Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ НА ТЕМУ:

«Визуализация планетарной системы»

Студент Ляпина Наталья Викторовна
фамилия, имя, отчество

Студент ИУ7-52Б _____ Ляпина Н.В.
группа *подпись, дата* *фамилия, и.о.*

Руководитель курсовой работы _____ Кострицкий А.С.
подпись, дата *фамилия, и.о.*

2022 г.

Содержание

Введение

Компьютерное моделирование является одним из эффективных методов изучения сложных систем. Компьютерные модели проще и удобнее исследовать в силу их возможности проводить вычислительные эксперименты, в тех случаях, когда реальные эксперименты затруднены из-за финансовых, физических препятствий, или могут дать непредсказуемый результат. [1]

Целью курсовой работы является разработка программного обеспечения для визуализации планетарной системы со звездой в центре.

Для достижения поставленной цели необходимо решить следующие задачи:

— формально описать структуру объектов синтезируемой сцены;

— выбрать или модифицировать существующие алгоритмы трехмерной графики, необходимые для визуализации модели планетарной системы;

— описать используемые типы данных;

— описать структуру разрабатываемого ПО;

— реализовать выбранные алгоритмы визуализации;

— разработать ПО для визуализации модели планетарной системы;

— исследовать время работы реализации алгоритма от количества полигонов.

1. Аналитический раздел

В данном разделе приведен анализ предметной области: рассмотрены существующие решения поставленной задачи, проведена формализация объектов синтезируемой сцены, приведен обзор существующих методов и алгоритмов для решения поставленной задачи.

1.1. Обзор существующих решений

На рисунке 1.1 показана модель солнечной системы в программе Galaxy3D. Эта программа позволяет пользователям наблюдать за солнечной системой в движении, но не позволяет контролировать ход времени в программе. Также у пользователей имеется возможность просмотреть информацию о планете, кликнув на нее. Данная программа является одной из самых популярных в сфере симуляции движения планет. [2]

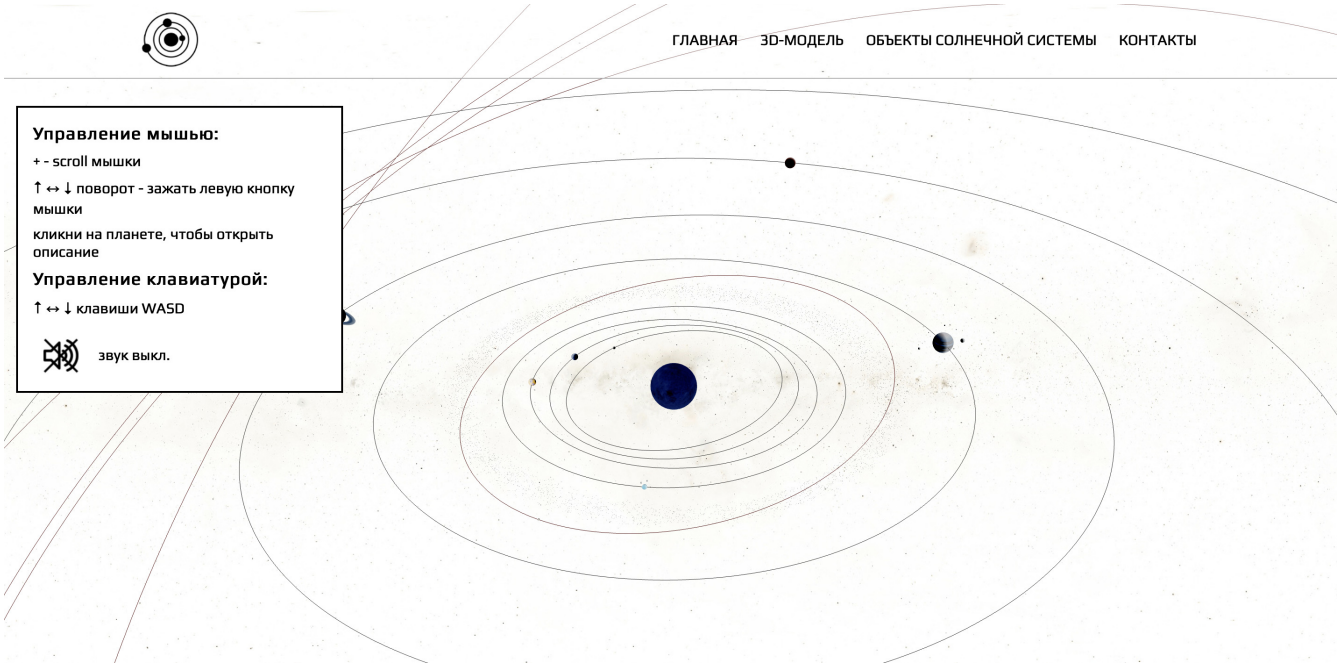


Рисунок 1.1 — Модель солнечной системы в Galaxy3D

На рисунке 1.2 показана модель солнечной системы в программе SpaceGid. Эта программа, также как и Galaxy3D, позволяет пользователям наблюдать за солнечной системой в движении, но, в отличии от Galaxy3D, она позволяет контролировать ход времени в программе. Функционал в данной программе дает возможность наблюдать за какой-то конкретной планетой, чего пользователь не мог сделать в Galaxy3D. Также имеется возможность контролировать скорость анимации и масштаб планет. [3]

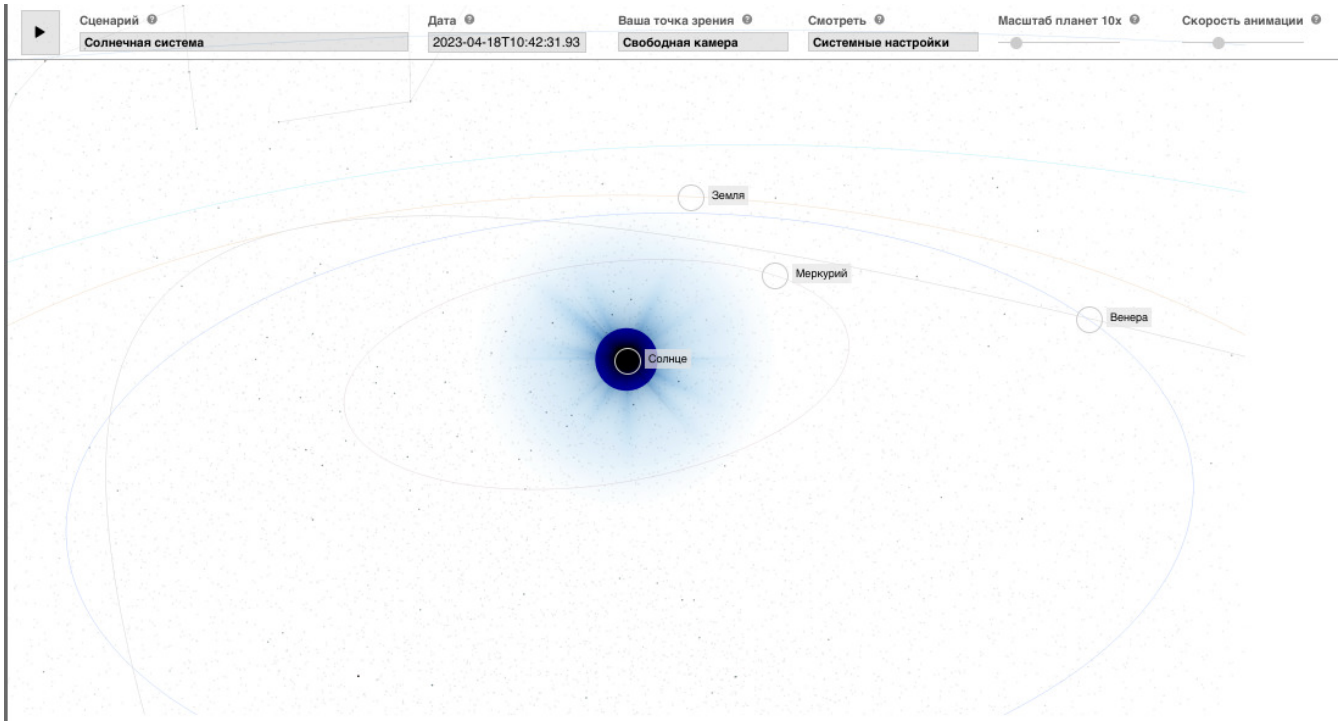


Рисунок 1.2 — Модель солнечной системы в SpaceGid

1.2. Формализация объектов сцены

Сцена состоит из следующих объектов:

1) Звезда – источник света, который является сферическим объектом;
2) Планеты – объект, который является выпуклым симметричным многогранником.

Ниже представлены характеристики для объектов типа **Звезда**.

1) Координаты центра масс в экранной СК;
2) Радиус;
3) Масса;
4) Параметр, используемый для вычисления количества узлов для построения полигональной сетки.

Ниже представлены характеристики для объектов типа **Планета**.

1) Начальные координаты центра масс;
2) Текущие координаты центра масс в экранной СК;
3) Начальная скорость;
4) Цвет поверхности;
5) Масса планеты;
6) Параметр, используемый для вычисления количества узлов для построения полигональной сетки.

1.3. Анализ способов задания трехмерных моделей

Модель можно задать несколькими способами:

1) Каркасная модель – модель, состоящая из вершин и рёбер, которые определяют форму отображаемого объекта;

2) Поверхностная модель – совокупность ограничивающих модель поверхностей, называемых полигонами;

3) Объемная модель – модель, состоящая из полигонов. Отличается от поверхностной модели тем, что усточняется информация о материале модели.

В таблице 1.1 приведено сравнение способов задания трехмерных моделей.

Таблица 1.1 — Сравнение способов задания трехмерных моделей

Характеристика	КМ	ПМ	ОМ
Применимость к сферическим объектам	нет	да	да
Применимость к произвольным выпуклым объектам	нет	да	да

Обозначения: КМ – каркасная модель, ПМ – поверхностная модель, ОМ – объемная модель. Для оценки используется бинарный признак.

Для решения поставленной задачи был выбран поверхностный способ задания трехмерных моделей, так как для реализации поставленной задачи не требуется информация о материале модели.

1.4. Алгоритм построения поверхностной модели

Для хранения информации о полигонах был выбран способ хранения с использованием списка граней. Каждый элемент этого списка представляет из себя список вершин, составляющих данную грань.

В качестве примера будет рассмотрен алгоритм для сферы.

На рисунке 1.3 рассматривается сечение сферы плоскоостью OXY в экранной СК. Сечение сферы любой плоскостью всегда даст окружность или точку в частном случае.

Рассмотрим алгоритм генирации вершин полигональной сетки для $\frac{1}{4}$ окружности. Остальные вершины можно получить в виду симметрии сферы.

Пусть

1) $center$ – центр сферы

2) R – радиус сферы

3) $points$ – массив вершин полигональной сетки

4) n – параметр, опередеяющий количество вершин между левой и верхней вершинами окружности.

