Содержание

Введение

Компьютерная графика используется практически во всех научных и инженерных дисциплинах для наглядности восприятия и передачи информации. Трехмерные изображения используются в медицине, картографии, геофизике, ядерной физике и других областях.

Компьютерные модели физических явлений позволяют получить более полную информацию об изменяющихся физических величинах, построить соответствующие графики, траектории, увидеть исследуемые процессы в динамике. Последнее особо важно для формирования наглядного образа изучаемого явления.

В наше время астрономия приобрела особо важное значение. Без нее оказались бы невозможными многие достижения науки и техники, в том числе успехи человечества в освоении космоса. Для решения фундаментальных проблем астрономии и астрофизики — происхождение и эволюция звезд, строение галактик и история звездообразования, необходимо изучать звезды.

Кратная звездная система — гравитационно-связанная система из нескольких звёзд с замкнутыми орбитами. Кратность звёздной системы ограничена. Невозможно создать долгоживущую систему из трех, четырех и более равноправных звезд. Устойчивыми оказываются только иерархические системы.

Тройные звездные системы — наиболее распространённый тип кратных систем. В соответствии с иерархическим принципом тройные звездные системы обычно состоят из пары близко расположенных звезд, вращающихся вокруг друг друга, и третьей, более отдаленной, которая вращается вокруг центра масс первых двух.

Целью данной курсовой работы является создание ПО для моделирования движения звезд и планеты в тройной звездной системе.

Для достижения поставленной цели необходимо решить следующие задачи:

— выделить объекты сцены и выбрать модель их представления;

— проанализировать и выбрать алгоритмы построения реалистичного трехмерного изображения;

— разработать физическую модель поведения объектов;

— разработать программу на основе выбранных алгоритмов;

— провести исследование на основе разработанной программы.

1 Аналитический раздел

1.1 Описание моделей объектов сцены

В качестве объектов визуализируемой сцены можно выделить следующие сущности.

1. Камера — наблюдатель. Характеризуется своим пространственным положением и направлением просмотра.

2. Планета — сфера. Характеризуется положением центра, массой и радиусом.

3. Звезды — сферы, излучающие свет во все стороны (источники света). Характеризуются положением центра, массой, радиусом и интенсивностью света.

Для описания трёхмерных объектов существуют три модели: каркасная, поверхностная и объёмная. Для реализации поставленной задачи более подходящей моделью будет поверхностная. По сравнению с каркасной моделью она даст более реалистичное изображение. С другой стороны, ей требуется меньше памяти, чем объемной.

Поверхностная модель может задаваться параметрическим представлением или полигональной сеткой.

1. При параметрическом представлении поверхность можно получить при вычислении параметрической функции, что удобно при просчете поверхностей вращения.

2. В случае полигональной сетки форма объекта задается совокупностью вершин, ребер и граней.

Выбрано полигональное представление объектов, так как оно сочетается с основными алгоритмами для построения сцен.

В случае полигональной сетки форма объекта задается некоторой совокупностью вершин, ребер и граней, что позволяет выделить несколько способов представления.

1. Вершинное представление — при таком представлении вершины хранят указатели на соседние вершины. В таком случае для отрисовки нужно будет обойти все данные по списку, что может занимать достаточно много времени при переборе.

2. Список граней — при таком представлении объект хранится, как множество граней и вершин. В таком случае достаточно удобно производить различные манипуляции над данными.

3. Таблица углов — при таком представлении вершины хранятся в предопределенной таблице, такой, что обход таблицы неявно задает полигоны. Такое представление более компактное и более производительное для нахождения полигонов, однако, операции по замене достаточно медлительны.

Наиболее подходящим в условиях поставленной задачи будет представление в виде списка граней, так как оно позволяет эффективно манипулировать данными, а также проводить явный поиск вершин грани и самих граней, которые окружают вершину.

1.2 Анализ алгоритмов удаления невидимых линий и поверхностей

В данной задаче акцент делается на частоту вывода изображения на экран. Движение объектов должно выглядеть как анимация, а не просто набор кадров. Поэтому алгоритм, который будет использоваться, должен работать быстро.

1.2.1 Алгоритм, использующий z-буфер

Данный алгоритм работает в пространстве изображений.

Используются два буфера: буфер кадра для запоминания цвета каждого пикселя и буфер глубины для запоминания глубины каждого пикселя. Вначале в z-буфер заносятся минимально возможные значения z, а буфер кад-

ра заполняется значениями пикселя, описывающими фон. В процессе работы глубина (значение координаты z) каждого нового пикселя, который надо занести в буфер кадра, сравнивается с глубиной того пикселя, который уже занесён в z-буфер. Если это сравнение показывает, что новый пиксель расположен ближе к наблюдателю, чем пиксель, уже находящийся в буфере кадра, то новый пиксель заносится в буфер кадра. Кроме того, производится корректировка z-буфера: в него заносится глубина нового пикселя. Если же глубина нового пикселя меньше глубины хранящегося в буфере, то никаких действий производить не надо [1].

Преимущества:

— простота реализации;

— линейная зависимость от числа визуализируемых объектов.

Недостатки:

— трудоемкость устранения лестничного эффекта;

— большой объем требуемой памяти.

1.2.2 Алгоритм обратной трассировки лучей

Данный алгоритм работает в пространстве изображения.

Алгоритм предлагает рассмотреть следующую ситуацию: через каждый пиксел изображения проходит луч, выпущенный из камеры, и программа должна определить точку пересечения этого луча со сценой. Первичный луч — луч, выпущенный из камеры. На рисунке 1.1 представлена ситуация, когда первичный луч пересекает объект в точке H1.

