Оглавление

Введение

В современном мире компьютерная графика используется достаточно широко. Типичная область ее применения – это кинематография и компьютерные игры.

На сегодняшний день большое внимание уделяется алгоритмам получения реалистичного изображения. Такие алгоритмы являются одними из самых затратных по времени, потому что они должны предусматривать множество физических явлений, таких как преломление, отражение, рассеивание света. Для создания еще более реалистичного изображения также учитывается дифракция, вторичное, троичное отражение света, поглощение.

Можно заметить, что чем качественнее мы получаем изображение на выходе алгоритма, тем больше времени и памяти мы используем для синтеза. Это и становится проблемой при создании динамической сцены, так как на каждом временном интервале необходимо производить расчеты заново.

Целью данной курсовой работы является создание реалистичной сцены, на которой присутствуют дом и молния, с использованием выбранных алгоритмов, и при этом оптимизировать данные алгоритмы.

Чтобы достигнуть поставленной цели, требуется решить следующие задачи:

- описать структуру трехмерной сцены, включая объекты, из которых состоит сцена;
- проанализировать и выбрать необходимые существующие алгоритмы для построения сцены;
- проанализировать и выбрать варианты оптимизаций ранее выбранных алгоритмов (если есть необходимоть);
- реализовать выбранные квантовые алгоритмы;
- разработать программное обеспечение, которое позволит отобразить трехмерную сцену и визуализировать удар молнии;
- провести сравнение реализованных оптимизированных алгоритмов и неоптимизированных.

1 Аналитическая часть

надо что-то написать В данном разделе будут выбраны алгоритмы для разработки программного обеспечения.

1.1 Описание объектов сцены

Сцена состоит из источника света, молнии, дома и плоскости земли. Источник свет представляет собой материальную точку, испускающую лучи света во все стороны (если источник расположен в бесконечности, то он имеет направление). В моей программе источником света будет молния.

Молния представляет собой ломаную линию, которая имеет начало и конец, а также несколько ветвей.

Дом – сооружение, для которого пользователь должен задать этажность, а также указать, в каких окнах включен свет.

Плоскость земли – это некая ограничивающая плоскость. Предполагается, что под такой плоскостью не расположено никаких объектов. Располагается на максимальной координате по оси Y.

1.2 Анализ и выбор формы задания трехмерных моделей

Отображением формы и размеров объектов являются модели. Обычно используются три формы задания моделей.

1. Каркасная (проволочная) модель.

   Одна из простейших форм задания модели, так как мы храним информацию только о вершинах и ребрах нашего объекта. Недостаток данной формы состоит в том, что модель не всегда точно передает представление о форме объекта.

2. Поверхностная модель.

Такой тип модели часто используется в компьютерной графике. Поверхности можно задавать разными способами: либо аналитически, либо задавать участки поверхности, как поверхность того или иного вида (использовать полигональную аппроксимацию). Недостаток данной формы состоит в том, что мы не знаем с какой стороны находится материал.

3. Объемная (твердотельная) модель.

   Данная форма отличается от поверхностной тем, что у нас есть информация о том, где расположен материал. Это делается с помощью указания направления внутренней нормали.

Таким образом, можно сделать вывод о том, что для решение данной задачи нам подойдут поверхностные модели, так как каркасные модели могут привести к неправильному восприятию формы, а объемные модели будут излишеством, так как будут тратить больше памяти.

1.2.1 Анализ способа задания поверхностных моделей

Также необходимо определить каким образом лучше всего задавать поверхностные модели.

1. Аналитический способ.

   Этот способ задания модели характеризуется описанием модели объекта, которое доступно в неявной форме, то есть для получения визуальных характеристик необходимо дополнительно вычислять некоторую функцию, которая зависит от параметра.

2. Полигональная сетка.

   Данный способ характеризуется совокупностью вершин, граней и ребер, которые определяют форму многогранного объекта в трехмерной компьютерной графике.

Для более верного выбора также следует перечислить способы хранения информации о сетке.

1. Список граней.

   Объект – это множество граней и множество вершин. В каждую грань входят как минимум 3 вершины;

2. «Крылатое» представление.

   Каждая точка ребра указывает на две вершины, две грани и четыре ребра, которые её касаются.

3. Полурёберные сетки.

   То же «крылатое» представление, но информация обхода хранится для половины грани.

4. Таблица углов.

   Это таблица, хранящая вершины. Обход заданной таблицы неявно задаёт полигоны. Такое представление более компактно и более производительно для нахождения полигонов, но, в связи с тем, что вершины присутствуют в описании нескольких углов, операции по их изменению медленны.

5. Вершинное представление.

   Хранятся лишь вершины, которые указывают на другие вершины. Простота представления даёт возможность проводить над сеткой множество операций.

Стоит отметить, что одним из решающих факторов в выборе способа задания модели в данном проекте является скорость выполнения преобразований над объектами сцены.

При реализации программного продукта наиболее удобным представлением является модель, заданная полигональной сеткой – это поможет избежать проблем при описании сложных моделей. При этом способ хранения полигональной сетки – список граней, так как он предоставляет явное описание граней, что поможет при реализации алгоритма удаления невидимых рёбер и поверхностей. Также этот способ позволит эффективно преобразовывать модели, так как структура будет включать в себя список вершин.

1.3 Анализ и выбор алгоритма удаления невидимых ребер и поверхностей

Перед выбором алгоритма удаления невидимых ребер выделим несколько свойств, которыми должен обладать выбранный алгоритм, чтобы обеспечить оптимальную работу и реалистичное изображение.

Свойства:

- алгоритм может работать как в объектном пространстве, так и в пространстве изображений;
- алгоритм должен быть достаточно быстрым и использовать мало памяти;
- алгоритм должен иметь высокую реалистичность изображения.

Алгоритм, использующий Z-буфер

Суть данного алгоритма – это использование двух буферов: буфера кадра, в котором хранятся атрибуты каждого пикселя, и Z-буфера, в котором хранятся информация о координате Z для каждого пикселя.

Первоначально в Z-буфере находятся минимально возможные значения Z, а в буфере кадра располагаются пиксели, описывающие фон. Каждый многоугольник преобразуется в растровую форму и записывается в буфер кадра.

В процессе подсчета глубины нового пикселя, он сравнивается с тем значением, которое уже лежит в Z-буфере. Если новый пиксель расположен ближе к наблюдателю, чем предыдущий, то он заносится в буфер кадра и происходит корректировка Z-буфера.

Для решения задачи вычисления глубины Z каждый многоугольник описывается уравнением $ax + by + cz + d = 0$. При $c = 0$ многоугольник для наблюдателя вырождается в линию.

Для некоторой сканирующей строки y=const, поэтому имеется возможность рекуррентно высчитывать z' для каждого $x' = x + dx$: $z' - z = -\frac{ax'+d}{c} + \frac{ax+d}{c} = \frac{a(x-x')}{c}$.

Получим $z' = z - \frac{a}{c}$, так как $x - x' = dx = 1$.

