СОДЕРЖАНИЕ

ВВЕДЕНИЕ **5**

1 Аналитическая часть **6**

1.1 Анализ предметной области 6

1.2 Формализация объектов синтезируемой сцены 8

1.3 Анализ методов рендера модели 9

1.3.1 Растеризация 9

1.3.2 Трассировка лучей 10

1.3.3 Маршрирование лучей 10

1.3.4 Сравнение алгоритмов 12

1.4 Анализ алгоритмов закраски 12

1.4.1 Простая закраска 12

1.4.2 Метод закраски Гуро 13

1.4.3 Метод закраски Фонга 13

1.4.4 Выбор алгоритма 14

2 Конструкторская часть **15**

2.1 Требования к программному обеспечению 15

2.2 SDF-функции 15

2.3 Схема алгоритма маршрирования лучей 17

2.4 Модель освещения сцены 19

2.5 Выбор структур данных 21

2.6 Структура программы 21

3 Технологическая часть **23**

3.1 Средства реализации 23

3.2 Проектирование программы 23

3.3 Реализация алгоритмов 26

3.4 Пользовательский интерфейс 29

3.4.1 Взаимодействие с элементами интерфейса 29

3.4.2 Управление камерой 30

3.5 Демонстрация работы программы 31

4 Исследовательская часть **33**
4.1 Цель исследования 33
4.2 Методика исследования 33
4.3 Технические характеристики 34
4.4 Результаты исследования 34
4.4.1 Влияние размеров сцены 34
4.4.2 Влияние количества итераций фрактала 36
4.5 Вывод 38

ЗАКЛЮЧЕНИЕ **39**

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ **40**

ПРИЛОЖЕНИЕ А **42**

ВВЕДЕНИЕ

Целью курсовой работы является разработка программного обеспечения для моделирования изображения трёхмерного фрактала "оболочка Мандельброта". Пользователь должен иметь возможность изменять параметры модели, управлять камерой и источником света.

Для достижения поставленной цели необходимо решить следующие задачи:

1) проанализировать существующие методы и алгоритмы для моделирования изображений аналитических поверхностей;

2) выбрать и описать алгоритмы реализации, необходимые для визуализации фрактала;

3) реализовать выбранные алгоритмы;

4) разработать интерфейс для управления параметрами сцены, изображаемого объекта и положением камеры;

5) провести исследование производительности программы при разных параметрах сцены.

1 Аналитическая часть

В данном разделе проводится анализ существующих алгоритмов для решения поставленной задачи. В результате анализа выбирается алгоритм для дальнейшей реализации.

1.1 Анализ предметной области

Понятия фрактал и фрактальная геометрия появились в конце 70-х годов XX века, и их основоположником принято считать Бенуа Мандельброта, который в 1977 году выпустил книгу «Фрактальная геометрия природы» [1]. В ней ученый дает следующее определение фрактала.

Фрактал — структура, состоящая из частей, которые в каком-то смысле подобны целому. Слово фрактал образовано от латинского *fractus* — состоящий из фрагментов [1].

По общепринятой классификации фракталы делят на геометрические, алгебраические и стохастические [2].

Геометрические фракталы (в двумерном случае) получают на основе некоторой исходной фигуры путём ее дробления и выполнения различных преобразований полученных фрагментов.

Одним из наиболее известных примеров геометрического фрактала является снежинка Коха. В качестве исходного объекта в ней выбран равносторонний треугольник со сторонами единичной длины. На каждой итерации стороны фигуры делятся на 3 равные части, центральная часть отбрасывается, а на её месте строятся две боковые стороны равностороннего треугольника.

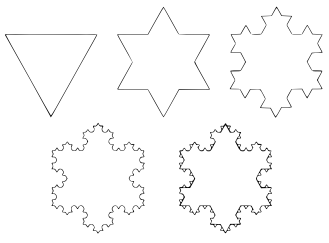

Рисунок 1.1 – Снежинка Коха на разных итерациях

Алгебраические фракталы получают с помощью итерационных процессов по выражениям вида

$$Z_{n+1} = f(Z_n),$$

где z – комплексное число, а f – некая функция.

Наиболее ярким представителем квадратичных алгебраических фракталов является множество Мандельброта. Его получают на комплексной плоскости при итерировании по формуле

$$Z_{n+1} = Z_n^2 + c, \tag{1}$$

где c – комплексная переменная.

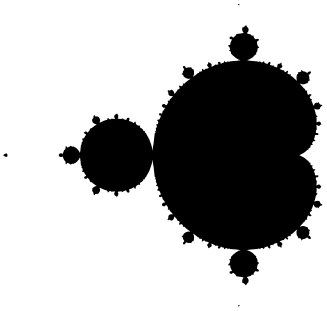

Рисунок 1.2 – Множество Мандельброта

Стохастические фракталы получают при хаотическом изменении каких-либо параметров в итерационном процессе. В отличие от алгебраических и геометрических фракталов, они не являются детерменированными.

Ярким примером стохастического фрактала является траектория броуновского движения на плоскости.

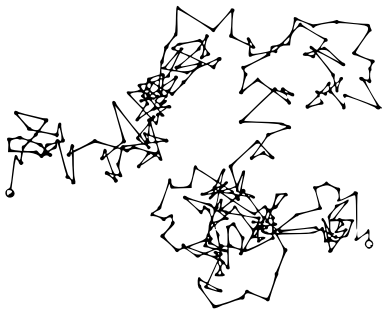

Рисунок 1.3 – Траектория броуновского движения

Новой ступенью развития фрактальной геометрии являются трёхмерные фракталы.

Оболочка Мандельброта — трёхмерный фрактал, аналог множества Мандельброта, созданный Д. Уайтом и П. Ниландером с использованием гиперкомплексной алгебры, основанной на сферических координатах.

Формула для n-ой степени трехмерного гиперкомплексного числа x, y, z следующая:

$$\langle x, y, z\rangle^n = r^n\langle cos(n\theta)cos(n\phi), sin(n\theta)cos(n\phi), sin(n\phi)\rangle, \qquad (2)$$

где

$$r = \sqrt{x^2 + y^2 + z^2};$$
$$\theta = arctan(y/x);$$
$$\phi = arctan(z/\sqrt{x^2 + y^2}) = arcsin(z/r);$$

r – модуль, θ и ϕ – аргументы гиперкомплексного числа.

В модели Уайта и Ниландера используется итерация $z \mapsto z^n + c$, где z и c — трехмерные гиперкомплексные числа [3], на которых операция возведения в натуральную степень выполняется аналогично 2. Для $n > 3$, результатом является трёхмерный фрактал.

1.2 Формализация объектов синтезируемой сцены

Сцена состоит из следующего набора объектов.

1) точечный источник света — задается положением в пространстве и интенсивностью; в зависимости от характеристик источника определяется освещенность объектов сцены; характеристики имеют значения по умолчанию, но имеется возможность их изменить;

2) камера — задается положением в пространстве, углом поворота вокруг каждой из трех координатных осей;

3) изображаемый фрактал — задаётся координатами центра объекта, степенью уравнения фрактала, количеством итераций для построения объекта.

1.3 Анализ методов рендера модели

Рендеринг (*англ. «rendering» — «визуализация»*) — термин, обозначающий процесс получения изображения по модели с помощью компьютерной программы.

Фрактал «оболочка Мандельброта» является алгебраическим, т. е. описывается математическими уравнениями и не имеет явной геометрии (например, треугольников или вокселей).

Рассмотрим основные методы рендера моделей.

1.3.1 Растеризация

Растровое изображение — это изображение, представляющее собой сетку пикселей – цветных точек на мониторе, бумаге и других отображающих устройствах.

Растеризация — это процесс получения растрового изображения.

Технология основана на обходе лучем вершин треугольного полигона, который сохраняет свою форму даже после попадания из трехмерного пространства в двухмерное. Каждая точка объекта в трехмерном пространстве переводится в точку на экране, а затем точки соединяются и получается изображение исходного объекта [4].

Преимуществом метода является то, что современные компьютеры оптимизированы для рендеринга растровых изображений, из-за чего процесс растеризации достаточно быстрый.

Недостатки:

— изображение на выходе получается ступенчатым, требуется дополнительное сглаживание;

— метод работает с примитивами, следовательно требуется ресурсоемкая полигонализация сложных объектов;

— для моделей сцены могут использоваться миллионы полигонов, каждый из которых должен быть подвержен растеризации;

— каждый пиксель может обрабатываться множество раз (например, при наложении полигонов);

1.3.2 Трассировка лучей

Трассировка лучей (*англ. «ray tracing»*) — это технология отрисовки трехмерной графики, симулирующая физическое поведение света [5].

Суть алгоритма заключается в моделировании пути света для создания изображения. Лучи света, исходящие из камеры, проверяются на пересечение с полигонами и примитивами, принадлежащими объектам сцены. Для каждого пересечения вычисляются параметры освещения, и испускаются вторичные лучи, которые используются для определения отражений и теней.