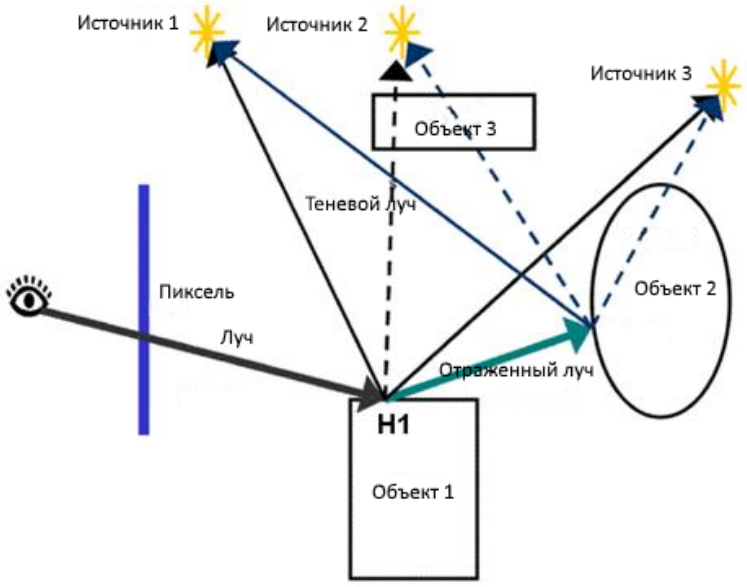

Рисунок 1.1 – Схема обратной трассировки лучей

Для источника света определяется, видна ли для него эта точка. Чтобы это сделать, испускается теневой луч из точки сцены к источнику. Если луч пересек какой-либо объект сцены, то значит, что точка находится в тени, и ее не надо освещать. В обратном случае требуется рассчитать степень освещенности точки. Затем алгоритм рассматривает отражающие свойства объекта: если они есть, то из точки H1 выпускается отраженный луч, и процедура повторяется рекурсивно. Тоже самое происходит при рассмотрении свойств преломления [2].

Преимущества:

— линейная зависимость от количества объектов на сцене;

— позволяет передавать множество разных оптических явлений (отражение, преломление).

Недостатки:

— производительность (большое количество вычислений для каждого луча).

1.2.3 Алгоритм Робертса

Данный алгоритм работает в объектном пространстве.

Алгоритм работает только с выпуклыми телами. Если тело изначально не выпуклое, то нужно его разбить на выпуклые составляющие.

Основные этапы [3].

1. Подготовка исходных данных.
2. Удаление ребер, экранируемых самим телом.
3. Удаление ребер, экранируемых другими телами.
4. Удаление линий пересечения тел, экранируемых самими телами и другими телами, связанными отношением протыкания.

Преимущества:

— точность вычислений (за счет работы в объектном пространстве).

Недостатки:

— вычислительная трудоемкость алгоритма растет как квадрат количества объектов;

— тела должны быть выпуклыми (иначе нужно разбивать на выпуклые составляющие).

1.2.4 Алгоритм Варнока

Данный алгоритм работает в пространстве изображений.

Алгоритм анализирует область на экране дисплея (окно) на наличие видимых элементов. Если в окне нет изображения, то оно просто закрашивается фоном. Если же в окне имеется элемент, то проверяется, достаточно ли он прост для визуализации. Если объект сложный, то окно разбивается на более

мелкие, для каждого из которых выполняется тест на отсутствие или простоту изображения. Рекурсивный процесс разбиения может продолжаться до тех пор, пока не будет достигнут предел разрешения экрана [4].

Преимущества:

— эффективен для простых сцен (будет немного разбиений).

Недостатки:

— неэффективен для большого количества небольших по размеру многоугольников;

— неэффективен для большого числа пересечений объектов.

1.2.5 Вывод

Проанализировав данные алгоритмы, можно прийти к выводу, что наиболее подходящим алгоритмом для данной программы будет алгоритм z-буфера. Он позволит выполнять синтез изображения достаточно быстро.

1.3 Анализ алгоритмов закрашивания

1.3.1 Простая закраска

Вся грань закрашивается с одинаковой интенсивностью, которая вычисляется по формуле (1.1):

$$I = I_0 \cdot k \cdot \cos(\alpha), \tag{1.1}$$

где I — интенсивность света в точке, I_0 — интенсивность источника света, k — коэффициент диффузного отражения, α — угол падения луча.

Метод позволяет получать изображения, сравнимые по качеству с реальными объектами, лишь при выполнении следующих условий:

— предполагается, что источник света находится в бесконечности;

— наблюдатель находится в бесконечности;

— закрашиваемая грань является реально существующей, а не полученной в результате аппроксимации поверхности.

Преимущества:

— простой в реализации;

— небольшие требования к ресурсам.

Недостатки:

— плохо подходит для тел вращения;

— плохо учитывает отраженный свет.

1.3.2 Закраска по Гуро

В данном методе используется интерполяция интенсивности [5]. Рассматривается отдельная грань, вычисляются нормали к вершинам грани. Используя нормали в вершинах, вычисляется интенсивность каждой вершины, выполняется первая линейная интерполяция вдоль ребер. Вторая интерполяция (тоже линейная) выполняется, когда вычисляется интенсивности пикселей, расположенных на сканирующей строке.

Преимущества:

— подходит для фигур вращения, аппроксимированных полигонами;

— хорошо сочетается с диффузным отражением.

Недостатки:

— могут быть потеряны ребра (получится плоское изображение).

1.3.3 Закраска по Фонгу

Данный метод схож с закраской по Гуро. Вместо интенсивности происходит интерполяция по значению самой нормали.

Преимущества:

— улучшенная аппроксимация кривизны поверхности по сравнению с Гуро;

— хорошо передает блики.

Недостатки:

— наиболее трудоемкий алгоритм.

1.3.4 Вывод

Для поставленной задачи лучше всего подойдет закраска по Гуро, так как она сможет обеспечить достаточно реалистичное изображение закругленных объектов. Так же этот алгоритм достаточно быстр.

1.4 Выбор модели освещения

Существует две модели освещения, которые используются для построения света на трёхмерных сценах: локальная и глобальная. Исходя из поставленной задачи, необходимо выбрать вариант, который позволит достаточно быстро просчитывать освещения на сцене.

1. Локальная модель является самой простой, не рассматривает процессы светового взаимодействия объектов сцены между собой и рассчитывает освещённость только самих объектов.

2. Глобальная модель рассматривает трёхмерную сцену, как единую систему и описывает освещение с учётом взаимного влияния объектов,

что позволяет рассматривать такие явления, как многократное отражение и преломление света, а также рассеянное освещение.

Для поставленной задачи лучше подходит локальная модель освещённости, так как она является более быстродействующей. Также в сцене отсутствуют объекты, обладающие зеркальными или преломляющими свойствами, поэтому использование более качественной глобальной модели не требуется.

Далее будут рассмотрены только локальные модели освещения.

1.4.1 Модель Ламберта

Модель Ламберта [6] моделирует идеальное диффузное освещение. Свет при попадании на поверхность рассеивается равномерно во все стороны. При расчете такого освещения учитывается только ориентация поверхности (нормаль) и направление на источник света.