При этом стоит отметить, что для невыпуклых многогранников предварительно потребуется удалить нелицевые грани.

Преимущества

- простота реализации;
- оценка трудоемкости линейна.

Недостатки

- сложная реализация прозрачности;
- большой объем требуемой памяти.

Вывод

Данный алгоритм не подходит для решения поставленной задачи, так как требует большой объем памяти, что не удовлетворяет требованиям.

Алгоритм обратной трассировки лучей

Суть данного алгоритма состоит в том, что наблюдатель видит объект с помощью испускаемого света, который согласно законам оптики доходит до наблюдателя некоторым путем. Отслеживать пути лучей от источника к наблюдателю неэффективно с точки зрения вычислений, поэтому наилучшим способом будет отслеживание путей в обратном направлении, то есть от наблюдателя к объекту.

Преимущества

- высокая реалистичность синтезируемого изображения;
- работа с поверхностями в математической форме;
- вычислительная сложность слабо зависит от сложности сцены.

Недостатки

- производительность.

Вывод

Данный алгоритм не отвечает главному требованию – скорости работы, но при некоторой адаптации можно добиться большей скорости работы.

Алгоритм Робертса

Данный алгоритм работает в объектном пространстве, решая задачу только с выпуклыми телами.

Алгоритм выполняется в 3 этапа.

Этап подготовки исходных данных На данном этапе должна быть задана информация о телах. Для каждого тела сцены должна быть сформирована матрица тела V. Размерность матрицы - $4 * n$, где n – количество граней тела.

Каждый столбец матрицы представляет собой четыре коэффициента уравнения плоскости $ax + by + cz + d = 0$, проходящей через очередную грань.

Таким образом, матрица тела будет представлена в следующем виде

$$V = \begin{pmatrix} a_1 & a_2 & \dots & a_n \\ b_2 & b_2 & \dots & b_n \\ c_2 & c_2 & \ddots & c_n \\ d_2 & d_2 & \dots & d_n \end{pmatrix} \tag{1.1}$$

Матрица тела должна быть сформирована корректно, то есть любая точка, расположенная внутри тела, должна располагаться по положительную сторону от каждой грани тела. В случае, если для очередной грани условие не выполняется, соответствующий столбец матрицы надо умножить на -1.

Этап удаления рёбер, экранируемых самим телом

На данном этапе рассматривается вектор взгляда $E = \{0, 0, -1, 0\}$.

Для определения невидимых граней достаточно умножить вектор E на матрицу тела V. Отрицательные компоненты полученного вектора будут соответствовать невидимым граням.

Этап удаления невидимых рёбер, экранируемых другими телами сцены

На данном этапе для определения невидимых точек ребра требуется построить луч, соединяющий точку наблюдения с точкой на ребре. Точка будет невидимой, если луч на своём пути встречает в качестве преграды рассматриваемое тело.

Преимущества

- работа в объектном пространстве;
- высокая точность вычисления.

Недостатки

- рост сложности алгоритма – квадрат числа объектов;
- тела сцены должны быть выпуклыми (усложнение алгоритма, так как нужна будет проверка на выпуклость);
- сложность реализации.

Вывод

Данный алгоритм не подходит для решения поставленной задачи из-за высокой сложности реализации как самого алгоритма, так и его модификаций, отсюда низкая производительность.

Алгоритм художника

Данный алгоритм работает аналогично тому, как художник рисует картину – то есть сначала рисуются дальние объекты, а затем более близкие. Наиболее распространенная реализация алгоритма – сортировка по глубине, которая заключается в том, что произвольное множество граней

сортируется по ближнему расстоянию от наблюдателя, а затем отсортированные грани выводятся на экран в порядке от самой дальней до самой ближней. Данный метод работает лучше для построения сцен, в которых отсутствуют пересекающиеся грани.

Преимущества

- требование меньшей памяти, чем, например, алгоритм Z-буффера.

Недостатки

- недостаточно высока реалистичность изображения;
- сложность реализации при пересечения граней на сцене.

Вывод

Данный алгоритм не отвечает главному требованию – реалистичность изображения. Также алгоритм художника отрисовывает все грани (в том числе и невидимые), на что тратится большая часть времени.

Алгоритм Варнока

Алгоритм Варнока является одним из примеров алгоритма, основанного на разбиении картинной плоскости на части, для каждой из которых исходная задача может быть решена достаточно просто.

Поскольку алгоритм Варнока нацелен на обработку картинки, он работает в пространстве изображения. В пространстве изображения рассматривается окно и решается вопрос о том, пусто ли оно, или его содержимое достаточно просто для визуализации. Если это не так, то окно разбивается на фрагменты до тех пор, пока содержимое фрагмента не станет достаточно простым для визуализации или его размер не достигнет требуемого предела разрешения.

Сравнивая область с проекциями всех граней, можно выделить случаи, когда изображение, получающееся в рассматриваемой области, определяется сразу:

- проекция ни одной грани не попадает в область;
- проекция только одной грани содержится в области или пересекает область, то в этом случае проекции грани разбивают всю область на две части, одна из которых соответствует этой проекции;
- существует грань, проекция которой полностью накрывает данную область, и эта грань расположена к картинной плоскости ближе, чем все остальные грани, проекции которых пересекают данную область, то в данном случае область соответствует этой грани.

Если ни один из рассмотренных трех случаев не имеет места, то снова разбиваем область на четыре равные части и проверяем выполнение этих условий для каждой из частей. Те части, для которых таким образом не удалось установить видимость, разбиваем снова и т. д.

Преимущества

- меньшие затраты по времени в случае области, содержащий мало информации.

Недостатки

- алгоритм работает только в пространстве изображений;
- большие затраты по времени в случае области с высоким информационным содержимым.

Вывод

Данный алгоритм не отвечает требованию работы как в объектном пространстве, так и в пространстве изображений, а также возможны большие затраты по времени работы.

1.3.1 Вывод

Для удаления невидимых линий выбран алгоритм обратной трассировки лучей. Данный алгоритм позволит добиться максимальной реалистич-

ности и даст возможность смоделировать распространение света в пространстве, учитывая законы геометрической оптики. Данный алгоритм можно модернизировать, добавив в него обработку новых световых явлений. Также этот алгоритм позволяет строить качественные тени с учетом большого числа источников. Стоит отметить тот факт, что алгоритм трассировки лучей не требователен к памяти, в отличие, например, от алгоритма Z-буфера.

1.4 Анализ и выбор модели освещения

Физические модели материалов стараются аппроксимировать свойства некоторого реального материала. Такие модели учитываются особенности поверхности материала или же поведение частиц материала.

Эмпирические модели материалов устроены иначе, чем физически обоснованные. Данные модели подразумевают некий набор параметров, которые не имеют физической интерпретации, но которые позволяют с помощью подбора получить нужный вид модели.

Рассмотрим эмпирические модели, а конкретно модель Ламберта и модель Фонга.