Преимущества:

— вычислительная сложность метода слабо зависит от сложности сцены [5];

— отсечение невидимых поверхностей, перспектива и корректное изменение поля зрения являются следствием алгоритма;

Недостатки:

— алгоритм может работать только с примитивами, следовательно для сложных объектов необходимо проводить полигонализацию;

— метод имеет крайне высокую вычислительную сложность и является крайне ресурсоёмким [5].

1.3.3 Марширование лучей

Марширование лучей (*англ. «ray marching»*) — разновидность алгоритма трассировки лучей.

Как и трассировка лучей, этот метод испускает луч в сцену для каждого пикселя экрана. Однако, в отличии от трассировщика лучей, данный метод не пытается вычислить пересечение луча и объекта аналитически. В

нём происходит смещение текущего положения вдоль луча, пока не найдется точка, пересекающая объект. Такая операция переноса является простой и нересурсозатратной, в отличие от аналитического вычисления пересечения, однако данный способ является менее точным, чем обычная трассировка.

Для сцен, состоящих из непрозрачных объектов, можно ввести ещё одну оптимизацию в данный алгоритм. Она заключается в использовании полей расстояний со знаком.

Поле расстояний со знаком (*англ. «signed distance field»*) — это функция, получающая на вход точку пространства и возвращающая кратчайшее расстояние от этой точки до поверхности каждого объекта в сцене [6]. Если точка находится внутри объекта, то функция возвращает отрицательное число.

Эта оптимизация позволяет заметно ограничить количество шагов при движении вдоль луча, что увеличивает эффективность метода.

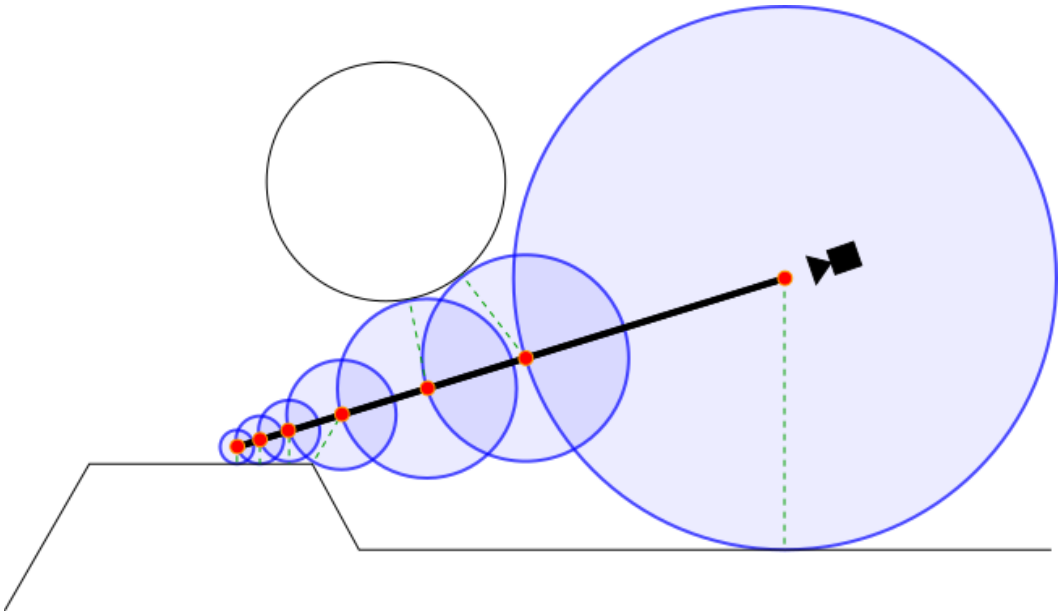

Рисунок 1.4 – Визуализация марширования лучей с использованием SDF. Красным отмечены все проверяемые точки

Преимущества:

— решает проблему производительности трассировки лучей за счёт оптимизаций;

— не требует разбиения поверхностей на примитивы, подходит для рендера сложных объектов;

— качество получаемого изображения сопоставимо с получаемым при трассировке лучей.

Недостаток метода заключается в том, что пересечение вычисляется менее точно, чем при трассировке лучей, имеется некоторая погрешность.

1.3.4 Сравнение алгоритмов

Сравнение методов рендеринга будет проводиться по следующим критериям.

1) качество изображения;

2) эффективность по времени выполнения рендера;

3) возможность рендерить сложные объекты без разбиения на примитивы.

В таблице 1.1 приведено сравнение рассмотренных методов по указанным выше критериям.

Таблица 1.1 – Сравнение алгоритмов

Метод	Качество	Быстродействие	Сложные объекты
Растеризация	3	1	–
Ray Tracing	1	3	–
Ray Marching	2	2	+

Алгоритм маршрирования лучей подходит для рендера фракталов лучше прочих из рассмотренных, так как он не требует разбиения модели на примитивы для работы, не требует хранения информации о вершинах и гранях объекта, а также обладает высоким быстродействием.

1.4 Анализ алгоритмов закраски

1.4.1 Простая закраска

Алгоритм подразумевает однотонную закраску кажого многоугольника, принадлежащего объекту, в зависимости от интенсивности в одной

из его точек. При этом предполагается следующее:

— источник света расположен в бесконечности;

— наблюдатель находится в бесконечности;

— многоугольник представляет собой реальную моделируемую поверхность, а не является аппроксимацией криволинейной поверхности.

Ключевым недостатком данного алгоритма является то, что все точки грани будут иметь одинаковую интенсивность.

1.4.2 Метод закраски Гуро

Метод Гуро позволяет устранить дискретность изменения интенсивности и создать иллюзию гладкой криволинейной поверхности [7]. Он основан на интерполяции интенсивности. Сам алгоритм можно разбить на четыре этапа.

1) вычисление нормалей ко всем полигонам объекта;

2) определение нормали в вершинах путём усреднения нормалей по всем граням, принадлежащим рассматриваемым вершинам;

3) вычисление значения интенсивности в вершинах на основе выбранной модели освещения;

4) закраска каждого многоугольника путём линейной интерполяции в вершинах сначала вдоль ребер, а затем между ними.

Достоинством метода является большая реалистичность получаемого изображения по сравнению с простой закраской.

1.4.3 Метод закраски Фонга

При такой закраске, в отличие от метода закраски Гуро, интерполируется значение вектора нормали, а не интенсивности [8]. Алгоритм также имеет четыре основных этапа:

1) вычисление векторов нормалей к каждой грани и к каждой вершине грани;

2) интерполяция векторов нормалей вдоль ребер грани;

3) линейная интерполяция векторов нормалей вдоль сканирующей строки;

4) вычисление интенсивности в очередной точке сканирующей строки.

Достоинства:

— можно достичь лучшей аппроксимации кривизны поверхности;

— более реалистичные блики, чем в методе Гуро [7].

Недостатки:

— высокая ресурсоёмкость;

— высокая вычислительная сложность.

1.4.4 Выбор алгоритма

В качестве метода закраски был выбран метод Фонга, так как он позволяет получить лучшую аппроксимацию сложной поверхности, чем простая закраска и метод закраски Гуро.

Вывод

В данном разделе было проведено описание объектов сцены, рассмотрены алгоритмы визуализации сцены, отвечающие заданным требованиям. В качестве алгоритма для рендера изображения был выбран алгоритм маршрирования лучей как быстродейственный алгоритм, имеющий небольшую погрешность и позволяющий визуализировать сложные объекты сцены. В качестве алгоритма закраски был выбран метод закраски Гуро, так как он является быстродейственным и позволяет достаточно качественно закрашивать объекты.

2 Конструкторская часть

В данном разделе описаны используемые поля расстояний со знаком, алгоритм марширования лучей, модель освещения сцены, а также структура программы.

2.1 Требования к программному обеспечению

Программа должна предоставлять следующую функциональность:

— добавление объекта из набора и настройка его параметров;

— изменение положения камеры в пространстве;

— изменение характеристик источника света.

2.2 SDF-функции

Поля расстояний со знаком (SDF) используются для нахождения кратчайшего расстояния от точки в пространстве до поверхности тела. Знак результата определяет положение точки относительно данного тела:

— вне объекта, если $SDF(p) > \varepsilon$;

— на границе объекта, если $|SDF(p)| \leq \varepsilon$;

— внутри объекта, если $SDF(p) < -\varepsilon$;

где $\varepsilon \to 0$ — допустимая погрешность при работе с числами с плавающей точкой.

SDF функции определяются для конкретных видов тел. Так, SDF для сферы имеет вид:

$$SDF_{sphere}(x, y, z) = \sqrt{(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2)} - r, \qquad (3)$$

где

— $C(x_0, y_0, z_0)$ — координаты центра сферы;

— r — радиус сферы.

Фрактальные структуры имеют итеративную природу, и поэтому SDF для них зачастую задаются в итеративном виде. Так, на рисунке 2.1 представлена схема поля расстояния со знаком для оболочки Мандельброта.