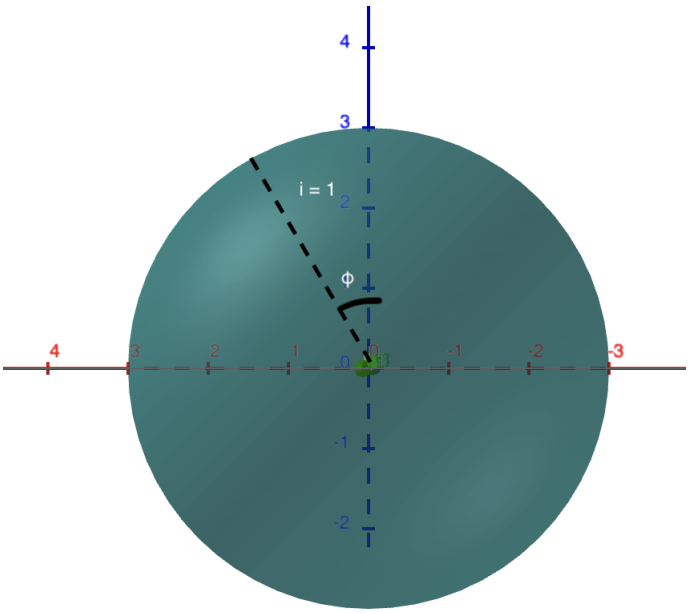


Рисунок 1.3 — Сечение сферы плоскостью OXY

Таким образом, вычисление i-ой вершины сетки $\frac{1}{4}$ окружности можно описать следующим образом:

$$tmp = \sqrt{2 \cdot R^2 \cdot \left(1 - \left(\cos \frac{90}{n+1} \cdot i\right)\right)}, \tag{1.1}$$

$$dx = \sin\left(\frac{90}{n+1} \cdot i\right) \cdot R, \tag{1.2}$$

$$dy = \sqrt{tmp^2 - dx^2}, \tag{1.3}$$

$$point_i = (center_x - dx, center_y + R - dy, center_z). \tag{1.4}$$

Для получения полной сетки необходимо:

1) получить набор вершин для верхней полуокружности;

2) произвести серию поворотов полуокружности относительно оси OX с шагом, равным $\frac{90}{n+1}$. Результат каждого поворота – набор вершин, определяющих очередную полуокружность;

3) на основе списка вершин создается список граней. Для каждого полигона в этом списке хрантся не сами вершины, а их индексы в массиве вершин, что значительно экономит ресурсы. [4]

Таким образом можно создавать не только сферы, но и любые другие симметричные выпуклые фигуры.

1.5. Анализ алгоритмов удаления невидимых ребер и поверхностей

В поставленной задаче требуется создать анимацию движения планет, следовательно время – наиболее важный ресурс. Таким образом алгоритм удаления невидимых ребер и поверхностей должен обладать характеристикой быстродействия.

1.5.1. Алгоритм Варнока

Данный алгоритм работает в пространстве изображений.

Сцена является начальным окном. Затем, алгоритму требуется определить, что изображать в очередном окне. Если нельзя однозначно ответить на этот вопрос, то окно делится на два. Так продолжается либо пока не достигнут предел в один пиксель, либо пока окно не будет закрашено.

Существуют возможные оптимизации данного алгоритма: сортировка многоугольников по координате z и хранение информации о многоугольниках. Данные оптимизации помогают увеличить быстродействие алгоритма, но они создают другую проблему – увеличение затрачиваемой памяти. [5]

Особенностями этого алгоритма являютмя возможность устранить лестничный эффект, рекурсивный вызов и недостаточная быстродействественность.

1.5.2. Алгоритм Z-буфера

Данный алгоритм работает в пространстве изображений, используя буфер кадра для заполнения интенсивности каждого пикселя, здесь вводится некоторый Z-буфер (буфер глубины каждого пикселя).

Значение каждого нового пикселя, который нужно занести в буфер кадра, сравнивается с глубиной пикселя, занесенного в Z-буфер. Если сравнение показывает, что новый пиксель расположен ближе к наблюдателю, то новое значение z заносится в буфер и корректируется значение интенсивности.

Алгоритм, использующий Z-буфер крайне прост в своей реализации по сравнению с другими анализируемыми алгоритмами, также не тратится время на сортировку элементов сцены.

Но несмотря на его быстродействие увеличиваются затраты по памяти при использовании данного алгоритма, запоминается информация по каждому пикселю изображения. [6]

Вычислительная сложность данного алгоритма равна $O(n \cdot m \cdot k)$, где $n \cdot m$ – количество пикселей в буфере кадра, k – количество полигонов.

Алгоритм

1) Всем элементам буфера кадра присвоить фоновое значение;
2) Инициализировать Z-буфер минимальными значениями глубины;
3) Выполнить растровую развертку каждого многоугольника сцены:

а) Для каждого i-го пикселя, связанного с многоугольником вычислить его глубину $z(x_i, y_i)$;

б) Сравнить глубину пикселя со значением, хранимым в Z-буфере. Если $z(x_i, y_i) > z_b(x_i, y_i)$, то $z_b(x_i, y_i) = z(x_i, y_i)$ и цвет(x_i, y_i) = цвет i-го пикселя;

Математические основы алгоритма

При известном уравнении плоскости, несущей конкретный многоугольник, вычисление глубины каждого пикселя могут быть произведены пошаговым способом.

Уравнение плоскости имеет следующий вид (выражено через z):

$$z = \frac{ax + by + d}{c} \neq 0, \tag{1.5}$$

где a, b, d, c – коэффициенты уравнения.

Для сканирующей строки $y = \text{const}$, глубина пикселя, у которого $x_1 = x + \Delta x$, поэтому равна:

$$z_1 - z = -\frac{ax_1 + d}{c} + \frac{ax + d}{c} = \frac{a(x - x_1)}{c}, \tag{1.6}$$

Отсюда получаем, что

$$z_1 = z - \frac{a}{c}\Delta x,\ \Delta x = 1, \text{ поэтому } z_1 = z - \frac{a}{c}. \tag{1.7}$$

Поскольку в данной задаче для визуализации используется только один вид многоугольников, а именно треугольник, то нахождение абсцисс точек пересечения горизонтали со сторонами треугольника будет выглядеть следующим образом:

1) Для каждой из сторон треугольника будут записываться параметрические уравнения вида:

$$x = x_H + (x_K - x_H)t, \tag{1.8}$$

$$y = y_H + (y_K - y_H)t, \tag{1.9}$$

2) После этого для каждой стороны находится параметр tпри пересечении с горизонталью $y = c$:

$$c = y_H + (y_K - y_H)t, \text{ где } t = \frac{c - y_H}{y_K} - y_H \tag{1.10}$$

3) Если $t \in (0, 1)$, то рассчитывается абсцисса точки пересечения горизонтали со стороной треугольника:

$$x = x_H + (x_K - x_H)\frac{c - y_H}{y_K - y_H} \tag{1.11}$$

Особенностями алгоритма являются возможность работы со сценами любой сложности, отсутствие требования предварительной сортировки объектов по глубине, сравнительно большой объем затрачиваемой памяти, возникновение лестничного эффекта.

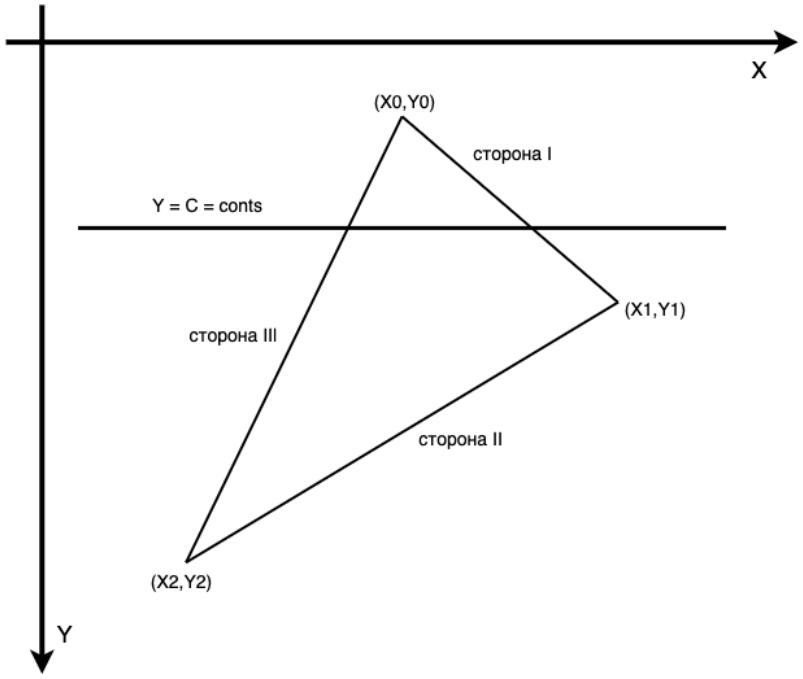


Рисунок 1.4 — Поиск абсцисс точек пересечения

1.5.3. Алгоритм Робертса

Алгоритм работает только с выпуклыми телами. Если тело изначально является невыпуклым, то предварительно его нужно разбить на выпуклые составляющие. [6]

Ниже приведены основные этапы работы алгоритма.

0) Подготовка исходных данных;
1) Удаление рёбер или граней, экранируемых самим телом;
2) Удаление рёбер, экранируемых другими телами;
3) Вычисление отрезков, которые образуют новые ребра при протыкании других объектов друг друга.

Особенностями алгоритма является высокая точность вычислений, невозможность работы с прозрачными и просвечивающими объектами, невозможность передачи падающих теней, метод строго ориентирован на выпуклые многогранники. [13]

Вывод

В таблице 1.2 приведено сравнение алгоритмов удаления ребер и поверхностей.

Таблица 1.2 — Сравнение алгоритмов удаления ребер и поверхностей

Характеристика	АВ	АБ	АР
Скорость вычислений	$O(S_{pic} \cdot n)$	$O(S_{pic} \cdot n)$	$O(n^2)$
Применимость к произвольным выпуклым объектам	да	да	да
Объемные вычислительные затраты	да	нет	да

Обозначения: АВ – алгоритм Варнока, АБ – алгоритм Z-буфера, АР – алгоритм Робертса.

Для решения поставленной задачи был выбран алгоритм Z-буфера.

1.6. Анализ методов и алгоритмов закрашивания

Главное требование, выдвигаемое к алгоритму закраски в рамках поставленной задачи, также как и в разделе выше, – быстродействие. [8]

Типы источников света

Основные типы источников света: точечные, прожекторы и бесконечно удаленные (направленные).

В посталвенной задаче Звезда не является точетным источником света. Также Звезда не является направленным источником света, так как в задаче не допускается считать её лучи параллельными. При этом Звезда не является прожектором, так как она не излучает свет конусом.

Поверхность Звезды будет рассматриваться как множество точечных источников.

