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:

«Разработка программного обеспечения для визуализации трехмерных фрактальных поверхностей»

Студент ИУ7-53Б _____ Г. А. Миронов
(Группа) (Подпись, дата) (И. О. Фамилия)

Руководитель курсовой работы _____ А. А. Оленев
(Подпись, дата) (И. О. Фамилия)

2021 г.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ

В настоящей расчетно-пояснительной записке применяют следующие термины с соответствующими определениями.

Axis Aligned Bounding Box — параллелипипед, выравненный по координатным осям

Поле расстояний со знаком — это функция, получающая на входе точку и возвращающая кратчайшее расстояние от этой точки до поверхности каждого объекта в сцене. Кроме того, поле расстояний со знаком дополнительно возвращает отрицательное число, если точка находится внутри объекта.

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В настоящей расчетно-пояснительной записке применяют следующие сокращения и обозначения.

ПО — Программное обеспечение

AABB — Axis Aligned Bounding Box

SDF — Поле расстояний со знаком

Введение

Физические тела, окружающие нас, обладают различными оптическими свойствами. Они, к примеру, могут отражать или пропускать световые лучи, также они могут отбрасывать тень. Эти и другие свойства нужно уметь наглядно показывать при помощи электронно-вычислительных машин. Этим и занимается компьютерная графика.

Компьютерная графика – представляет собой совокупность методов и способов преобразования информации в графическое представление при помощи ЭВМ. Без компьютерной графики не обходится ни одна современная программа. В течении нескольких десятилетий компьютерная графика прошла долгий путь, начиная с базовых алгоритмов, таких как вычерчивание линий и отрезков, до построения виртуальной реальности.

Целью данного курсового проекта является разработка ПО, визуализирующего трехмерные фрактальные поверхности.

Для достижения данной цели необходимо решить следующие задачи:

- описать структуру синтезируемой трехмерной сцены;
- описать существующие алгоритмы построения реалистичных изображений;
- выбрать и обосновать выбор реализуемых алгоритмов;
- привести схемы реализуемых алгоритмов;
- определить требования к программному обеспечению;
- описать использующиеся структуры данных;
- описать структуру разрабатываемого ПО;
- определить средства программной реализации;
- реализовать соответствующее ПО;
- провести экспериментальные замеры временных характеристик разработанного ПО.

1 Аналитическая часть

В данном разделе рассматриваются существующие алгоритмы построения реалистичных изображений. Так же, обосновывается выбор реализуемого алгоритма и указывается список ограничений, в рамках которых будет работать разрабатываемое ПО.

1.1 Формализация объектов синтезируемой сцены

Сцена состоит из следующих объектов:

- источников света:
  - источник рассеянного света - задается интенсивностью. Используется для освещения всей сцены;
  - источник направленного света - задается вектором направленности и интенсивностью. Предполагается, что такой источник расположен в бесконечности;
  - точечный источник света - задается положением в пространстве и интенсивностью;
- фрактальной поверхности.

Существует несколько способов представления фрактальной поверхности. На пример, можно представлять поверхность аналитически или же преобразовать ее в полигональную поверхность и представить в виде списка граней и т.д.. Представление поверхности в виде аналитического выражения является более предпочтительным в данной работе, так как позволяет синтезировать сцену сколь угодно высокой точности, в отличие от остальных способов представления поверхностей.

1.2 Описание алгоритмов удаления невидимых линий и поверхностей

1.2.1 Некоторые теоретические сведения

Прежде чем описывать алгоритмы, нужно дать некоторые определения, которые будут использованы в данном разделе.

Алгоритмы удаления невидимых линий и поверхностей служат для определения линий ребер, поверхностей, которые видимы или невидимы для наблюдателя, находящегося в заданной точке пространства [1].

Решать задачу можно в:

- объектном пространстве - мировая системе координат, высокая точность. Обобщенный подход, основанный на анализе пространства объектов, предполагает попарное сравнение положения всех объектов по отношению к наблюдателю.

- пространстве изображений - в экранных координатах, системе координат, связанной с тем устройством в котором отображается результат. (Графический дисплей).

Под экранированием подразумевается загораживание одного объекта другим.

Под глубиной подразумевается значение координаты Z, направленной от зрителя, за плоскость экрана.

1.2.2 Алгоритм Робертса

Алгоритм Робертса решает задачу удаления невидимых линий. Работает в объектном пространстве. Данный алгоритм работает исключительно с выпуклыми телами. Если тело изначально является не выпуклым, то нужно его разбить на выпуклые составляющие. Алгоритм целиком основан на математических предпосылках [1].

Из-за сложности математических вычислений, используемых в данном алгоритме, а так же из-за дополнительных затраты ресурсов на вычисление матриц данный алгоритм является довольно медленным.

1.2.3 Алгоритм Варнака

Алгоритм Варнака [1—3] позволяет определить, какие грани или части граней объектов сцены видимы, а какие заслонены гранями других объектов. Так же как и в алгоритме Робертса анализ видимости происходит в пространстве изображения. В качестве граней обычно выступают выпуклые многоугольники, алгоритмы работы с ними эффективнее, чем с произвольными многоугольниками. Окно, в котором необходимо отобразить сцену, должно

быть прямоугольным. Алгоритм работает рекурсивно, на каждом шаге анализируется видимость граней и, если нельзя легко определить видимость, окно делится на 4 части и анализ повторяется отдельно для каждой из частей (см. рис. 1.1).

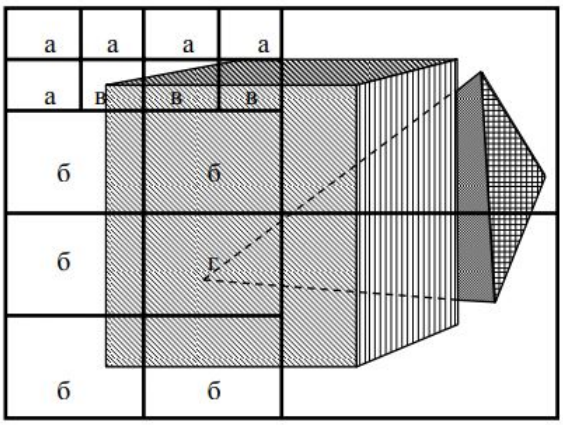

Рисунок 1.1 – Пример разбиения Алгоритмом Варнока

Так как данный алгоритм основывается на рекурсивном разбиении экрана, в зависимости от расположения объектов это может вызвать, как положительной, так и отрицательный эффект. Чем меньше пересечений объектов, тем быстрее алгоритм завершит свою работу.

1.2.4 Алгоритм Z–буфера

Алгоритм Z–буфера [1; 2] позволяет определить, какие пикселы граней сцены видимы, а какие заслонены гранями других объектов. Z–буфер -– это двухмерный массив, его размеры равны размерам окна, таким образом, каждому пикселу окна, соответствует ячейка Z-буфера. В этой ячейке хранится значение глубины пиксела (см. рис. 1.2). Перед растеризацией сцены Z–буфер заполняется значением, соответствующим максимальной глубине. В случае, когда глубина характеризуется значением w, максимальной глубине соответствует нулевое значение. Анализ видимости происходит при растеризации граней, для каждого пиксела рассчитывается глубина и сравнивается со значением в Z–буфере, если рисуемый пиксел ближе (его `w` больше значения в Z–буфере), то пиксел рисуется, а значение в Z–буфере заменяется его глубиной. Если пиксел дальше, то пиксел не рисуется и Z–буфер не изменяется, текущий пиксел дальше того, что нарисован ранее, а значит невидим.

К недостаткам алгоритма следует отнести довольно большие объемы

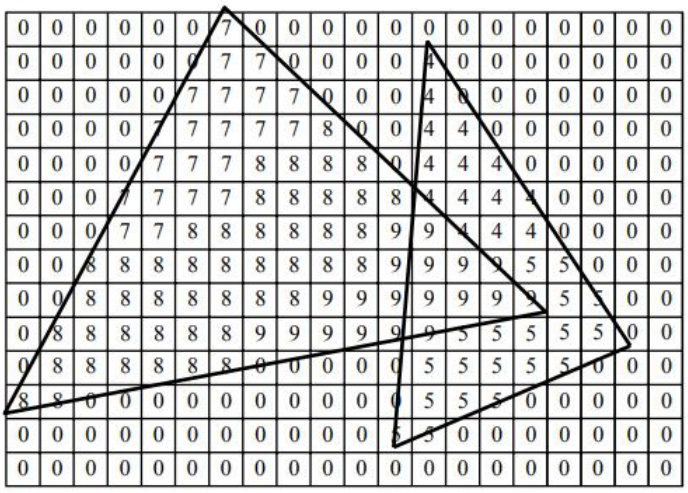

Рисунок 1.2 – Пример работы алгоритма Z–буфера

требуемой памяти, а также имеются другие недостатки, которые состоят в трудоемкости устранения лестничного эффекта и трудности реализации эффектов прозрачности.

1.2.5 Алгоритм прямой трассировки лучей

Основная идея алгоритма прямой трассировки лучей [2] состоит в том, что наблюдатель видит объекты благодаря световым лучам, испускаемым некоторым источником, которые падают на объект, отражаются, преломляются или проходят сквозь него и в результате достигают зрителя.

Основным недостатком алгоритма является излишне большое число рассматриваемых лучей, приводящее к существенным затратам вычислительных мощностей, так как лишь малая часть лучей достигает точки наблюдения. Данный алгоритм подходит для генерации статических сцен и моделирования зеркального отражения, а так же других оптических эффектов [4].

1.2.6 Алгоритм обратной трассировки лучей

Алгоритм обратной трассировки лучей отслеживает лучи в обратном направлении (от наблюдателя к объекту) [2]. Такой подход призван повысить эффективность алгоритма в сравнении с алгоритмом прямой трассировки лучей. Обратная трассировка позволяет работать с несколькими источниками света, передавать множество разных оптических явлений [5].

Пример работы данного алгоритма приведен на Рисунке 1.3.