Интенсивность можно рассчитать по формуле (1.2):

$$I_d = k_d \cdot \cos(\overrightarrow{L}, \overrightarrow{N}) \cdot i_d, \tag{1.2}$$

где I_d — рассеянная составляющая освещенности в точке, k_d — свойство материала воспринимать рассеянное освещение, i_d — мощность рассеянного освещения, $\overrightarrow{L}$ — направление из точки на источник, $\overrightarrow{N}$ — вектор нормали в точке.

1.4.2 Модель Фонга

Идея модели заключается в предположении, что освещенность каждой точки разлагается на 3 компоненты [6]: фоновое освещение (ambient), рассеянный свет (diffuse), бликовая составляющая (specular) (рисунок 1.2).

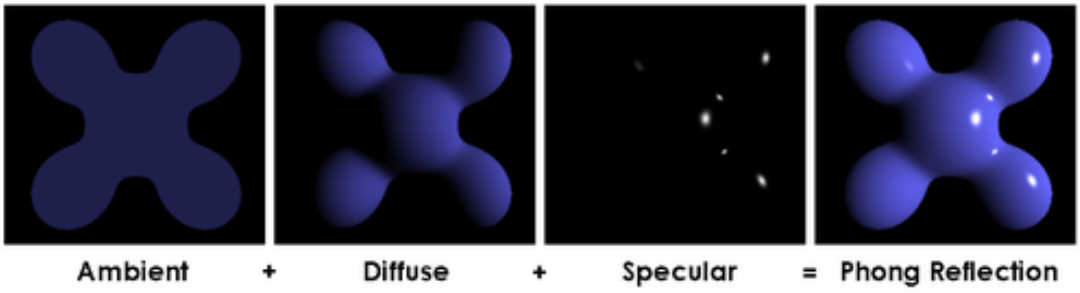

Рисунок 1.2 – Модель освещения Фонга

Свойства источника определяют мощность излучения для каждой из компонент, а свойства материала — способность объекта воспринимать свет.

Интенсивность света рассчитывается по формуле (1.3):

$$I = k_a \cdot I_a + k_d \cdot (\overrightarrow{N}, \overrightarrow{L}) + k_s \cdot (\overrightarrow{N}, \overrightarrow{V})^p, \tag{1.3}$$

где $\overrightarrow{N}$ — вектор нормали к поверхности в точке, $\overrightarrow{L}$ — направление проецирования (направление на источник света), $\overrightarrow{V}$ — направление на наблюдателя, k_a — коэффициент фонового освещения, k_s — коэффициент зеркального освещения, k_d — коэффициент диффузного освещения, p — степень блеска.

1.4.3 Вывод

Для поставленной задачи лучше всего подойдет модель Ламберта, так как программа должна иметь наиболее высокую производительность.

1.5 Текстурирование граней объектов с использованием барицентрических координат

Барицентрические координаты — это координаты, в которых точка треугольника (грани объекта) описывается как линейная комбинация вершин [1], то есть произвольная точка треугольника может быть найдена по формуле (1.4):

$$m = u \cdot p_0 + v \cdot p_1 + w \cdot p_2, \tag{1.4}$$

где u, v, w — барицентрические координаты, p_0, p_1, p_2 — вершины треугольника.

Барицентрические координаты точки неотрицательны и их сумма равна единице. Первые две координаты равны отношению площадей треугольников, которые образует точка внутри треугольника и вершины, к общей площади треугольника. Третья координата вычисляется через две известные.

Барицентрические координаты позволяют интерполировать значение любого параметра (в том числе и текстуры) в произвольной точке треугольника по формуле (1.5):

$$t = u \cdot t_0 + v \cdot t_1 + w \cdot t_2, \tag{1.5}$$

где t_0, t_1, t_2 — значения параметра в вершинах треугольника.

1.6 Физическая модель поведения объектов

В наиболее типичных тройных звездах две компоненты обычно образуют тесную бинарную систему (двойную звезду), обращаясь одна вокруг другой на сравнительно небольшом расстоянии, а третья звезда обращается вокруг тесной пары по орбите значительно большего размера. Гравитационное действие тесной пары на удаленного третьего компаньона такое же, каким было бы действие единственной массы. Третья звезда находится так далеко, что ее гравитационное поле не в состоянии повлиять сколько-нибудь значительно на устойчивое относительное движение партнеров внутренней тесной пары.

Рассмотрим движение звезд в тесной паре. Движение компонент двойных звезд происходит в соответствии с законами Кеплера: оба компонента описывают в пространстве подобные эллиптические орбиты вокруг общего центра масс, который находится в одном из фокусов каждой из орбит. Значения больших полуосей этих эллипсов обратно пропорциональны массам звезд.

Обращение в системах двойных звезд подчиняется закону всемирного тяготения Ньютона, так как законы Кеплера, как доказал еще сам Ньютон, являются следствием единого закона тяготения, выражающегося формулой (2.5):

$$F = G \cdot \frac{m_1 \cdot m_2}{R^2}, \tag{1.6}$$

где F — гравитационная сила, действующая между двумя объектами, m_1, m_2 — массы объектов, R — расстояние между центрами их масс, G — гравитационная постоянная.

Третья более удаленная звезда также движется по эллиптической орбите, в одном из фокусов которой находится центр масс тесной пары.

Теперь рассмотрим вопрос движения планеты, обращающейся вокруг третьей звезды. В данной программе будет моделироваться система, математическая модель которой соответствует ограниченной задаче трех тел: масса одного из тел (планета) пренебрежимо мала по сравнению с массами двух других тел (масса тесной пары и третья звезда). В такой системе можно не принимать во внимание влияние тела малой массы на движение двух других тел.

При описании движения близкой к третьей звезде планеты рассматривается ее движение относительно звезды, а не относительно центра масс тесной пары. Система отсчета, связанная с третьей звездой, не является инерциальной: она подвержена ускорению, направленному к центру масс тесной пары. Так как планета находится близко к своей звезде, гравитационное притяжение тесной пары сообщает ей почти такое же ускорение, как и самой третьей звезде. Поэтому влияние притяжения к тесной паре на движение планеты относительно третьей звезды оказывается незначительным: в главных чертах это движение описывается законами Кеплера.

В итоге, при расчете движения планеты вблизи третьей звезды оказывается возможным учитывать ее притяжение только этой звездой, т.е. исследовать движение планеты относительно звезды-хозяйки в рамках ограниченной задачи двух тел. Для планеты, масса которой много меньше массы звезды, это будет просто кеплерово движение в ньютоновском поле тяготения. Орбита планеты относительно третьей звезды может быть эллипсом или окружностью.