Модели освещения

Существует два типа моделей освещения, используемых в синтезе трехмерных изображений: локальная модель освещения и глобальная модель освещения. Модель называется локальной, если не учитывает перенос света между поверхностями. Иначе модель называют глобальной. Далее будут рассмотреты только локальные модели освещения, поскольку для частой смены кадров важна производительность.

Простая модель освещения

Модель Ламберта моделирует идеальное диффузное освещение.

Интенсивность освещенности точки в этой модели находится по закону Ламберта:

$$I_d = I_0 \cdot \cos(\alpha), \tag{1.12}$$

где I_d – уровень освещенности в ассматриваемой точке, I_0 – максимальный уровень освещенности, α – угол между вектором нормали к плоскости и вектором, направленным от рассматриваемой точки к источнику освещения.

Модель освещения по Фонгу

Свет в модели освещения Фонга состоит из нескольких составляющих:

— Диффузная (рассеянная) – рассчитывается по закону Ламберта (1.12), исходя из предположения, что при попадании наповерхность свет рассеивается равномерно во все стороны;

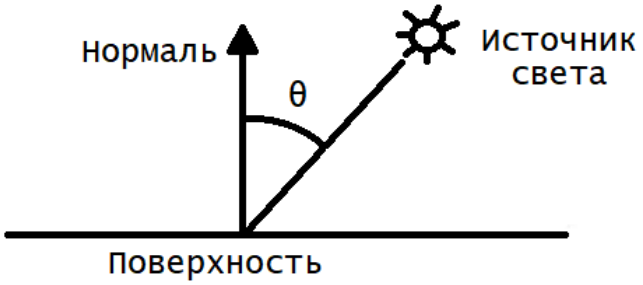


Рисунок 1.5 — Простая модель освещения

— Окружающая (фоновая) – некоторая константа, которая прибавляется к освещенности в каждой точке освещаемой модели;

— Зеркальная – придает объектам блеск.

Итоговая формула освещенности в точке выглядит следующим образом:

$$I = I_d + I_{env} + I_{mir}, \tag{1.13}$$

где I – интенсивность в ассматриваемой точке, I_{env} – интенсивность окружающего света, I_{mir} – интенсивность зеркального света.

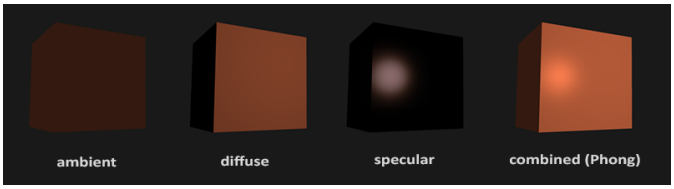


Рисунок 1.6 — Модель освещения Фонга

Для учета фоновой составляющей вводят небольшой коэффициент фонового освещения.

Так как планеты не обладают зеркальными свойствами, в рамках поставленной задачи $I_{mir} = 0$.

Так как поверхности планет не обладают зеркальными эффектами, а фоновая составлюща сравнительно мала, было решено использовать простую модель освещения.

Простая закраска

Весь полигон закрашивается одним уровнем интенсивности, который вычисляется по закону Ламберта.

При использовании данного метода все плоскости будут закрашены однотонно, и в результате вращения могут возникнуть ложные ребра в виду того, что цвета соседних граней могут сильно отличаться. Этот эффект можно сгладить, если использовать большое число плоскостей.

Данный метод не требователен к ресурсам, и у модели, полученной данным методом, четко видны переходы между полигонами.

Закраска по Гуро

Полигон закрашивается не одним цветом, а плавно изменяющимися оттенками одного цвет, вычисляемыми путем интерполяции цветов примыкающих граней.

Основные этапы:

1) для каждой грани вычислется вектор нормали;

2) вычисляется нормаль каждой вершины как среднее арифметическое между нормалями всех граней, пересекающихся в данной вершине;

3) рассчитывается интенсивность освещения в вершинах пропорционально косинусу угла между нормалью в вершине и направлением света;

4) каждый многоугольник закрашивается путем линейной интерполяции значений интенсивностей в вершинах сначала вдоль каждого ребра, а затем и между ребрами вдоль каждой сканирующей строки.

Закраска по Гуро имеет высокое качество построения зеркальных бликов и матовых поверхностей, но при этом имеет сравнительно большие вычислительные затраты и возможность осуществления ситуации, когда ребро многоугольника при закраске может стать незаметным.

Закраска по Фонгу

Данный метод аналогичен закраске по Гуро, однако было предложено интерполиговать не интенсивности в вершинах, а нормали.

Этот метод более трудоемкий, чем аналогичный ему метод закраски по Гуро, описанный выше: для каждой точки поверхности необходимо совершать больше вычислительных операций.

Закраска по Фонгу строит изображения с высоким качеством, особенно для поверхностей с зеркальными свойствами.

Вывод

В таблице 1.3 приведено сравнение алгоритмов закрашивания полигонов.

Таблица 1.3 — Сравнение алгоритмов закрашивания полигонов

Характеристика	ПЗ	ЗГ	ЗФ
Учёт зеркальных и матовых бликов	нет	да	да
Высокие вычислительные затраты	нет	да	да
Применимость к произвольным выпуклым объектам	да	да	да

Обозначения: ПЗ – алгоритм простой закраски, ЗГ – алгоритм закраски по Гуро, ЗФ – алгоритм закраски по Фонгу.

Для решения поставленной задачи был выбран алгоритм простой закраски т.к. основным критерием для данной задачи является быстродействие.

1.7. Физическая составляющая

В данной работе требуется реализовать движение планет по физическим законам.

По закону всемирного тяготения вычисляем силу тяготения в момент времени t для каждого объекта на сцене.

$$F_i^t = \sum_{j=0}^{n} G \cdot \frac{m_i \cdot m_j}{R_{ij}^2}, j \neq i, \tag{1.14}$$

где n – количество объектов на сцене, m_i – масса i-го объекта, m_j – масса j-го объекта, R_{ij} – расстояние между i-м и j-м объектом. [14]

Используя явную схему Эйлера решения дифференциальных уравнений и второй закон Ньютона, скорость в момент времени t вычисляется по формуле (1.15).

$$\vec{v_i^t} = \frac{F_i^{t-1} \cdot dt}{m_i}, \tag{1.15}$$

где m_i – масса i-го объекта, F_i^{t-1} – сила тяготения для i-го объекта в момент времени $t-1$, dt – приращение по времени.

Тогда кординаты следующего положения объекта в момент времени t вычисляются по формуле (1.16).

$$\vec{r_i^t} = \vec{v_i^{t-1}} \cdot dt + \vec{r_i^{t-1}}. \tag{1.16}$$

1.8. Выводы из аналитического раздела

В данном разделе была изучена предметная область, формализованы объекты сцены, рассмотрены алгоритмы удаления невидимых ребер и поверхностей, методы закрашивания поверхностей.

В качестве модели была выбрана поверхностная модель, в качестве алгоритма удаления невидимых рёбер и поверхностей был выбран алгоритм Z-буфера, в качестве метода закрашивания был выбран метод простой закраски.

2. Констукторский раздел

В данном разделе описаны разработанные алгоритмы, а также используемые классы.

2.1. Разработка алгоритмов решения поставленной задачи

2.1.1. Общий алгоритм

Общий алгоритм задачи подразумевает, что основной функционал программы выполняется в бесконечном цикле.

Рисунок 2.1 — Схема Общего алгоритма

2.1.2. Модифицированный алгоритм Z-буфера

Модификация алгоритма Z-буфера заключается в том, что в буфере будут храниться координаты z только для центров планет.

Так как планеты не должны пересекаться, по координатам их центра можно однозначно определить их расположение относительно экранной плоскости. Также предлагается отрисовывать только те точки сетки сферы, глубина которых не меньше глубины центра сферы.

На рисунке 2.2 представлена схема алгоритма модифицированного Z-буфера.

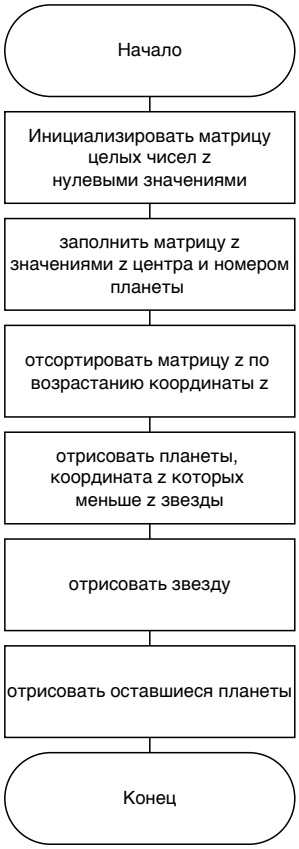


Рисунок 2.2 — Схема модифицированного Z-буфера

2.1.3. Алгоритм построения полигональной сетки

На рисунке 2.3 представлена схема алгоритма построения полигональной сетки.

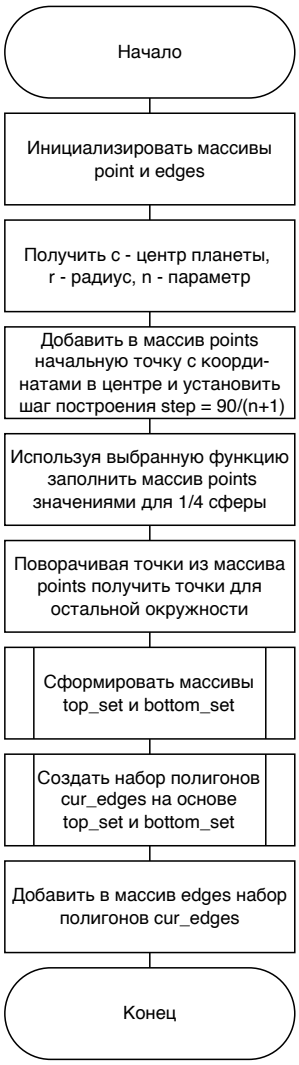


Рисунок 2.3 — Схема алгоритма полигональной сетки

2.1.4. Алгоритм простой закраски

На рисунке 2.4 представлена схема алгоритма простой закраски.

