Содержание

Введение

В современном мире компьютерная графика все больше проникает в обычную человеческую жизнь и используется в различных сферах. Типичные области ее применения – кинематография, компьютерные игры, наглядное отображение различных данных, а также моделирование экспериментов.

Одним из быстро развивающихся направлений компьютерной графики является моделирование и визуализация реалистичного трехмерного изображения. Для удовлетворения растущих потребностей в скорости синтеза и реалистичности полученного изображения разрабатываются новые алгоритмы и совершенствуются уже существующие.

Целью данной курсовой работы является обоснование выбора алгоритмов, которые можно использовать для получения трехмерных моделей растений с помощью фракталов, их практическая реализация и адаптация (при необходимости) к условиям решаемой задачи.

1 Аналитический раздел

В данном разделе представлено описание объектов сцены, а также обоснован выбор алгоритмов, которые будут использованы для ее визуализации.

1.1 Описание объектов сцены

Сцена состоит из одного или нескольких источников света, а также модели растения.

Источник света представляется в виде материальной точки, которая испускает лучи во все стороны. Если он устанавливается на «бесконечном» удалении от сцены, на сцену попадает множество параллельных лучей. В программе источники света будут играть роль студийных проекторов, однако визуализировать их не планируется.

Модель растения будет задана набором правил, описывающих в математическом виде процесс построения этого растения, также цветом и формой примитива, которым нужно отрисовать растение.

1.2 Анализ и выбор формы задания трехмерных моделей

Модель является отображением формы и размеров объекта. В основном используются три вида моделей:

1. Каркасная (проволочная) модель.

   Такой тип модели использует информацию о вершинах и ребрах объектов. Это простейшая форма задания модели, так как мы храним минимум информации. Недостаток такого подхода состоит в том, что модель не всегда точно передает представление о форме объекта.

2. Поверхностная модель.

   Наиболее часто используемый тип моделей в компьютерной графике. Поверхности можно задавать разными способами: либо аналитически, либо задавать участки поверхности, как поверхность того или иного вида (использовать полигональную аппроксимацию). Недостаток - мы не знаем, с какой стороны находится материал.

3. Объемная (твердотельная) модель.

   При твердотельном моделировании учитывается еще материал, из которого изготовлен объект. То есть у нас есть информация о том, с какой стороны поверхности расположен материал. Это делается с помощью указания направления внутренней нормали.

Вывод

Для решения данной задачи были выбраны поверхностные модели, так как у каркасных моделей есть серьезный недостаток – неправильное восприятие формы, а объемные модели слишком информативны и ресурсоемки.

1.3 Анализ способа задания поверхностных моделей

На следующем шаге необходимо определиться со способом задания поверхностной модели.

1.3.1 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.

1.3.2 Полигональная сетка

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

Возможные 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.

Вывод

Стоит отметить, что одним из решающих факторов в выборе способа задания модели в данном проекте является скорость выполнения преобра-

зований над объектами сцены. Поэтому при реализации программного продукта в данной работе наиболее удобным представлением является модель, заданная полигональной сеткой – это поможет избежать проблем при описании сложных моделей. При этом способ хранения полигональной сетки – это списком полигонов из трех вершин, что поможет при реализации алгоритма удаления невидимых рёбер и поверхностей. Этот способ позволит эффективно преобразовывать модели, так как структура будет включать в себя список вершин.

1.4 Анализ и выбор алгоритма удаления невидимых ребер и поверхностей

Выбирая алгоритм удаления невидимых ребер и поверхностей, в первую очередь нужно выделить свойства, которыми должен обладать итоговый алгоритм, чтобы обеспечить оптимальную работу программы и реалистичное изображение. При анализе были выделены следующие свойства:

- алгоритм должен быть достаточно быстрым;
- алгоритм должен использовать как можно меньше памяти;
- алгоритм должен иметь высокую реалистичность изображения.

1.4.1 Алгоритм, использующий Z-буфер

Суть алгоритма Z-буфера [1] – 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.

Преимуществами данного алгоритма являются простота реализации, а также линейная оценка трудоемкости.

Недостатки алгоритма – большой объем требуемой памяти и сложная реализация прозрачности.

1.4.2 Алгоритм обратной трассировки лучей

Суть данного алгоритма – 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 лучей в обратном направлении, то есть от наблюдателя к объекту.

Метод обратной трассировки лучей позволяет значительно сократить перебор световых лучей. В этом случае трассируется определенное число лучей, равное разрешению картинки [2].

Преимуществами данного алгоритма являются высокая реалистичность синтезируемого изображения и возможность работать с поверхностями в математической форме. Также важно отметить, что вычислительная сложность данного алгоитма слабо зависит от сложности сцены.

Основным недостатком алгоритма является низкая производительность.

1.4.3 Алгоритм Робертса

Данный алгоритм работает в объектном пространстве, решая задачу только с выпуклыми телами. [3]

Алгоритм выполняется в три этапа:

1. Этап подготовки исходных данных (информации о телах).
2. Этап удаления ребер, экранируемых самим телом.
3. Этап удаления невидимых ребер, экранируемых другими телами сцены.

Преимущество алгоритма Робертса в том, что он целиком основан на математических предпосылках, которые просты, точны и мощны. Он работает в объектном пространстве и позволяет получить высокую точность вычислений.

К недостаткам этого алгоритма можно отнести большую трудоемкость, невозможность работы с невыпуклыми телами, а также тот факт, что без модификации и привлечения сторонних методов данный алгоритм не позволяет учитывать тени и зеркальные эффекты.

1.4.4 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 [4].

Основным преимущетсвом данного алгоритма является тот факт, что он требует меньше памяти, чем, например, алгоритм Z-буфера.