Итоговая траектория планеты в системе отсчета, связанной с центром масс тесной пары, является результатом сложения двух относительно простых движений: регулярного движения по большому эллипсу вместе с третьей звездой вокруг центра масс тесной пары, и одновременного обращения вокруг третьей звезды по малой окружности (или малому эллипсу) [7].

1.7 Вывод

В данном разделе был проведен анализ алгоритмов удаления невидимых линий и моделей освещения, которые возможно использовать для построения сцены. В качестве ключевых алгоритмов выбраны алгоритм z-буфера и алгоритм закраски по Гуро. Для представления объектов сцены выбрана поверхностная модель (путем хранения списка граней).

2 Конструкторский раздел

2.1 Аппроксимация трёхмерных объектов

В программе будет генерироваться икосфера — это многогранная сфера, состоящая из треугольников.

Алгоритм триангуляции сферы следующий.

1. Задать число сечений (параллелей) и количество разбиений для каждого сечения (меридиан).

2. На каждом сечении подсчитать углы поворота радиуса вектора для очередной точки грани по формулам следующей системы уравнений (2.1):

$$\begin{cases} \alpha = \frac{\pi}{k_1}, \\ \beta = \frac{2 \cdot \pi}{k_2}, \end{cases} \tag{2.1}$$

   где k_1, k_2 — число сечений и разбиений для каждого сечения.

3. На каждом сечении найти нужные точки, разделив окружность на равные части.

4. Соединив точки, получим сферу, разбитую на треугольники.

Координаты вершин аппроксимирующих треугольников можно рассчитать по формулам следующей системы уравнений (2.2):

$$\begin{cases} x = r * \sin(\alpha \cdot i) \cdot \cos(\beta \cdot j), i = \overline{0, k_1}, j = \overline{0, k_2 - 1} \\ y = r * \cos(\alpha \cdot i), \\ z = r * \sin(\alpha \cdot i) \cdot \sin(\beta \cdot j), \end{cases} \tag{2.2}$$

где r — радиус сферы.

2.2 Объединенный алгоритм удаления невидимых граней с использованием z-буфера и закраски по Гуро

На рисунке 2.1 приведена схема алгоритма z-буфера. Совместно с алгоритмом z-буфера применяется закраска по Гуро.

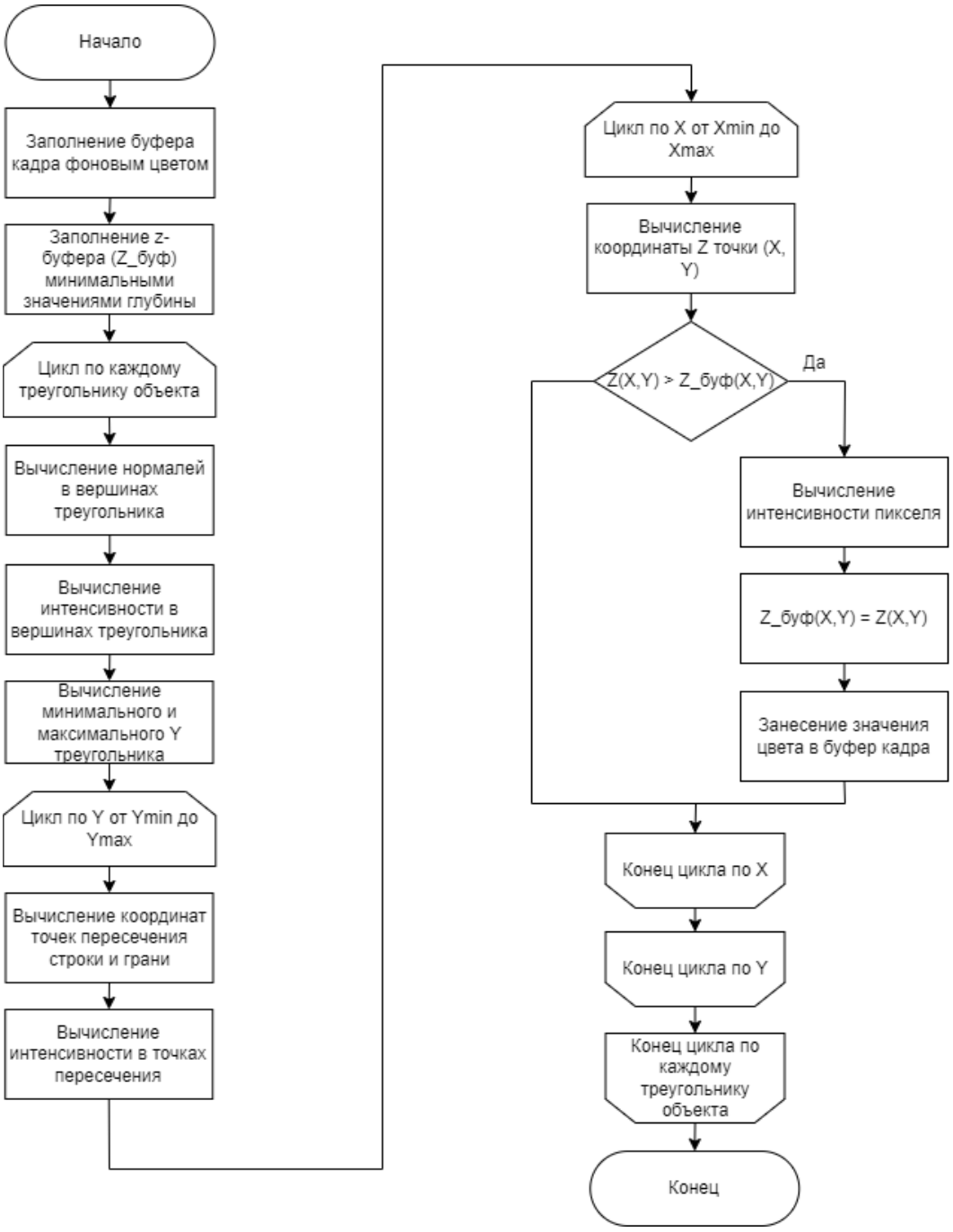

Рисунок 2.1 – Схема алгоритма z-буфера с закраской по Гуро

2.3 Алгоритм перемещения сферы

Для нахождения точек орбиты сферы используется метод численного интегрирования Верле [8]. Это алгоритм позволяет определить следующее положение тела, зная текущее и предыдущее положения без использования скорости.