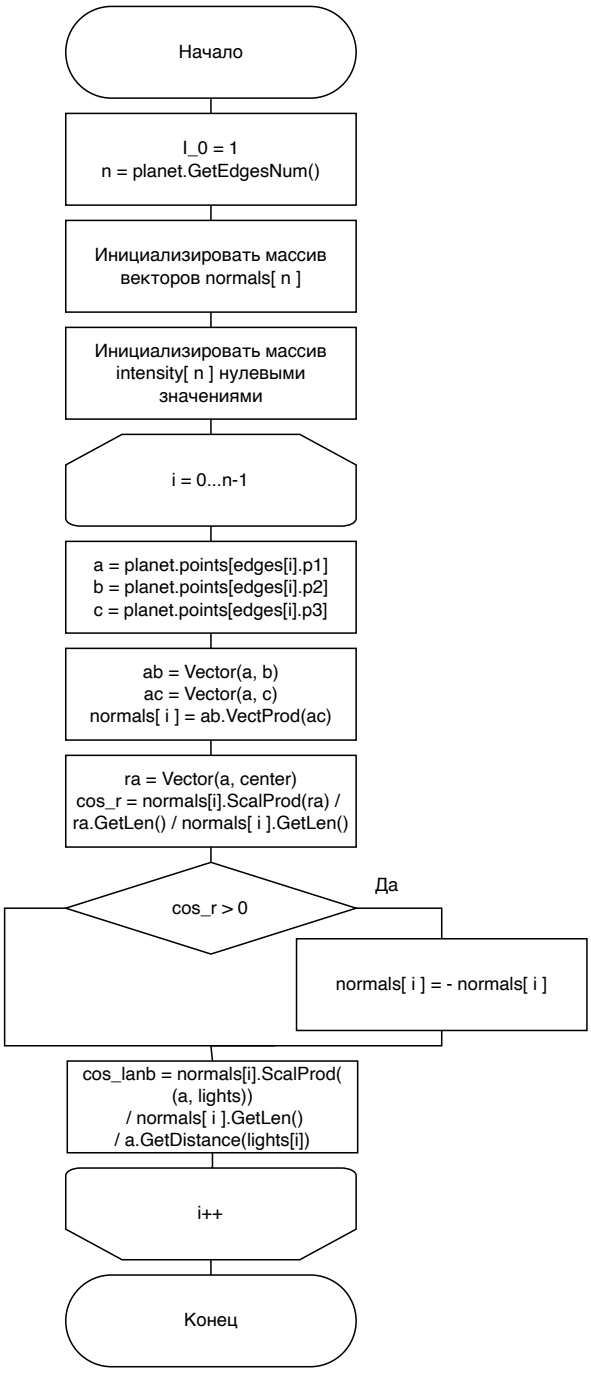


Рисунок 2.4 — Схема алгоритма простой закраски

2.1.5. Используемые типы и структуры данных

В таблице 2.1 представлены типы и структуры данных, которые будут реализованы для разрабатываемого ПО.

Таблица 2.1 — Используемые типы и структуры данных

Сущность	Структура
Точка с координатами X, Y, Z	Структура Point
Вектор с начальной и конечной точками	Структура Vector
Полигон с индексом трех точек	Структура Edge
Объект с координатами центра, радиусом, типом объекта, параметром узлов сетки, массивом точек, массивом полигонов	Структура Object
Планета со всеми параметрами объекта, цветом поверхности и списком интенсивности граней	Структура Planet
Звезда со всеми параметрами объекта и списком координат источников света	Структура Star
Сцена со звездой, массивом планет и с таймером для покадровой анимации	Структура MyGraphicsView

2.2. Выводы из конструкторской части

В данном разделе были описаны схемы реализуемых алгоритмов, определены используемые сущности и структуры данных.

3. Технологический раздел

В данном разделе представлены требования к ПО, средства, использованные в процессе разработки для реализации задачи, а также листинг кода программы. Кроме того описан пользовательский интерфейс и показаны результаты тестирования.

3.1. Требования к ПО

На рисунке (3.1) представлена IDEF0 диаграмма реализуемой задачи.

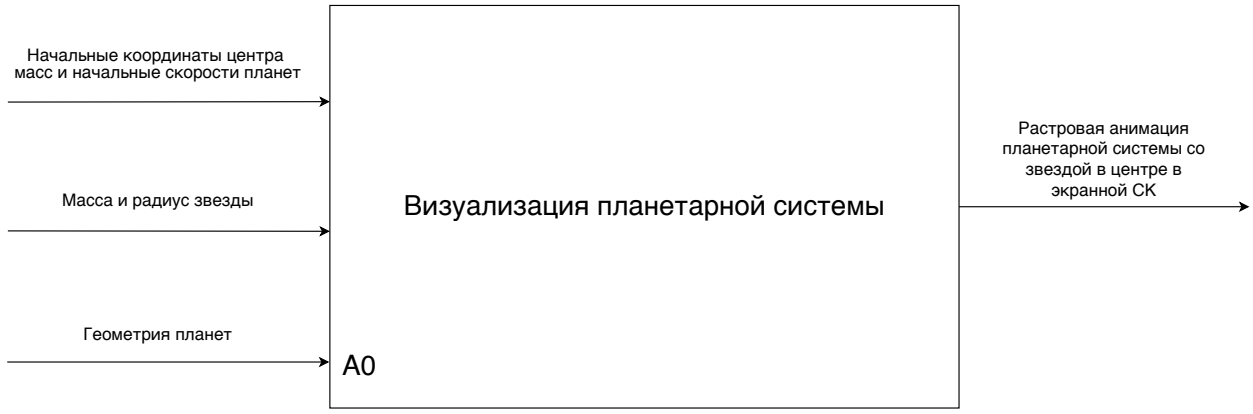


Рисунок 3.1 — IDEF0 диаграмма разрабатываемого ПО

Программа должна предоставлять доступ к следующему функционалу:

1) добавление звезды на сцену;
2) добавление планет на сцену;
3) пауза, старт и сброс анимации;
4) перемещение камеры на сцене.

К программе предъявляются следующие требования:

1) программа должна давать отклик на действия пользователя за минимальное время;
2) программа должна корректно реагировать на любые действия пользователя.

3.2. Средства реализации разработанного ПО

Для реализации был использован язык C++, так как он предоставляет весь необходимый функционал для решения поставленной задачи, а также для него существует фреймворк QT со встроенным графическим редактором QT Design, который позволит создать пользовательский интерфейс.

3.3. Листинг программы

В приведенных ниже листингах представлены следующие реализации:

1) модифицированный алгоритм Z-буфера (листинг 3..1);
2) алгоритм простой закраски (листинг 3..2);
3) алгоритм построения полигональной сетки (листинг 3..3).

Листинг 3..1 — Модифицированный алгоритм Z-буфера

```
1  void myGraphicsView::DrawSystem()
2  {
3      scene->clear();
4      DrawStars();
5
6      int z[MAX_OBJECTS_SIZE][Z_MATRIX_ROW_SIZE];
7      for (auto i = 0; i < objects.size(); i++)
8      {
9          z[i][0] = (objects[i]).GetCenter().GetZ();
10         z[i][1] = i;
11     }
12
13     if (objects.size() > 0)
14     {
15         for (auto i = 0; i < objects.size() - 1; i++)
16             for (auto j = 0; j < objects.size() - i - 1; j++)
```

```
                if (z[j][0] > z[j + 1][0])
                {
                    std::swap(z[j][0], z[j + 1][0]);
                    std::swap(z[j][1], z[j + 1][1]);
                }
    }
    int i = 0;

    while (i < objects.size() && z[i][0] < star[0].GetCenter().GetZ())
        DrawPlanet(objects[z[i++][1]]);
    DrawStar();
    while (i < objects.size())
        DrawPlanet(objects[z[i++][1]]);
}

//Пример отрисовки Звезды
void myGraphicsView::DrawStar()
{
    Edge* net = star[0].GetEdges();
    Point* points = star[0].GetPoints();
    int n = star[0].GetNetNodesParam();
    int cz = star[0].GetCenter().GetZ();
    Point p1, p2, p3;
    double t = star[0].GetCenter().GetZ() / Z_K;

    std::vector<Point> orbit = star[0].GetOrbitNum();
    int no = orbit.size();
    int j;
    for (int i = 0; i < no - 1; i++)
    {
        DrawLine(orbit[i], orbit[i+1]);
    }
    for (int i = 0; i < star[0].GetEdgesNum(); i++)
    {
        p1 = points[net[i].GetP1()];
        p2 = points[net[i].GetP2()];
        p3 = points[net[i].GetP3()];
        if (p1.GetZ() >= cz || p2.GetZ() >= cz || p3.GetZ() >= cz)
            DrawPolygon(p1, p2, p3, 1 + t, star[0].GetCenter());
    }
}
```

Листинг 3..2 — Алгоритм простой закраски

```
void PlanetA::CalcIntensities(std::vector<Point> lights)
{
```

```
3       Edge* edges = GetEdges();
4       Point* points = GetPoints();
5       int n = GetNetNodesParam();
6
7       Vector *normals = new Vector[GetEdgesNum()];
8       Point a, b, c;
9       Vector ab, ac, ra;
10      int I_0 = 1, I;
11      double cos_lamb, cos_r;
12      double len_n, len_a, len_r;
13      double sn, sr;
14
15      Point center = GetCenter();
16
17      for (int i = 0; i < GetEdgesNum(); i++)
18      {
19          intensities[i] = 0;
20          for (int j = 0; j < GetEdgesNum(); j++)
21          {
22              a = points[edges[i].GetP1()];
23              b = points[edges[i].GetP2()];
24              c = points[edges[i].GetP3()];
25
26              ab.Set(a, b);
27              ac.Set(a, c);
28              normals[i] = ab.VectProd(ac);
29
30              ra.Set(a, center);
31              len_r = ra.GetLen();
32              sr = normals[i].ScalarProd(ra);
33              len_n = normals[i].GetLen();
34              len_a = a.GetDistance(lights[j]);
35
36              cos_r = sr / len_r / len_n;
37              if (cos_r > 0)
38                  normals[i].Neg();
39
40              sn = normals[i].ScalarProd(Vector(a, lights[j]));
41              cos_lamb = sn / len_n / len_a;
42
43              intensities[i] += I_0 * cos_lamb;
44          }
45      }
46      delete [] normals;
47  }
```