Считается, что наблюдатель расположен на положительной полуоси z в

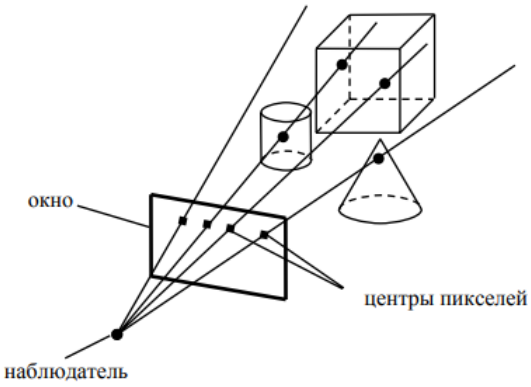

Рисунок 1.3 – Пример работы алгоритма обратной трассировки лучей

бесконечности, поэтому все световые лучи параллельны оси z. В ходе работы испускаются лучи от наблюдателя и ищутся пересечения луча и всех объектов сцены [3]. В результате пересечение с максимальным значением z является видимой частью поверхности и атрибуты данного объекта используются для определения характеристик пикселя, через центр которого проходит данный световой луч.

Для расчета эффектов освещения сцены проводятся вторичные лучи от точек пересечения ко всем источникам света. Если на пути этих лучей встречается непрозрачное тело, значит, данная точка находится в тени.

Несмотря на более высокую эффективность алгоритма в сравнении с прямой трассировкой лучей, данный алгоритм считается достаточно медленным, так как в нем происходит точный расчет сложных аналитических выражений для нахождения пересечения с рассматриваемыми объектами.

1.2.7 Алгоритм Ray Marching

У алгоритмов прямой и обратной трассировки имеется ряд преимуществ, а именно:

- возможность обрабатывать множество различных объектов, имеющих сложные формы;
- возможность строить реалистичные тени и отраения объектов.

Тем не менее, данные алгоритмы требуют задания уравнений поверхности для каждого из рассматриваемых объектов .

В случаях, когда некоторая потеря точности вычислений является некритичной, можно воспользоваться алгоритмом `Ray Marching`'га.

Основная идея данного алгоритма состоит в том, что, вместо аналитического вычисления точки пересечения луча и рассматриваемого объекта, мы «шагаем» точкой вдоль луча, пока не найдём точку пересечения с объектом.

На Рисунке 1.4 приведен пример работы данного алгоритма.

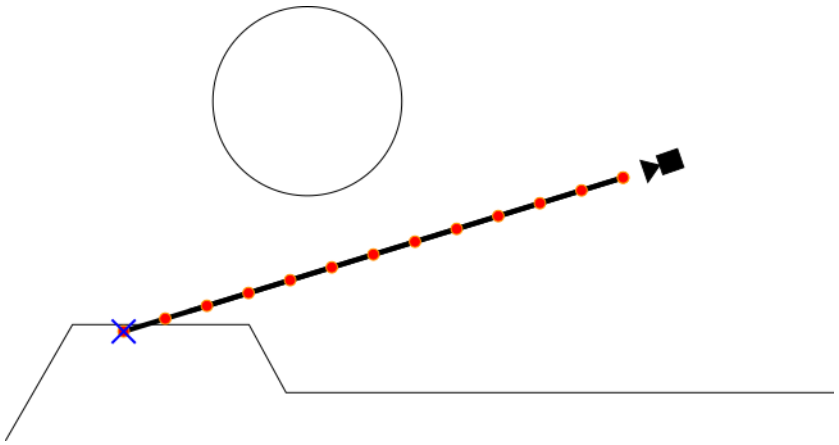

Рисунок 1.4 – Пример работы алгоритма `Ray Marching`

Функция `raymarching`-а с фиксированным интервалом, такая например, как показана на Рисунке 1.4, вполне достаточна для множества областей применения, например, объёмных и прозрачных поверхностей.

Тем не менее, для непрозрачных объектов можно ввести ещё одну оптимизацию. Для этой оптимизации требуется использование `SDF`.

На Рисунке 1.5 приведен пример работы данного оптимизации алгоритма.

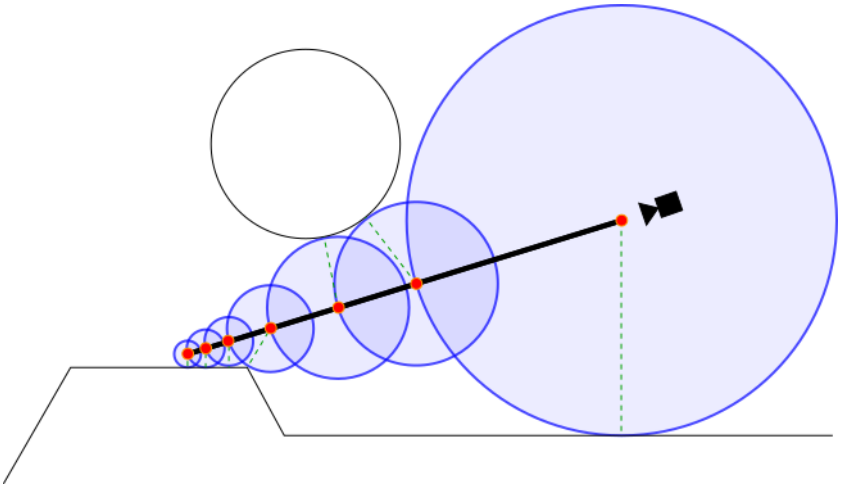

Рисунок 1.5 – Пример работы алгоритма `Ray Marching` с `SDF`

Из Рисунка 1.5 видно, что такая оптимизация позволяет уменьшить количество «шагов», позволяя динамически изменять размер «шага».

1.3 Описание алгоритмов построения теней

Для создания реалистичного изображения в компьютерной графике применяются различные алгоритмы освещения.

Модель освещения предназначена для расчета интенсивности отраженного к наблюдателю света в каждой точке изображения.

Модель освещения может быть:

- локальной - в данной модели учитывается только свет от источников и ориентация поверхности.
- глобальной - в данной модели, помимо составляющих локальной, учитывается еще и свет, отраженный от других поверхностей или пропущенный через них.

Локальная модель включает 3 составляющих:

1. Диффузную составляющую отражения.
2. Отражающую составляющую отражения.
3. Рассеянное освещение.

Выбор алгоритма построения теней напрямую зависит от выбора алгоритма отсечения невидимых ребер и поверхностей, а так же - от выбора модели освещения.

Так в алгоритме трассировки лучей тени получаются практически без дополнительных вычислений, а в алгоритме с Z буфером, к примеру, можно получить тени, используя второй буфер, полученный подменой точки наблюдения на точку источника света.

1.4 Вывод

Оценив все изложенные выше алгоритмы, можно сделать вывод, что для данной работы, предполагающей визуализацию реалистического изображения, учитывая тени, различные источники освещения и т.д., подходит алгоритм `Ray Marching`, так как он позволяет достичь высокой реалистичности, а также точности построенного изображения.

Данный алгоритм будет использоваться, несмотря на указанные недостатки, так как он достаточно полно отражает суть физических явлений с приемлемой потерей точности вычисления.

2 Конструкторская часть

В данном разделе будут рассмотрены требования к программному обеспечению, а также схемы алгоритмов, выбранных для решения поставленной задачи. Так же, будут описаны пользовательские структуры данных и приведена структура реализуемого программного обеспечения.

2.1 Требования к программному обеспечению

ПО должно предоставлять доступ к следующему функционалу:

- выбор фрактальной поверхности из предложенного списка;
- задание интервала построения для выбранной фрактальной поверхности по ося `X`, `Y`, `Z`;
- задание цвета и свойств поверхности;
- поворот и перемещение камеры (точки наблюдения);
- изменение положения, интенсивности и ориентации источника света.

К ПО предъявляются следующие требования:

- время отклика программы не должно превышать 30 секунд для корректной работы в интерактивном режиме;
- программа должна корректно реагировать на люые действия пользователя.

2.2 Разработка алгоритмов

В данном проекте алгоритм `Ray Marching` будет применяться к фрактальным поверхностям.

Эффективность процедуры определения пересечений луча с фрактальной поверхностью оказывает самое большое влияние на эффективность всего алгоритма. Чтобы избавиться от излишнего поиска пересечений, рассматривают пересечение луча с объемной оболочкой рассматриваемого объекта. В данном случае под оболочкой понимается некоторый простой объект, внутрь

которого можно поместить рассматриваемый объект, к примеру параллелепипед или сфера.

На рис. 2.1 приведены примеры сферической и прямоугольной оболочки.

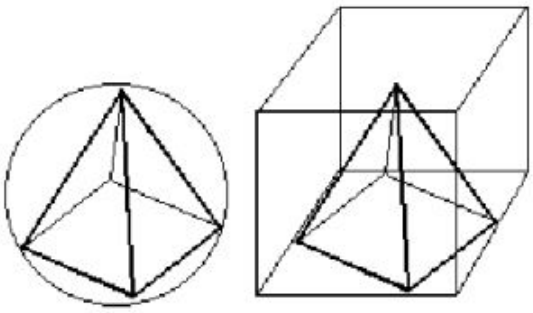

Рисунок 2.1 – Сферическая и прямоугольная оболочки

Если такого пересечения нет, то и, соответственно, пересечения луча и самого рассматриваемого объекта нет, и наоборот, если найдено пересечение, то, возможно, существует пересечение луча и рассматриваемого объекта.

Для повышения эффективности процедуры определения пересечений луча с прямоугольной оболочкой, данную оболочку часто заменяют на `AABB`, что позволяет упростить вычисления при незначительной потере точности.

2.2.1 Пересечение луча и сферы

Уравнение луча представлено ниже:

$$P = O + t\overrightarrow{D}, t \geq 0, \tag{2.1}$$

где $\overrightarrow{D}$ – направление луча.