Модель Ламберта

Модель Ламберта моделирует идеальное диффузное освещение, то есть свет при попадании на поверхность рассеивается равномерно во все стороны. При такой модели освещения учитывается только ориентация поверхности (N) и направление источника света (L). Иллюстрация данной модели представлена на рисунке 1.1.

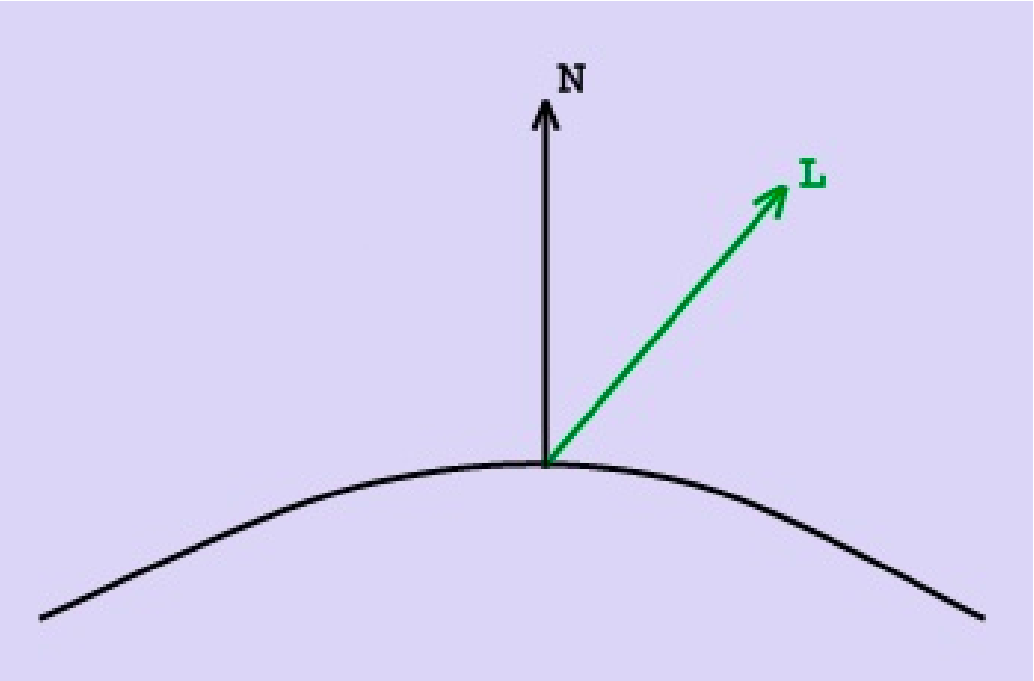

Рис. 1.1: Направленность источника света

Эта модель является одной из самых простых моделей освещения и очень часто используется в комбинации с другими моделями. Она может быть очень удобна для анализа свойств других моделей, за счет того, что ее легко выделить из любой модели и анализировать оставшиеся составляющие.

Модель Фонга

Это классическая модель освещения. Модель представляет собой комбинацию диффузной и зеркальной составляющих. Работает модель таким образом, что кроме равномерного освещения на материале могут появляться блики. Местонахождение блика на объекте определяется из закона равенства углов падения и отражения. Чем ближе наблюдатель к углам отражения, тем выше яркость соответствующей точки.

Падающий и отраженный лучи лежат в одной плоскости с нормалью к отражающей поверхности в точке падения (рисунок 1.2). Нормаль делит угол между лучами на две равные части. L – направление источника света, R – направление отраженного луча, V – направление на наблюдателя.

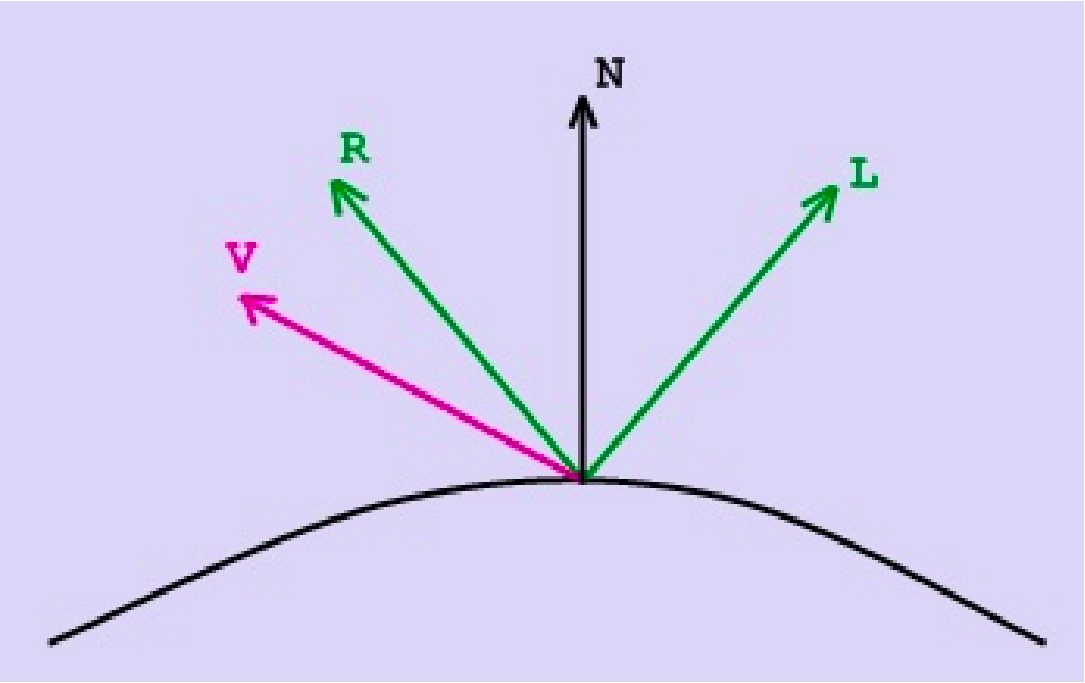

Рис. 1.2: Направленность источника света

1.5 Вывод

В данном разделе был проведен анализ алгоритмов удаления невидимых линий и методов освещения, которые возможно использовать для достижения поставленных задач. В качестве ключевого алгоритма, который также можно оптимизировать, выбран алгоритм трассировки лучей, который будет реализован в рамках данного курсового проекта.

2 Конструкторская часть

В данном разделе будут рассмотрены требования к программе и алгоритмы визуализации сцены и молнии.

2.1 Требования к программе

Программа должна предоставлять следующие возможности:

- визуальное отображение сцены;
- запуск и остановка ударов молнии;
- возможность включения и выключения света во всех окнах дома;
- поворот исходной сцены.

2.2 Общий алгоритм решения поставленной задачи

1. Задать объекты сцены (дом, окна, в которых включен свет).
2. Задать ионизацию облака.
3. Рассчитать координаты конца молнии, и в зависимости от этого рассчитать куда ударит молния.
   (a) Если конец молнии находится в области дома, то молния ударит в молниезащиту дома.
   (b) Если конец молнии находится вне области дома, то молния ударит в землю в случае, если молния-лидер.
4. Изобразить молнию.