Недостаки – недостаточно высокая реалистичность изображения и сложность реализации при пересечении граней на сцене.

1.4.5 Алгоритм Варнока

Алгоритм Варнока [5] работает в пространстве изображения и основывается на рекурсивном разбиении экрана. Главная идея - на каждом шаге найти ответ на вопрос о том, что изображать в очередном окне. Если нельзя точно дать ответ, то окно делится на части, пока не сможем решить, что изображать, или окно не дойдёт до размеров в один пиксель.

В простейшей версии алгоритма окно делится на подокна всякий раз, если это окно не пусто. В более сложных версиях делается попытка решения задачи для окон большего размера. Для этого проводится классификация многоугольников по отношению к ячейке: внешний, внутренний, пересекающий или охватывающий. Затем определяются действия, которые нужно предпринять в том или ином случае взаимного расположения ячейки и полигонов.

Достоинством данного алгоритма является простота реализации и высокая эффективность в случае, если размеры перекрываемых областей невелики.

К недостаткам алгоритма относится трудоемкость в случае, когда синтезируемая сцена сложная и число разбиений становится очень большим, а также отсутствие учета оптических свойств объектов.

Вывод

Для удаления невидимых линий был выбран алгоритм Z-буфера. Данный алгоритм работает достаточно быстро из-за отсутствия сортировок, позволяет добиться хорошей реалистичности, а также достаточно прост в реализации

1.5 Анализ и выбор модели освещения

Существует два вида моделей материалов: физические модели и эмпирические.

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

Эмпирические модели материалов устроены иначе. Подобные модели подразумевают некий набор параметров, не имеющих физической интерпретации, но позволяющих с помощью подбора получить нужный вид модели.

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

1.5.1 Модель Ламберта

Модель Ламберта [6] 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.

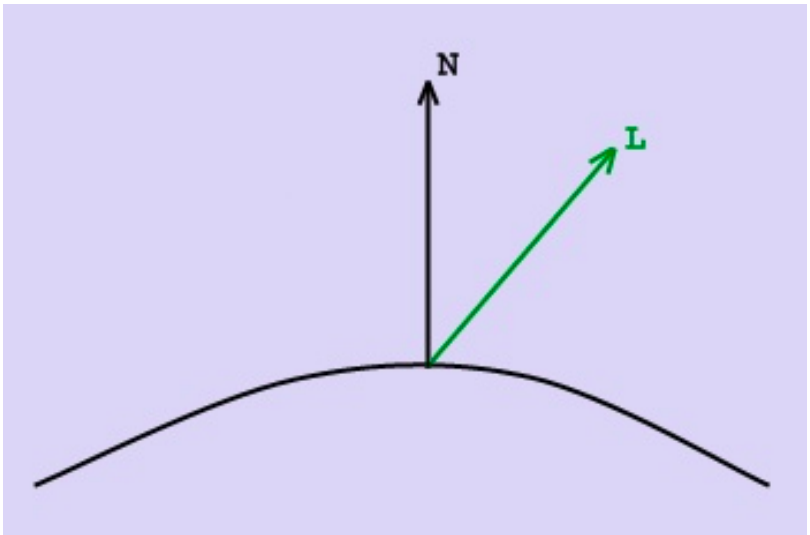

Рисунок 1.1 – Направленность источника света

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.

1.5.2 Модель Фонга

Модель Фонга [6] представляет собой комбинацию диффузной и зеркальной составляющих. Падающий и отраженный лучи лежат в одной плоскости с нормалью к отражающей поверхности в точке падения (1.2). Нормаль делит угол между лучами на две равные части. L – направление источника света, R – направление отраженного луча, V – направление на наблюдателя.

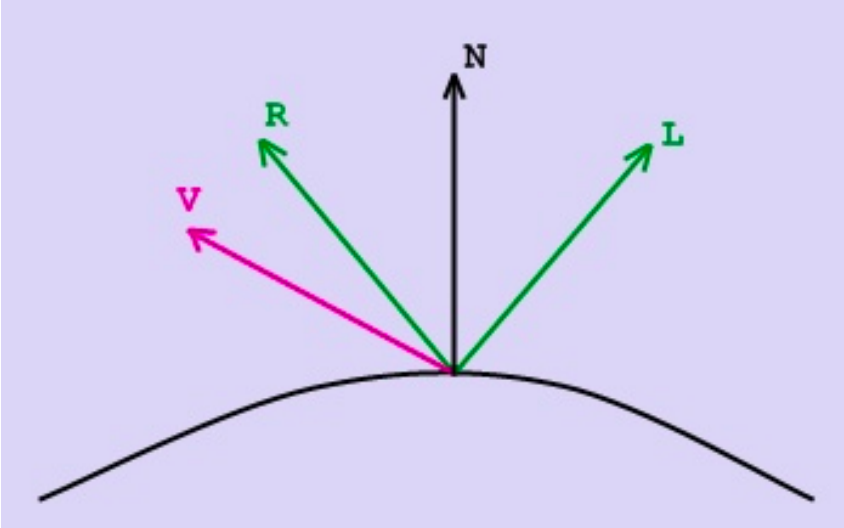

Рисунок 1.2 – Направленность источника света

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.

Вывод

Для освещения была выбрана модель Ламберта, так как по сравнению с моделью Фонга она требует меньше вычислительных затрат, и, следовательно, работает быстрее.

Вывод

В данном разделе был проведен анализ алгоритмов удаления невидимых линий и модели освещения, которые возможно использовать для решения поставленных задач. В качестве ключевого алгоритма удаления невидимых граней выбран алгоритм Z-буфера, а в качестве модели освещения - модель освещения Ламберта.

