 6.67430e^{-11} * infl.mass * body.mass/distance$;
5) возврат $force$.

2.5 Разработка алгоритма отсечения частей вне видимости камеры

Алгоритм отсечения полигона по плоскости

Входные данные:

— $plane$ – плоскость, заданная:

- $normal$ – внутренняя нормаль плоскости;
- d – свободный член;

— $poly$ – полигон, заданный 3-мя точками из массива $vertices$ – $v1$, $v2$, $v3$ и вектором внутренней нормали $normal$.

Выходные данные:

— $polys$ – массив отсечённых полигонов;

— $seen$ – флаг, что полигон виден целиком или частично относительно плоскости;

Алгоритм

1) создать пустой массив полигонов $polys$ размером массива 2;
2) создать пустые массивы точек see и $nsee$ размерами 3;
3) если скалярное произведение $plane.normal$ и $poly.v1$ + $plane.d$ >= 0:
   3.1) добавить $v1$ в массив see;
4) иначе:
   4.1) добавить $v1$ в массив $nsee$;
5) аналогично для $v2$ и $v3$;
6) если размер see = 0:
   6.1) вернуть $polys$, false;
7) иначе если размер see = 3:
   7.1) добавить $poly$ в $polys$
8) иначе если размер see = 2:

8.1) $v1$, $v2$, $v3$ = $see[0]$, $see[1]$, $nsee[0]$;

8.2) добавить в $polys$ полигон, образованный Алгоритм заключается в последовательном отсечении каждого полигона объекта по каждой из плоскостей.$v1$, $v2$ и точкой пересечения ребра $v1v3$ с плоскостью;

8.3) добавить в $polys$ полигон, образованный $v2$ и точками пересечений рёбер $v1v3$ и $v2v3$ с плоскостью;

9) иначе

9.1) добавить в $polys$ полигон, образованный видимой точкой и точками пересечения рёбер между видимой точки и невидимыми;

10) вернуть $polys$, true

Алгоритм отсечения частей объектов вне поля видимости камеры

При отсечении по зоне видимости используются плоскости, образующие пирамиду видимости. Каждая такая плоскость описывается парой: внутренняя нормаль и скалярное значение. Методом, описанном в [7], были найдены плоскости для используемой модели камеры:

— задняя – $((0, 0, 1), -d)$;

— левая – $((\frac{d}{px}, 0, 1), 0)$;

— правая – $((-\frac{d}{px}, 0, 1), 0)$;

— нижняя – $((0, \frac{d}{py}, 1), 0)$;

— верхняя – $((0, -\frac{d}{py}, 1), 0)$;

Алгоритм отсечения частей объектов вне поля видимости камеры заключается в последовательном отсечении каждого полигона объекта по каждой из плоскостей, по описанному выше алгоритму.

2.6 Разработка алгоритма z-буфера

2.7 Разработка алгоритма теневого буфера

Вывод

3 Технологическая часть

Вывод

4 Исследовательская часть

4.1 Вывод

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Маркеев, А. П. ЗАДАЧА ТРЕХ ТЕЛ И ЕЕ ТОЧНЫЕ РЕШЕНИЯ [Текст] / А. П. Маркеев // Соровский образовательный журнал. — 1999. — № 9. — С. 112-117.

2. Рубинштейн А.И., Городецкая Т.А., Серебренников П.С., Шипов Н.В., Шмаков А.В. ОБ ОДНОМ ЧАСТНОМ СЛУЧАЕ РЕШЕНИЯ ЗАДАЧИ ТРЕХ ТЕЛ // Проблемы современной науки и образования. - Иваново: Проблемы науки, 2017. - С. 6-9.

3. SOME THEORETICAL AND NUMERICAL ASPECTS OF THE N-BODY PROBLEM // LUND UNIVERSITY LIBRARIES. URL: https://lup.lub.lu.se/student-papers/search/publication/4780668 (дата обращения: 12.09.2024).

4. Самарский А. А. ВВЕДЕНИЕ В ЧИСЛЕННЫЕ МЕТОДЫ. - СПб: Лань, 2009. - 288 с.

5. Правильные многогранники. // Большая Российская энциклопедия URL: https://bigenc.ru/c/pravil-nye-mnogogranniki-6cab23?ysclid=m3t3ptp0y231309393 (дата обращения: 12.09.2024).

6. Казанцев А.В. Основы компьютерной графики. Часть 1. Математический аппарат компьютерной графики. - Казань: 2001. - 62 с.

7. Гэбриел Гамбетта. Компьютерная графика. Рейтрейсинг и растеризация. — СПб.: Питер, 2022. — 224 с.: ил. — (Серия «Библиотека программиста»).

8. Лекция 9. Удаление невидимых линий. // НГТУ - компьютерные голографические измерительные системы URL: http://optic.cs.nstu.ru/files/CC/CompGraph/L6_удаление невидимых линий.pdf (дата обращения: 30.11.2024).

9. On Vertex-Vertex Systems and Their Use in Geometric and Biological Modelling : Диссертация на соискание доктора технических наук / Colin Smith ; THE UNIVERSITY OF CALGARY. — Calgary, 2006. — 204 с.

10. Роджерс Д. Алгоритмические основы машинной графики [Текст] / Роджерс Д. — Москва: Мир, 1989 — 512 с.

11. OpenGL Projection Matrix // songho.ca URL: https://songho.ca/opengl/gl_projectionmatrix.html (дата обращения: 30.10.2024).

12. OpenGL Transformation // songho.ca URL: https://songho.ca/opengl/gl_transform.html (дата обращения: 30.10.2024).

13. Design Patterns // Refactoring.Guru URL: https://refactoring.guru (дата обращения: 22.10.2024).

Приложение А