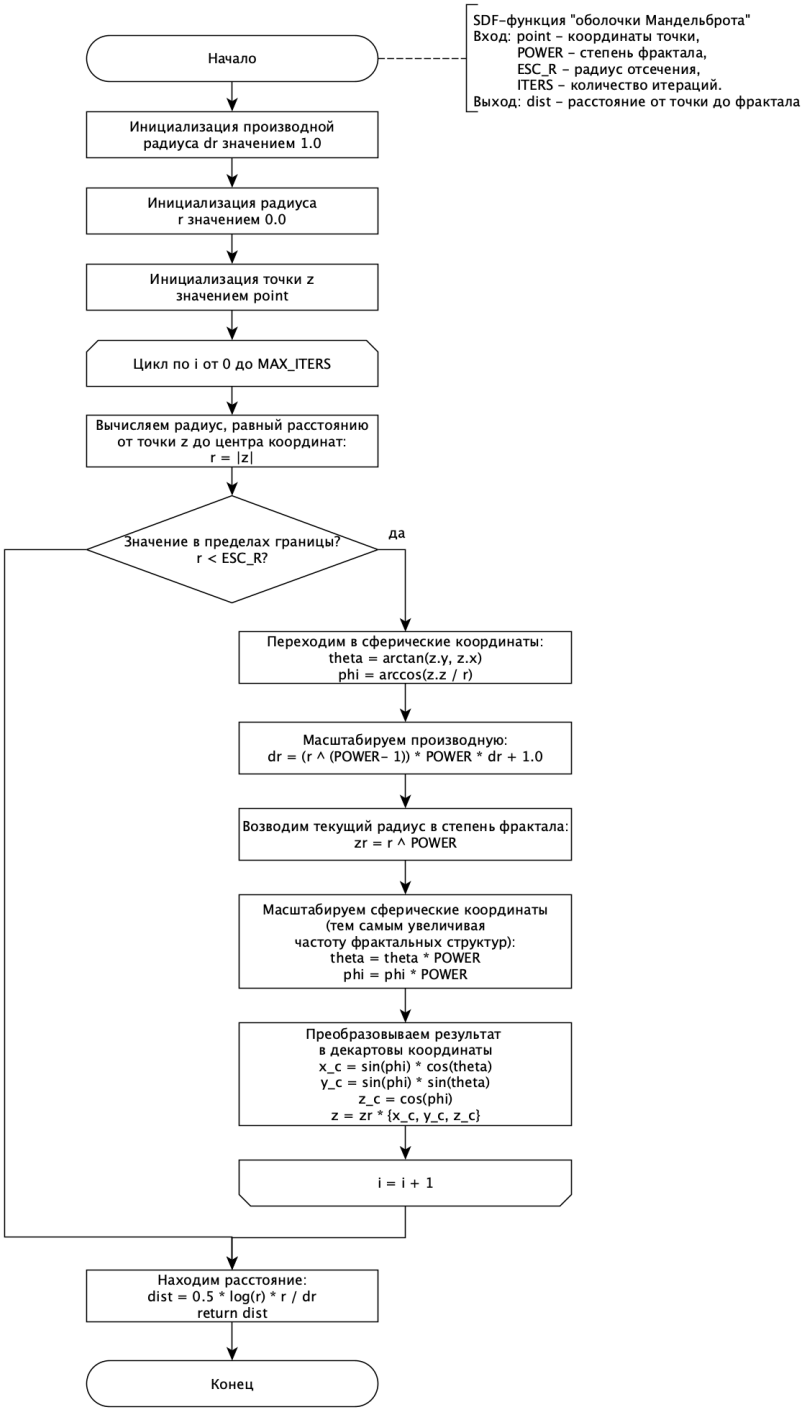

Рисунок 2.1 – Схема алгоритма SDF для оболочки Мандельброта

2.3 Схема алгоритма маршрирования лучей

Алгоритм маршрирования лучей отрисовывает объекты сцены при помощи полей расстояния со знаком (SDF). Маршрирование лучей предполагает итеративное перемещение точки вдоль направления обзора (от камеры к объекту) и проверку результата на основе полученного из SDF-функции значения.

Схема алгоритма маршрирования лучей представлена на рисунке 2.2.

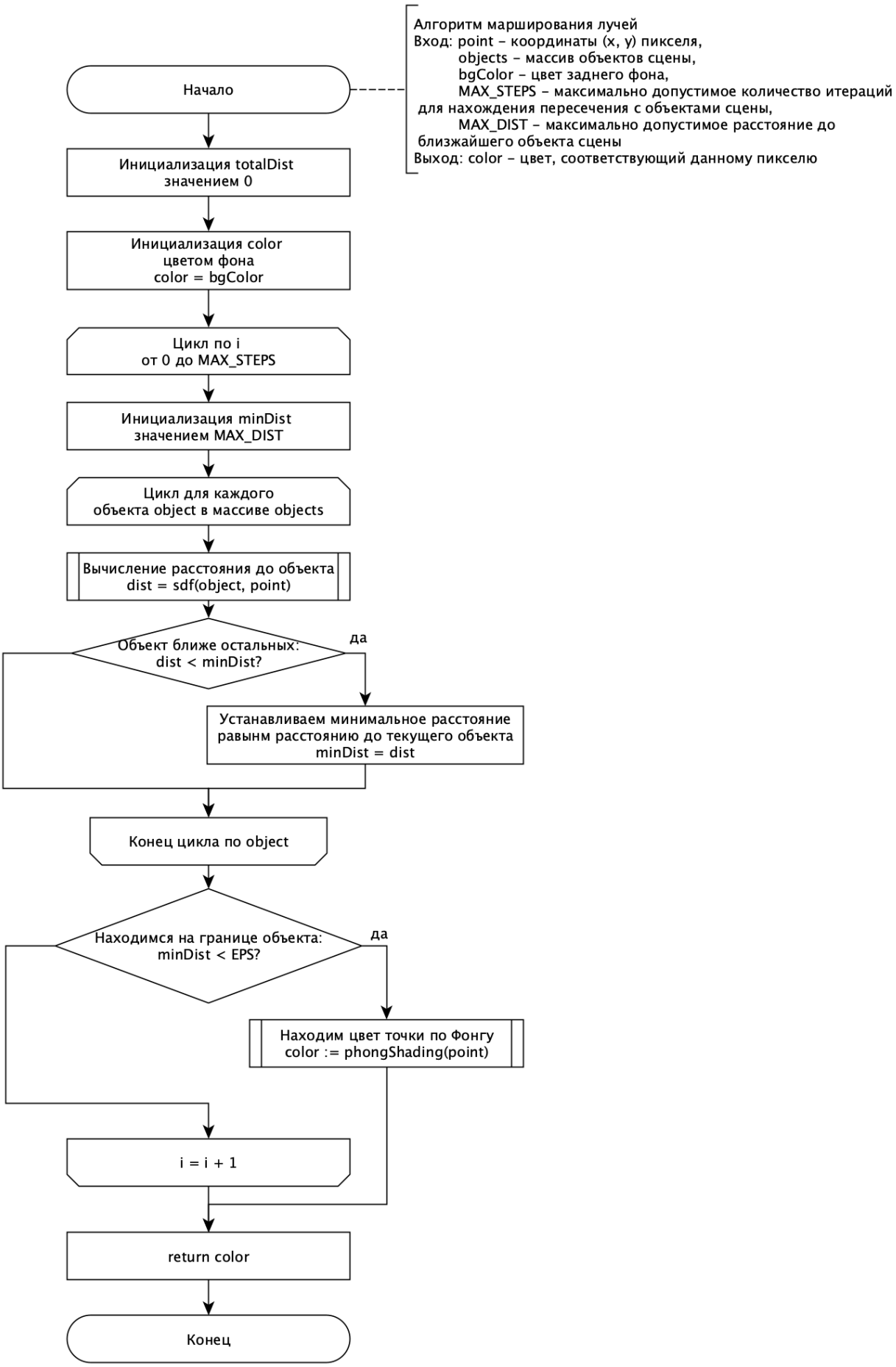

Рисунок 2.2 – Схема алгоритма маршрирования лучей

2.4 Модель освещения сцены

Для освещения сцены используется модель Фонга. Согласно этой модели, освещенность сцены является суммой трех компонент:

— *ambient* — фоновое освещение;

— *diffuse* — диффузное освещение;

— *specular* — бликовое освещение.

Общая формула расчета освещенности в точке представлена в формуле (4).

$$I = k_a i_a + k_d i_d (\vec{N} * \vec{L}) + k_s i_s (\vec{N} * \vec{R})^{\alpha}, \tag{4}$$

где

— k_a, k_d, k_s — коэффициенты фонового, диффузного и бликового освещения соответственно;

— i_a, i_d, i_s — интенсивности фонового, диффузного и бликового освещения соотвественно;

— $\vec{N}$ — вектор нормали к поверхности в точке;

— $\vec{L}$ — падающий луч (направление на источник света);

— $\vec{R}$ — отраженный луч;

В стандартном случае, для вычисления вектора нормали использую нормированную взвешенную сумму векторов нормали граней, которым эта точка принадлежит:

$$\vec{N} = \frac{a_1\vec{n_1} + \cdots + a_k\vec{n_k}}{||a_1\vec{n_1} + \cdots + a_k\vec{n_k}||} \tag{5}$$

Этот метод вычисления нормали работает только в тех случаях, когда информация об объектах хранится в виде полигональной сетки, и есть возможность определить, к каким граням приналдлежит указанная точка.