Раскладывается вектор местоположения точки $\overrightarrow{r}(t)$ в моменты времени $(t + \triangle t)$ и $(t - \triangle t)$ по формуле Тейлора (2.3):

$$\begin{cases} \overrightarrow{r}(t + \triangle t) = \overrightarrow{r}(t) + \overrightarrow{v}(t) \cdot \triangle t + \frac{\overrightarrow{a}(t)}{2} \cdot \triangle t^2 + \frac{\overrightarrow{b}(t)}{6} \cdot \triangle t^3 + O(\triangle t^4), \\ \overrightarrow{r}(t - \triangle t) = \overrightarrow{r}(t) - \overrightarrow{v}(t) \cdot \triangle t + \frac{\overrightarrow{a}(t)}{2} \cdot \triangle t^2 - \frac{\overrightarrow{b}(t)}{6} \cdot \triangle t^3 + O(\triangle t^4), \end{cases} \tag{2.3}$$

где $\overrightarrow{v}(t)$ — вектор скорости, $\overrightarrow{a}(t)$ — вектор ускорения, $\overrightarrow{b}(t)$ — производная ускорения по времени.

Затем складываются полученные формулы и выражается $\overrightarrow{r}(t + \triangle t)$ по формуле (2.4):

$$\overrightarrow{r}(t + \triangle t) = 2 \cdot \overrightarrow{r}(t) - \overrightarrow{r}(t - \triangle t) + \overrightarrow{a}(t) \cdot \triangle t^2 + O(\triangle t^4). \tag{2.4}$$

Небесные сферы двигаются по действием силы гравитационного взаимодействия, которая расчитывается по формуле (2.5):

$$\overrightarrow{F} = G \cdot \frac{m_1 \cdot m_2}{r^3} \cdot \overrightarrow{r}(t), \tag{2.5}$$

где $\overrightarrow{r}(t)$ — вектор, соединяющий звезды, m_1, m_2 — массы звезд, G — гравитационная постоянная.

Тогда из второго закона Ньютона выражается ускорение по формуле (2.6):

$$\overrightarrow{a}(t) = \frac{\overrightarrow{F}(t)}{m}. \tag{2.6}$$

Возникает проблема с поиском положения в самый первый момент и последующий за ним. Предыдущему положению можно задать начальное условие (положение старта), а текущее вычисляется по формуле равноускоренно-

го движения (2.7):

$$\overrightarrow{r}(t + \triangle t) = \overrightarrow{r}(t) + \overrightarrow{v}(t) \cdot \triangle t + \frac{\overrightarrow{a}(t)}{2} \cdot \triangle t^2. \quad (2.7)$$

2.4 Вывод

В данном разделе были описаны алгоритм передвижения сфер, способ моделирования многогранной сферы, приведена схема алгоритма, объединяющего z-буфер и закраску по Гуро.

3 Технологический раздел

3.1 Выбор языка и среды программирования

В качестве языка программирования был выбран C++, по следующим причинам:

— поддержка ООП;

— благодаря своей вычислительной мощности язык обеспечивает высокую скорость исполнения кода;

— в стандартной библиотеке имеется множество готовых контейнеров и алгоритмов.

В качестве среды программирования был выбран «QtCreator», так как:

— в рамках фреймворка Qt представлена многофункциональная и производительная графическая библиотека;

— данная среда бесплатна для студентов.

3.2 Структура программы

Структура программы представлена на рисунке 3.1 в виде схемы последовательных преобразований.

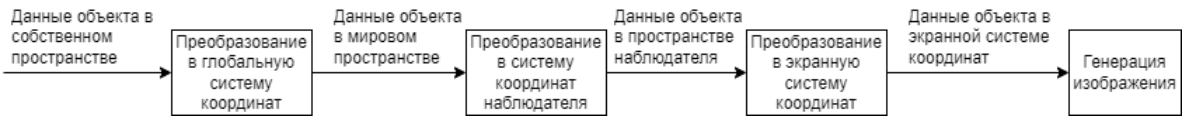

Рисунок 3.1 – Структура программы

Разработанная программа состоит из следующих классов.

1. Классы объектов сцены:
    - Camera — класс камеры с возможностью перемещения по сцене;

- Sphere — класс трёхмерных объектов (сфер), содержащий характеристики сфер и их орбиты.

2. Класс для работы с моделями сфер:

   - Mesh — класс полигональной сетки, описывающий представление трёхмерного объекта в программе и методы работы с ним.

3. Вспомогательные классы сцены:

   - Scene — класс, содержащий набор объектов;
   - Painter — класс отрисовки сцены.

4. Класс интерфейса пользователя:

   - MainWindow — класс главного окна сцены.

На рисунке 3.2 приведена диаграмма классов.

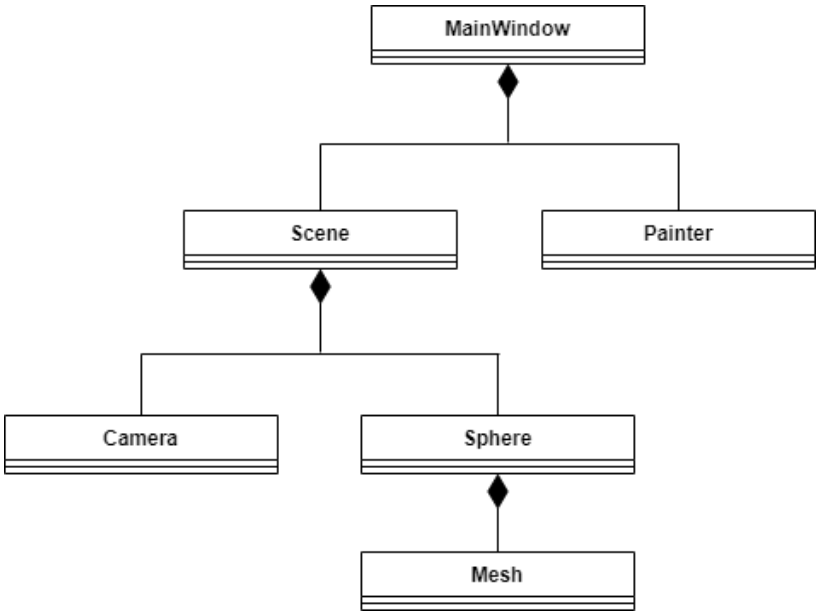

Рисунок 3.2 – Диаграмма классов

3.3 Реализации алгоритмов

В листинге 3.1 представлен алгоритм расчета точек орбиты сферы, в листинге 3.2 — алгоритм z-буфера, совмещенный с закраской по Гуро.