Листинг 3..3 — Алгоритм построения полигональной сетки

```
1  void Object::CreateNet()
2  {
3      CreatePoints();
4      CreateEdges();
5  }
6
7  void Object::CreatePoints()
8  {
9      points = new Point [GetPointsNum()];
10     int k = 0;
11     int r = radius_a;
12     int cx = center.GetX(); int cy = center.GetY(); int cz = center.GetZ();
13
14     points[k++] = Point(cx - r, cy, cz);
15
16     int step_degree = 90 / (net_nodes_param + 1);
17     double radians, a, b, c;
18     for (int i = 1; i <= net_nodes_param; i++)
19     {
20         radians = (90 - step_degree * i) * M_PI / 180;
21         if (type == 0)
22         {
23             a = sqrt(2 * r * r * (1 - cos(radians)));
24             b = sin(radians) * r;
25             c = sqrt(a * a - b * b);
26             points[k++] = Point(cx - b, cy + r - c, cz);
27         }
28         else if (type == 1)
29         {
30             int r_tmp = round(sqrt((radius_a*radius_a*radius_b*radius_b) /
                   (cos(radians)*cos(radians)
31             *radius_a*radius_a+sin(radians)*sin(radians)*radius_b*radius_b)));
32             a = sqrt(2 * r_tmp * r_tmp * (1 - cos(radians)));
33             b = sin(radians) * r_tmp;
34             c = sqrt(a * a - b * b);
35             points[k++] = Point(cx - b, cy + radius_a - c, cz);
36         }
37     }
38     points[k++] = Point(cx, cy + r, cz);
39
40     for (int i = 0; i <= net_nodes_param; i++)
41     {
42         points[k++] = Point(2 * cx - points[net_nodes_param - i].GetX(),
43                             points[net_nodes_param - i].GetY(),
44                             points[net_nodes_param - i].GetZ());
```

```
45      }
46  }
47
48  void Object::CreateEdges()
49  {
50      int n = 3 + net_nodes_param * 2;
51      int m = 2 + net_nodes_param * 2 * 2;
52
53      Edge cur_edges[m];
54      edges = new Edge[GetEdgesNum()];
55
56      int j = 0;
57      int top_set[n];
58      int bottom_set[n];
59      double step_degree = 90 / (net_nodes_param + 1);
60      int k = n;
61      int cur, right, left;
62
63      for (int i = 0; i < n; i++)
64      {
65          top_set[i] = i;
66      }
67
68      int s = n;
69
70      for (double degree = step_degree; degree <= 360; degree += step_degree)
71      {
72          for (int i = 1; i < n - 1; i++)
73          {
74              Point temp = points[i];
75              points[s++] = temp.RotateX(degree, center);
76          }
77          for (int i = 0; i < n - 2; i++)
78          {
79              bottom_set[i] = k + i;
80          }
81          for (int i = 0; i < m; i++)
82          {
83              cur_edges[i] = Edge();
84          }
85
86          cur_edges[0].Append(top_set[0]);
87          cur_edges[n - 2].Append(top_set[n - 1]);
88
89          for (int i = 1; i < n - 1; i++)
90          {
91              cur_edges[i - 1].Append(top_set[i]);
```

```
92              cur_edges[i].Append(top_set[i]);
93          }
94
95          for (int i = 2; i < n - 1; i++)
96          {
97              cur_edges[n - 1 + i - 2].Append(top_set[i]);
98          }
99          for (int i = 0; i < n - 2; i++)
100         {
101             cur = (n + i) % (n - 1);
102             right = (n - 1) + i;
103             left = right - 1;
104             if (cur == 1)
105             {
106                 left = 0;
107             }
108             cur_edges[cur].Append(bottom_set[i]);
109             cur_edges[left].Append(bottom_set[i]);
110             if (cur != n - 2)
111             {
112                 cur_edges[right].Append(bottom_set[i]);
113             }
114         }
115         for (int i = j; i < j + m; i++)
116         {
117             edges[i] = cur_edges[i % m];
118         }
119         j += m;
120         top_set[0] = 0;
121         top_set[n - 1] = n - 1;
122         for (int i = 0; i < n - 2; i++)
123         {
124             top_set[i + 1] = bottom_set[i];
125         }
126         k += n - 2;
127     }
128
129     for (int i = 0; i < n - 2; i++)
130     {
131         bottom_set[i] = top_set[i + 1];
132     }
133     for (int i = 0; i < n; i++)
134     {
135         top_set[i] = i;
136     }
137     for (int i = 0; i < m; i++)
138     {
```

```
139             cur_edges[i] = Edge();
140         }
141
142         cur_edges[0].Append(top_set[0]);
143         cur_edges[n - 2].Append(top_set[n - 1]);
144
145         for (int i = 1; i < n - 1; i++)
146         {
147             cur_edges[i - 1].Append(top_set[i]);
148             cur_edges[i].Append(top_set[i]);
149         }
150
151         for (int i = 2; i < n - 1; i++)
152         {
153             cur_edges[n - 1 + i - 2].Append(top_set[i]);
154         }
155         for (int i = 0; i < n - 2; i++)
156         {
157             cur = (n + i) % (n - 1);
158             right = (n - 1) + i;
159             left = right - 1;
160             if (cur == 1)
161             {
162                 left = 0;
163             }
164             cur_edges[cur].Append(bottom_set[i]);
165             cur_edges[left].Append(bottom_set[i]);
166             if (cur != n - 2)
167             {
168                 cur_edges[right].Append(bottom_set[i]);
169             }
170         }
171         for (int i = j; i < j + m; i++)
172         {
173             edges[i] = cur_edges[i % m];
174         }
175
176 }
```

3.4. Описание интерфейса программы

Программа запускается с помощью среды разработки Qt Creator: необходимо только нажать на кнопку сборки и запуска в левом нижнем углу.

Существует альтернативный способ запуска с помощью CMake и использования следующей команды: ./build_release.sh (листинг 3..4)

Листинг 3..4 — Release сборка программы для терминала

```
#!/bin/bash

cd .. && rm -rf build && cd planet_system

mkdir ../build && cd ../build

cmake -DCMAKE_BUILD_TYPE=Release ../planet_system

cmake --build .

./planet_system
```

На рисунках (3.2, 3.3, 3.4) представлен интерфейс программного продукта. Три верхние кнопки используются для остановки, запуска и сброса анимации. Следующие два окна для ввода исходных данных для моделирования движения. Затем две кнопки реализуют масштабирование относительно центра Звезды: вперед и назад. Последняя кнопка отвечает за завершение приложения.