2.3 Алгоритм обратной трассировки лучей

Алгоритмы трассировки лучей на сегодняшний день считаются наиболее мощными при создании реалистичных изображений.

Изображение формируется из-за того, что свет попадает в камеру. Выпустим из источников света множество лучей (первичные лучи). Часть этих лучей “улетит” в свободное пространство, а часть попадет на объекты. На них лучи могут преломляться и отражаться. При этом часть энергии луча поглотится. Преломленные и отраженные лучи образуют новое поколение лучей. Далее эти лучи опять же преломятся, отразятся и образуют новое поколение лучей. В конечном итоге часть лучей попадет в камеру и сформирует изображение. Это описывает работу прямой трассировки лучей.

Метод обратной трассировки лучей позволяет значительно сократить перебор световых лучей. В этом методе отслеживаются лучи не от источников, а из камеры. Таким образом, трассируется определенное число лучей, равное разрешению картинки.

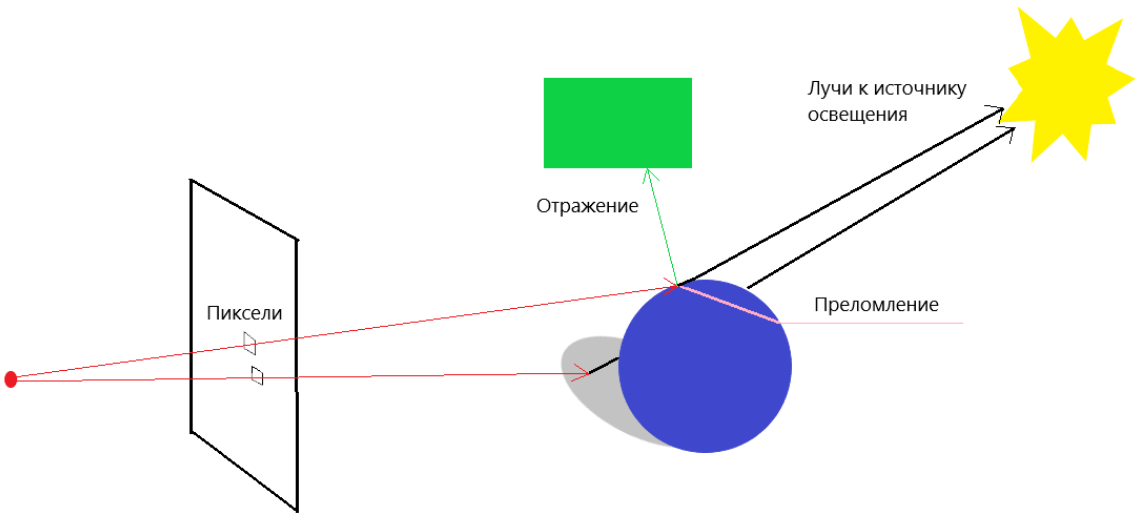

Рис. 2.1: Пример работы алгоритма трассировки лучей

Предположим, что у нас есть камера и экран, находящийся на расстоянии h от нее (рисунок 2.1). Разобьем экран на квадратики. Дальше будем по очереди проводить лучи из камеры в центр каждого квадратика

(первичные лучи). Найдем пересечение каждого такого луча с объектами сцены и выберем среди всех пересечений самое близкое к камере. Далее, применив нужную модель освещения, можно получить изображение сцены. Это самый простой метод трассировки лучей. Он позволяет лишь отсечь невидимые грани.

Но если надо смоделировать такое явление как отражение, то необходимо из самого близкого пересечения пустить вторичные лучи. Например, если поверхность отражает свет, и она идеально ровная, то необходимо отразить первичный луч от поверхности и пустить по этому направлению вторичный луч. Если же поверхность неровная, то необходимо пустить множество вторичных лучей.

На рисунке 2.2 представлена схема алгоритма.

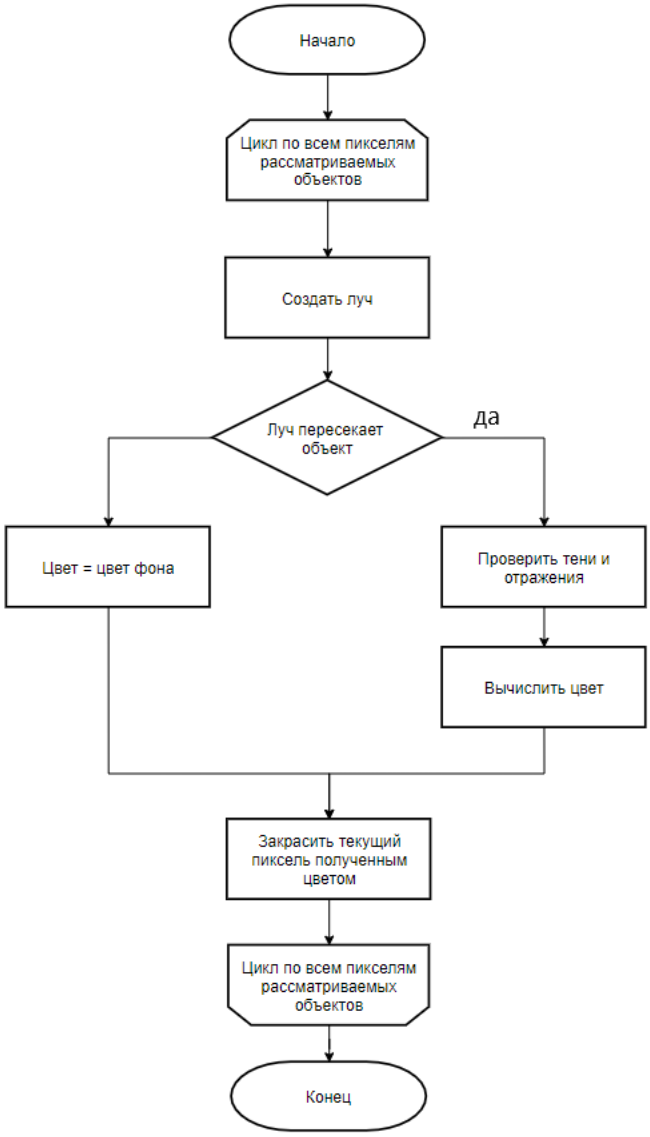

Рис. 2.2: Схема алгоритма трассировки лучей

2.4 Способ оптимизации алгоритма трассировки лучей

Для уменьшения времени работы алгоритма при реализации данной программы необходимо испускать лучи из камеры не по всей сцене, а в отдельные ее участки. Во-первых, лучи пускаются во все сегменты молнии для того, чтобы узнать, где находится молния за домом или перед ним. Во-вторых, лучи пускаются в черные окна, чтобы затем пустить из них вторичный луч и понять отражается в них молния или нет. Таким образом можно получить большой выигрыш по времени.