Так как в алгоритме маршрирования лучей обхекты задаются в аналитическом виде, и получить информацию о вершинах и гранях объекта не представляется возможным. Исходя из факта, что градиент функции $F(x, y, z)$ коллинеарен её нормали, мы можем с высокой точностью аппроксимировать нормаль:

$$\vec{N} = gradSDF = \begin{pmatrix} SDF(x+\Delta, y, z) - SDF(x-\Delta, y, z) \\ SDF(x, y+\Delta, z) - SDF(x, y-\Delta, z) \\ SDF(x, y, z+\Delta) - SDF(x, y, z-\Delta) \end{pmatrix}^T, \quad (6)$$

Схема алгоритма закраски по Фонгу представлена на рисунке 2.3.

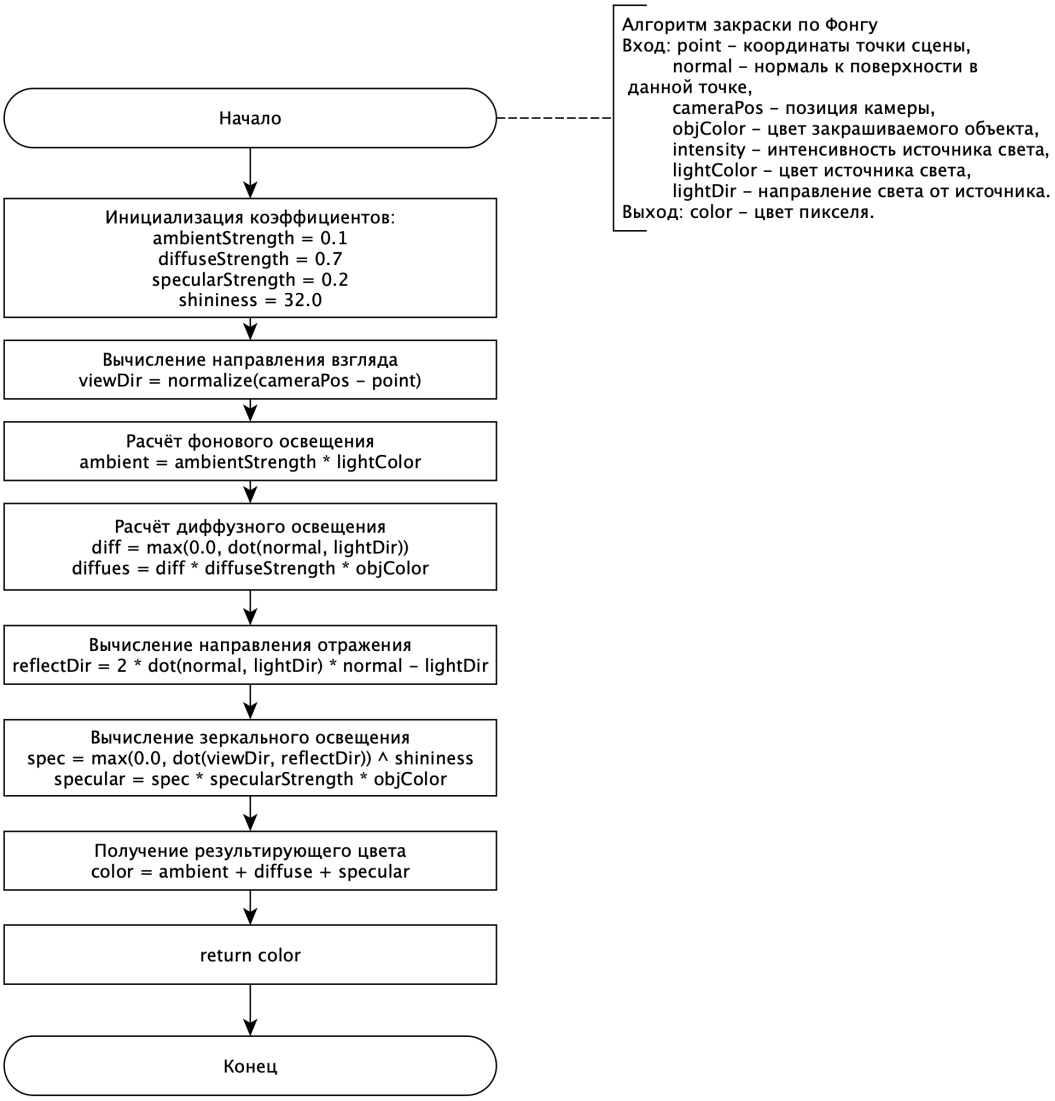

Рисунок 2.3 – Схема алгоритма закраски по Фонгу

2.5 Выбор структур данных

В работе будут использованы следующие типы и структуры данных:

— точка — вектор из трех вещественных чисел соответствующих координатам по каждой из осей трехмерной декартовой системы координат;

— цвет — вектор из трех целых чисел, принимающих значение в диапазоне $[0, 255]$, соответствует цветовой модели RGB;

— сфера — объект сцены, содержит радиус, цвет объекта;

— фрактал — объект сцены, содержит степень, используемую в формуле фрактала, количество итераций для построения, радиус отсечения, цвет объекта;

— источник света — содержит интенсивность и точку своего положения в пространстве;

— камера — содержит точку своего положения в пространстве, угол обзора и тройку векторов, задающую направление камеры;

— сцена — хранит в себе набор объектов, камеру, источник света.

2.6 Структура программы

Программа состоит из следующих модулей:

— Scene — модуль отвечающий за хранение данных о сцене и её объектах;

— Camera — модуль для работы с камерой, вычисления направления взгляда, обработки движения камеры в пространстве;

— Shading — модуль, отвечающий за расчет освещенности;

— Rendering — модуль визуализации, выполняющий отрисовку сцены;

Вывод

В данном разделе описаны алгоритмы, используемые для моделирования трехмерного фрактала «оболочка Мандельброта», его рендеринга и закраски.

3 Технологическая часть

В данном разделе представлены средства разработки программы, детали её реализации и описание пользовательского интерфейса.

3.1 Средства реализации

В качестве основного языка программирования выбран C++ [9]. Выбор языка обусловлен следующими факторами:

— поддержка объектно-ориентированной парадигмы;

— высокая производительность;

— наличие строгой типизации;

Для реализации графического интерфейса (GUI) программы был использован фреймворк Qt [10]. Он предоставляет возможность создавать кроссплатформенные динамические интерфейсы, и содержит широкий набор инструментов для работы с трехмерными объектами, полезных в рамках данной задачи, таких, как собственные реализации трехмерных векторов и матриц.

Для оптимизации работы алгоритма маршрирования лучей была дополнительно реализована версия алгоритма, использующая графический процессор для вычислений. Для взаимодействия с графическим процессором была использована библиотека OpenGL [11]. Её выбор обусловлен кроссплатформенностью и богатым набором функций для работы с графическим процессором.

3.2 Проектирование программы

Программа спроектирована с использованием объектно-ориентированного подхода. Компонентами системы являются классы, отвечающие за отрисовку и закраску объектов, изменение параметров сцены.

Основные классы разработанной программы:

— Camera — абстрактный класс, определяющий базовое поведение камеры;

— FreeCamera — камера, способная свободно перемещаться в пространстве;

— SphereCamera — камера, двигающаяся по сферической траектории вокруг объекта сцены;

— LightSource — класс, описывающий источник света;

— SceneObject — абстрактный класс, определяющий базовый набор параметров объекта сцены, в том числе наличие SDF;

— Sphere — объект-сфера;

— Mandelbulb — фрактал «оболочка Мандельброта»;

— RenderingAlgorithm — абстрактный класс, реализующий паттерн «Стратегия» для алгоритма рендеринга;

— RayMarching — алгоритм маршрирования лучей (однопоточный);

— ConcurrentRayMarching — алгоритм маршрирования лучей (многопоточный);

— OpenGLRenderer — алгоритм маршрирования лучей (графический процессор, OpenGL);

— ShadingAlgorithm — абстрактный класс, реализующий паттерн «Стратегия» для алгоритма закраски;

— SimpleShading — простая модель закраски;

— PhongShading — модель закраски по Фонгу;

— GouraudShading — модель закраски по Гуро;

— Scene — класс, представляющий сцену. Содержит камеру, источник света, объекты сцены, а также класс, определяющий стратегию рендеринга.

Диаграмма разработанных классов приведена на рисунке (3.1).