Листинг 3.1 – Алгоритм расчета точек орбиты сферы

```
1  void Sphere::calc_trajectory(Point3D& s2, Point3D& vs1, Point3D&
       vs2, double m2)
2  {
3      Point3D fs, es, s1 = center;
4      double m1 = m;
5      double rasts = sqrt((s1.x - s2.x) * (s1.x - s2.x) + (s1.z -
           s2.z) * (s1.z - s2.z));
6      double dt = 0.025;
7      points.clear();
8
9      double xp2 = s2.x, xp1 = s1.x;
10     double zp2 = s2.z, zp1 = s1.z;
11     double xc1 = xp1 + vs1.x * dt - (dt * dt * 0.5 * G * m2 *
           (s1.x - s2.x)) / (rasts * rasts * rasts);
12     double xc2 = xp2 + vs2.x * dt + (dt * dt * 0.5 * G * m1 *
           (s1.x - s2.x)) / (rasts * rasts * rasts);
13     double zc1 = zp1 + vs1.z * dt - (dt * dt * 0.5 * G * m2 *
           (s1.z - s2.z)) / (rasts * rasts * rasts);
14     double zc2 = zp2 + vs2.z * dt + (dt * dt * 0.5 * G * m1 *
           (s1.z - s2.z)) / (rasts * rasts * rasts);
15
16     s2.x = xc2;
17     s2.z = zc2;
18     s1.x = xc1;
19     s1.z = zc1;
20
21     for (int i = 0; i < 207; i++)
22     {
23         points.push_back(s1);
24
25         rasts =  sqrt((s1.x - s2.x) * (s1.x - s2.x) + (s1.z -
               s2.z) * (s1.z- s2.z));
26         es.x = (s1.x - s2.x) / rasts;
27         es.z = (s1.z - s2.z) / rasts;
28         fs.x = (G * m1 * m2 * es.x) / (rasts * rasts);
29         fs.z = (G * m1 * m2 * es.z) / (rasts * rasts);
30
31         s1.x = 2 * xc1 - xp1 - (fs.x) * dt * dt / m1;
32         s1.z = 2 * zc1 - zp1 - (fs.z) * dt * dt / m1;
33         xp1 = xc1;
```

```
34          xc1 = s1.x;
35          zp1 = zc1;
36          zc1 = s1.z;
37
38          s2.x = 2 * xc2 - xp2 + (fs.x) * dt * dt / m2;
39          s2.z = 2 * zc2 - zp2 + (fs.z) * dt * dt / m2;
40          xp2 = xc2;
41          xc2 = s2.x;
42          zp2 = zc2;
43          zc2 = s2.z;
44      }
45      cur = 0;
46  }
```

Листинг 3.2 – Алгоритм z-буфера с закраской по Гуро

```
1  void Painter::textureSpTriangle(Mesh mesh, int a, int b, int c,
       std::vector<QPoint> points, std::vector<QPoint>
       texturePoints, QImage *img, QImage *img2, double light,
       Point3D p)
2  {
3       double u,v,w;
4       QColor kolor;
5       double Xa = points[0].x(),Xb = points[1].x(),Xc =
           points[2].x();
6       double Ya = points[0].y(),Yb = points[1].y(),Yc =
           points[2].y();
7
8       double div = (Xb - Xa)  * (Yc - Ya) - (Yb - Ya) * (Xc - Xa);
9       if(div == 0.0)
10          return;
11
12      Point3D Na = Point3D(mesh.Tnodes[a].x - p.x,
           mesh.Tnodes[a].y - p.y, mesh.Tnodes[a].z - p.z);
13      Point3D Nb = Point3D(mesh.Tnodes[b].x - p.x,
           mesh.Tnodes[b].y - p.y, mesh.Tnodes[b].z - p.z);
14      Point3D Nc = Point3D(mesh.Tnodes[c].x - p.x,
           mesh.Tnodes[c].y - p.y, mesh.Tnodes[c].z - p.z);
15      Point3D pa = mesh.Tnodes[a],pb = mesh.Tnodes[b],pc =
           mesh.Tnodes[c];
16      double Ia = countStarLightIntense(Na, pa);
17      double Ib = countStarLightIntense(Nb, pb);
```

```
18      double Ic = countStarLightIntense(Nc, pc);
19
20      Point3D wall =
           mesh.countNormalVector(mesh.Tnodes[a],mesh.Tnodes[b],
           mesh.Tnodes[c]);
21
22      QPoint A = mesh.points[a], B = mesh.points[b],C =
           mesh.points[c];
23      if (A.y() > B.y())
24      {
25          std::swap(A, B);
26          std::swap(Ia, Ib);
27      }
28      if (A.y() > C.y())
29      {
30          std::swap(A, C);
31          std::swap(Ia, Ic);
32      }
33      if (B.y() > C.y())
34      {
35          std::swap(B, C);
36          std::swap(Ib, Ic);
37      }
38
39      int xmin = (int)minim(Xa, Xb, Xc), ymin = (int)minim(Ya, Yb,
           Yc);
40      int xmax = (int)maxim(Xa, Xb, Xc), ymax = (int)maxim(Ya, Yb,
           Yc);
41      for(int y = ymin; y < ymax; y++)
42      {
43          bool half2 = (y - ymin) > B.y() - A.y() || B.y() ==
              A.y();
44          int h = half2 ? C.y() - B.y() : B.y() - A.y();
45          float alpha = (float)(y - ymin) / (ymax - ymin);
46          float betta = (float)(y - ymin - (half2 ? B.y() - A.y()
              : 0)) / h;
47
48          QPoint minp = A + (C - A) * alpha;
49          QPoint maxp = half2 ? B + (C - B) * betta : A + (B - A)
              * betta;
50          double Id, If;
```

```
51          Id = Ia + (Ic - Ia) * alpha;
52          If = half2 ? Ib + (Ic - Ib) * betta : Ia + (Ib - Ia) *
                betta;
53
54          if (minp.x() > maxp.x())
55          {
56              std::swap(minp, maxp);
57              std::swap(Id, If);
58          }
59          minp.setX(std::max(xmin, minp.x()));
60          maxp.setX(std::min(xmax, maxp.x()));
61          int d = maxp.x() - minp.x(), f = minp.x(), l = maxp.x();
62          for(int x = f; x < l; x++)
63          {
64              double Z = (-wall.x * (x - mesh.Tnodes[a].x) +
                    -wall.y * (y - mesh.Tnodes[a].y))/wall.z +
                    mesh.Tnodes[a].z;
65              if (zBuffer[x][y] < Z)
66              {
67                  v = ((x - Xa) * (Yc - Ya) - (y - Ya) * (Xc -
                        Xa))/div;
68                  w = ((Xb - Xa) * (y - Ya) - (Yb - Ya) * (x -
                        Xa))/div;
69                  u = 1.0 - v - w;
70                  if(u < 0 || u > 1 || v < 0 || v > 1 || w < 0 ||
                        w > 1) continue;
71
72                  double xT = u * texturePoints[0].x() + v *
                        texturePoints[1].x() + w *
                        texturePoints[2].x();
73                  double yT = u * texturePoints[0].y() + v *
                        texturePoints[1].y() + w *
                        texturePoints[2].y();
74                  if(yT >= img->height()) yT = img->height() -1;
75                  if(yT < 0) yT = 0;
76                  if(xT >= img->width()) xT = img->width() -1;
77                  if(xT < 0) xT = 0;
78
79                  QRgb rgb = img->pixel(xT, yT);
80                  kolor = QColor(qRed(rgb), qGreen(rgb),
                        qBlue(rgb), qAlpha(rgb));
```

```
81
82                  float phi = l == f ? 1. : (float)(x - f) /
                        (float)(l - f);
83                  double l = ld + (lf - ld) * phi;
84                  light = l;
85                  setColor(x, y, kolor, light, img2);
86                  zBuffer[x][y] = Z;
87              }
88          }
89      }
90  }
```