2 Конструкторский раздел

В данном разделе были рассмотрены требования к программе и алгоритмы визуализации модели растения.

2.1 Общий алгоритм решения поставленной задачи

Алгоритм решения поставленной задачи выглядит следующим образом:

1. Задать набор правил для построения модели растения.
2. Рассчитать параметры поверхностей отдельных частей растения согласно установленным правилам.
3. Соединить полученные поверхности в единую модель.
4. Изобразить модель растения на сцене.

2.2 Разработка алгоритма Z-буффера

Формальное описание алгоритма:

1. Заполнить буфер кадра фоновым значением интенсивности.
2. Заполнить z-буфер минимальным значением z.
3. Для каждого пиксела (x, y) грани вычислить его глубину $z(x, y)$ и сравнить вычисленное значение со значением $z-$буфер(x, y), хранящимся в z-буфере в этой же позиции.
4. Если $z(x, y) > z - (x, y)$, то записать атрибут этого многоугольника (интенсивность, цвет и т.п.) в буфер кадра и заменить $z-$буфер(x, y) на $z(x, y)$.

Схема алгоритма представлена на рисунке 2.1 соответственно.

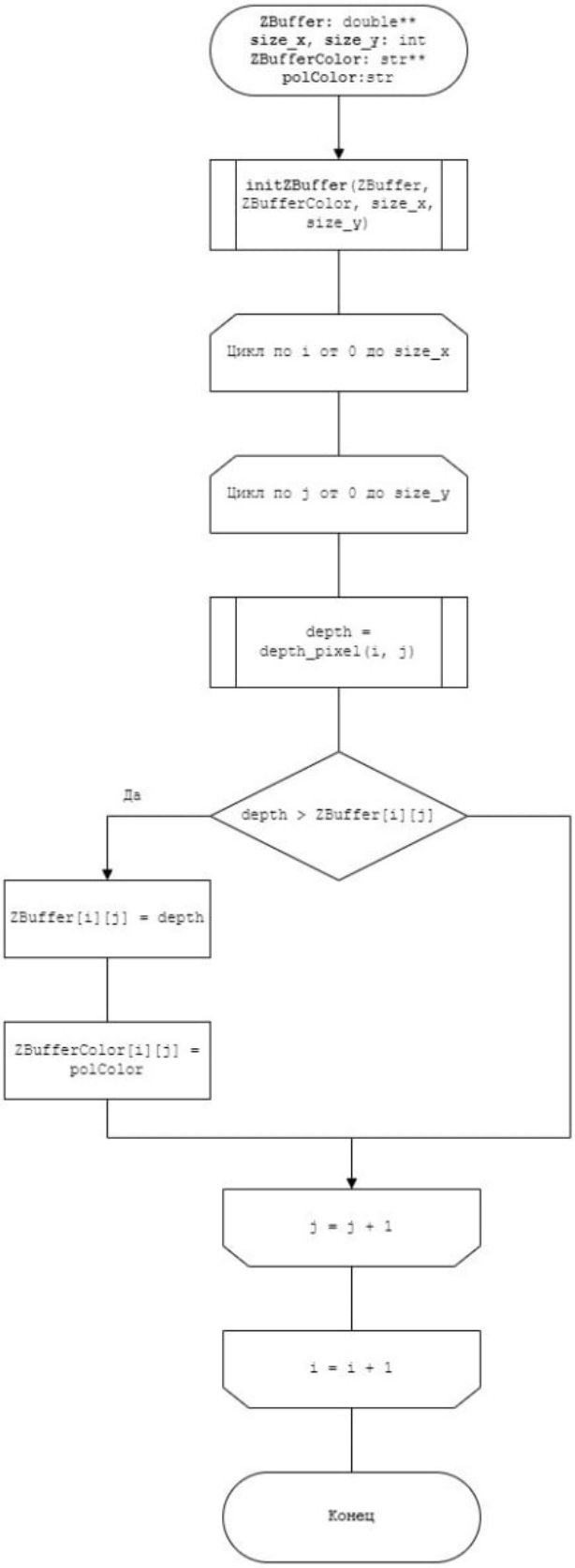

Рисунок 2.1 – Схема алгоритма Z-буфера

2.3 Разработка алгоритма 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.

Схема алгоритма простой закраски, основанного на принципах модели освещения Ламберта, представлена на рисунке 2.2.