Сфера — это множество точек P, лежащих на постоянном расстоянии r от фиксированной точки C. Тогда можно записать уравнение, удовлетворяющее этому условию:

$$distance(P, C) = r \tag{2.2}$$

Запишем расстояние (2.2) между P и C как длину вектора из P в C.

$$|P - C| = r \tag{2.3}$$

Заменим на скалярное произведение вектора на себя:

$$\sqrt{\langle P - C\rangle, \langle P - C\rangle} = r \tag{2.4}$$

Избавимся от корня:

$$\langle P - C\rangle, \langle P - C\rangle = r^2 \tag{2.5}$$

В итоге есть два уравнения - уравнение луча и сферы. Найдем пересечение луча со сферой. Для этого подставим (2.1) в (2.5)

$$\langle O + t\overrightarrow{D} - C\rangle, \langle O + t\overrightarrow{D} - C\rangle = r^2 \tag{2.6}$$

Разложим скалярное произведение и преобразуем его. В результате получим:

$$t^2\langle\overrightarrow{D}, \overrightarrow{D}\rangle + 2t\langle\overrightarrow{OC}, \overrightarrow{D}\rangle + \langle\overrightarrow{OC}, \overrightarrow{OC}\rangle - r^2 = 0 \tag{2.7}$$

Представленное квадратное уравнение (2.7) имеет несколько возможных случаев решения. Если у уравнения одно решение, это обозначает, что луч касается сферы. Два решения обозначают, то что луч входит в сферу и выходит из неё. И если нет решений, значит, луч не пересекается со сферой.

2.2.2 Пересечение луча и AABB

Пример `AABB` приведен на Рисунке 2.2.

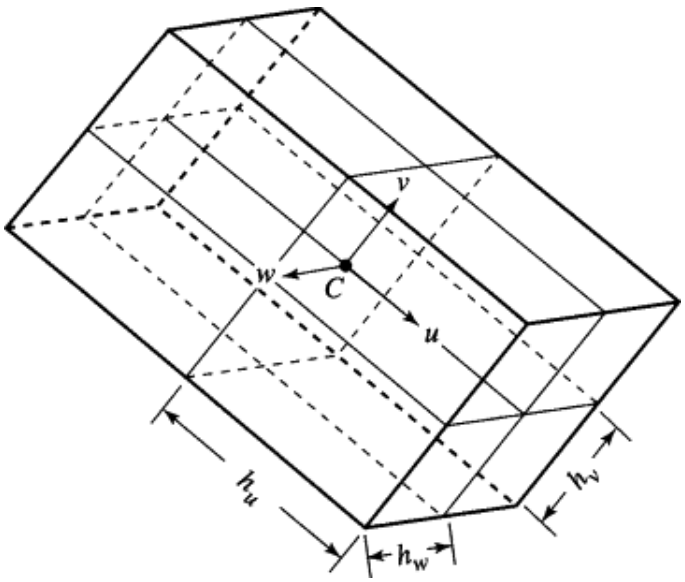

Рисунок 2.2 – Пример `AABB`

Уравнение (2.8) эквивалентно уравнению (2.1)

$$\begin{cases} x(t) = x_O + tx_D \\ y(t) = y_O + ty_D \\ z(t) = z_O + tz_D \end{cases} \tag{2.8}$$

Из определения AABB следует, что каждая из граней такой оболочки задается некоторым константным выражением, в связи с чем из уравнения (2.8) можно выразить значение параметра t, соответствующее точке пересечения луча и соответствующей плоскости:

$$t = \frac{P_x - x_O}{x_D} \tag{2.9}$$

$$t = \frac{P_y - y_O}{y_D} \tag{2.10}$$

$$t = \frac{P_z - z_O}{z_D} \tag{2.11}$$

Теперь найдем значения параметра t для каждой из пар параллельных плоскостей, содержащих грани рассматриваемой AABB, воспользовавшись уравнениями (2.9) – (2.11). Конкретное уравнение выбирается в зависимости от положения рассматриваемой пары плоскостей в пространстве.

Обозначим вычисленные значения как T_{near} и T_{far}, определяющие значение параметра t в точке пересечения с ближайшей и дальней плоскостями из рассматриваемой пары соответственно.

В результате, может быть сделан следующий вывод:

$$\begin{cases} \text{AABB расположен вне зоны видимости,} & T_{far} < 0 \\ \text{луч не пересекает AABB,} & T_{far} < T_{min} \\ \text{обнаружено пересечение с AABB,} & \text{иначе} \end{cases} \tag{2.12}$$

2.2.3 Описание алгоритма Ray Marching-a

Суть алгоритма состоит в следующем: Из некоторой точки пространства, называемой виртуальным глазом, или камерой, через каждый пиксель изображения испускается луч и находится точка пересечения с ограничивающей

оболочкой фрактальной поверхности (2.7 и 2.12). При обнаружении точки пересечения, вычисляется точка пересечения луча с фрактальной поверхностью. Далее из найденной точки пересечения испускаются лучи до каждого источника освещения. Если данные лучи пересекают другие объекты сцены, значит точка пересечения находится в тени относительно рассматриваемого источника освещения и освещать ее не нужно. Освещение со всех видимых источников света складываются (по интенсивности).

На Рисунке 2.3 представлена схема синтеза изображения с применением данного алгоритма.

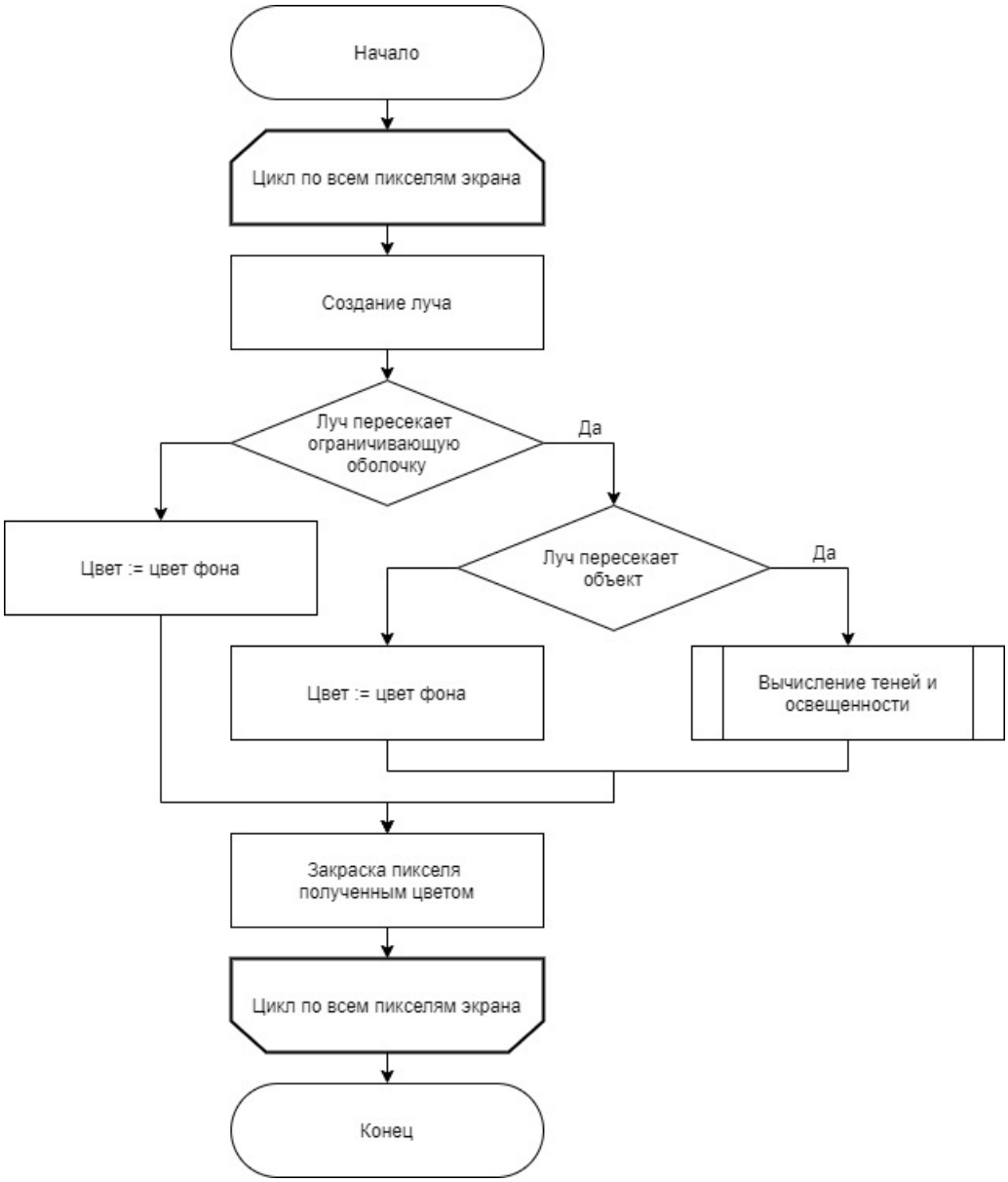

Рисунок 2.3 – Схема алгоритма синтеза изображения с применением алгоритма `ray marching`-а

2.3 Описание используемых типов и структур данных

В данной работе используются следующие типы и структуры данных:

- источник света – задается расположением, направленностью и интенсивностью света;
- математические абстракции:
  - точка — хранит координаты x, y, z;
  - вектор -– хранит направление по x, y, z;
- цвет – хранит три составляющие `RGB` модели цвета.

2.4 Описание структуры программного обеспечения

На Рисунке 2.4 представлена диаграмма классов реализуемого программного обеспечения.