3.4 Описание интерфейса

На рисунке 3.3 представлен интерфейс программы.

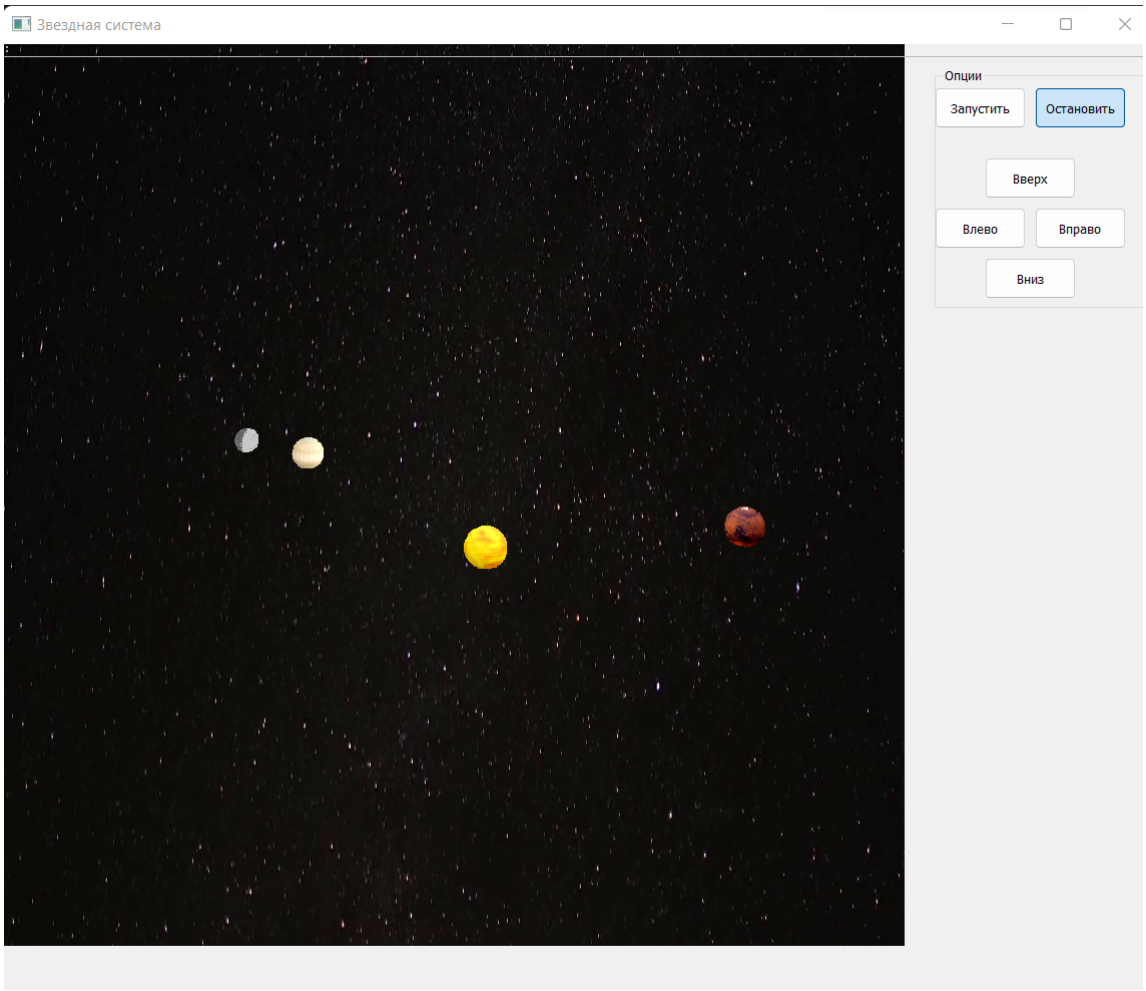

Рисунок 3.3 – Интерфейс программы

Функции представленных кнопок следующие:

— кнопки «Запустить/Остановить» позволяют запустить/остановить моделирование движения сфер;

— кнопки «Вправо/Влево» позволяют перемещать камеру по оси OX, то есть вправо/влево;

— кнопки «Вверх/Вниз» позволяют перемещать камеру по оси OY, то есть вверх/вниз.

Так же для перемещения камеры вправо/влево можно использовать кнопки «E» И «Q» соответственно, для перемещения камеры вверх/вниз — кнопки «A» и «D».

3.5 Вывод

В этом разделе был выбран язык программирования и среда разработки, рассмотрена диаграмма основных классов, приведены реализации алгоритмов и разобран интерфейс приложения.