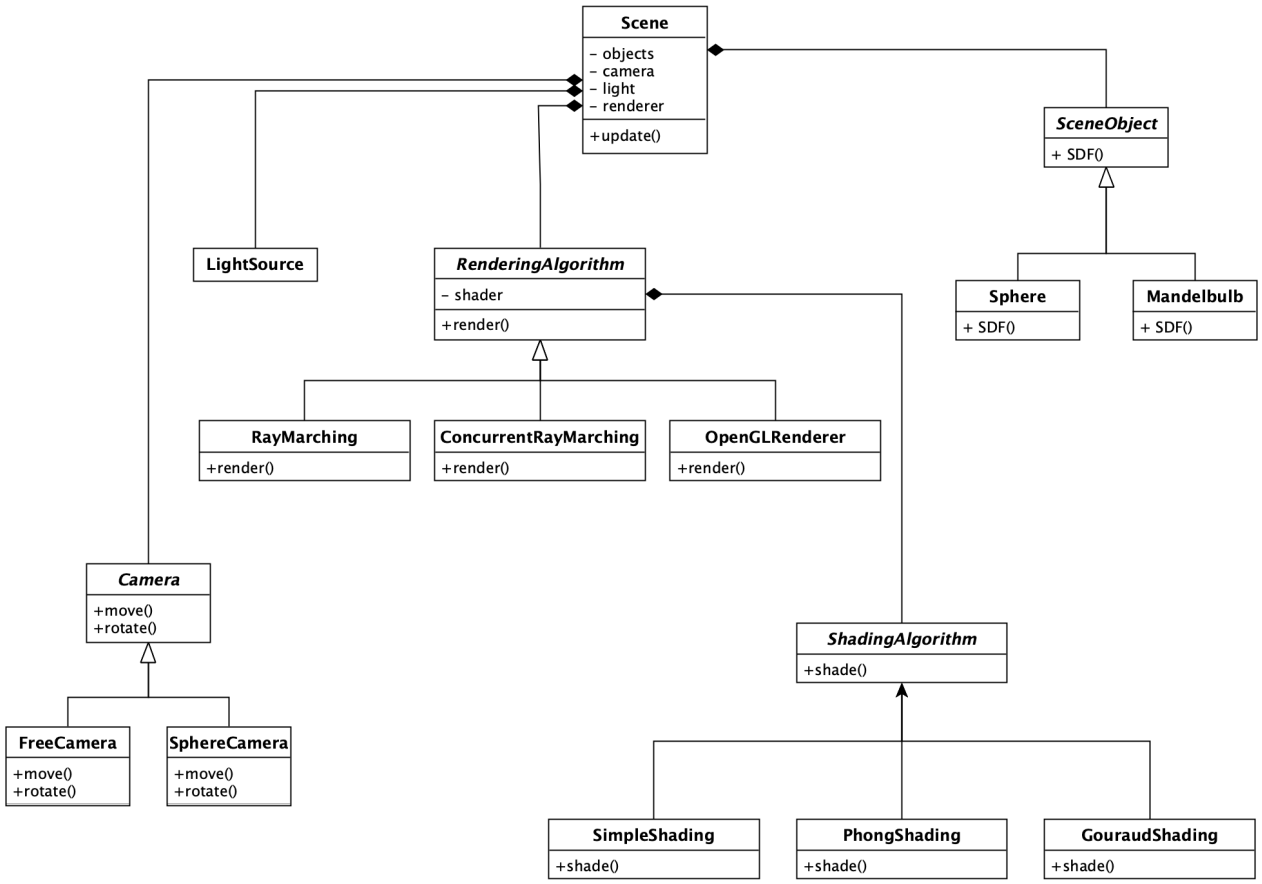

Рисунок 3.1 – Диаграмма разработанных классов

3.3 Реализация алгоритмов

В листинге 1 представлена функция SDF для оболочки Мандельброта. Она вычисляет расстояние от точки до поверхности фрактала.

Листинг 1 – Функция SDF для оболочки Мандельброта

```
qreal Mandelbulb::SDF(const QVector3D &point) {
    QVector3D z = (point - m_pos);
    qreal dr = 1.0;
    qreal r = 0.0;
    for (int i = 0; i < m_iters; ++i) {
        r = z.length();
        if (r > m_rad) break;
        qreal theta = std::atan2(z.y(), z.x());
        qreal phi = std::acos(z.z() / r);
        dr = std::pow(r, m_power - 1.0) * m_power * dr + 1.0;
        qreal zr = std::pow(r, m_power);
        theta *= m_power;
        phi *= m_power;
        z = zr * QVector3D(std::sin(phi) * std::cos(theta),
                           std::sin(phi) * std::sin(theta),
                           std::cos(phi))
            + (point - m_pos);
    }
    return 0.5 * std::log(r) * r / dr;
}
```

В листинге 2 представлена реализация алгоритма маршрирования лучей.

Листинг 2 – Алгоритм маршрирования лучей

```
QColor RayMarching::castRay(const QVector3D &origin, const
    QVector3D &direction, const QColor &bgColor, Camera *cam,
    QList<SceneObject *> *objects, LightSource *lightSource)
{
    qreal totalDist = 0.0f;
    qreal k = cam->getRadius() / cam->DEFAULT_RADIUS;
    qreal adjustedEPS = qMax(0.0001f, EPS * k);
    int adjustedSteps = MAX_STEPS + static_cast<int>(k / 50);
    qreal adjustedMaxDist = MAX_DIST * k;

    for (qsizetype step = 0; step < adjustedSteps; ++step)
    {
        QVector3D point = origin + direction * totalDist;
        qreal minDist = adjustedMaxDist;
        SceneObject *closestObj = nullptr;

        for (SceneObject *obj : *objects) {
            qreal dist = obj->SDF(point);
            if (dist < minDist) { minDist = dist; closestObj =
                obj; }
        }

        if (minDist < adjustedEPS) {
            if (closestObj) {
                QVector3D normal = this->normal(point,
                    closestObj);
                return m_shading->shade(cam, normal,
                    closestObj->getColor(), direction,
                    lightSource);
            }
        }
        if (totalDist > adjustedMaxDist) break;
        totalDist += std::min(minDist * 0.8,
            (double)adjustedMaxDist);
    }
    return bgColor;
}
```

В листинге 3 представлена реализация алгоритма закраски по Фонгу.

Листинг 3 – Алгоритм закраски по Фонгу

```
QColor PhongShading::shade(Camera *cam,
                           const QVector3D &normal,
                           const QColor &baseColor,
                           const QVector3D &hitPoint,
                           LightSource *lightSource)
{
    QVector3D cameraPos = cam->getPos();
    QVector3D ambientLight(0.1f, 0.1f, 0.1f);
    qreal ambientStrength = 0.1f;
    QVector3D lightDir = lightSource->getDirection(hitPoint);
    QVector3D viewDir = (cameraPos - hitPoint).normalized();
    qreal diffuseStrength = 0.7f;
    qreal specularStrength = 0.2f;
    qreal shininess = 32.0f;
    QVector3D ambient = ambientStrength * ambientLight;
    qreal diff = std::max(0.0f, QVector3D::dotProduct(normal,
       lightDir));
    QVector3D diffuse = diff * diffuseStrength *
       QVector3D(baseColor.redF(), baseColor.greenF(),
       baseColor.blueF());
    QVector3D reflectDir = 2 * QVector3D::dotProduct(normal,
       lightDir) * normal - lightDir;
    qreal spec = std::pow(std::max(0.0f,
       QVector3D::dotProduct(viewDir, reflectDir)), shininess);
    QVector3D specular = spec * specularStrength *
       QVector3D(1.0f, 1.0f, 1.0f);
    QVector3D result = (ambient + diffuse + specular) *
       lightSource->getIntensity();
    QColor color;
    color.setRedF(std::min(result.x(), 1.0f));
    color.setGreenF(std::min(result.y(), 1.0f));
    color.setBlueF(std::min(result.z(), 1.0f));

    return color;
}
```

3.4 Пользовательский интерфейс

На рисунке 3.2 представлен интерфейс программы, включающий в себя элементы для взаимодействия пользователя с симуляцией. С их помощью пользователь может управлять параметрами сцены, редактировать параметры активных объектов и менять используемые методы рендеринга и закраски.