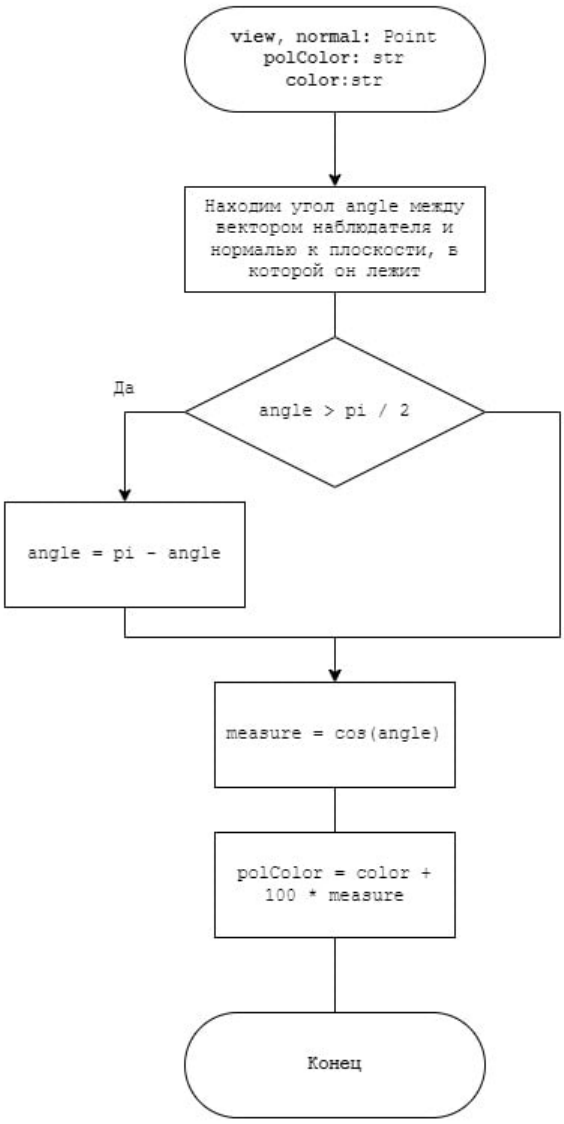

Рисунок 2.2 – Схема алгоритма простой закраски

2.4 Разработка алгоритма генерации растения

Для генерации моделей растений будем использовать L-системы.

Lindenmayer System или просто L-system [7] – это математическая модель, предложенная в 1968 году венгерским биологом и ботаником Аристидом Линденмайером, для изучения развития простых многоклеточных организмов.

Позже эта модель была расширена и стала использоваться для моделирования сложных ветвящихся структур – разнообразных деревьев и цветов.

Основная идея L-систем – постоянная перезапись элементов строки (rewriting). Другими словами, это способ получения сложных объектов путем замены частей простого начального объекта по некоторым правилам.

Подобный подход удобен для описания растений, так как они обладают свойствами рекурсивности и самоподобия при росте. Например, маленькие листочки, являющиеся частью большого взрослого составного листа, имеют ту же форму, что весь лист имел на раннем этапе формирования.

Язык L-систем очень прост, он состоит из символов (алфавита) и продукционных правил. Первое состояние предложения называется аксиомой. К этой аксиоме можно применить продукционные правила для эволюции или роста системы. Например, если у нас есть система с аксиомой A и единственным правилом A -> ABB, то после первой итерации предложение сменится на ABB, потом на $ABBBB$ и т.д.

2.4.1 Классификация L-систем

1. Детерминированные контекстно-свободные L-системы.

   Это самый простой вид L-систем. Пример такой системы представлен на рисунке 2.3 (при построении использовались следующие параметры: аксиома F, правило $F : F[-F]F[+F][F]$, количество итераций 3, угол поворота 30).

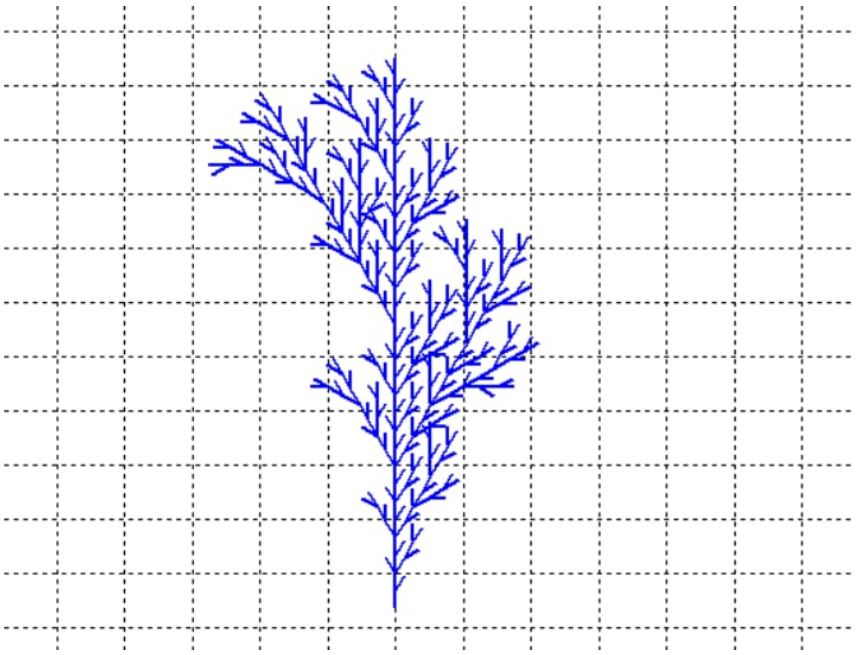

Рисунок 2.3 – Пример простой L-системы

2. Параметрические L-системы.

   Каждому символу добавляем переменную (возможно не одну), которая позволяет указывать некоторые параметры для этого символа (возможно величину угла поворота, длину шага, толщину линии и т.п.) Пример такой системы представлен на рисунке 2.4 (параметры системы: аксиома $BA(0,0)$, правила $A(x,y)x < y : BA(x+1,y)$ и $A(x,y)x >= y : +BA(0,y+1)$, количество итераций 118, угол поворота 90).