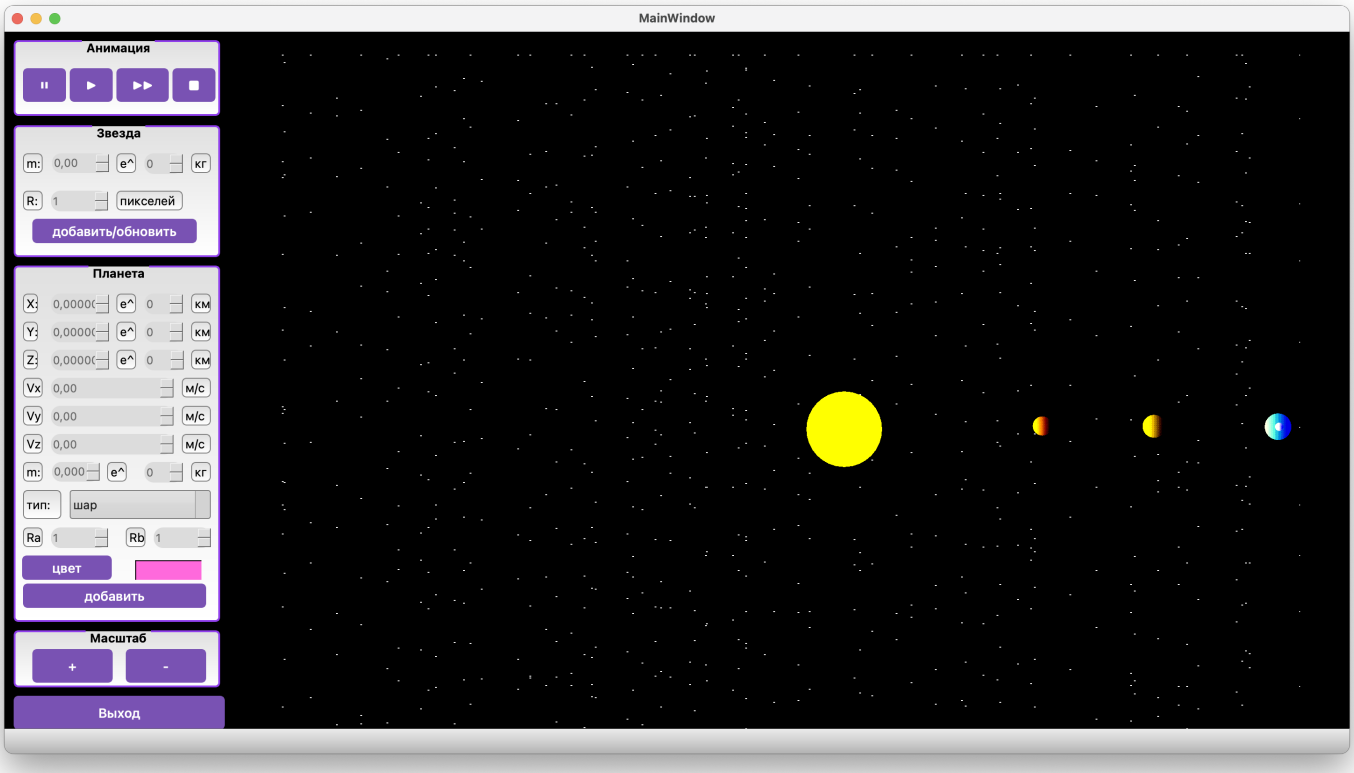


Рисунок 3.2 — Интерфейс ПО

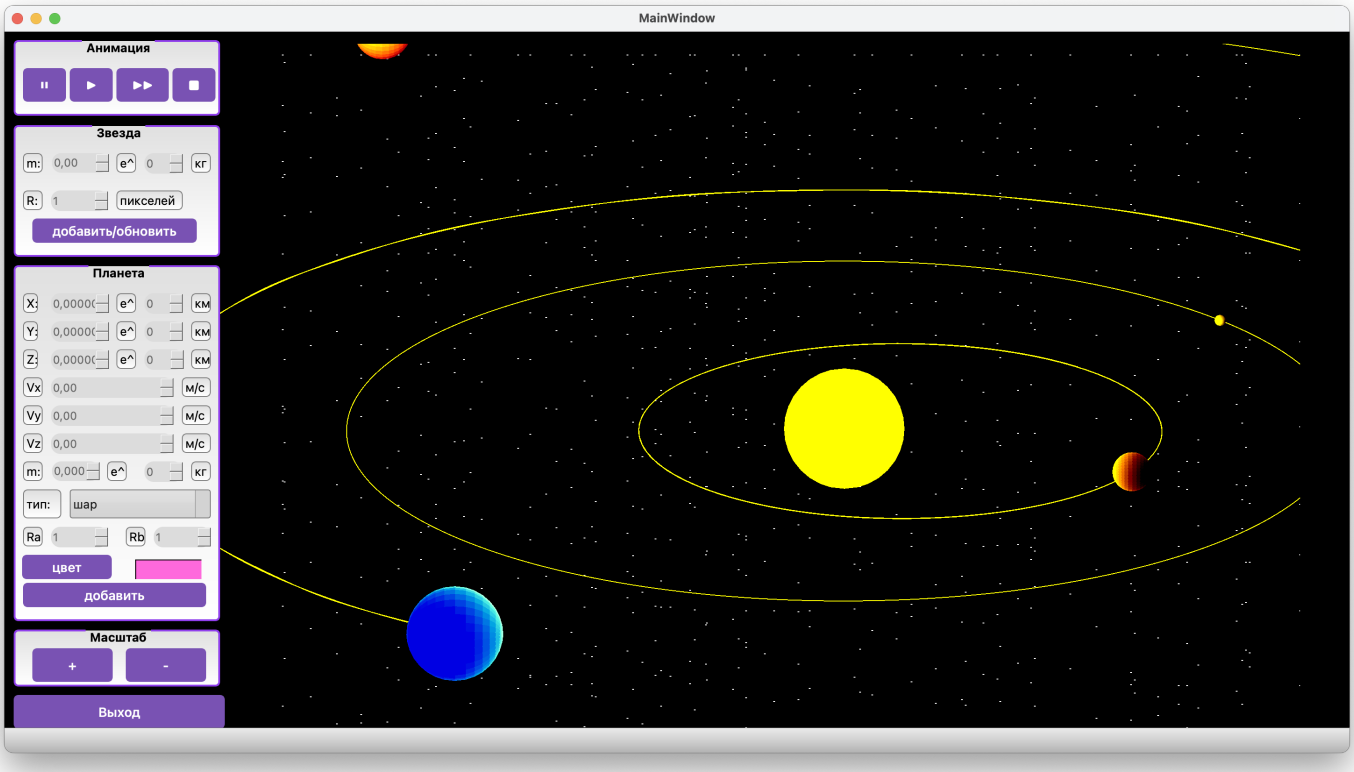


Рисунок 3.3 — Интерфейс ПО

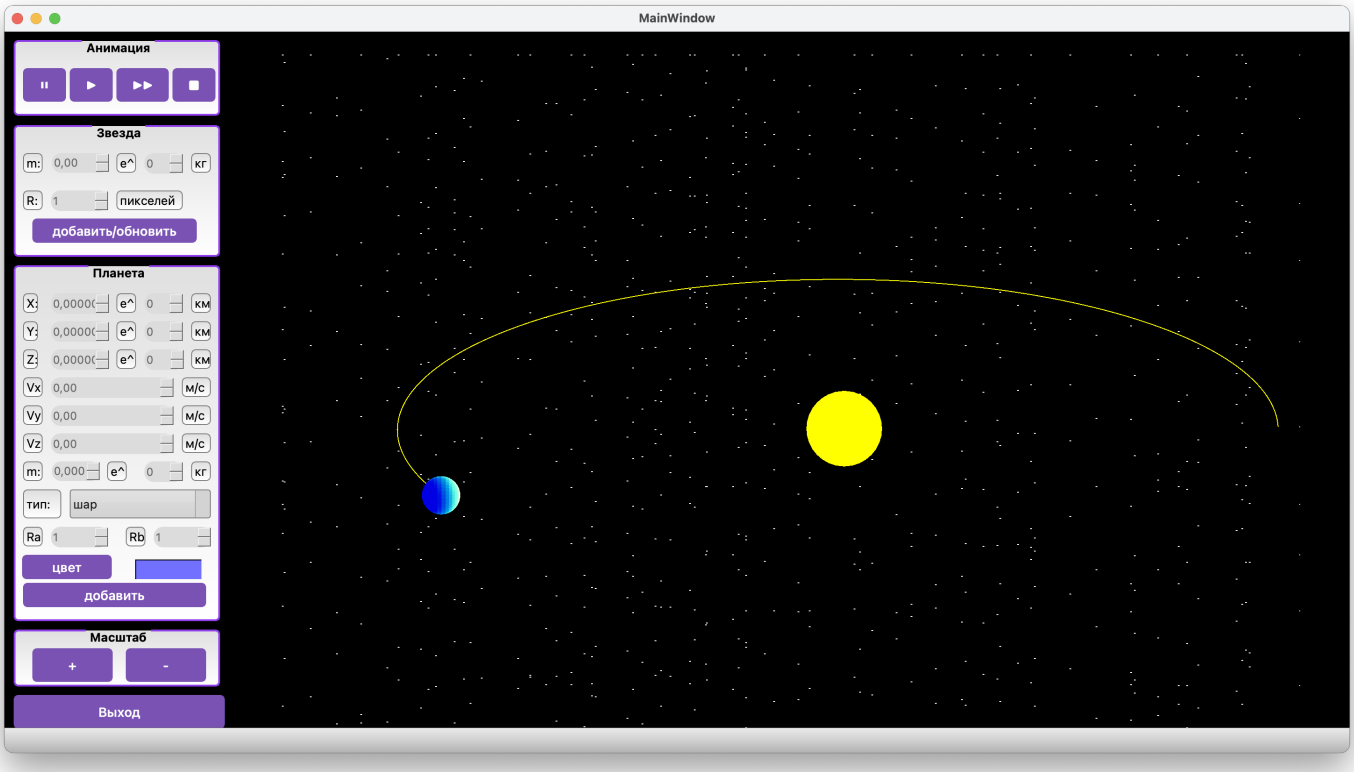


Рисунок 3.4 — Интерфейс ПО

3.5. Модульное тестирование

Было произведено тестирование одного из модулей программы с помощью библиотеки testlib и фреймворка Qt Test.

Для тестирования был выбран модуль myGraphicsView, из которого были выбраны функцыы CalcNewPos, AddNewPlanet и UpdateStar. Создан класс Test, наследуемый от класса QObject. В публичном слоте находится конструктор класса, а в приватном — непосредственно создаваемые тесты. [9]

Листинг 3..5 — Класс TestG для тестирования модуля Vector

```
1  class TestG : public QObject {
2      Q_OBJECT
3  public:
4      explicit TestG(QObject* parent = nullptr, myGraphicsView* g = nullptr);
5
6  private:
7      myGraphicsView* m_g;
8  private slots:
9      void test_g_01();
10     void test_g_02();
11     void test_g_03();
12     void test_g_04();
13 };
```

Таким образом, чтобы добавить новый тест, достаточно создать новый приватный слот. В данном случае добавление теста будет выглядеть следующим образом:

Листинг 3..6 — пример добавления нового модульного теста

```
1  private slots:
2      void test_g_01();
3      void test_g_02();
4      void test_g_03();
5      void test_g_04();
```

Ниже приведен листинг реализации конкретного теста проверки рассчета нормали к заданной поверхности. Проверка корректности полученного результата производится с помощью макросов фреймворка Qt Test. В данном примере используется макрос QCOMPARE, сравнивающий ожидаемое значение с полученным.

Листинг 3..7 — реализация модульного теста test_g_03

```
1  void TestG::test_g_03()
2  {
3      Star s (Point(600,400,5), 10, 0, 2e30) ;
4      Point * test = s.GetPoints();
5      std::vector<Point> test_tmp = {Point(590, 400, 5),
6                                     Point(600, 410, 5),
```

```
                                    Point(610, 400, 5),
                                    Point(600, 400, 15),
                                    Point(600, 390, 5),
                                    Point(600, 400, -5),
                                    Point(600, 410, 5),
                                    Point(0, 0, 0),
                                    Point(0, 0, 0),
                                    Point(0, 0, 0),
                                    Point(0, 0, 0)};
    for(int i = 0; i < s.GetPointsNum(); i++)
    {
        QCOMPARE(qFuzzyCompare(test[i].GetX(), test_tmp[i].GetX()), true);
        QCOMPARE(qFuzzyCompare(test[i].GetY(), test_tmp[i].GetY()), true);
        QCOMPARE(qFuzzyCompare(test[i].GetZ(), test_tmp[i].GetZ()), true);
    }
}
```

Функция qFuzzyCompare используется для корректного сравнения чисел с плавающей точкой, так как QCOMPARE сравнивает числа с помощью оператора «==». Такое сравнение не используется для чисел с плавающей точкой, но подходит для сравнения значений типа bool, что и показано в приведенном примере.

3.6. Реализация управления из командной строки

Для автоматизации работы программы и использования сценария было необходимо реализовать передачу в программу аргументов командной строки и их распознавание самой программой. Для этого была использована библиотека getopt [10], позволяющая выполнить «сканирование» аргументов командной строки и их обработку. Конкретно была использована одноименная функция getopt, которая последовательно перебирает переданные параметры в программу в соответствии с заданной строкой параметров, содержащей имена параметров и признак наличия передаваемого значения («:»). Перечень используемых параметров:

1) Флаг «-t» — указание программе о том, что нужно провести замер времени и вызвать python скрипт, который сгенерирует графики;

2) Флаг «-e» — указание программе о том, что проводится модульное тестирование;

3) Флаг «-i n» — указание программе о том, что нужно отрисовать n сцен и сохранить их в формате *png*.

Листинг 3..8 — пример вызова программы для создания изображений

```
#!/bin/bash

./planet_system -i 20
```

3.7. Выводы из технологического раздела

В данном разделе были рассмотрены средства реализации, описаны основные моменты программной реализации, рассмотрен интерфейс программного продукта, методы тестирования и управления из командной строки.

4. Исследовательский раздел

В данном разделе приведено описание планирования экспериментов и их результаты.

4.1. Технические характеристики

Технические характеристики машины, на которой выполнялось исследование:

— операционная система: macOS Ventura 13.0; [11]
— оперативная память: 8 Gb;
— процессор: Apple M1. [12]

4.2. Постановка исследования

Целью эксперимента является определение зависимости времени генерации изображения модели планетарной системы в зависимости от количества точек, аппроксимирующих поверхности объектов сцены. Количество точек для всех объектов является одинаковым. Эксперимент проводится при параметрах сетки (см 4.), принимающих значение от 0 до 20.

Во время тестирования устройство было подключено к блоку питания и не нагружено никакими приложениями, кроме встроенных приложений окружения, окружением и системой тестирования. Оптимизация компилятора была отключена.

По результатам эксперимента составляется сравнительная таблица, а также строится график зависимостей.

4.3. Результат эксперимента

В таблице 4.1 приведены результаты эксперимента. На рисунке (4.1) представлены результаты эксперимента в графическом виде.

Таблица 4.1 — Зависимость процессорного времени генерации изображения от кол-ва точек

Кол-во точек	Параметр сетки	Время (мс)
10	0	0.000
53	1	1.659
127	2	4.272
233	3	7.564
371	4	13.761
541	5	26.961
743	6	42.740
977	7	72.107
1243	8	97.179
1541	9	159.422
1871	10	199.826
2233	11	283.439
2627	12	363.550
3053	13	459.466
3511	14	627.598
4001	15	823.686
4523	16	1041.410
5077	17	1278.590
5663	18	1590.800
6281	19	1956.390
6931	20	2414.680

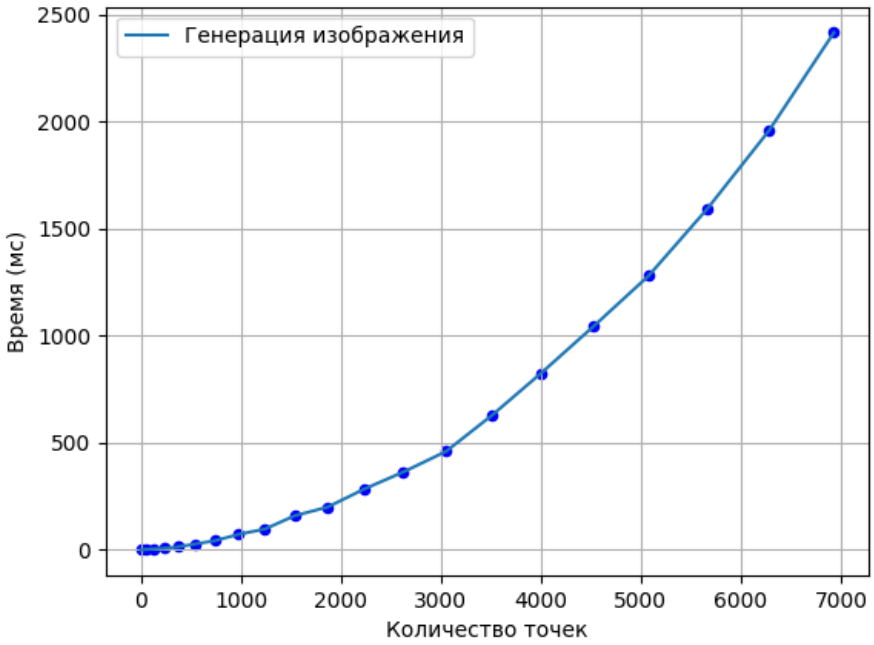


Рисунок 4.1 — Зависимость замеренного времени генерации изображения от кол-ва точек объекта

Так как $\log_{\frac{6931}{53}} \frac{2414}{1} \approx 1.598$, то зависимость стремится к квадратичной.