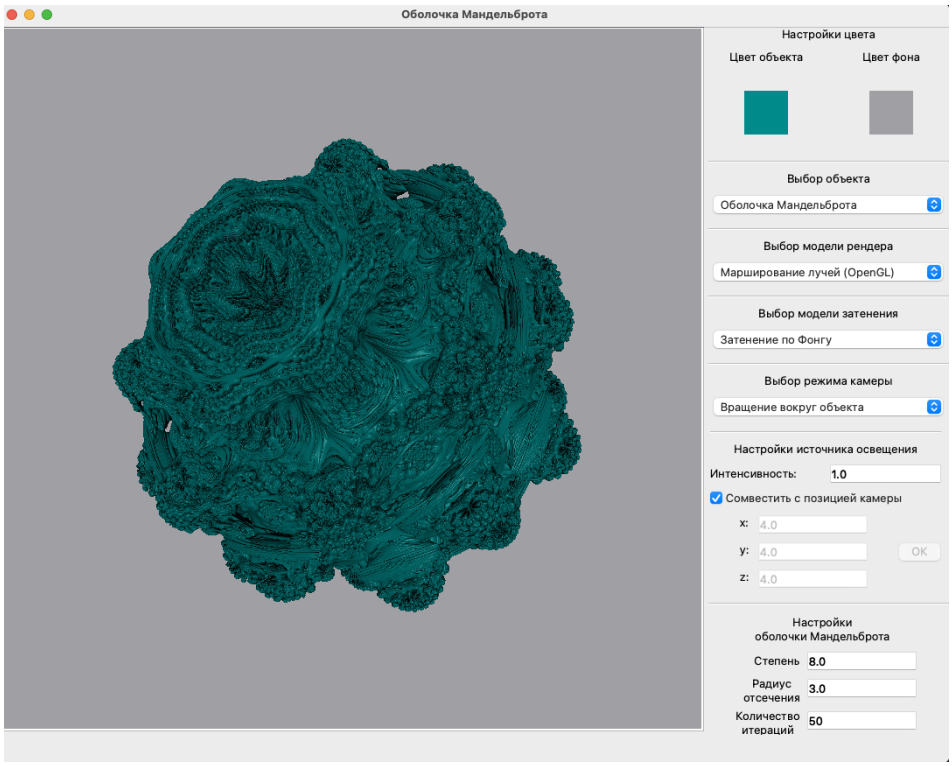

Рисунок 3.2 – Пользовательский интерфейс

3.4.1 Взаимодействие с элементами интерфейса

Функции базовых элементов интерфейса.

— кнопки «цвет объекта» и «цвет фона» — при нажатии открывают диалоговое окно с выбором нового цвета;

— меню «выбор объекта» — позволяет выбрать моделируемый объект из выпадающего списка;

— меню «выбор модели рендера» — позволяет выбрать алгоритм рендеринга из выпадающего списка;

— меню «выбор модели закраски» — позволяет выбрать алгоритм закраски из выпадающего списка;

— «интенсивность» — отвечает за интенсивность источника света, число с плавающей точкой от 0 до 1;

— кнопка «совместить с позицией камеры» — позволяет совместить источник света с камерой;

— поля «x», «y», «z»> — отвечают за положение источника света в пространстве сцены.

При включенном режиме совмещения источника света с позицией камеры, поля, отвечающие за ручное изменение положения источника являются неактивными и их значения недоступны к редактированию. Также, в зависимости от выбранного объекта, пользователю предоставляется возможность изменять его индивидуальные характеристики.

Для фрактала «оболочка Мандельброта»:

— «степень» — степень используемая в итерации фрактала, число с плавающей точкой от 3 до 30, влияет на форму фрактала;

— «радиус отсечения» — радиус отсечения, используемый при построении фрактала, число с плавающей точкой от 1 до 5, влияет на количество «артефактов»;

— «количество итераций» — количество итераций, используемых в функции SDF для уточнения границ фрактала, число от 1 до 1000, влияет на детализацию фрактала.

Для сферы:

— «радиус» — радиус сферы.

3.4.2 Управление камерой

В программе реализовано два вида управления камерой: свободное перемещение камеры по пространству сцены и режим вращения камеры

вокруг объекта. Переключение между режимами осуществляется с помощью меню «режим камеры».

В режиме свободного перемещения пользователю доступны следующие операции:

— перетаскивание мышью — позволяет поворачивать камеру вокруг своей оси;

— клавиши W, A, S, D — позволяют перемещать камеру вперёд, влево, назад или вправо относительно текущего направления взгляда.

В режиме вращения вокруг объекта:

— перетаскивание мышью — позволяет вращать камеру вокруг центра сцены, меняя ракурс;

— колесико мыши — позволяет приближать и отдалять камеру;

3.5 Демонстрация работы программы

На рисунке 3.3 представлены результаты работы программы, где отображен процесс моделирования фрактальной структуры в трехмерном пространстве. Для построения фрактала используется итерация $z \mapsto z^{12}$, рендеринг осуществляется с помощью алгоритма маршрирования лучей с использованием графического процессора, закраска осуществляется по модели Фонга.

Вывод

В данном разделе была описана технология разработки структуры программы, приведены примеры реализации используемых алгоритмов и работы реализованной программы.

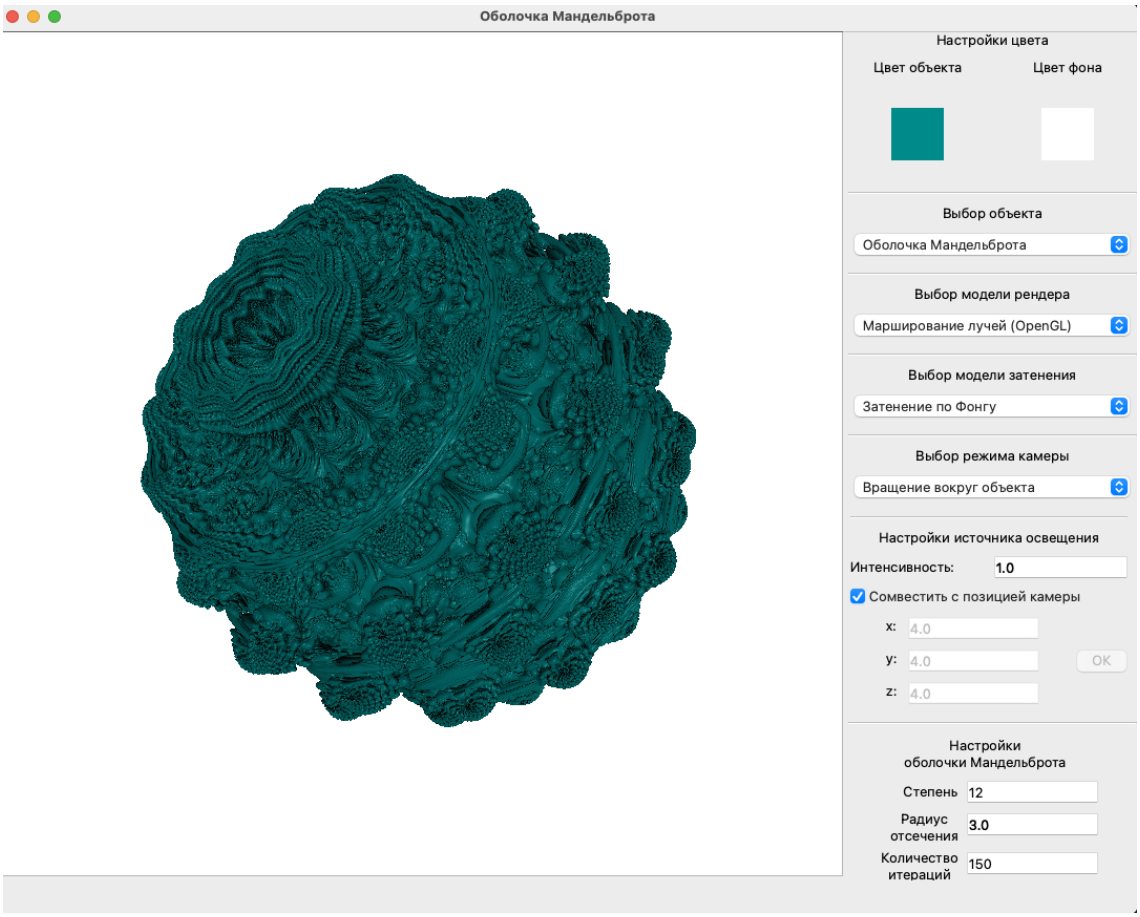

Рисунок 3.3 – Демонстрация работы программы

4 Исследовательская часть

В данном разделе представлены результаты исследования влияния параметров объектов сцены и разрешения окна сцены на производительность программы при визуализации фрактала «оболочка Мандельброта» разными методами рендеринга. Производительность оценивалась по времени, необходимому для рендера одного кадра.

4.1 Цель исследования

Цель исследования — определение зависимости производительности программы от разрешения экрана и количества итераций при построении фрактала для всех реализованных алгоритмов рендеринга.

4.2 Методика исследования

При проведении исследования для каждого из алгоритмов рендеринга создавалась сцена с фиксированным количеством итераций для построения фрактала и фиксированным разрешением экрана, после чего замерялось время, за которое программа создаст новый кадр.

Параметры исследования:

— iterations — количество итераций (изменяется от 1 до 15 с шагом 1);

— size — размер стороны экрана в пикселях (изменяется от 200 до 1000 с шагом 100);

— renderer — реализация алгоритма маршрирования лучей (однопоточная, многопоточная, с использованием OpenGL).

Для каждой исследуемой конфигурации сцены вида (iterations, size, renderer) проводилось по 50 замеров, после чего результаты измерений усреднялись. Все измерения проводились на одном устройстве.

4.3 Технические характеристики

Для проведения исследования использовалось устройство со следующими параметрами.