2.5 Алгоритм генерации молнии

Первоначально случайным образом задаются две координаты на молнии, ее конец и начало. По данным двум точками строим прямую, путем вычитания из координат конца координаты начала молнии, также находим расстояние от нее до громоотвода. Если расстояние это меньше нужного, то нужно поменять координаты конца на координаты вершины громоотвода.

Существует два вида молнии.

1. Обычная молния – это молния, которая не доходит до объекта или земли (рисунок 2.3).

2. Молния лидер – это молния, которая доходит до какого-то объекта либо до земли (рисунок 2.4).

Далее генерацию молнии можно разделить на два случая.

1. Молния бьет в землю – в данном случае молния имеет более непредсказуемый характер и может себя вести произвольно.

2. Молния бьет в громоотвод – в данном случае молния движется от начала удара до вершины громоотвода с небольшими колебаниями.

Каждую итерацию каждый сегмент делится пополам, с небольшим сдвигом центральной точки.

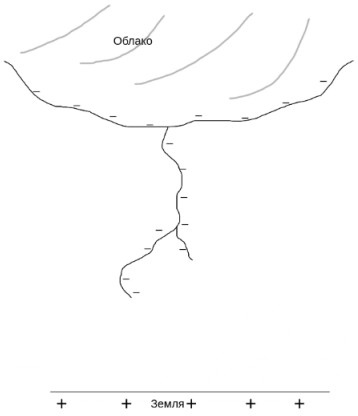

Рис. 2.3: Обычная молния

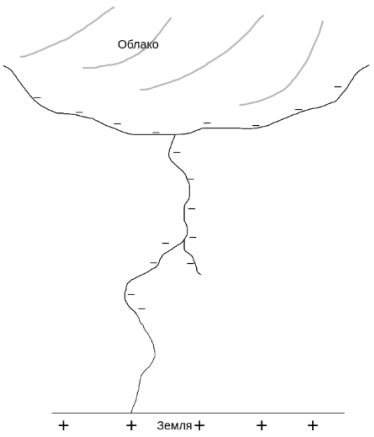

Рис. 2.4: Молния-лидер

Чтобы создать ветви, когда разделяем сегмент молнии, вместо добавления двух сегментов надо добавить три. Третий сегмент – это продолжение молнии в направлении первого с небольшим отклонением.

На каждом сегменте с вероятностью в 1% появляется побочная ветвь, которая строится по таким же законам, как и главная в том случае, если молния бьет в землю. Для каждой такой побочной ветви генерируется угол на который она повернута относительно главной ветви. Длина побочного сегмента зависит от того, в каком месте молнии она появляется: чем ближе к концу, тем короче она будет.

Пример генерации молнии без побочных сегментов (ветвей) представлен на рисунке 2.5 и с побочными сегментами (ветвями) – на рисунке 2.6.

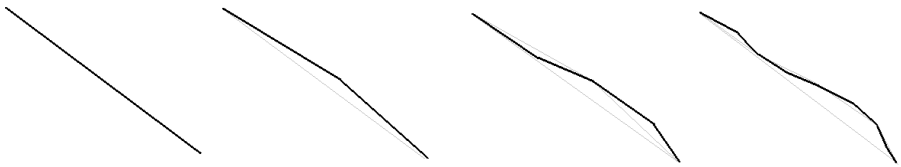

Рис. 2.5: Генерация молнии без побочных ветвей

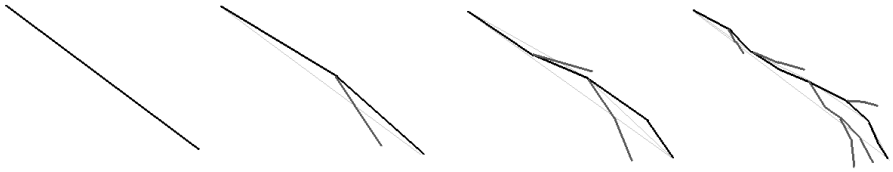

Рис. 2.6: Генерация молнии с побочными ветвями

2.6 Модель освещения Ламберта

Данная модель вычисляет цвет поверхности в зависимости от того как на нее светит источник света. Согласно данной модели, освещенность точки равна произведению силы источника света и косинуса угла, под которым он светит на точку.

$$I_d = k_d cos(L, N) i_d, \tag{2.1}$$

где:

- I_d – рассеянная составляющая освещенности в точке;
- k_d – свойство материалла воспринимать рассеянное освещение;
- i_d – мощность рассеянного освещения;
- L – направление из точки на источник;
- N – вектор нормали.

2.7 Генерация дома

Дом удобнее генерировать с помощью массива точек, ограничивающих сторону дома.

Дом состоит из 5 объектов – 4 стороны и крыша. Они задаются путем задания координат для каждой стороны. Для каждой координаты задается три параметра – координаты X, Y, Z. Высота дома зависит от этажности. После задания данных параметров создаются и накладываются окна.

Каждое окно, как и сторона дома, ограничено массивом точек. Для каждого окна задаются ограничивающие его 4 точки. Задаются они путём задания трёх координат для каждой стороны. В зависимости от количества этажей создаются окна. На каждый этаж приходится по 8 окон.

Также создается громоотвод (молниезащита дома).

2.8 Выбор используемых типов и структур данных

Для разрабатываемого ПО необходимо реализовать следующие типы и структуры данных.

1. Сцена - список объектов.
2. Объекты сцены - набор вершин и граней.
3. Источник света - положение и направление света.
4. Цвет - вектор из трех числе (синий, красный, зеленый).
5. Математические обстракции:
   (a) точка - хранит положение, задается координатами x, y, z;
   (b) вектор - хранит направление, задается x, y, z;
   (c) многоугольник - хранит вершины, нормаль, цвет.

2.9 Вывод

В данном разделе были описаны требования к программе, подробно рассмотрены алгоритмы трассировки лучей, генерации молнии и дома, описаны типы и структуры данных, которые будут реализованы, и приведена схема алгоритма трассировки лучей. Также был указан способ оптимизации алгоритма трассировки лучей, который будет применен в данном проекте для уменьшения времени работы алгоритма.

3 Технологическая часть

В данном разделе представленны средства разработки программного обеспечения, детали реализации и тестирование функций.

3.1 Выбор языка программирования и среды разработки

Существует множество языков, а также сред программирования, многие из которых обладают достаточно высокой эффективностью, удобством и простотой в использовании. Для разработки данной программы был выбран язык C#. Данный выбор обусловлен следующими факторами.

1. Этот язык предоставляет программисту широкие возможности реализации самых разнообразных алгоритмов. Он обладает высокой эффективностью и большим набором стандартных классов и процедур.

2. C# является полностью объектно-ориентированным. Он позволяет использовать множественное наследование, абстрактные и параметризованные классы.

3. Язык является строго типизированным, что позволяет защититься от непроконтролированных ошибок.

4. В данном языке имеется большое количество библиотек и шаблонов, позволяющих не тратить время на изобретение готовых конструкций.

В качестве среды разработки была выбрана Visual Studio 2019. Некоторые факторы по которым была выбрана данная среда.