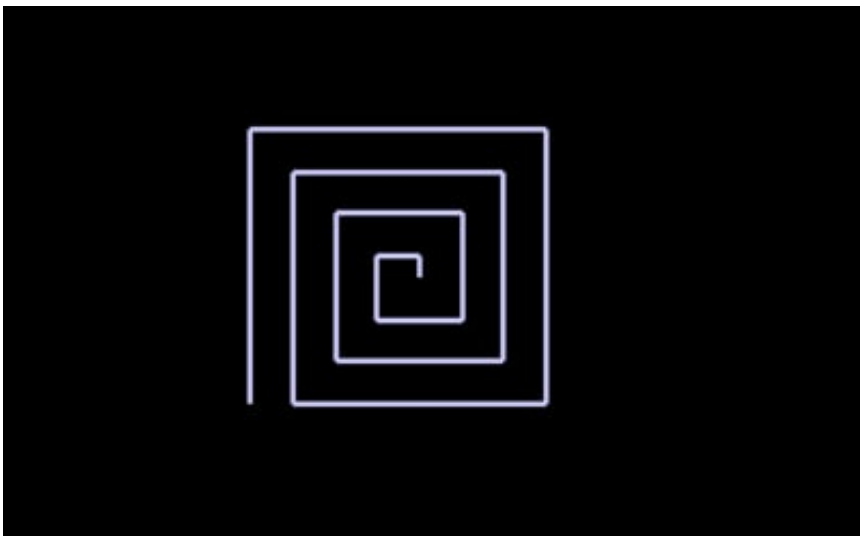

Рисунок 2.4 – Пример параметрической L-системы

3. Контекстно-зависимые L-системы.

   Такие системы принимают во внимание окружение заменяемого символа.

4. Стохастические L-системы.

   Представляют собой простые системы с возможностью задания вероятности выполнения того или иного правила. Подобный подход вносит некоторый элемент случайности в получающиеся структуры.

2.4.2 Алгоритм визуализации L-системы

Для визуализации предложения используется система рендеринга, называемая “черепашьей графикой” [8].

Черепашья графика характеризуется размещением в декартовой системе “черепашки” и передачей ей инструкций. Черепашка движется в соответствии с полученными ею инструкциями и оставляет за собой след. В нашем случае черепашке отправляется каждый символ из предложения L-системы. Ключ к черепашьему языку представлен в таблице 4.1.

Таблица 2.1 – Алфавит для отрисовки трехмерной системы Линденмайера

[A..Z]	Любая (не являющаяся константой буква алфавита перемещает черепашку вперед на фиксированное расстояние, перемещаясь черепашка рисует линию)
[a..z]	Переместить черепашку на фиксированное число шагов, не рисуя линию
+	Поворот черепашки вправо на фиксированный угол
-	Поворот черепашки влево на фиксированный угол
/	Крен черепашки вправо на фиксированный угол
\	Крен черепашки влево на фиксированный угол
^	Тангаж черепашки вверх на фиксированный угол
_	Тангаж черепашки вниз на фиксированный угол
[	Запись текущего состояния черепашки в стек
]	Извлечение состояния черепашки из стека и присвоение этого состояния
~	Нарисовать поверхность, идентифицированную модулем сразу после ~в текущем местоположении и ориентации черепахи.
{	Создать пустой полигон и положить его в стек полигонов (для рисования поверхностей)
G	Переместить черепашку вперед на фиксированное расстояние и провести линию, не добавляя вершину к текущему многоугольнику.
g	Переместить черепашку вперед, не проводя линию и не добавляя вершину к текущему многоугольнику.
.	Добавить положение черепахи к текущему многоугольнику
}	Извлечь текущий полигон из стека полигонов и нарисовать его, используя указанные вершины.

2.4.3 Итоговый алгоритм генерации

Таким образом в итоговом алгоритме генерации модели растения можно выделить следующие шаги:

1. Получить параметры растения, представленного в виде L-системы (аксиому, набор продукционных правил, количество итераций роста, начальные данные, длину шага и т.д.)

2. Используя количество итераций, аксиому и правила получить предложение, по которому будет строиться итоговое растение.

3. Если в предложении есть вхождения некоторых поверхностей, убедится, что их представления заданы в программе или сгенерировать их.

4. “Прочитать” предложение с помощью черепашки и таблицырасшифровки черепашьего языка.

5. Получить итоговую модель растения, которую уже можно будет выводить на экран.

Вывод

В данном разделе были подробно рассмотрены алгоритмы, которые будут реализованы, и приведены схемы алгоритмов Z-буфера и простой модели освещения, указан способ генерации модели растения.

3 Технологический раздел

В этом разделе будет обоснован выбор языка программирования и среды разработки, рассмотрена диаграмма основных классов и разобран интерфейс, предлагаемый пользователю.

3.1 Требования к вводу

На вход программы подается набор параметров, описывающих L-систему, визуальной интерпретацией которой является модель растения. В этот набор параметров входят стиль отрисовки, цвет, аксиома, угол поворота на каждой итерации, количество итераций, правила преобразования аксиомы.

3.2 Требования к программе

Программа должна иметь несколько уже готовых наборов "исходных данных". При этом программа должна предоставлять следующие возможности:

- визуальное отображение сцены;
- изменение параметров L-системы, описывающей растение;
- добавление источника света;
- поворот, масштабирование, перемещение исходной сцены;
- очистка сцены.

Программа не должна аварийно завершаться при некорректном вводе. Вывод программы - визуальное отображение модели растения, заданной пользователем, на сцене.

3.3 Выбор языка программирования и среды разработки

Для реализации данной программы был выбран язык программирования C++ [9]. На этот выбор повлияли следующие факторы:

1. Этот язык предоставляет программисту широкие возможности реализации самых разнообразных алгоритмов.
2. C++ обладает высокой эффективностью.
3. Язык является строго типизированным, что позволяет защититься от неконтролируемых ошибок.
4. В данном языке имеется большое количество библиотек и шаблонов, позволяющих не тратить время на изобретение готовых конструкций.

В качестве среды разработки был выбран Clion 2022 [10]. Некоторые факторы по которым была выбрана данная среда:

1. Включает весь основной функционал: параллельная сборка, отладчик, поддержка точек останова, сборки и т.д.
2. Имеет удобный редактор кода со всеми полезными функциями: подсветкой синтаксиса, автоматическим форматированием, дополнением и отступами.
3. Имеет удобный интерфейс для работы с git.
4. Данная среда разработки бесплатна для студентов.