— операционная система macOS Sequoia [12];

— центральный процессор Apple M1 Pro, частота 3.2 ГГц, 10 ядер [13];

— графический процессор 16 ядер, пропускная способность 200 ГБ/с [13];

— оперативная память 16 ГБ LPDDR4X [13].

4.4 Результаты исследования

4.4.1 Влияние размеров сцены

Результаты исследования зависимости времени отрисовки кадра от разрешения экрана представлены в таблице 4.1 и на рисунках 4.1—4.2.

Таблица 4.1 – Зависимость времени отрисовки кадра от разрешения экрана

Разрешение	Однопоточный	Многопоточный	OpenGL
200	65895366	23999063	1641956
300	147749941	53905087	2084237
400	261793954	95804413	2651755
500	407500025	149992919	3569735
600	588554860	215305649	3995359
700	797428490	293810629	4269103
800	1039635932	383417703	5459058
900	1314811132	487414066	6760236
1000	1620256803	600612492	7875581

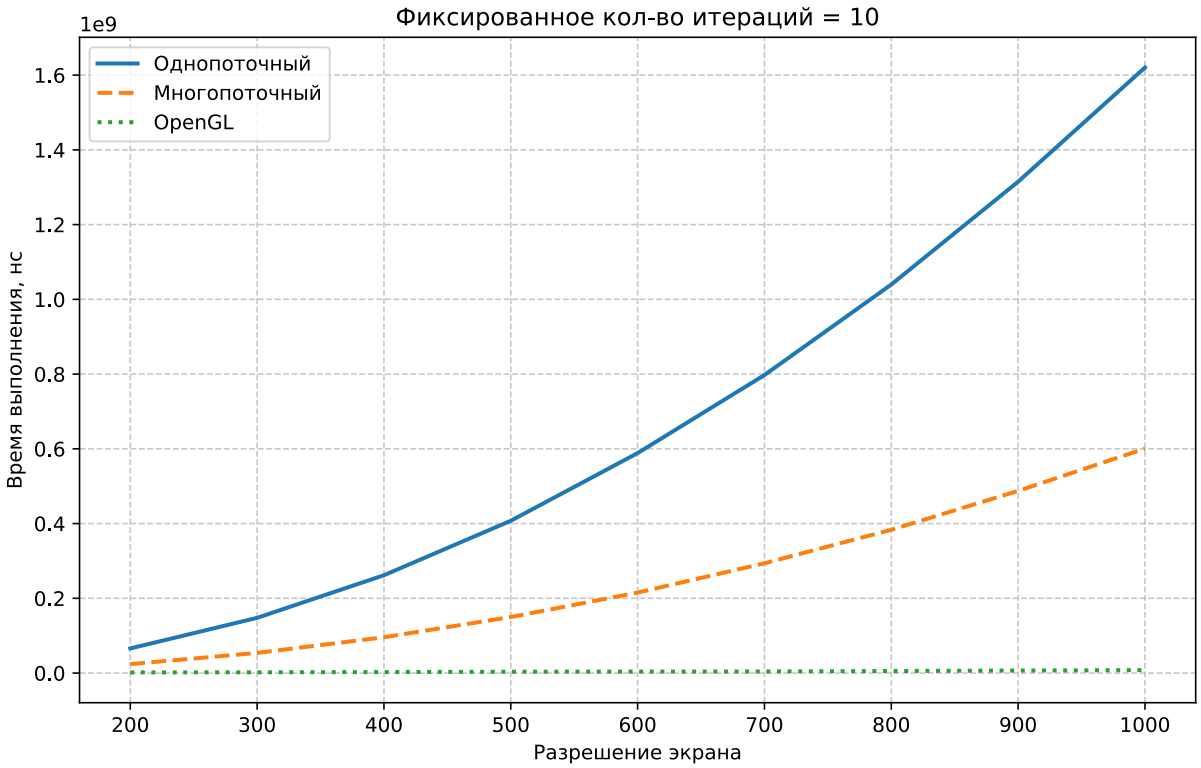

Рисунок 4.1 – Графики зависимости времени отрисовки кадра от разрешения экрана

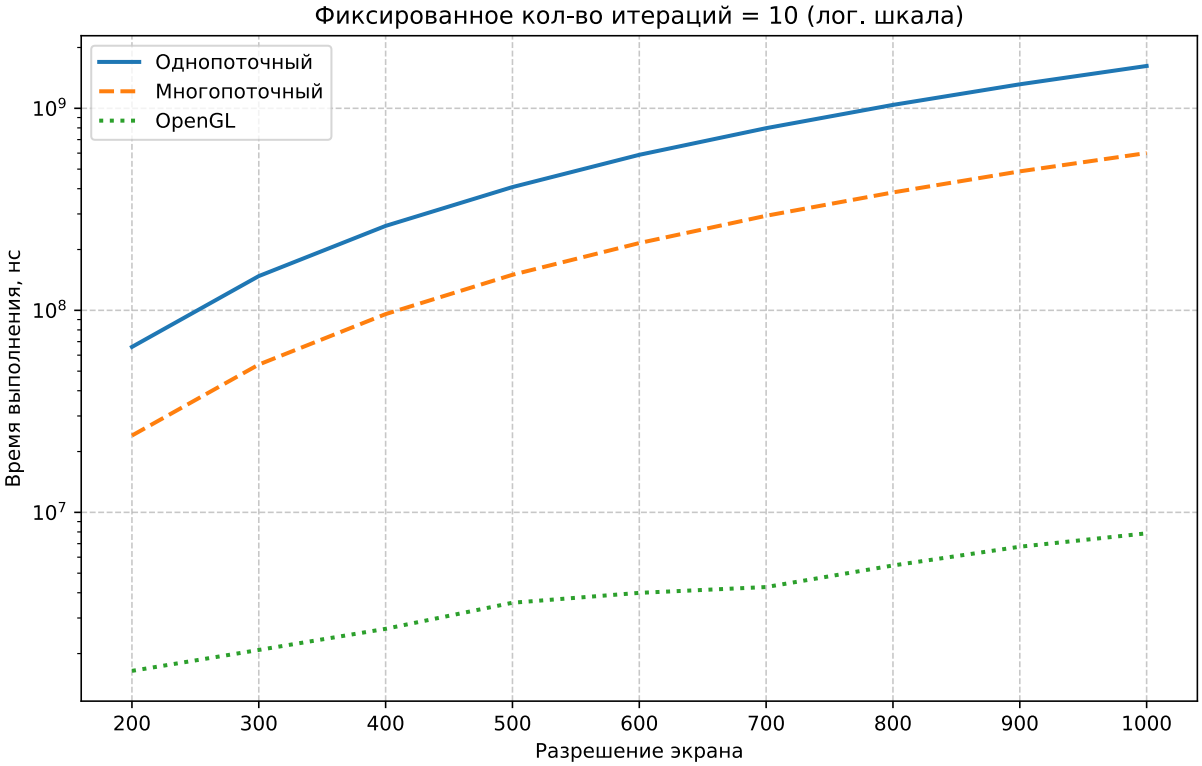

Рисунок 4.2 – Графики зависимости времени отрисовки кадра от разрешения экрана (логарифмическая шкала)

4.4.2 Влияние количества итераций фрактала

Результаты исследования зависимости времени отрисовки кадра от количества итераций фрактала представлены в таблице 4.2 и на рисунках 4.3—4.4.

Таблица 4.2 – Зависимость времени отрисовки кадра от количества итераций

Кол-во итераций	Однопоточный	Многопоточный	OpenGL
1	47593287	153015446	2887503
2	109548455	168974550	3554830
3	175185833	185803137	3778188
4	228460870	195374127	4016154
5	290343735	205713838	3971579
6	350660671	212353142	4293275
7	410860620	214479487	4147245
8	470393082	215181351	4007472
9	529079199	215591820	4205380
10	589931072	214973965	4030680
11	649500219	215758280	4273324
12	710061214	215801329	4374386
13	769308330	216255990	4215994
14	827978309	217695584	4459785
15	886998022	218102497	4228779