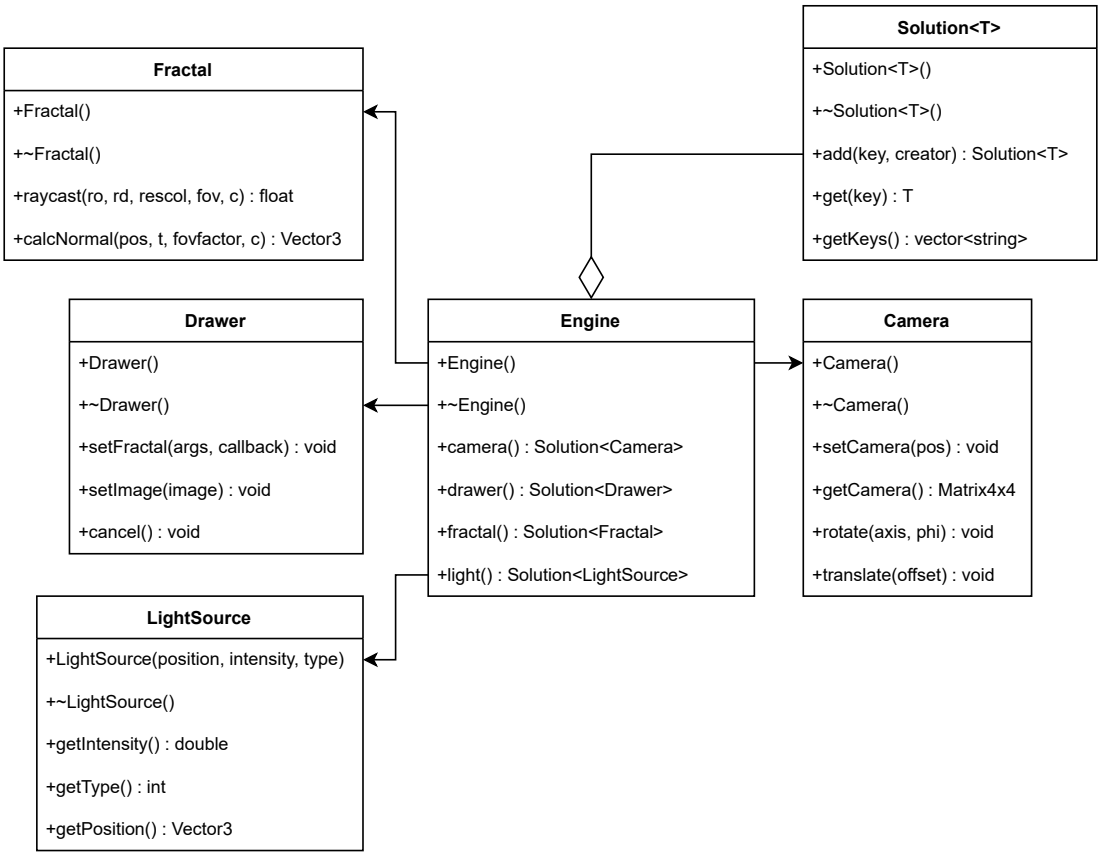

Рисунок 2.4 – Схема алгоритма `ray marching`-а

2.5 Описание оптимизаций временных характеристик

Из схемы алгоритма, представленной на Рисунке 2.3 следует, что каждый пиксель экрана обрабатывается независимо от остальных, в связи с чем данный алгоритм можно оптимизировать, распараллелив вычисления.

В связи с аппаратными ограничениями [6], обработка каждого отдельного пикселя в отдельном потоке приведет лишь к снижению производительности разрабатываемого ПО. В связи с этим, следует разбить рассматриваемые пиксели на группы по "строкам"или "столбцам"и обрабатывать каждую группу отдельным потоком.

2.6 Вывод

В данном разделе были представлены требования к разрабатываемому программному обеспечению и разработана схема разрабатываемого алгоритма. Так же, были описаны пользовательские структуры данных и приведена структура реализуемого программного обеспечения.

3 Технологическая часть

В данном разделе будут представлены средства разработки программного обеспечения, детали реализации и процесс сборки разрабатываемого программного обеспечения.

3.1 Выбор средств реализации

В качестве языка программирования для разработки программного обеспечения был выбран язык `C++` [7]. Данный выбор обусловлен тем, что данный язык предоставляет весь функционал требуемый для решения поставленной задачи.

Для создания пользовательского интерфейса ПО был использован фреймворк `QT` [8]. Данный фреймворк содержит в себе объекты, позволяющие напрямую работать с пикселями изображения, а так же возможности создания интерактивных пользовательских интерфейсов, что позволит в интерактивном режиме управлять изображением.

В процессе работы был использован инструмент `Clang` [9], позволяющий форматировать исходные коды, а так же в процессе их написания обнаруживать наличие синтаксических ошибок. Так же, данный инструмент предоставляет возможность статического анализа кода [10].

Кроме того, использовался инструмент `Valgrind` [11], позволяющий отслеживать утечки памяти в ходе работы программного обеспечения.

Для сборки программного обеспечения использовались инструменты `make` [12] и `CMake` [13].

В качестве среды разработки был выбран текстовый редактор `Visual Studio Code` [14], поддерживающий возможность установки плагинов [15], в том числе для работы с `C++` и `CMake`.

3.2 Интерфейс ПО

Интерфейс реализуемого ПО представлен на Рисунках 3.1 – 3.3.

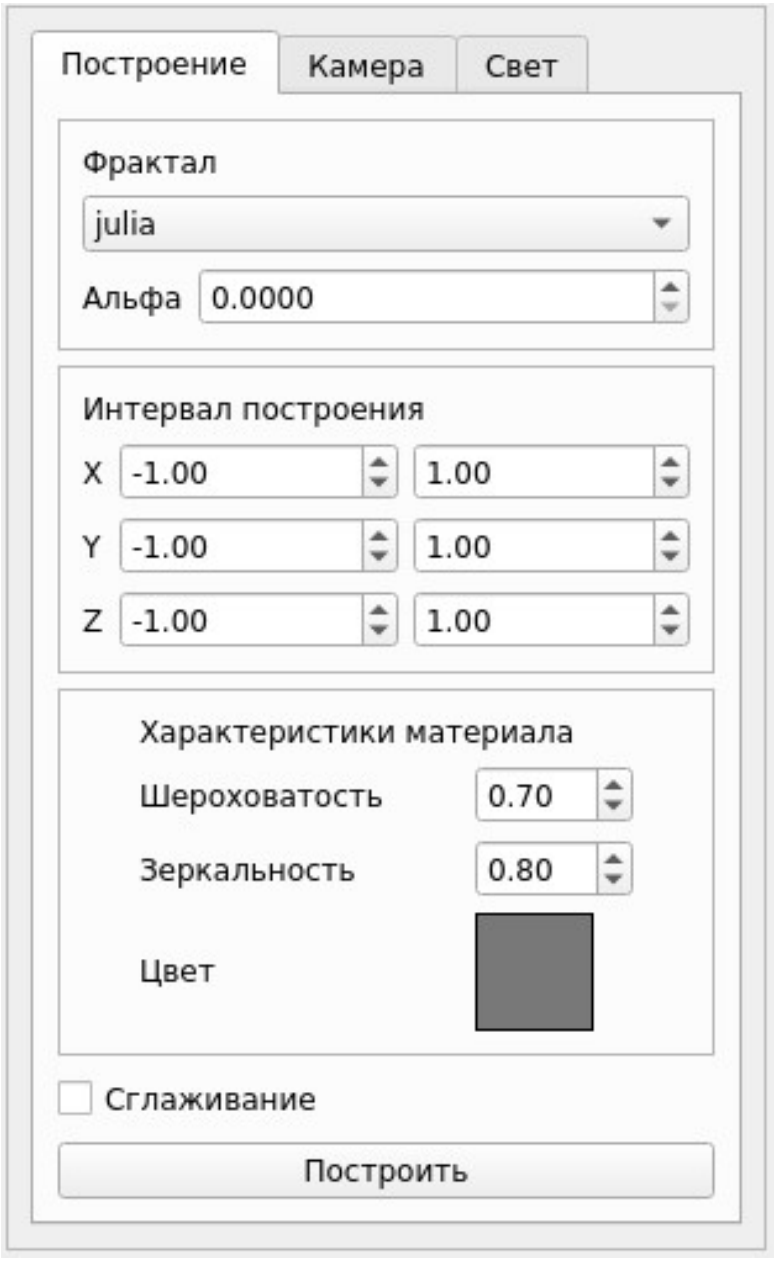

Рисунок 3.1 – Интерфейс программы - группы настроек построения поверхности

На Рисунке 3.1 представлен интерфейс настройки параметров построения фрактальной поверхности, включающий в себя выбор поверхности для построения, специфичных для конкретной поверхности параметров, интервала построения по осям X, Y и Z, характеристик материала согласно ТЗ, а так же - признака использования сглаживания в процессе синтеза сцены.

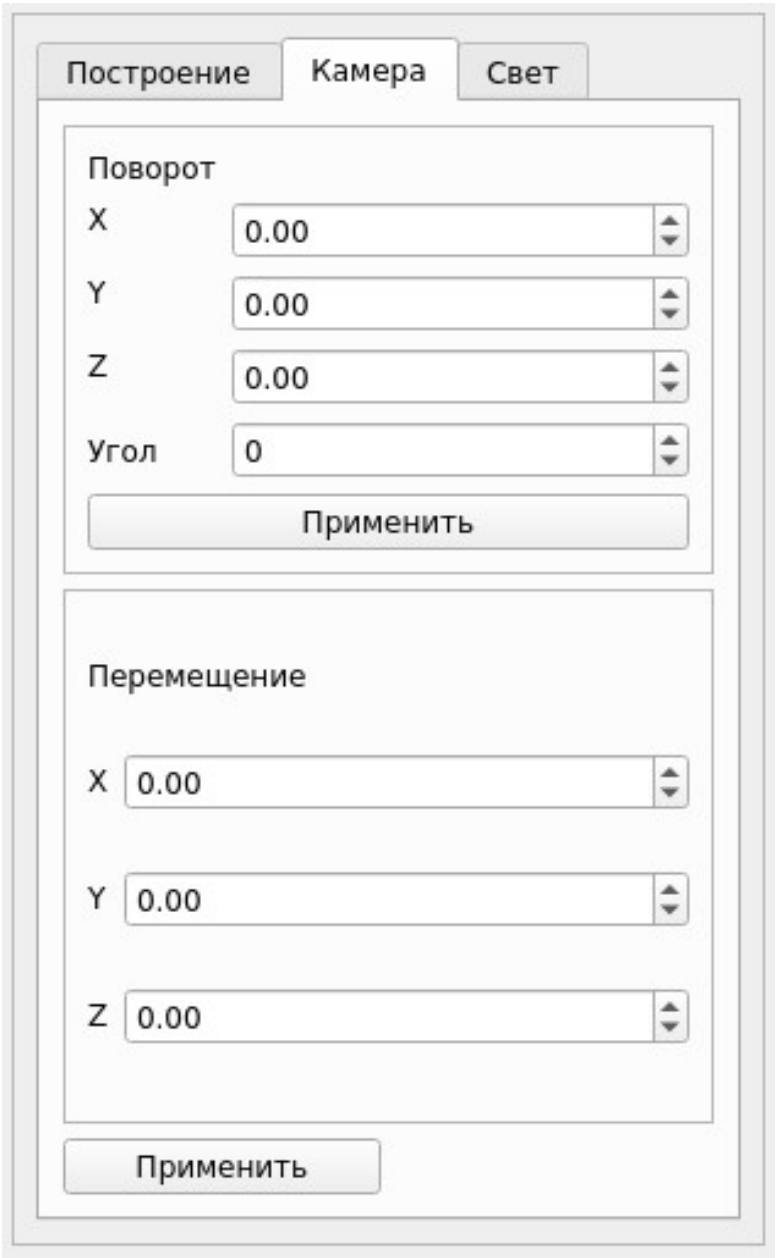

Рисунок 3.2 – Интерфейс программы - группы настроек камеры

На Рисунке 3.2 представлен интерфейс настройки параметров камеры (точки наблюдения), включающий в себя задание параметров и выполнение в соответствии с ними поворота камеры относительно вектора (X, Y, Z) на угол Угол, задание параметров и выполнение в соответствии с ними перемещения камеры относительно текущего положения камеры на вектор (X, Y, Z).