3.4 Структура классов программы

В программе можно выделить следующие классы:

- Object – абстракция базового объекта;
- Vertex – абстракция вершины;
- Triangle – формализация связи трех вершин;
- Plant – класс, реализующий работу с моделью растения;
- PlantGenerator – класс, отвечающий за генерацию модели растения;
- LightSource – абстракция источника света;
- Scene - класс для работы со сценой;
- InputForm – класс для взаимодействия с пользователем;
- MainWindow – основной класс для работы с приложением;

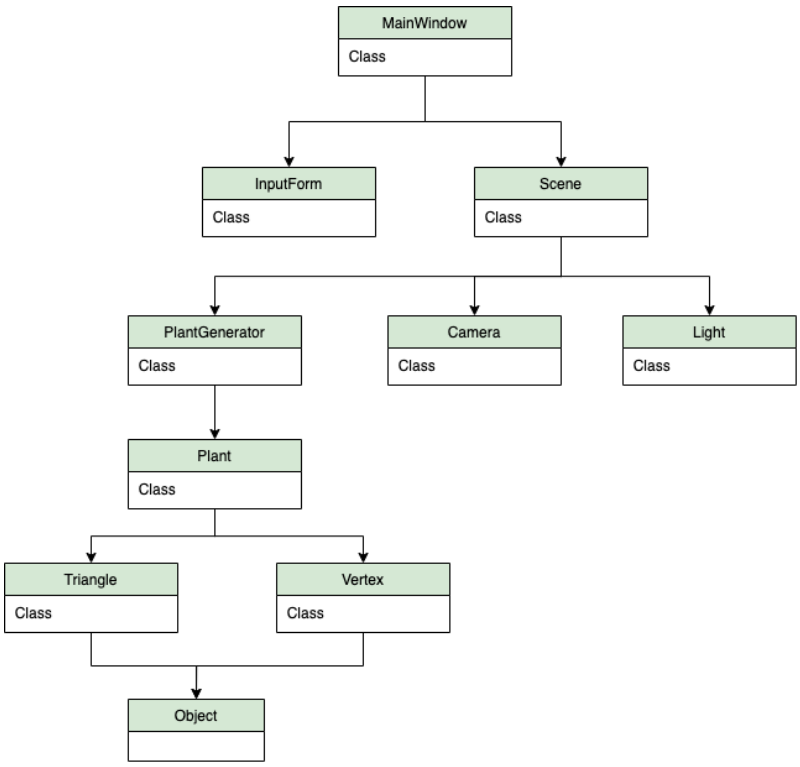

Рисунок 3.1 – Диаграмма классов

3.5 Интерфейс

Интерфейс предоставляет пользователю такие возможности, как добавление модели растения, изменение параметров модели, добавление источника света, поворот сцены, смещение сцены, масштабирование сцены. На рисунке 3.2 представлен интерфейс программы.

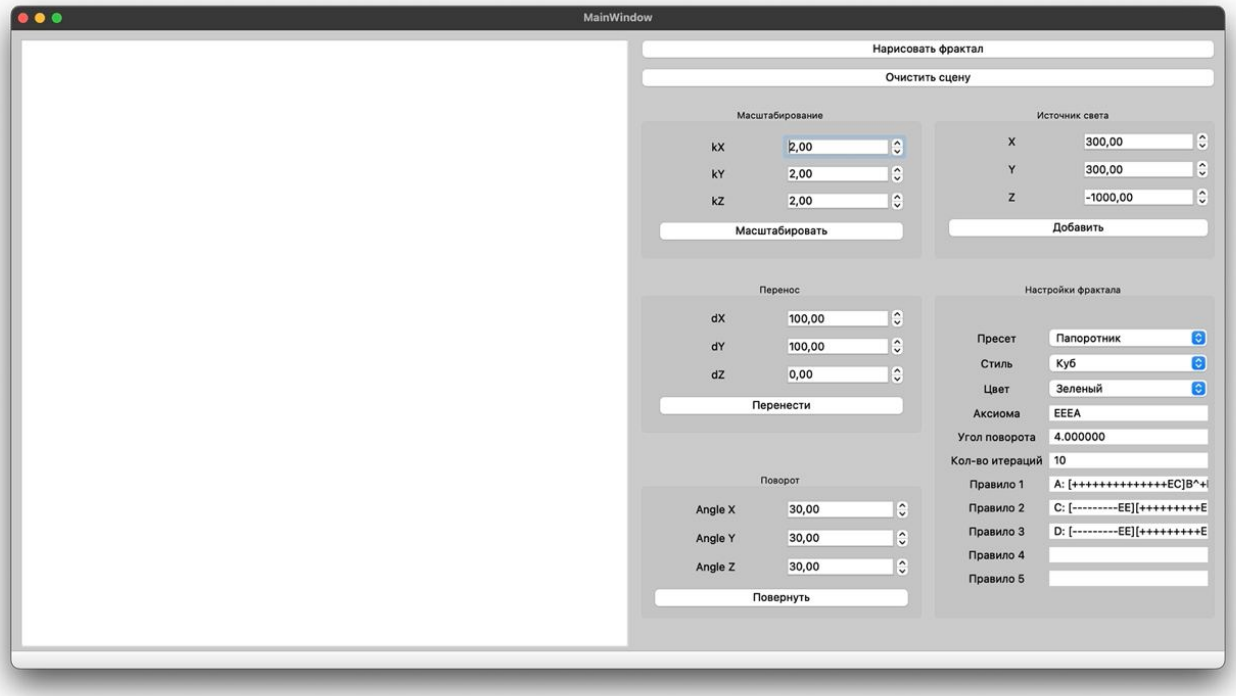

Рисунок 3.2 – Интерфейс программы

3.6 Результаты работы программного обеспечения

На рисунке 3.3 приведен результат генерации модели папоротника.