1. Включает весь основной функционал: параллельная сборка, отладчик, поддержка точек останова, сборки и т.д.

2. Разработчики имеют возможность расширить любой функционал, включая компиляцию, отладку.

3. Работает с интерфейсом Windows Forms, который очень удобен в использовании, а также позволяет без проблем создавать приложения.

3.2 Структура программы

Так как при написании программы используется язык C#, а это объектно-ориентированный язык, то особое внимание уделено структуре классов.

Условно классы в программе можно разделить на несколько групп по выполняемым функциям.

- Математические абстракции
  - Trace - структура, в которую входит точка пересечения луча с объектом расстояние от камеры до точки пересечения
  - Ray - трехмерный луч, задающийся точкой начала луча, направляющим вектором
  - Comparator - помогает сравнивать сегменты молнии
- Вспомогательные классы свойств трехмерных объектов
  - Texture - абстрактный класс с основными свойствами.
  - SimpleTexture и VectorTexture - обеспечивают загрузку из файла текстуры, ее интерпретацию на простую поверхность
- Трехмерные объекты
  - House – реализует работу с домом, генерацию, трехмерные преобразования
  - Lightning – реализует работу с молнией, генерацию, трёхмерные преобразования.
- Источники света
  - Shadow - класс, позволяющий работать с тенями. Одним из ключевых механизмов базового алгоритма тарссировки лучей является метод теневых лучей, позволяющие реализовать тени от загораживающих источники света объектов.
- Сцена

  - Scene - характеризует набор объектов и их свойств. Следует обратить внимание на то, что сцена использует динамические объекты, что позволяет уже после загрузки и построения сцены изменить ее состав (удалять/добавлять/изменять компоненты).

- Алгоритмы визуализации
  - Highlight - отрисовка вспышки
  - Lightning - отрисовка молнии
  - House - отрисовка дома
  - Shadow - отрисовка тени
  - Texture - отрисовка текстуры
  - Window - отрисовка окна

- Интерфейс пользователя
  - Взаимодействие с интерфейсом происходит через диалоговые окна, которые в свою очередь взаимодействуют с классом Scene

На рисунке 3.1 представлена структура и состав классов.