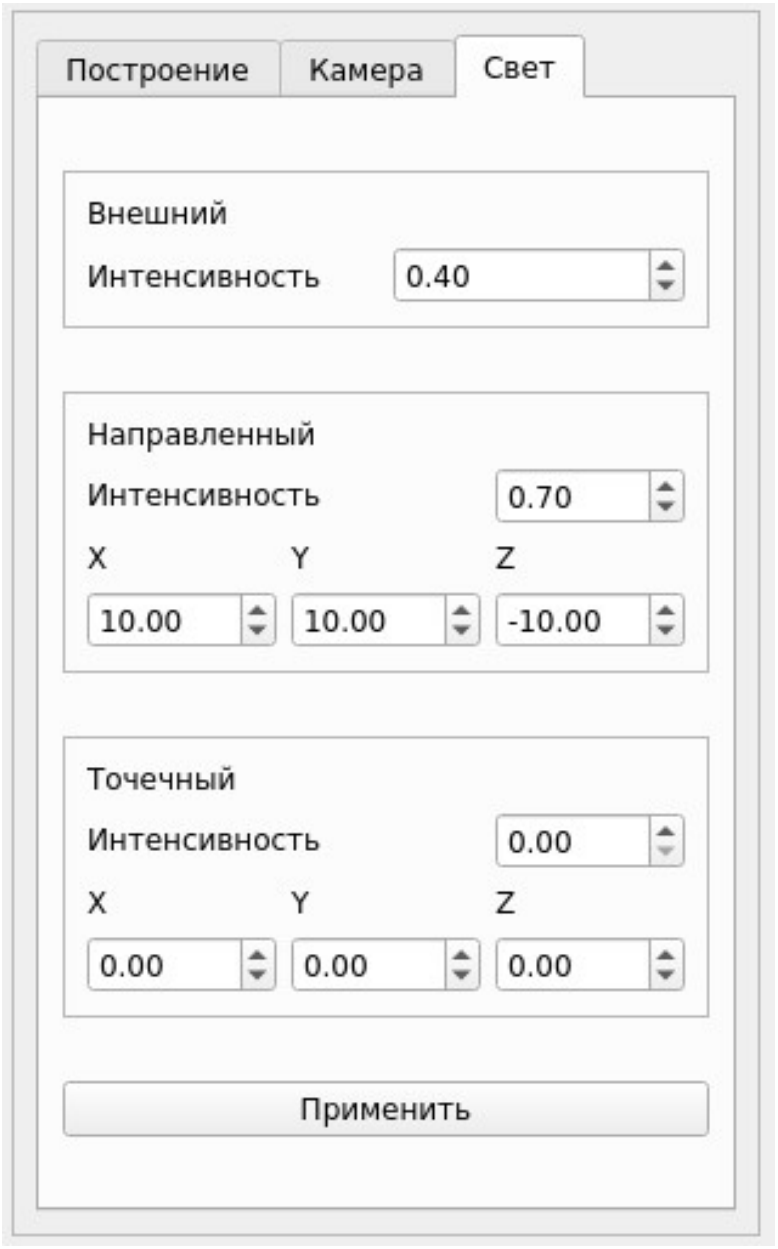

Рисунок 3.3 – Интерфейс программы - группы настроек освещения сцены

На Рисунке 3.3 представлен интерфейс настройки параметров освещения синтезируемой сцены, включающий в себя задание параметров различных источников освещения в соответствии с их описанием, приведенным ранее, а именно:

- Интенсивности для внешнего (рассеянного) света;
- Интенсивности, а так же вектора направления (X, Y, Z) направленного источника света;
- Интенсивности, а так же положения (X, Y, Z) точечного источника света;

3.3 Реализация алгоритмов

В листинге 3.1 представлена реализация алгоритма `Ray Marching`'а. В листинге 3.2 представлена реализация алгоритма расчета освещенности.

Листинг 3.1 – Реализация алгоритма Ray Marching

```
float Fractal::raymarching(
        math::Vector3 const &ro,
        math::Vector3 const &rd,
        math::Vector4 &rescol,
        float fov,
        math::Vector3 const &c) {
    float res = -1.0;

    auto dis = bounder_->intersect(ro, rd);

    if (dis.y() < 0.0) return -1.0;
    dis.setX(std::max(dis.x(), 0.0f));

    // raymarch fractal distance field
    math::Vector4 trap;

    float fovfactor = 1.0 / sqrt(1.0 + fov * fov);
    float t = dis.x();
    for (int i = 0; i < MAX_RAYMARCH_STEP; i++) {
        auto pos = ro + rd * t;
        float surface = std::clamp(1e-3f * t * fovfactor, 1e-4f,
           0.1f);

        float dt = map(pos, c, trap);
        if (t > dis.y() || dt < surface) break;
        t += std::min(dt, 0.05f);
    }

    if (t < dis.y()) {
        rescol = trap;
        res = t;
    }

    return res;
}
```

Листинг 3.2 – Реализация алгоритма расчета освещенности

```
double Drawer::computeLighting(
        const math::Vector3 &point,
        const math::Vector3 &normal,
        const math::Vector3 &view,
        float fov,
        math::Vector3 const &c) {
    if (!lights_) return 0.0;

    double intensity = 0;
    auto length_n = normal.length();
    auto length_v = view.length();

    for (const auto &light : *lights_) {
        if (light->getType() == light::LightType::Ambient) {
            intensity += light->getIntensity();
            continue;
        }

        auto vec_l = math::Vector3();
        double t_max = 1.0;

        if (light->getType() == light::LightType::Point) {
            vec_l = light->getPosition() - point;
        } else {
            vec_l = light->getPosition();
            t_max = std::numeric_limits<double>::max();
        }

        math::Vector4 tra;
        if (fractal_->raycast(point, vec_l, tra, fov, c) > 0)
           continue;

        // diffuse reflection
        auto n_dot_l = math::Vector3::dotProduct(normal, vec_l);
        if (n_dot_l > 0)
            intensity += fractal_->getRoughness() *
                light->getIntensity() * n_dot_l /
                (length_n * vec_l.length());

        // specular reflection
        auto vec_r = normal * (2.f * n_dot_l) - vec_l;
        auto r_dot_v = math::Vector3::dotProduct(vec_r, view);
        if (r_dot_v > 0) {
            intensity += light->getIntensity() *
                std::pow(r_dot_v / (vec_r.length() * length_v),
                SPECULARITY);
        }
    }

    return intensity;
}
```

3.4 Описание процесса сборки приложения

Для сборки программного обеспечения использовались инструменты `make` [12] и `CMake` [13].

Действия, необходимые для сборки проекта приведены приведены в листинге 3.3:

Листинг 3.3 – Сборка реализуемого программного обеспечения

```
1 $ mkdir -p build && cd build
2 $ cmake ..
3 $ make
```

В результате выполнения данных действий будет скомпилирован исполняемый файл `bmstu_iu7_cg_course`.

3.5 Вывод

В данном разделе были представлены средства разработки программного обеспечения, детали реализации и процесс сборки разрабатываемого программного обеспечения.

4 Экспериментальная часть

В данном разделе будет поставлен эксперимент, в котором будут сравнены временные характеристики работы реализованного программного обеспечения в различных конфигурациях.

4.1 Цель эксперимента

Целью эксперимента является проверка правильности выполнения поставленной задачи, оценка эффективности при многопоточной реализации синтеза изображения, а так же - сравнение эффективности работы многопоточной реализации при различном количестве потоков.

4.2 Апробация

На Рисунках 4.1 – 4.5 представлены результаты синтеза изображения с различными параметрами освещенности сцены. Можно заметить, что сцены работают корректно.