На рисунке 3.4 приведен результат генерации модели синей водоросли.

На рисунке 3.5 приведен результат генерации модели вербены.

На рисунке 3.6 приведен результат генерации модели растениеподобной структуры.

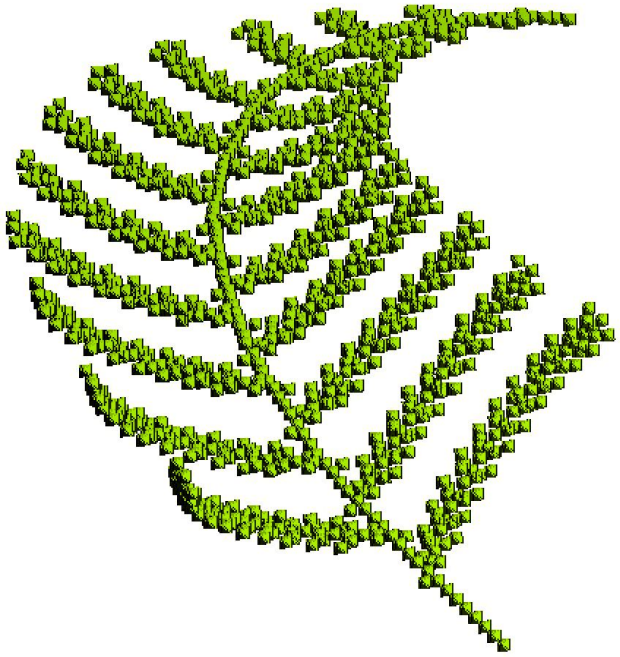

Рисунок 3.3 – Модель папоротника

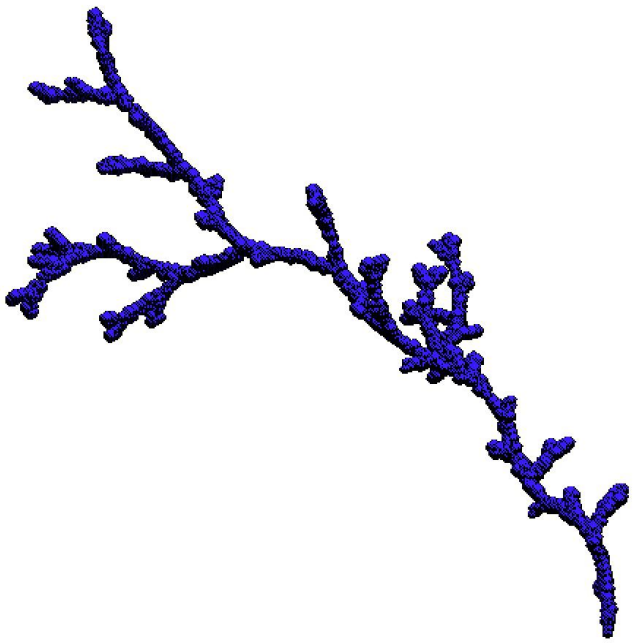

Рисунок 3.4 – Модель синей водоросли

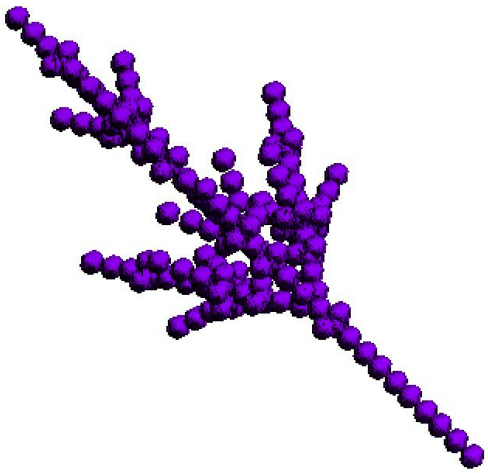

Рисунок 3.5 – Модель вербены

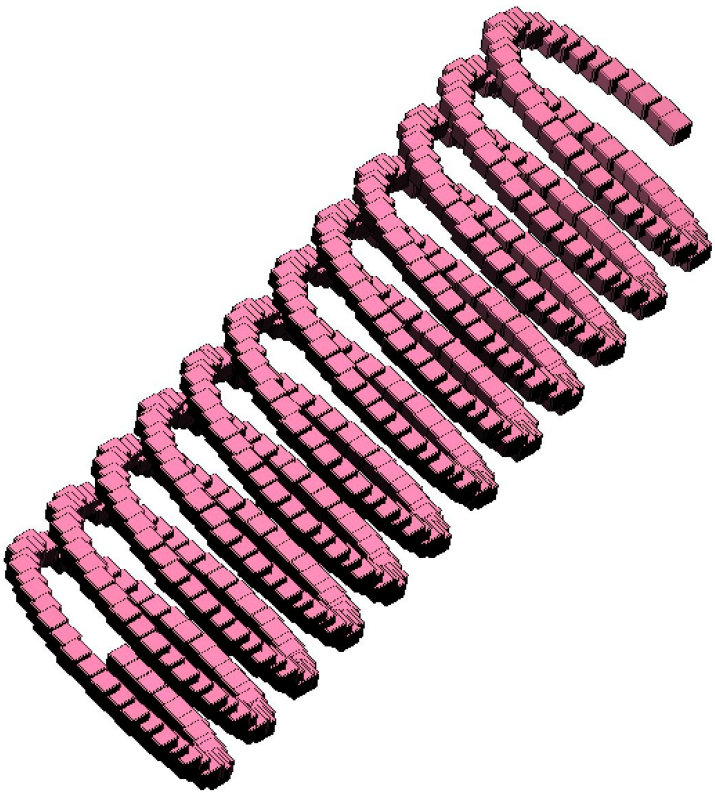

Рисунок 3.6 – Модель растениеподобной структуры

Вывод

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

4 Экспериментальный раздел

В данном разделе проведен эксперимент по сравнению времени работы алгоритма Z-буфера при его последовательной и параллельной реализациях.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- Операционная система macOS Monterey 12.5.1
- Память 16 Гб.
- Процессор 2,3 ГГц 4-ядерный процессор Intel Core i5.