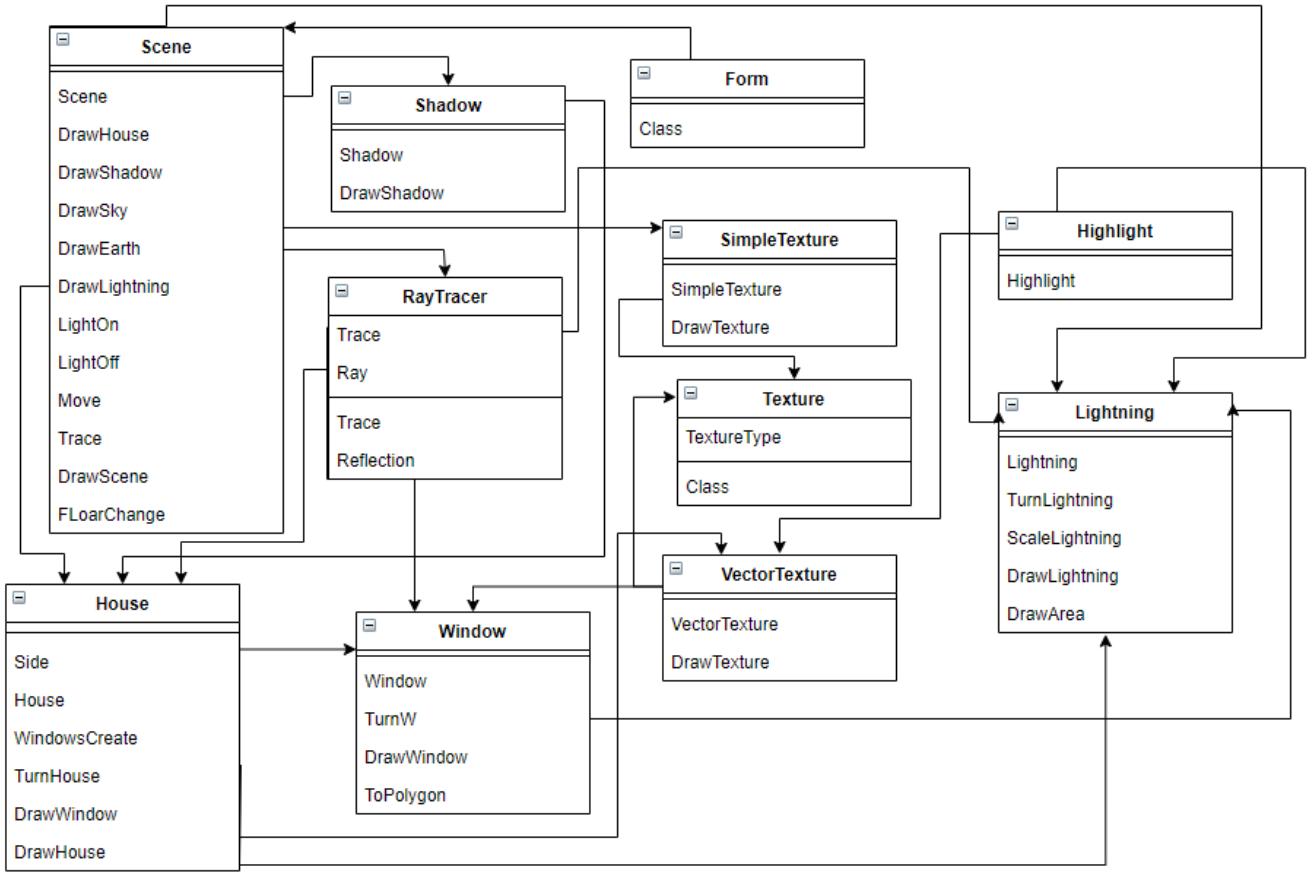

Рис. 3.1: Структура и состав классов

3.3 Разработка алгоритмов

На листинге 3.1 представлен алгоритм обратной трассировки лучей (улучшенный по времени).

```
1  static public List<PointF> Reflection(Window w, double[] nor, Point3D p,
       Lightning lightning)
2  {
3    var po = new Point3D(p.X, p.Y, -200);
4    var no = new double[4];
5    var refl = new List<PointF>();
6    double r;
7
8    float dx = w.Points[0].X - w.Points[3].X;
9    float dy = w.Points[0].Y - w.Points[3].Y;
10   float dz = w.Points[0].Z - w.Points[3].Z;
11
12   float dx2 = w.Points[0].X - w.Points[1].X;
13   float dy2 = w.Points[0].Y - w.Points[1].Y;
14   float dz2 = w.Points[0].Z - w.Points[1].Z;
15
16   double step = 1.0 / Math.Sqrt(dx * dx + dy * dy + dz * dz);
17
18   nor.CopyTo(no, 0);
19   for (int i = 0; i < 4; i++)
20   {
21     no[i] /= 100000;
22   }
23
24   for (r = 0.0; r <= 1; r = r + step)
25   {
26     var x = (float)(w.Points[3].X + r * dx + 0.5);
27     var y = (float)(w.Points[3].Y + r * dy + 0.5);
28     var z = (float)(w.Points[3].Z + r * dz + 0.5);
29     float x2 = x + dx2;
30     float y2 = y + dy2;
31     float z2 = z + dz2;
32
33     float dx3 = x2 - x;
34     float dy3 = y2 - y;
35     float dz3 = z2 - z;
36
37     double step2 = 1.0 / Math.Sqrt(dx3 * dx3 + dy3 * dy3 + dz3 * dz3);
38     double r2;
39
40     for (r2 = 0.0; r2 <= 1; r2 = r2 + step2)
41     {
```

```
42        var x3 = (float)(x - r2 * dx3 + 0.5);
43        var y3 = (float)(y - r2 * dy3 + 0.5);
44        var z3 = (float)(z - r2 * dz3 + 0.5);
45
46        var poi = new Point3D(x3 - po.X, y3 - po.Y, z3 - po.Z);
47        double a = Math.Abs(no[0] * poi.X + no[1] * poi.Y + no[2] * poi.Z) /
48          Math.Sqrt(no[0] * no[0] + no[1] * no[1] + no[2] * no[2]) /
49          Math.Sqrt(poi.X * poi.X + poi.Y * poi.Y + poi.Z * poi.Z);
50
51        a = Math.Asin(a);
52
53        var os = new Point3D((float)(poi.Y * no[2] - poi.Z * no[1]),
54          (float)(no[0] * poi.Z - poi.X * no[2]),
55          (float)(poi.X * no[1] - poi.Y * no[0]));
56
57        var vector = new Point3D();
58
59        double cos = Math.Cos(a), sin = Math.Sin(a);
60
61        vector.X = (float)(no[0] * (cos + (1 - cos) * os.X * os.X) +
62          no[1] * ((1 - cos) * os.Y * os.X + sin * os.Z) +
63          no[2] * ((1 - cos) * os.Z * os.X - sin * os.Y));
64        vector.Y = (float)(no[0] * (-sin * os.Z + (1 - cos) * os.X * os.Y) +
65          no[1] * ((1 - cos) * os.Y * os.Y + cos) +
66          no[2] * ((1 - cos) * os.Z * os.Y + sin * os.X));
67        vector.Z = (float)(no[0] * (sin * os.Y + (1 - cos) * os.X * os.Z) +
68          no[1] * ((1 - cos) * os.Y * os.Z - sin * os.X) +
69          no[2] * ((1 - cos) * os.Z * os.Z + cos));
70
71        var ray = new Ray(vector, new Point3D(x3, y3, z3));
72        Trace t2 = lightning.TraceM(ray);
73
74        if (t2.Dist != -1)
75        {
76          refl.Add(new PointF(x3, y3));
77        }
78      }
79    }
80
81    return refl;
82  }
```

Листинг 3.1: Функция заполнения карты достоверности

3.4 Интерфейс

На рисунке 3.2 представлен интерфейс программы.

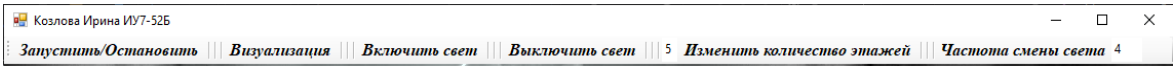

Рис. 3.2: Интерфей программы

Функции представленных кнопок следующие:

- кнопка "Запустить/Оставновить запускает или выключает (останавливает) автоматичускую генерацию молнии, непрерывно генерируются различные изображения;
- кнопка "Визуализация создает одно изображение с указанной частотой включения и выключения света в окнах;
- кнопка "Включить свет включает свет во всех окнах;
- кнопка "Выключить свет выключает свет во всех окнах;
- кнопка "Изменить количество этажей изменяет высоту дома (количество этажей), значение берется из поля слева, при этом генерируется новая сцена.

Также пользователь может взаимодействовать с программой при помощи стрелок вправо/влево на клавиатуре или при помощи указателя мышки. Таким образом будет производиться поворот сцены.

3.5 Тестирование

На рисунке 3.2 представлен интерфейс данном программы, на котором видно, что есть два поля ввода информации, а именно информации про количество этажей и частоту смены света.

В каждом поле стоит ограничение на ввод информации, а именно ввести можно только цифры от 1 до 9 (для лучшей отрисовки сцены).

Частоту смены света можно задать в диапазоне от 1 до 5000, где 1 – очень частая смена света, и 5000 – редкая.

На рисунке 3.3 представлен результат для входных данных: 3- количество этажей, 5000 - частота смена света (работу данного параметра можно посмотреть только при непосредсвенной непрерывной генерации сцены).

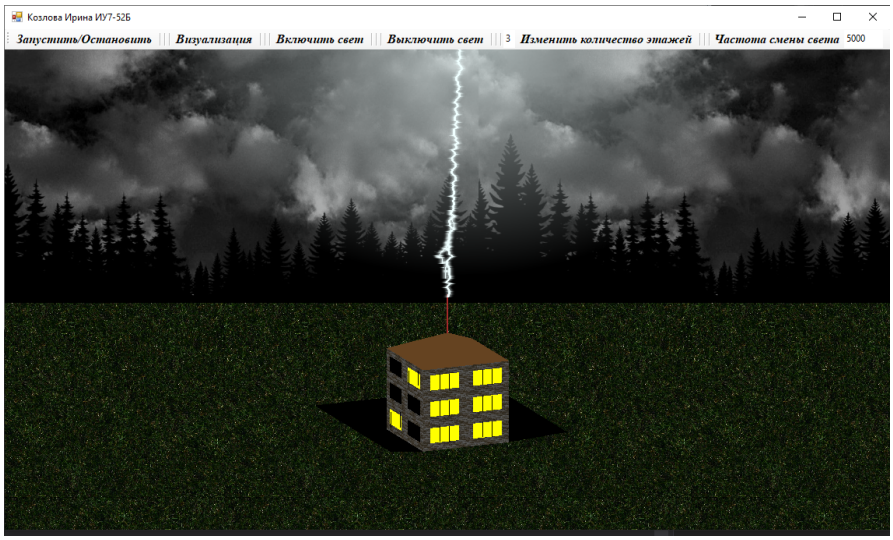

Рис. 3.3: Пример работы программы

3.6 Вывод

В данном разделе были рассмотренны средства реализации программного обеспечения и листинги исходных кодов программного обеспечения, разработанного на основе алгоритма трассировки лучей, изложенного в конструкторской части.

4 Исследовательская часть

В данном разделе будут приведены результаты работы разработанного программного обеспечения и поставлен эксперимент по оценки эффективности работы программы.

4.1 Результаты работы программного обеспечения

На рисунке 4.1 приведен результат генерации сцены, на которой показана молния-лидер.