4.4. Выводы из исследовательсного раздела

По результатам исследования можно сделать вывод, что время генерации изображения зависит от количества точек, аппроксимирующих поверхности объектов, причем зависимость стремится к квадратичной.

Заключение

Цель курсовой работы достигнута – было реализовано программное обеспечение для визуализации модели планетарной системы. Пользователь может динамически просматривать сцену: масштабировать, запускать и ставить на паузу движение модели планетарной системы.

В ходе выполнения работы:

1) были проанализированы существующие алгоритмы компьютерной графики — были рассмотрены алгоритмы удаления невидимых ребер и поверхностей, алгоритмы закраски полигонов и методы моделирования освещения. Существующие алгоритмы были адаптированы для решения поставленной задачи;

2) были реализованы выбранные алгоритмы удаления невидимых ребер и поверхностей, алгоритмы закраски полигонов и методы моделирования освещения;

3) был проведен эксперимент, определяющий зависимости процессорного времени генерации изображения от количества точек, аппроксимирующих поверхности объектов. Было определено, что эта зависимость стремятся к квадратичной.

В дальнейшем этот продукт можно улучшить – использовать реалистичные текстуры для изображения поверхностей космических объектов, добавить звезды, кольца планет и астероиды.

Список источников

1) Zharzhanova Zhainagul Kuandykovna Computer simulation is one of effective methods for the study of Informatics // International scientific review. 2019. №LVII. [Электронный ресурс]. Режим доступа: https://cyberleninka.ru/article/n/computer-simulation-is-one-of-effective-methods-for-the- (дата обращения: 20.09.2022).

2) Galaxy3D // [Электронный ресурс]. Режим доступа: https://galaxy3d.ru (дата обращения: 20.09.2022)

3) SpaceGid // [Электронный ресурс]. Режим доступа: https://spacegid.com/3d-model-solnechnoy- html (дата обращения: 20.09.2022)

4) Компьютерная графика: учебное пособие. – В.Е. Васильев, А.В. Морозов, 2005. – С. 11-27.

5) Computer graphics: theory and applications. – Tosiyasu L.Kunii, 1983. – С. 35-45.

6) Алгоритмические основы машинной графики. (Procedural Elements for Computer Graphics). – Роджерс Д.Ф., 1989. – С. 233-360

7) Компьютерная графика: учебное пособие. Т.О. Перемитина, 2012. – С. 79-92

8) Типы источников света. [Электронный ресурс]. Режим доступа: https://wikichi.ru/wiki/Computer_graphics_lighting (дата обращения: 20.09.2022).

9) Документация библиотеки Qt Test [Электронный ресурс]. Режим доступа: https://doc.qt.io/qt-6/qttest-index.html (дата обращения: 15.10.2022).

10) Документация по библиотеке getopt. [Электронный ресурс]. Режим доступа: https://ru.manpages.org/getopt/3 (дата обращения: 05.10.2022)

11) Документация по операционной системе macOS Ventura. [Электронный ресурс]. Режим доступа: https://www.apple.com/macos/ventura/ (дата обращения: 22.11.2022).

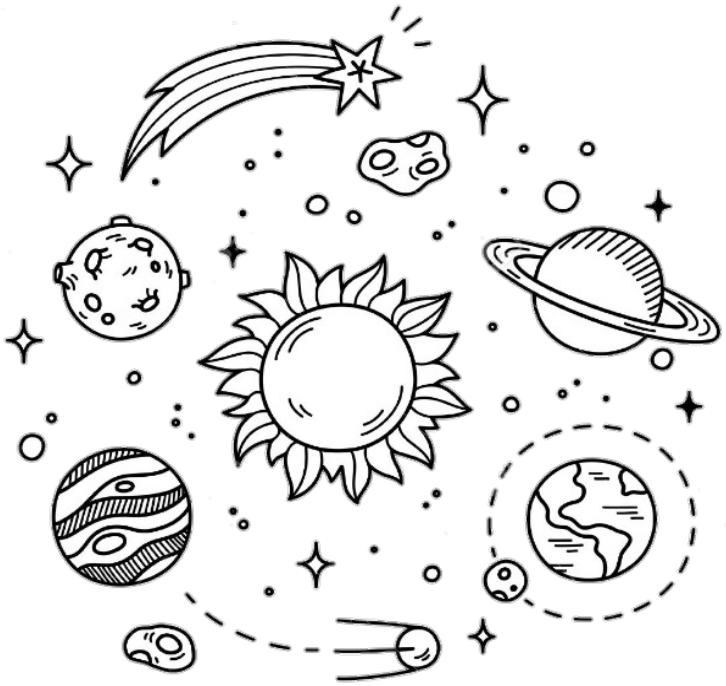
12) Документация по процессору Apple M1. [Электронный ресурс]. Режим доступа: https://www.apple.com/ru/newsroom/2020/11/apple-unleashes-m1/ (дата обращения: 22.11.2022).

13) Курс лекций по Компьютерной графике. – А.В. Куров 2021.

14) Моделирование движения космических тел для исследования устойчивости планетарных систем двойных звёзд – Д. Д. Колпащикова, 2018. — С. 24-30. // [Электронный ресурс]. Режим доступа: https://moluch.ru/archive/224/52740/ (дата обращения: 20.11.2022).

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана»

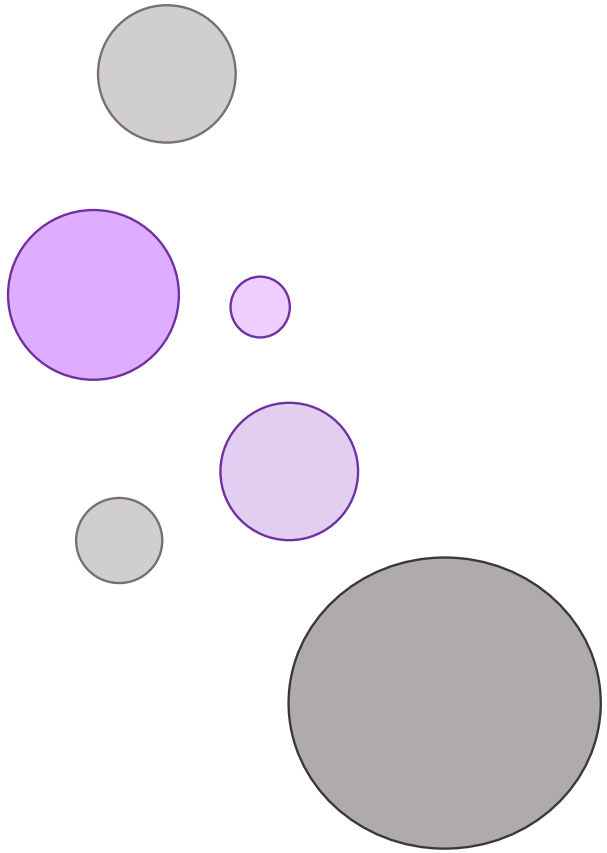
Разработка программного обеспечения для визуализации планетарной системы



Студент: Ляпина Н.В. ИУ7-52Б
Руководитель: Кострицкий А.С.

Москва. 2022

Содержание



Цель

разработка программного обеспечения для визуализации планетарной системы со Звездой в центре

Задачи

Формализация объектов сцены → Постановка задачи → Выбор и адаптация алгоритмов компьютерной Графики для решения задачи → Реализация ПО → Исследование

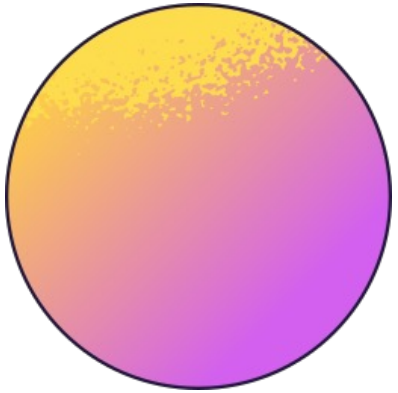
Формализация объектов сцены

Планета

- Представляется сферическим объектом
- Имеет цвет поверхности
- Имеет орбиту
- Имеет массу, скорость и координаты центра масс

Звезда

- Представляется сферой
- Имеет массу
- Поверхность является множеством точечных источников света



Представление модели

Для аппроксимации сферических объектов используется
полигональный способ задания модели

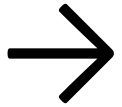
Для хранения полигональной сетки используется список граней

Элемент этого списка – список вершин, составляющих грань

Алгоритмы трехмерной графики

Удаление невидимых ребер и поверхностей

Объекты не пересекаются, редко перекрывают друг друга, определение большей части изображения не требует анализа, важна быстро действенность алгоритма



Выбран алгоритм Z-буфера

Алгоритмы трехмерной графики

Закраска полигонов

Модель освещения

Локальная простая модель освещения: поверхности планет матовые, а фоновое освещение мало. Перенос света между поверхностями пренебрежимо мал.