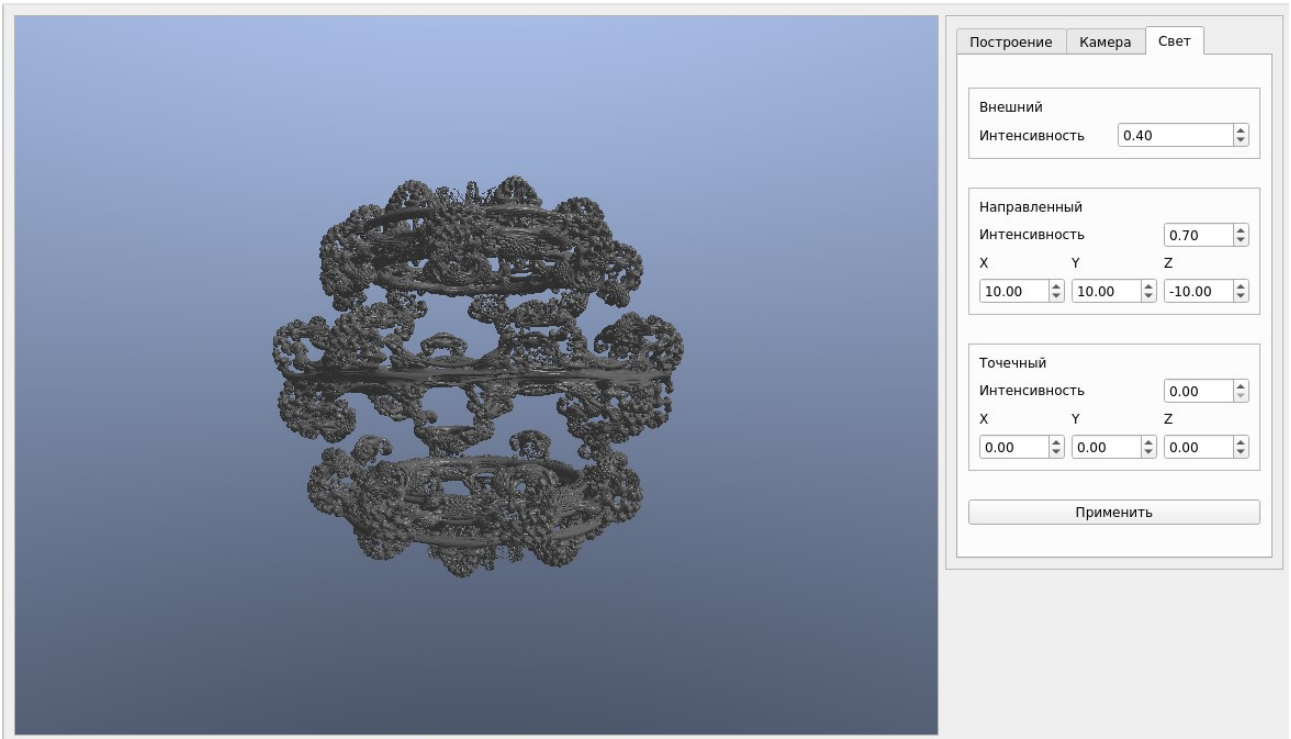

Рисунок 4.1 – Визуализация сцены в обычном режиме

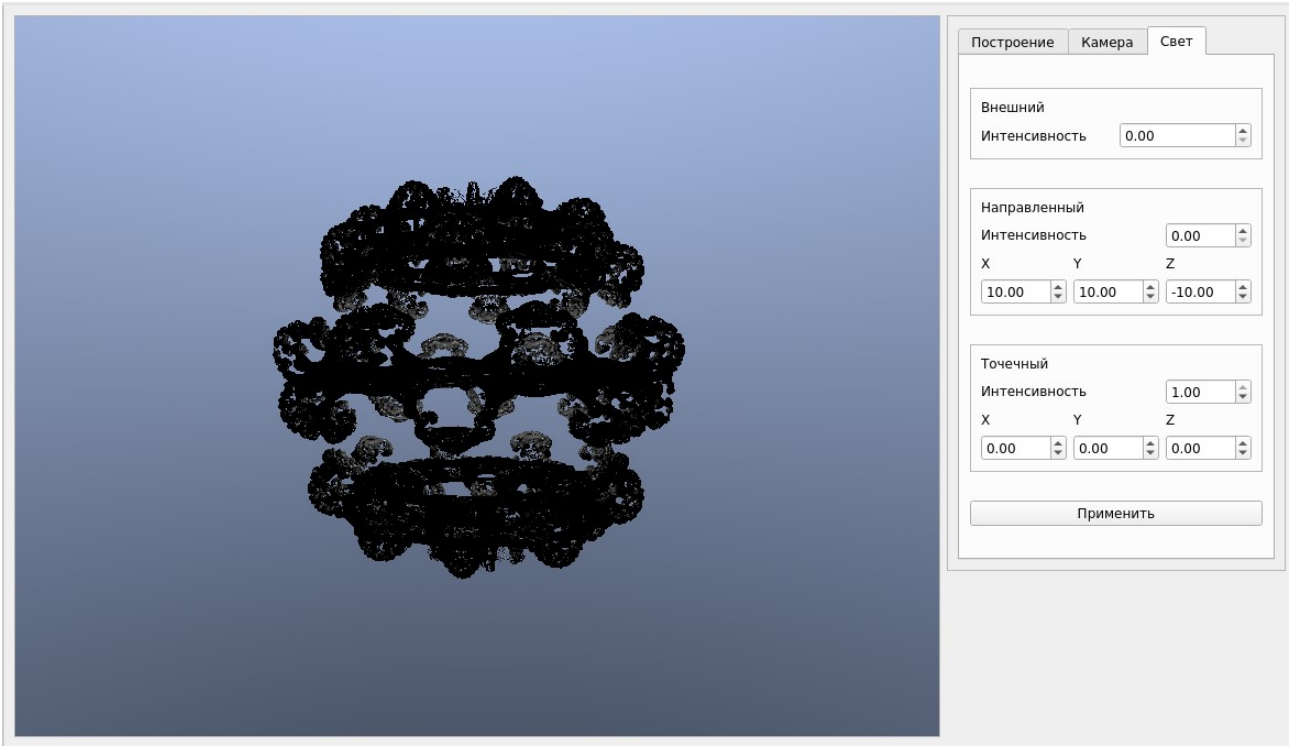

Рисунок 4.2 – Визуализация сцены только с точечным источником света максимальной интенсивности

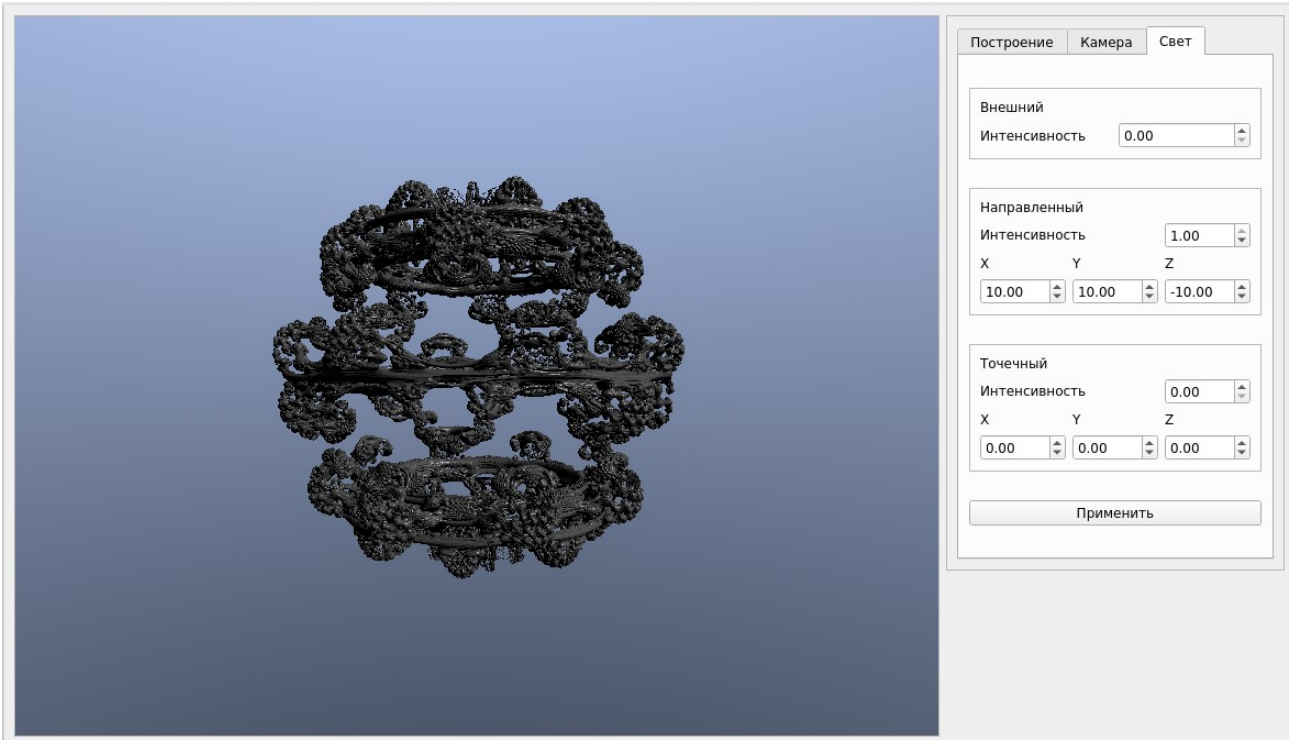

Рисунок 4.3 – Визуализация сцены только с направленным источником света максимальной интенсивности

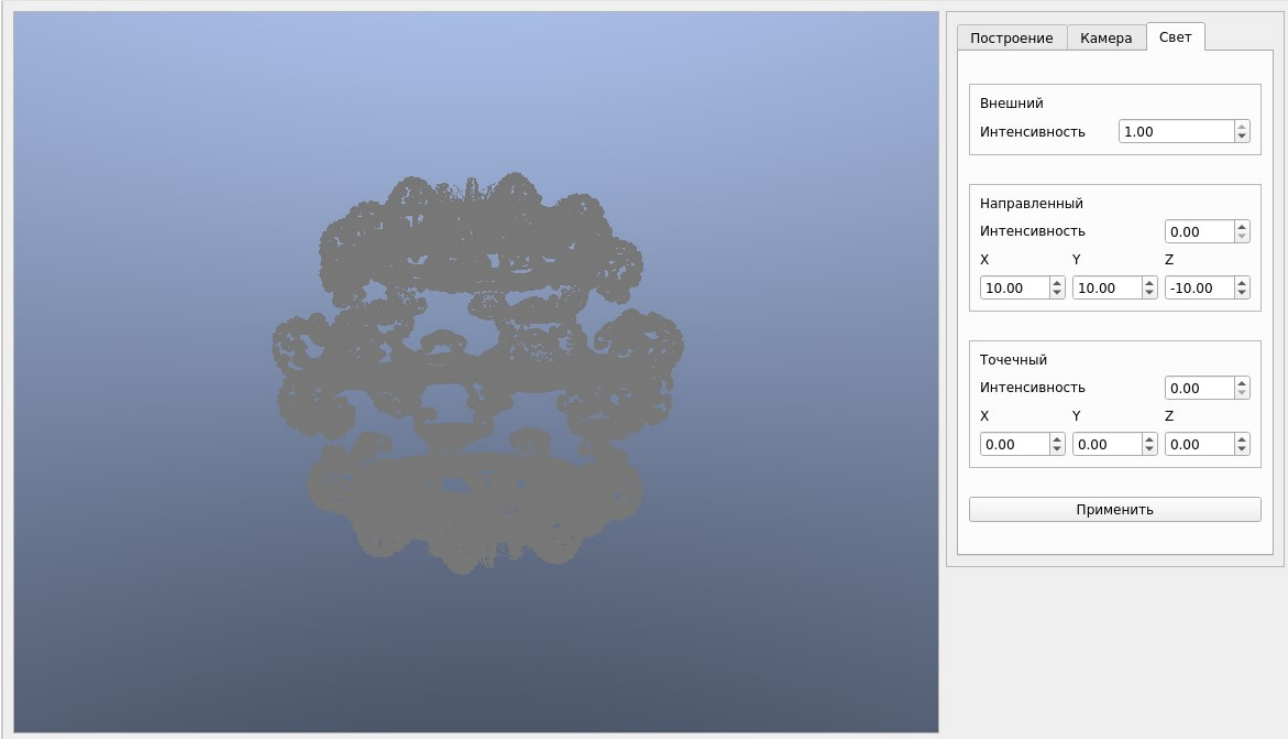

Рисунок 4.4 – Визуализация сцены только с рассеянным светом максимальной интенсивности