Рис. 4.1: Изображение с молнией-лидером

На изображении 4.2 приведен результат генерации сцены, на котором вижно отражение молнии от стекл окон дома.

Рис. 4.2: Изображение с отражением молнии от стекл окон дома

На изображении 4.3 приведен результат генерации сцены, на которой покзана молния, бьющая в громоотвод.

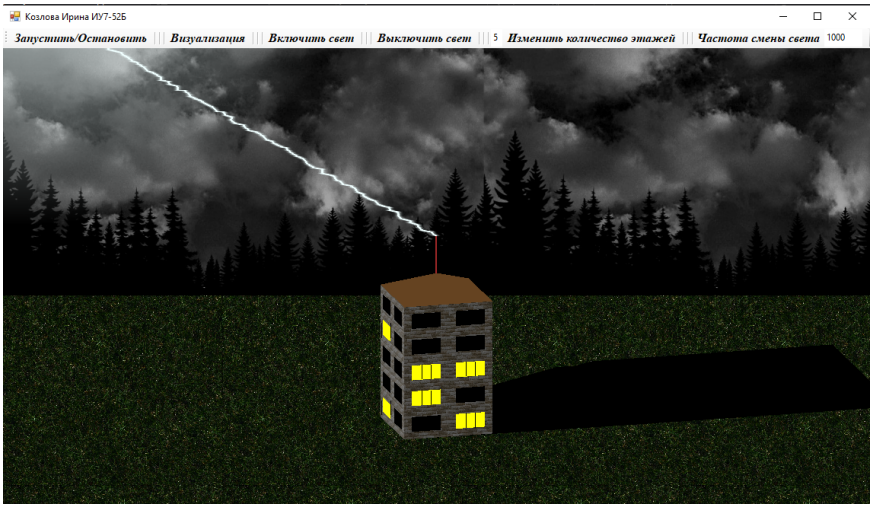

Рис. 4.3: Изображение с молнией, бьющей в громоотвод

На изображении 4.4 приведен результат генерации сцены, на которой покзана молния с большим количеством побочных ветвей.

Рис. 4.4: Изображение с молнией, у которой большое количество ветвей

4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование, следующие.

- Операционная система: Windows 10 [?] x86_64.
- Память: 8 GiB.
- Процессор: 11th Gen Intel® Core™ i5-1135G7 @ 2.40GHz [?].
- 4 физических ядра и 8 логических ядра.

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

4.3 Постановка эксперимента

4.3.1 Цель эксперимента

Целью эксперимента является попытка оптимизировать алгоритм обратной трассировки лучей для данной сцены.

Необходимо провести теоретическое сравнение оптимизированного и неоптимизированного алгоритма, а также подтвердить результаты на практике путем подсчета колличесва испускаемых лучей для построения сцены.

Эксперимент будет проводиться на различных сценах, с разным количесвом этажей, а также с разными видами молнии.

1. Количество этажей – 6, молния без побочных ветвей бьет в громоотвод, количество окон, в которых горит цвет – не имеет значения.

2. Количество этажей – максимальное, молния-лидер с большим количеством ветвей бьет в земплю, во всех окнах не горит свет.

3. Количество этажей – минимальное, молния без ветвей бьет в громоотвод, во всех окнах горит свет.

4. Количество этажей – 5, молния-лидер бьет в землю, количество окон, в которых горит свет – не имеет значение.

5. Количество этажей – макисмальное, молния-лидер бьет в землю, во всех окнах горит свет.

Также будет приведены:

- таблица и график зависимости испускаемых лучей от отношения количества окон, в которых не горит свет, к общему количеству окон;

- таблица и график зависимости времени построения от количества испускаемых лучей.

4.3.2 Сравнение алгоритма трассировки лучей

4.3.3 Сравнение характера шума

Из-за того что в квантовом случае пиксели получаемого изображения могут быть либо идеальными, либо крайне зашумленными [**?**], сравнение характера шума изображения можно провести подсчитав процент идеальный пикселей от всего изображения.

В таблице 4.1 приведено сравнение характера шума для разных размеров изображения. Размер сабпикселя равен 4x4.

Таблица 4.1: Сравнение характера шума классического и квантового алгоритма избыточной выборки.

Размер изображения	Классическая выборка	Квантовая выборка
64x64	58%	88%
128x128	71%	78%
256x256	63%	75%
512x512	62%	73%
1024x1024	61%	70%

4.3.4 Сравнение вычислительной сложности алгоритмов

Для того чтобы оценить время выполнения алгоритмов, нужно вывести их сложность. Вычислительная сложность алгоритма Монте – Карло напрямую зависит от размера изображения и составляет $O(n)$ [**?**]. На основе изложенного алгоритма в разделе **??**, выведем сложность выполнения квантового алгоритма избыточной выборки. При этом, примем что все квантовые операции выполняются за $O(1)$.

Опираясь на описание квантового алгоритм избыточной выборки из раздела **??**, построчно рассчитаем его сложность:

- перевести все пиксели холста в состояние 1 – O(1) (реализуется с помощью операции смены фазы, см. раздел **??**);

- сформировать и заполнить квантовую поисковую таблицу – $O(n * k)$, где k – глубина итераций усиления комплексной амплитуды. Так как k – константа (см. раздел **??**), мы имеем право отбросить эту константу [**?**], получая конечную сложность формирования таблицы $O(n)$;

- посчитать количество инвертируемых пикселей (для каждого пикселя холста) – $O(n^2)$;

- заполнить соответствующую ячейку карты достоверности (для всей таблицы) – $O(n^2)$.

Итоговая вычислительная сложность квантового алгоритма избыточной выборки составляет (4.1):

$$c * (O(1) + O(n) + O(n^2) + O(n^2)) = O(n^2) \tag{4.1}$$

где c – число цветов колоризации. Так как это число является константой (см. раздел **??**), в конечном счете мы можем откинуть его [**?**].

Вывод

Как и ожидалось, уровень зашумленности изображения примерно равен как для классической выборки, так и для квантовой. Так, например, средний процент ошибки на пиксель при размере синтезируемого изображения 512x512 для квантового алгоритма составляет 3%, а для классического 4% соотвественно.

Квантовая выборка на маленьких размерах изображения выдает большой процент идеальных пикселей. Например, для изображения размером 64x64 количество идеальных пикселей составляет 88% от всего изображения. Но, с увеличением изображения этот процент незначительно падает, и уже на размере изображения 512x512 составляет 73%. Несмотря на это,

классическая выборка даже в самом лучшем случае не добивается такого результата – ее лучший результат 71% идеальных пикселей для размера изображения 128x128. Квантовый алгоритм генерирует на 10%..30% идеальных пикселей больше, чем его классический аналог, из чего можно сделать вывод о сильных различиях в характере шума.

Была оценена вычислительная сложность квантового алгоритма избыточной выборки ($O(n^2)$). Исходя из того факта, что сложность выборки Монте – Карло составляет $O(n)$, можно сделать что квантовый алгоритм будет проигрывать по времени работы классическому аналогу даже на настоящем квантовом компьютере.