Во время тестирования устройство было подключено к сети электропитания, нагружено приложениями окружения и самой системой тестирования.

4.2 Время выполнения алгоритмов

Для замера процессорного времени использовалась функция *std::chrono::system_ clock::now(...)* из библиотеки *chrono* [11] на C++. Функция возвращает процессорное время типа float в секундах.

Контрольная точка возвращаемого значения не определена, поэтому допустима только разница между результатами последовательных вызовов.

Замеры времени для каждой длины входного массива полигонов проводились 1000 раз. В качестве результата взято среднее время работы алгоритма на данной длине. При каждом запуске алгоритма, на вход подавались случайно сгенерированные массивы полигонов.

4.2.1 Время выполнения однопоточной реализации

Результаты замеров приведены в таблице 4.1.

Таблица 4.1 – Результаты замеров времени однопоточной и последовательной реализаций в микросекундах (N - количество полигонов)

N	Однопоточное выполнение	Последовательное выполнение
10	30	28
20	35	31
30	40	36
50	43	39
100	57	49
200	75	69

4.2.2 Время выполнения многопоточной реализации

Результаты замеров приведены в таблице 4.2.

Таблица 4.2 – Результаты замеров времени выполнения программы, реализующей многопоточный алгоритм удаления невидимых граней, использующий Z-буфер, для разного количества потоков в микросекундах.

Количество треугольников	Количество потоков					
	1	4	6	12	24	48
10	30	33	25	23	21	27
100	57	75	76	73	53	0,78
1000	245	143	118	103	99	128
10000	2098	845	688	603	561	642
100000	19935	5606	5231	4689	4487	4992

При 6 потоках достигается пик, при котором все логические ядра процессора одновременно выполняют параллельные ветки алгоритма. Далее при увеличении числа потоков производительность падает. Это объясняется тем, что создается очередь потоков, которая замедляет работу программы.

Вывод

В данном разделе было произведено сравнение времени выполнения реализации алгоритма удаления невидимых ребер, при последовательной реализации и многопоточной. Результат показал, что выгоднее всего по времени использовать столько потоков, сколько у процессора логических ядер.

Заключение

В рамках данного курсового проекта были:

- описаны структуры трехмерной сцены, включая объекты, из которых состоит сцена;
- проанализированы и выбраны необходимые существующие алгоритмы для построения сцены;
- реализованы выбранные алгоритмы;
- разработано программное обеспечение, которое позволит отобразить трехмерную сцену и визуализировать модель растения;
- проведены сравнение однопоточного и многопоточного оптимизированного алгоритма удаления невидимых линий.

В ходе выполнения эксперимента было установлено, что многопоточная реализация алгоритма удаления невидимых граней, использующего Z-буфер, может показывать меньшее время выполнения, чем однопоточная или линейная реализация. Было выявлено, что выгоднее всего по времени использовать столько потоков, сколько у процессора логических ядер.

Список использованных источников

[1] Алгоритм Z-буфера. [Электронный ресурс]. Режим доступа: http://compgraph.tpu.ru/Oglavlenie.htm (дата обращения: 15.11.2022).

[2] Трассировка лучей из книги Джефа Проузиса [Электронный ресурс]. Режим доступа: https://www.graphicon.ru/oldgr/courses/cg99/notes/lect12/prouzis/raytrace.htm (дата обращения: 15.11.2022).

[3] Алгоритм Робертса. [Электронный ресурс]. Режим доступа: http://compgraph.tpu.ru/Oglavlenie.htm (дата обращения: 15.11.2022).

[4] Алгоритм, использующие список приоритетов (алгоритм художника). [Электронный ресурс]. Режим доступа: http://compgraph.tpu.ru/Oglavlenie.htm (дата обращения: 15.11.2022).

[5] Алгоритм Варнока. [Электронный ресурс]. Режим доступа: http://compgraph.tpu.ru/Oglavlenie.htm (дата обращения: 15.11.2022).

[6] Модели освещения. [Электронный ресурс]. Режим доступа: https://devburn.ru/2015/09/ (дата обращения: 15.11.2022).

[7] The alocithmic beauty of plants. Режим доступа: http://algorithmicbotany.org/papers/abop/abop.pdf (дата обращения 15.11.2022).

[8] Parametric L-systems and their application to the modelling and visualization of plants [Электронные ресурс]. Режим доступа: http://algorithmicbotany.org/papers/hanan.dis1992.pdf (дата обращения 15.11.2022).

[9] Рэнди Дэвис Стефан. C++ для чайников. Для чайников. Вильямс, 2018. с. 400.

[10] Clion 2022 [Электронный ресурс]. Режим доступа: https://www.jetbrains.com/clion/whatsnew/ (дата обращения 15.11.2022).

[11] Date and time utilities [Электронный ресурс]. Режим доступа: https://en.cppreference.com/w/cpp/chrono (дата обращения: 15.11.2022).