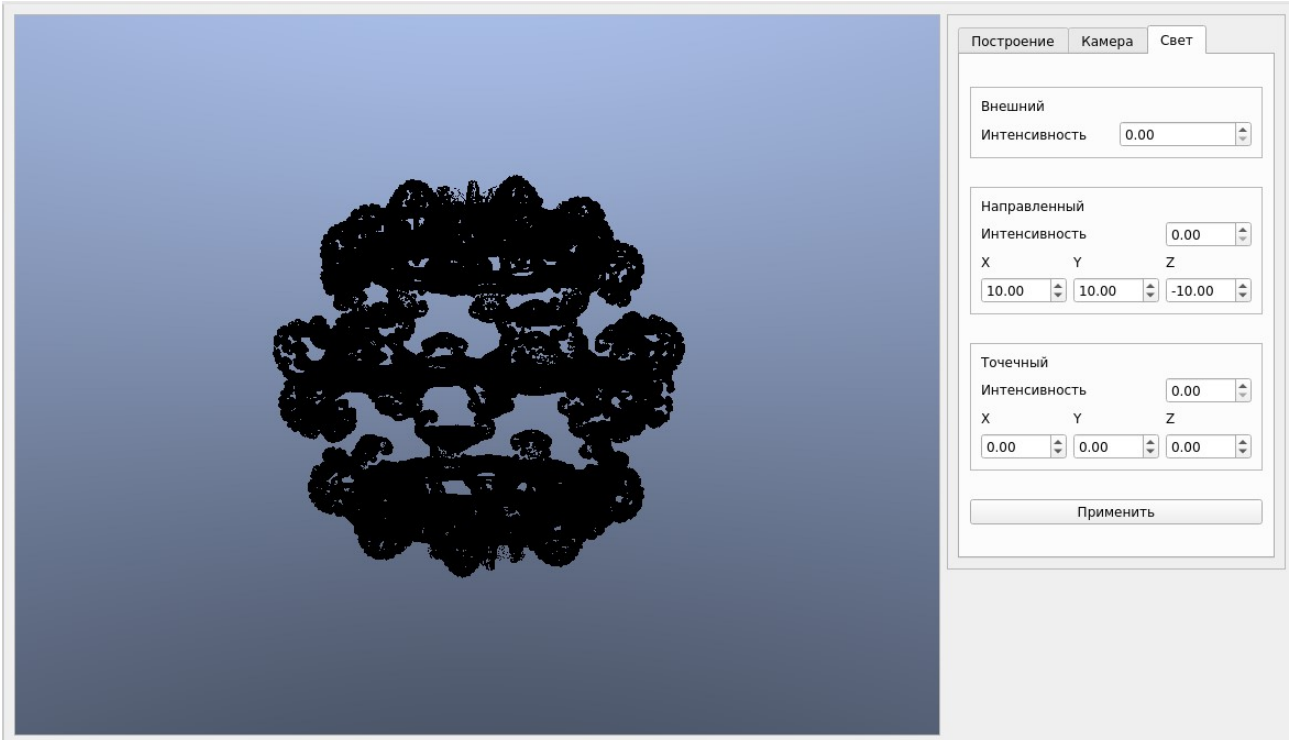

Рисунок 4.5 – Визуализация сцены без источников освещения

На Рисунках 4.6 – 4.7 показана одна и та же фрактальная поверхность при разном положении источника. Смена положения источника освещения работает корректно.

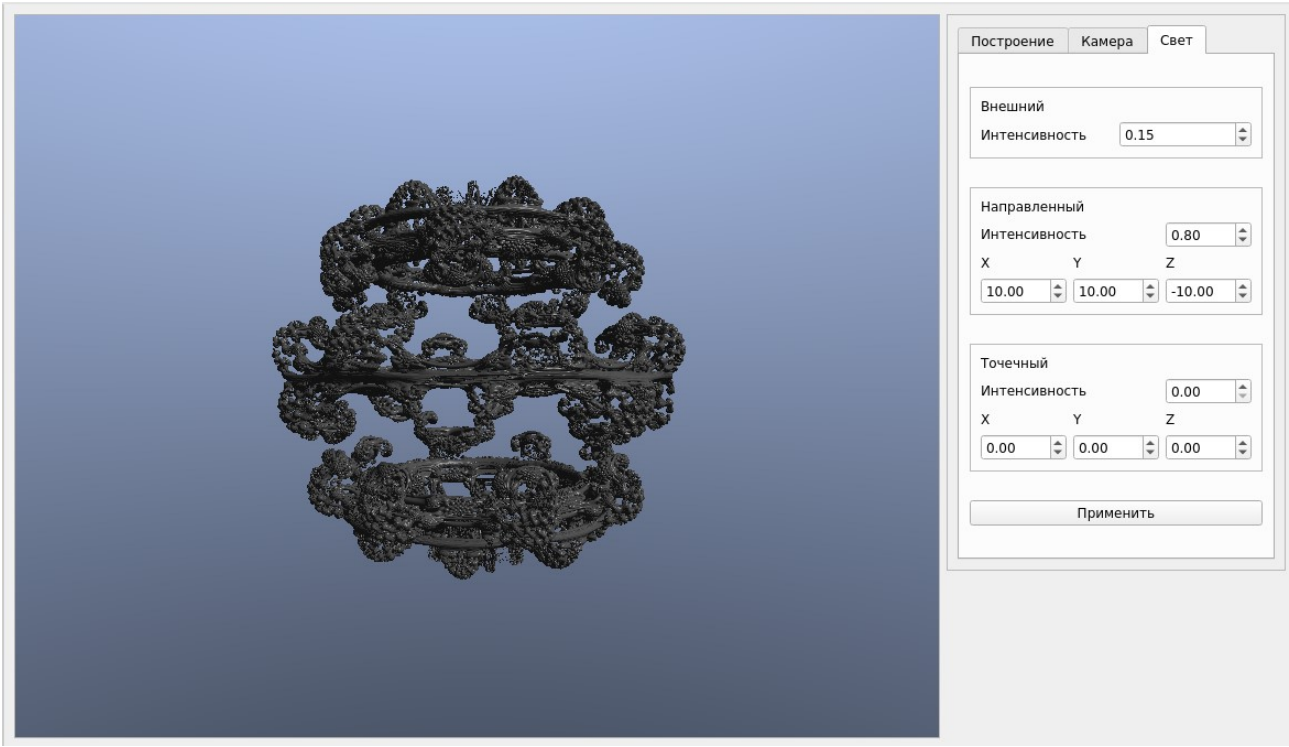

Рисунок 4.6 – Визуализация сцены

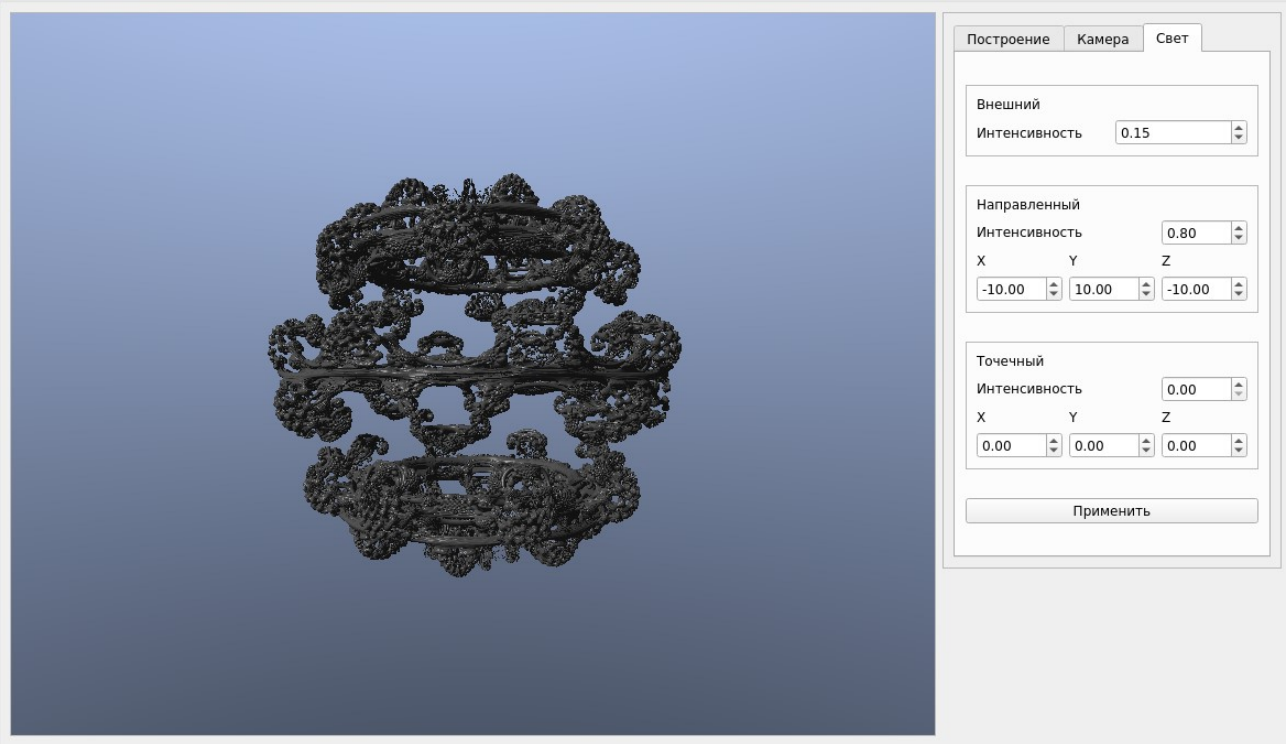

Рисунок 4.7 – Визуализация сцены со смещенным направленным источником

4.3 Технические характеристики

Технические характеристики устройства, на котором выполнялось исследование:

- процессор: Intel Core™ i5-8250U [16] CPU @ 1.60GHz;
- память: 32 GiB;
- операционная система: Manjaro [17] Linux [18] 21.1.4 64-bit.