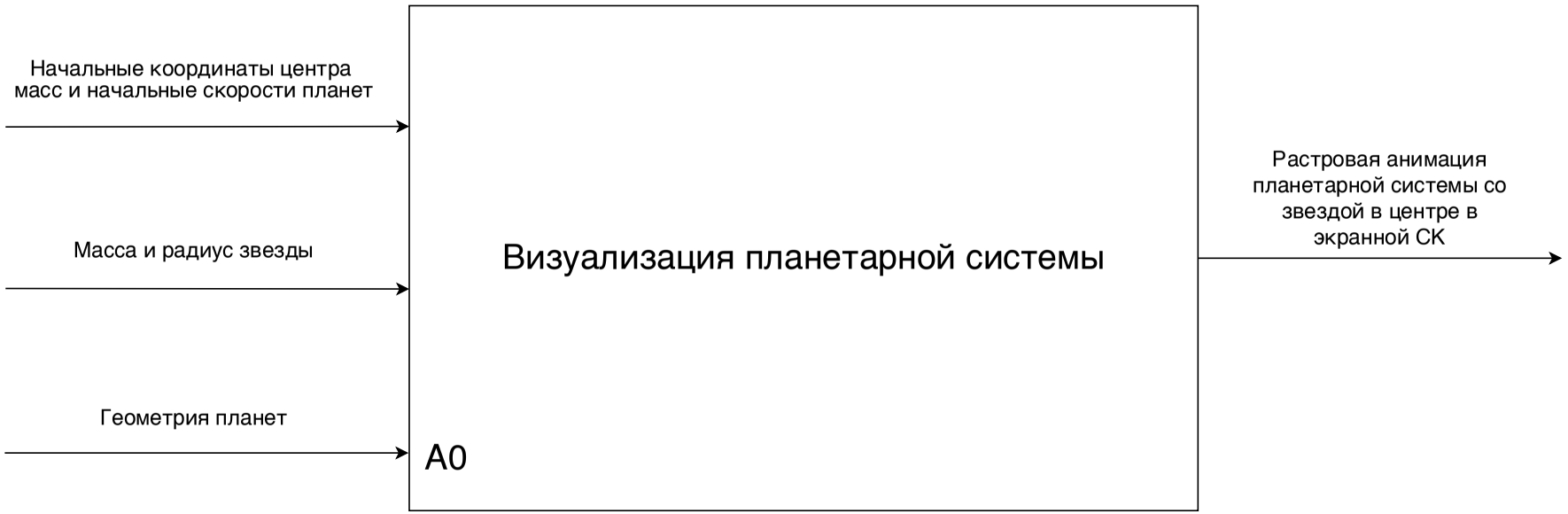
Интенсивность освещенности точки вычисляется по закону Ламберта

Модель закраски

Простая закраска: меньше нагружает ПО при покадровой анимации => быстрее отклик на действия пользователя и более плавная анимация

Ухудшение качества изображения некритично, так как поверхность гладкая и одноцветная

Требования к ПО



Программный продукт

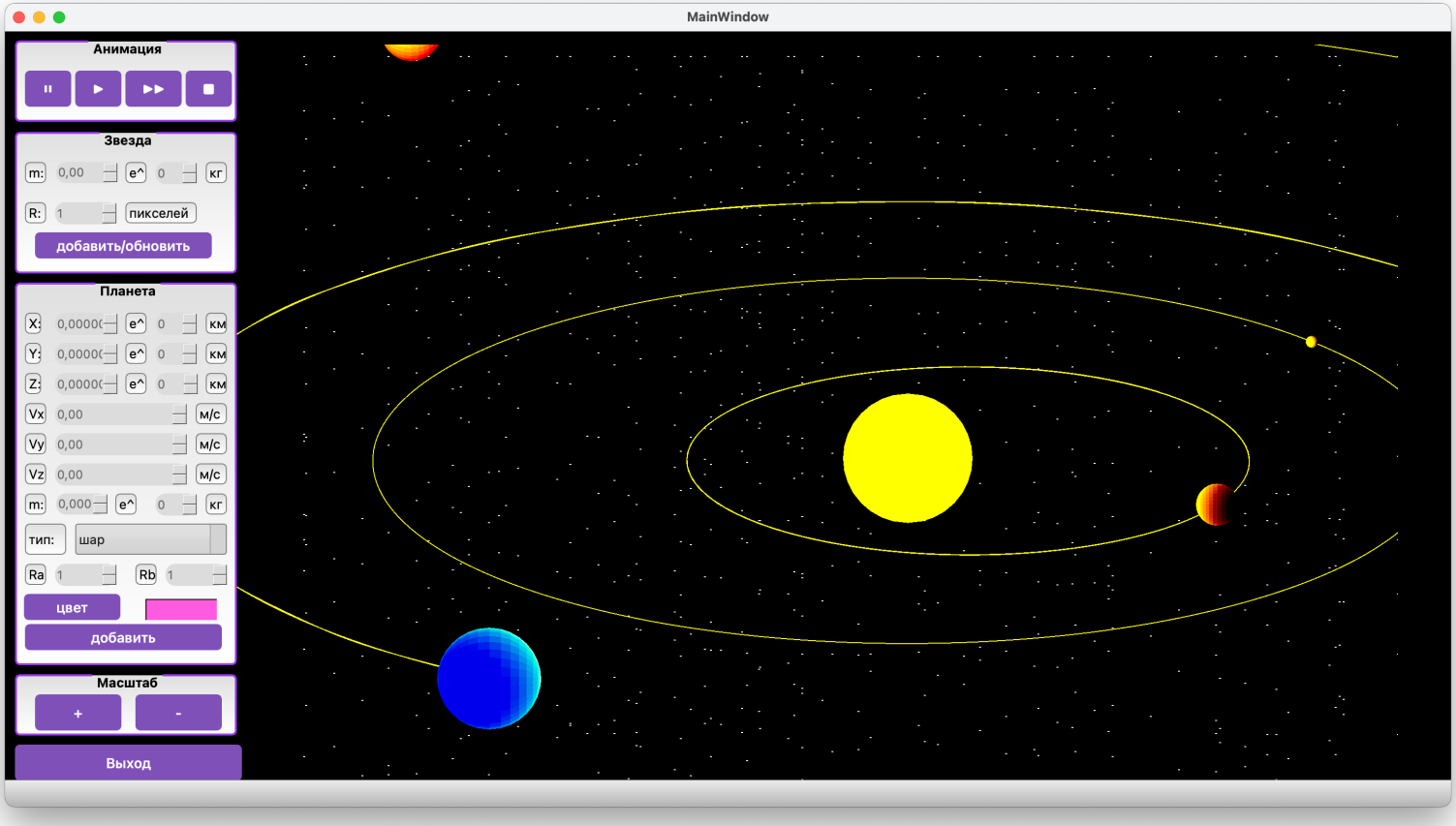
Группа кнопок, отвечающих за управление анимацией

Группа кнопок, отвечающих за ввод информации о Звезде

Группа кнопок, отвечающих за ввод информации о планете

Группа кнопок, отвечающих за масштабирование объектов

Кнопка выхода

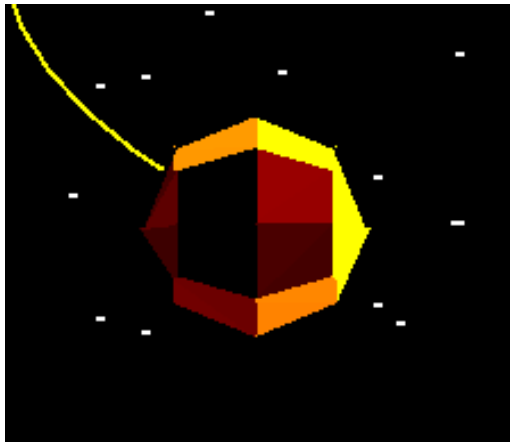


Москва. 2022

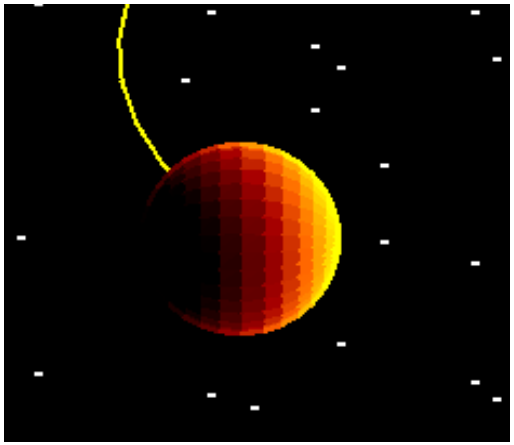
Исследование

Цель: определение зависимости времени генерации изображения модели планетарной˘ системы в зависимости от количества точек, аппроксимирующих поверхности объектов сцены.

Низкополигональная модель



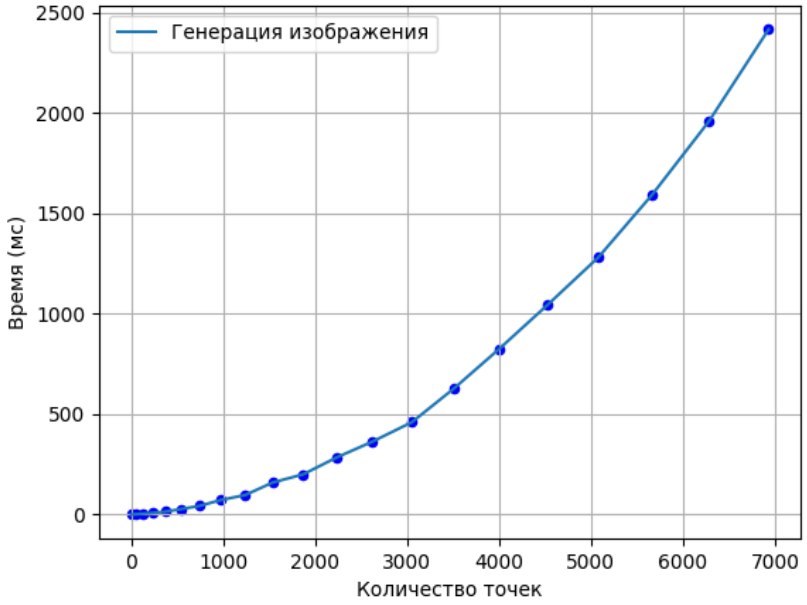
Высокополигональная модель



Исследование

Результаты

Зависимость процессорного времени генерации изображения от кол-ва точек. Причем зависимость стремится к квадратичной.

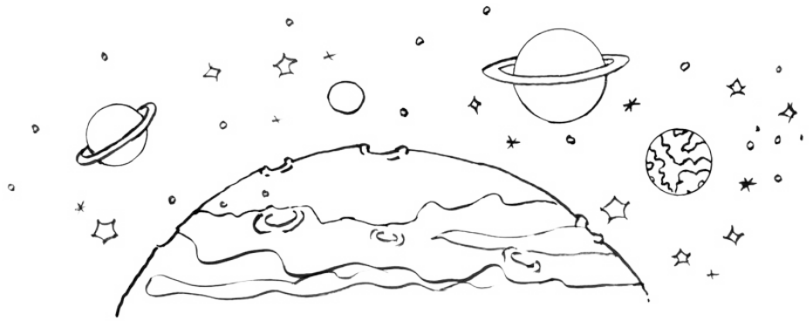


Кол-во точек	Параметр сетки	Время (мс)
10	0	0.000
53	1	1.659
127	2	4.272
233	3	7.564
371	4	13.761
541	5	26.961
743	6	42.740
977	7	72.107
1243	8	97.179
1541	9	159.422
1871	10	199.826
2233	11	283.439
2627	12	363.550
3053	13	459.466
3511	14	627.598
4001	15	823.686
4523	16	1041.410
5077	17	1278.590
5663	18	1590.800
6281	19	1956.390
6931	20	2414.680

Выводы

В ходе выполнения курсовой работы было разработано программное обеспечение для визуализации модели планетарной системы. Пользователь может динамически просматривать сцену: масштабировать, запускать и ставить на паузу движение модели планетарной системы.

В дальнейшем этот продукт можно улучшить – использовать реалистичные текстуры для изображения поверхностей космических объектов, добавить звезды, кольца планет и астероиды.



Москва. 2022