4 Экспериментальный раздел

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование, следующие:

— операционная система Windows 11 64–bit;

— оперативная память 16 ГБ;

— процессор 2.40 ГГц Intel Core i5–1135G7 [9].

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также системой тестирования.

4.2 Цель эксперимента

Для программы, изображающей динамичную сцену, важна скорость отрисовки сцены. На скорость отрисовки может повлиять изменение количества полигонов, аппроксимирующих объекты сцены.

В рамках данной курсовой работы проведено исследование зависимости времени отрисовки сцены от количества полигонов, аппроксимирующих объекты сцены. Критерием измерения будет среднее время отрисовки кадра.

4.3 План эксперимента

В ходе эксперимента меняется количество полигонов: 100, 300, 500, 800, 1000, 1200, 1400. Каждый замер производится 10 раз, за результат выбирается среднее арифметическое времени отрисовки кадра. Для замера времени используется библиотека chrono.

Так как оценивается не только скорость синтезирования изображения, но и быстродействие симуляции движения сфер, в эксперименте сферы будут

находится в движении. Камера для всех экспериментов находятся в одинаковой позиции и остается неподвижной на протяжении всех замеров.

4.4 Результат эксперимента

Результаты эксперимента представлены в виде таблицы 4.1 и графика 4.1.

Таблица 4.1 – Зависимость времени отрисовки сцены от количества полигонов, аппроксимирующих сферы

Количество полигонов сферы	Время, мкс
100	42052
300	48833
500	53089
800	60611
1000	67723
1200	70069
1400	75393

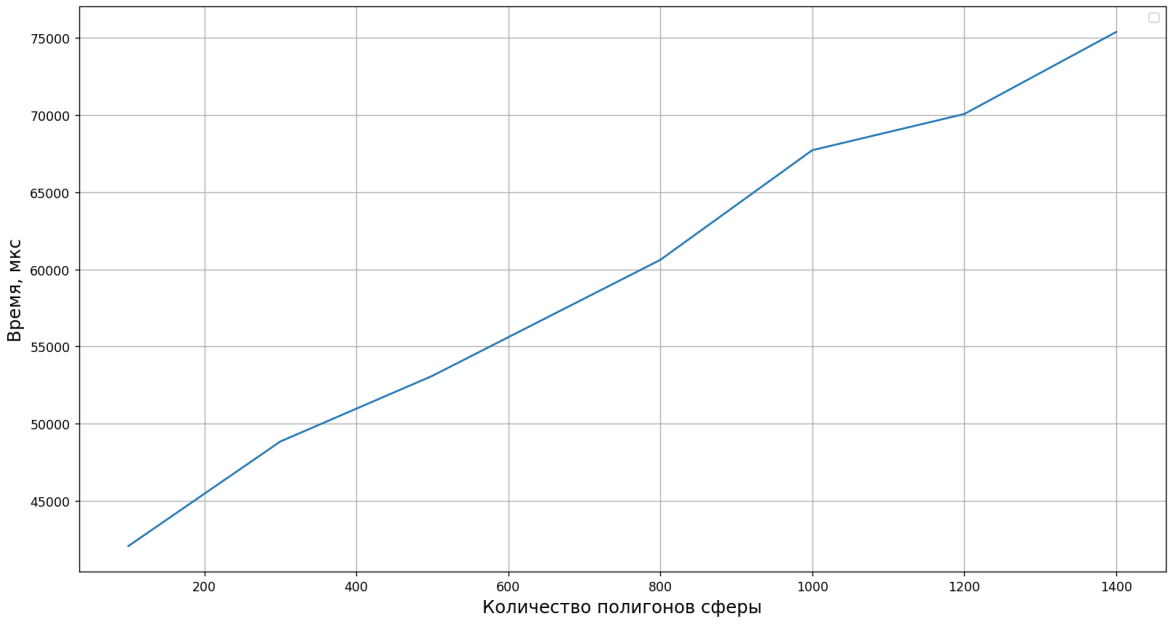

Рисунок 4.1 – Зависимость времени отрисовки сцены от количества полигонов, аппроксимирующих сферы

4.5 Вывод

Увеличение количества полигонов значительно влияет на скорость отрисовки сцены. При использовании сфер, каждая из которых представлена 1200 полигонов, становятся заметными задержки между кадрами. При меньших количествах данный эффект менее заметен.

Заключение

В ходе работы над курсовым проектом была достигнута поставленная цель: создание ПО для моделирования движения звезд и планеты в тройной звездной системе. Были выполнены следующие задачи:

— выделены объекты сцены и выбрана модель их представления;

— проанализированы и выбраны необходимые существующие алгоритмы построения реалистичного трехмерного изображения;

— разработана физическая модель поведения объектов;

— разработана программа на основе выбранных алгоритмов;

— проведено исследование на основе разработанной программы.

В ходе выполнения эксперимента были определены параметры системы, при которых сцена отрисовывается со скоростью, необходимой для формирования у пользователя эффекта плавности движения объектов сцены.

Список использованных источников

1. Роджерс Д. Адамс Дж. Математические основы машинной графики. М.: Мир, 1989.

2. Обратная трассировка лучей. Режим доступа: http://www.ray-tracing.ru/articles164.html (дата обращения: 09.10.2022).

3. Алгоритм Робертса. [Электронный ресурс]. Режим доступа: http://compgraph.tpu.ru/Oglavlenie.htm (дата обращения: 09.10.2022).

4. Алгоритм Варнока. [Электронный ресурс]. Режим доступа: http://compgraph.tpu.ru/Oglavlenie.htm (дата обращения: 09.10.2022).

5. Алгоритмические основы современной компьютерной графики. Режим доступа: https://intuit.ru/studies/courses/70/70/info (дата обращения: 09.10.2022).

6. Простые модели освещения. Режим доступа: http://grafika.me/node/344 (дата обращения: 12.10.2022).

7. Мороз В.И. Кононович Э.В. Общий курс астрономии. М.: Мир, 2001. с. 603.

8. Метод Стёрмера — Верле. Режим доступа: https://hleb-produkt.ru/stati/7809-metod-stermera-verle.html (дата обращения: 15.10.2022).

9. Процессор Intel® Core™ i5-1135G7 [Электронный ресурс]. Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/208658/intel-core-i51135g7-processor-8m-cache-up-to-4-20-ghz.htmll (дата обращения: 20.01.2023).