Исследование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения рабочего стола, окружением рабочего стола, а также непосредственно системой тестирования.

4.4 Описание эксперимента

Была реализована функция параллельного синтеза сцены. Для этого была использована библиотека `OpenMP` [19], директива препроцессора `#pragma omp parallel for`. Данная директива препроцессора преобразует код для выполнения итераций цикла параллельно.

В данном эксперименте полученное изображение разбивалось на массивы рядов, во избежание проблем с разделяемой памятью (класс `QImage` [20], предоставляемый фреймворком `QT` [8], не может использоваться для параллельных записи и чтения), а по окончании собирались обратно.

В рамках данного эксперимента будет производиться оценка влияния размерности изображения и количества потоков на время работы алгоритма. Для этого будем синтезировать квадратные изображения с размерностями равными [300, 400, 500, ..., 1000] элементов. Количество потоков будет задано равным [1, 2, 4, 8, 16] штук.

Для снижения погрешности измерений будем усреднять получаемые значения. Для этого каждое из измерений будет проводиться $N = 100$ раз, после чего будет вычисляться среднее арифметическое значение измеряемой величины.

4.5 Результат эксперимента

В таблице 4.1 приведены экспериментально полученные значения временных характеристик работы алгоритма в зависимости от размерности синтезируемого изображения и количества потоков.

Таблица 4.1 – Замеры времени для изображений с различными размерностями

Размерность изображения, пикс.	**Количество потоков, шт.**				
	1	**2**	**4**	**8**	**16**
300	1.47e+03	8.35e+02	4.22e+02	2.85e+02	2.89e+02
400	2.59e+03	1.36e+03	7.82e+02	5.11e+02	5.25e+02
500	4.05e+03	2.13e+03	1.20e+03	7.77e+02	8.14e+02
600	5.80e+03	3.10e+03	1.73e+03	1.11e+03	1.18e+03
700	7.90e+03	4.37e+03	2.36e+03	1.50e+03	1.75e+03
800	1.05e+04	5.61e+03	3.03e+03	1.95e+03	1.99e+03
900	1.34e+04	7.03e+03	3.83e+03	2.47e+03	2.50e+03
1000	1.61e+04	8.71e+03	4.76e+03	3.06e+03	3.10e+03

Согласно данным, приведенным в таблице 4.1, время синтеза изображения зависит от размерности данного изображения как $O(mn)$ или $O(n^2)$ т.к. в данном эксперименте синтезируются только квадратные изображения.

На рисунке 4.8 приведены графики зависимости времени синтеза изображения от размерности синтезируемого изображения для различного числа потоков, использующихся в алгоритме.

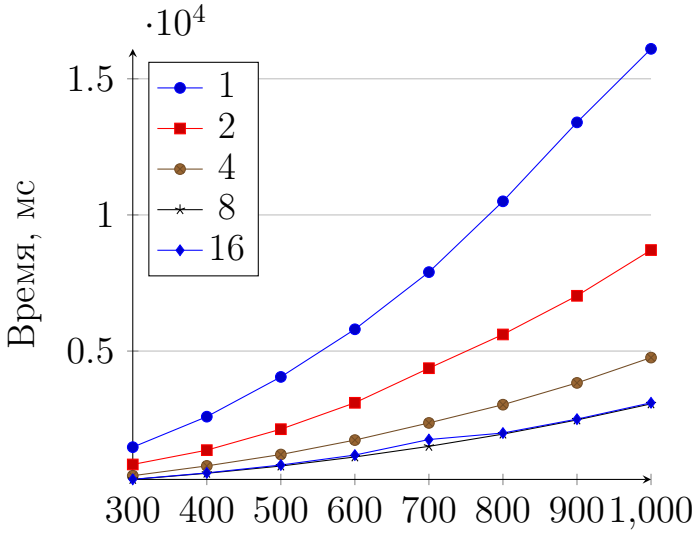

Рисунок 4.8 – Сравнение времени работы алгоритмов

Из рисунка 4.8 следует, что наиболее эффективным реализованный

алгоритм синтеза изображения становится при приближении числа потоков к количеству логических ядер используемой ЭВМ (8 для использованной в ходе эксперимента). Так, при синтезе изображений с размерностью равной 1000 пикселей, алгоритм использующий 8 потоков оказался в 5.25 раз эффективнее однопоточного алгоритма по времени.

Стоит отметить, что многопоточная реализация алгоритма оказалась более эффективной при всех рассматриваемых размерностях изображения. Связано это с большими размерностями синтезируемых изображений (от 300 до 1000 пикселей). Для изображения малых размерностей, однопоточный алгоритм окажется более эффективным в связи с отсутствием дополнительных затрат по времени и памяти, требуемых для реализации многопоточности (создание потоков, совместный лоступ к ресурсам).

4.6 Вывод

В данном разделе было произведено экспериментально сравнение временных характеристик реализованного программного обеспечения.

Время работы алгоритма имеет квадратичную зависимость от размерности синтезируемого изображения.

Наиболее эффективной по времени оказалась многопоточная реализация с числом потоков равным 8 (число логических ядер ЭВМ использованной в ходе эксперимента).

Заключение

В ходе курсового проекта было разработано программное обеспечение, предоставляющее возможность визуализации трехмерных фрактальных поверхностей. Разработанное программное обеспечение предоставляет функционал для изменения интервалов построения поверхностей, задания цвета и свойств материала поверхности, а так же задания и изменения в процессе работы положения точки наблюдения и источников света по их характеристикам (положению, интенсивности) в интерактивном режиме. В процессе выполнения данной работы были выполнены следующие задачи:

- описана структура синтезируемой трехмерной сцены;
- описаны существующие алгоритмы построения реалистичных изображений;
- были выбраны реализуемые алгоритмы;
- приведены схемы реализуемых алгоритмов;
- определены требования к программному обеспечению;
- описаны использующиеся структуры данных;
- описана структура разрабатываемого ПО;
- определены средства программной реализации;
- реализовано соответствующее ПО;
- проведены экспериментальные замеры временных характеристик разработанного ПО.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Роджерс Д.* Алгоритмические основы машинной графики. — 1-е изд. — Д. Роджерс. — Москва «Мир», 1989.
2. *Шикин Е. В.* Компьютерная графика. Динамика, реалистические изображения //. — М.: ДИАЛОГ–МИФИ, 1998. — С. 288.
3. Ю.М. Баяковский. Трассировка лучей из книги Джефа Проузиса [Электронный ресурс]. — Режим доступа: `https://www.graphicon.ru/oldgr/courses/cg99/notes/lect12/prouzis/raytrace.htm` дата обращения: 03.11.2021).
4. Проблемы трассировки лучей – из будущего в реальное время [Электронный ресурс]. — Режим доступа: `https://nvworld.ru/articles/ray-tracing/3/` дата обращения: 03.11.2021).
5. *Снижко Е. А.* Компьютерная графика. Динамика, реалистические изображения //. — Балт. гос. техн. ун-т. – СПб., 2005. — С. 132.
6. Александр Антонов. Под законом Амдала [Электронный ресурс]. — Режим доступа: `https://old.computerra.ru/2002/430/198481/` дата обращения: 05.11.2021).
7. Standard C++ [Электронный ресурс]. — Режим доступа: `https://isocpp.org/` (дата обращения: 24.10.2021).
8. Qt | Cross-platform software development for embedded & desktop [Электронный ресурс]. — Режим доступа: `https://www.qt.io/` (дата обращения: 24.10.2021).
9. Clang C Language Family Frontend for LLVM. — Режим доступа: `https://clang.llvm.org/` (дата обращения: 2.11.2021).
10. Clang Static Analyzer. — Режим доступа: `https://clang-analyzer.llvm.org/` (дата обращения: 2.11.2021).
11. Valgrind Home. — Режим доступа: `https://valgrind.org/` (дата обращения: 2.11.2021).
12. Make - GNU Project - Free Software Foundation. — Режим доступа: `https://www.gnu.org/software/make/` (дата обращения: 3.11.2021).

13. CMake. — Режим доступа: https://cmake.org/ (дата обращения: 3.11.2021).

14. Visual Studio Code - Code Editing. Redefined. — Режим доступа: https://code.visualstudio.com/ (дата обращения: 1.11.2021).

15. Managing Extensions in Visual Studio Code. — Режим доступа: https://code.visualstudio.com/docs/editor/extension-marketplace (дата обращения: 1.11.2021).

16. Процессор Intel® Core™ i5-8250U [Электронный ресурс]. — Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/124967/intel-core-i5-8250u-processor-6m-cache-up-to-3-40-ghz.html (дата обращения: 24.10.2021).

17. Manjaro - enjoy the simplicity [Электронный ресурс]. — Режим доступа: https://manjaro.org/ (дата обращения: 24.10.2021).

18. LINUX.ORG.RU – Русская информация об ОС Linux [Электронный ресурс]. — Режим доступа: https://www.linux.org.ru/ (дата обращения: 24.10.2021).

19. Директивы OpenMP | Microsoft Docs. — Режим доступа: https://docs.microsoft.com/ru-ru/cpp/parallel/openmp/reference/openmp-directives?view=msvc-170 (дата обращения: 24.10.2021).

20. QImage Class | Qt GUI 5.15.7 - Qt Documentation. — Режим доступа: https://doc.qt.io/qt-5/qimage.html (дата обращения: 24.10.2021).