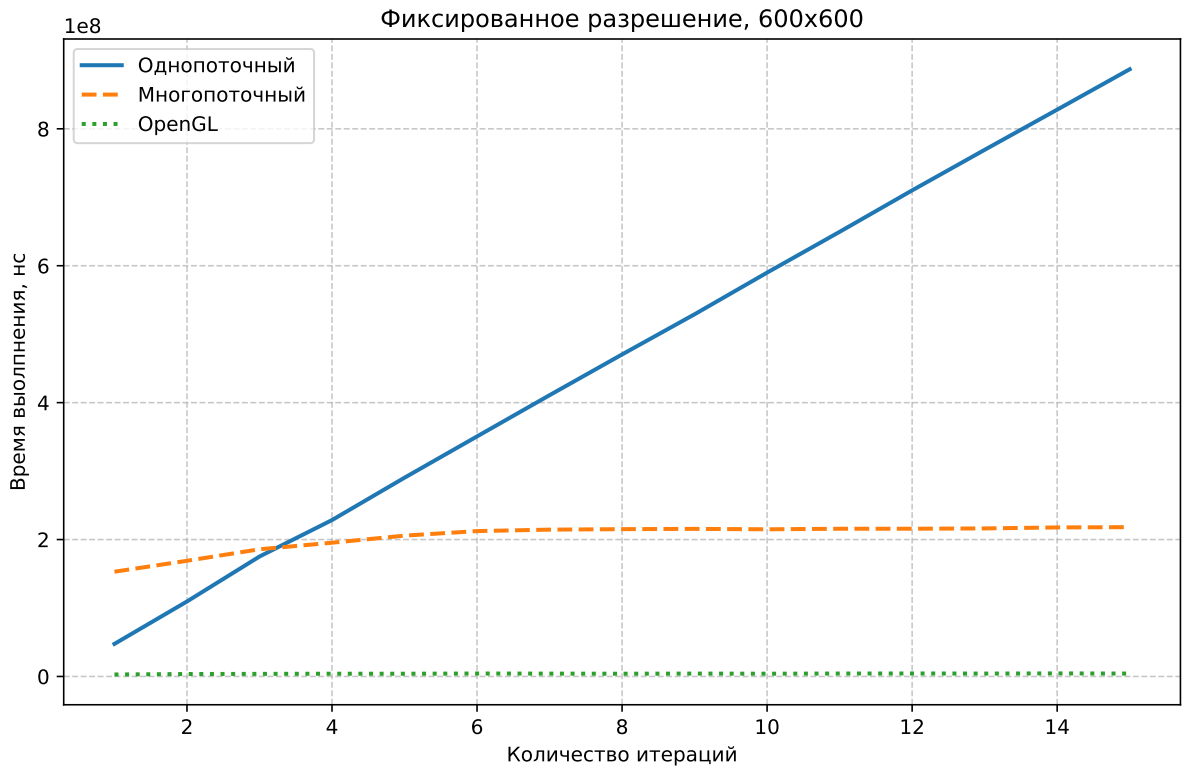

Рисунок 4.3 – Графики зависимости времени отрисовки кадра от количества итераций фрактала

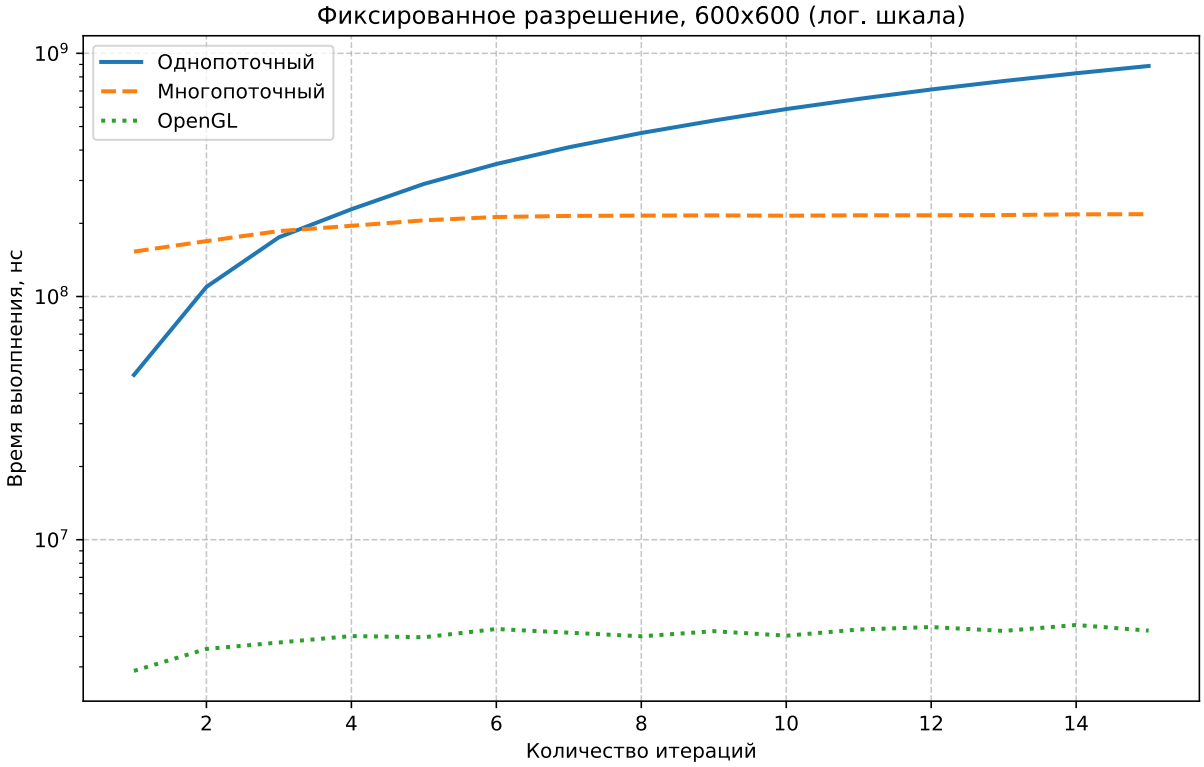

Рисунок 4.4 – Графики зависимости времени отрисовки кадра от количества итераций фрактала (логарифмическая шкала)

4.5 Вывод

На основании проведённых исследований можно сделать следующие выводы.

1) разрешение экрана оказывает наиболее существенное влияние на производительность программы;

2) увеличение количества итераций изображаемого фрактала оказывает существенное влияние на производительность однопоточного алгоритма, однако для остальных алгоритмов его влияние не столь существенно;

3) при любой конфигурации сцены версия алгоритма, выполняющая вычисления с помощью графического процессора, рендерит изображение сцены в десятки раз быстрее, чем однопоточная и многопоточная версии;

4) многопоточная версия программы работает в среднем в 5-9 раз быстрее однопоточной;

Таким образом, для оптимизации производительности следует использовать версию алгоритма маршрирования лучей, использующую графический процессор для вычислений. При необходимости дальнейшей оптимизации следует уменьшать разрешение экрана. Изменение количества итераций даст ощутимый прирост в производительности только для однопоточного алгоритма, в остальных реализациях его влияние незначительно.

ЗАКЛЮЧЕНИЕ

Цель данной курсовой работы была достигнута. Было разработано программное обеспечение для моделирования изображения трёхмерного фрактала «оболочка Мандельброта».

Для достижения цели были выполнены все поставленные задачи:

— проанализированы и выбраны методы и алгоритмы для визуализации трехмерных фракталов;

— реализованы алгоритмы рендера сцены, закраски объектов и нахождения расстояния до них;

— разработан пользовательский интерфейс для управления параметрами сцены, изображаемого объекта и положением камеры;

— проведено исследование производительности программы при разных параметрах сцены. Эффективность работы программы зависит от сложности изображаемого объекта и разрешения экрана.

Разработанная программа успешно решает задачу моделирования фрактала «оболочка Мандельброта», предоставляя пользователю возможность детальной настройки сцены с помощью взаимодействия с пользовательским интерфейсом.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Mandelbrot B. B. The Fractal Geometry of Nature. — San Francisco, 1982. — P. 462.

2. Морозов А. Д. Введение в теорию фракталов. — М. Ижевск: Институт компьютерных исследований, 2006. — С. 162.

3. Boily V., D. R. On the algebraic foundation of the Mandelbulb // Fractals. 2023. no. 31.

4. Процесс получения изображения по трехмерной модели в реальном времени / Д.А. Зенкевич, Д.В. Гончаров, Свиридова И.В. [и др.] // Экономика и социум. 2020. № 5.

5. Akenine-Moller T., Haines E., Hoffman N. Real-Time Rendering. 3 edition. — New York: CRC Press, 2008. — P. 1045.

6. Круглов М. А., Холкин А. В. Метод построения трехмерной графики на основе полей расстояний // Инженерный вестник Дона. 2024. № 7.

7. Д. Роджерс. Алгоритмические основы машинной графики. — Москва: Издательство Мир, 1989. — С. 512.

8. Phong B. T. Illumination of Computer-Generated Images // Communications of the ACM. 1975. June. Vol. 18. — P. 313 – 317.

9. Документация по C++ [Электронный ресурс]. — Режим доступа: `https://learn.microsoft.com/ru-ru/cpp/` (дата обращения: 22.11.2024).

10. Документация по Qt [Электронный ресурс]. — Режим доступа: `https://doc.qt.io/` (дата обращения: 22.11.2024).

11. Документация по OpenGL [Электронный ресурс]. — Режим доступа: `https://registry.khronos.org/OpenGL-Refpages/gl4/` (дата обращения: 01.12.2024).

12. MacOS Sequoia Developer Documentation [Электронный ресурс]. — Режим доступа: https://developer.apple.com/documentation/ (дата обращения: 25.11.2024).

13. MacBook Pro (14 дюймов, 2021 г.) - Спецификации [Электронный ресурс]. — Режим доступа: https://support.apple.com/ru-ru/111902 (дата обращения: 24.11.2024).

ПРИЛОЖЕНИЕ А

Презентация к курсовой работе состоит из 15 слайдов.