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

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Визуализация трёхмерных деревьев на основе L-систем с учётом параметров освещения и ветвления»

Студент ИУ7-53Б _____ Д. В. Павлов
(Группа) (Подпись, дата) (И. О. Фамилия)

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

2025 г.

РЕФЕРАТ

В ходе курсовой работы разработано программное обеспечение для процедурной генерации трёхмерных моделей деревьев и их визуализации. Реализованы алгоритмы построения структуры дерева на основе L-систем, программный растеризатор с z-буферизацией и метод карт теней для формирования теней в сцене.

Расчетно-пояснительная записка к курсовой работе содержит 60 страниц, 22 рисунка, 9 таблиц, 14 источников, 2 приложения.

Ключевые слова: процедурная генерация, L-системы, трехмерная графика, растеризация, z-буфер, карты теней, визуализация деревьев.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Создание реалистичных трёхмерных моделей растений является важной задачей в компьютерной графике, особенно для визуализации природных ландшафтов, архитектурных симуляций и игровых сред. Одним из эффективных подходов к процедурной генерации деревьев служат L-системы — формальные грамматики, предложенные Аристидом Линденмейером для моделирования роста биологических структур. Благодаря рекурсивному характеру и способности описывать самоподобные формы, L-системы широко применяются для генерации как двумерных, так и трёхмерных растительных объектов.

Цель курсовой работы — разработка программного обеспечения, с графическим пользовательским интерфейсом для визуализации трёхмерных деревьев на основе L-систем с учётом параметров освещения, правил L-системы, углов ветвления и количества итераций генерации.

Для достижения цели поставлены следующие задачи:

— провести сравнительный анализ существующих методов генерации трёхмерных деревьев и сделать выводы о применении L-систем для решения поставленной задачи;

— спроектировать метод генерации трёхмерных деревьев на основе L-систем и описать его;

— выбрать средства для разработки ПО и реализовать систему визуализации;

— провести исследование работы программы.

1 Аналитическая часть

1.1 Общая характеристика предметной области

Визуализация трёхмерных растительных объектов является одной из важнейших задач современной компьютерной графики. Деревья составляют неотъемлемую часть виртуальных сцен в природных ландшафтах, архитектурных симуляциях и игровых средах. Качество их визуального представления напрямую влияет на уровень погружения и реалистичность изображения.

Значимость задачи визуализации деревьев обусловлена широким спектром прикладных направлений. В игровой индустрии и симуляторах виртуальной реальности требуется отображать обширные сцены с растительностью в реальном времени, обеспечивая баланс между качеством картинки и производительностью. В архитектурной визуализации и ландшафтном дизайне точное визуальное представление деревьев позволяет оценить визуальную привлекательность проекта и его интеграцию в окружающую среду. В кинематографе и анимации требуется фотореалистичный рендеринг сложных растительных сцен с корректной обработкой освещения, теней и материалов.

Для эффективной визуализации деревьев необходимо решить несколько взаимосвязанных проблем. Во-первых, требуется компактное представление геометрии: дерево может содержать десятки тысяч ветвей и листьев, что создаёт высокую нагрузку на графический конвейер. Во-вторых, необходимо обеспечить управляемость формой объекта через параметры, влияющие на структуру кроны, углы ветвления, густоту листвы. В-третьих, система визуализации должна корректно обрабатывать параметры освещения и материалов, обеспечивая реалистичное взаимодействие света с поверхностями.

Традиционный подход, при котором художник-конструктор вручную создаёт полигональную модель дерева и затем визуализирует её стандартными средствами, сталкивается с рядом ограничений. Ручное моделирование крайне трудоёмко и не позволяет быстро экспериментировать с различными формами деревьев. Кроме того, созданные вручную модели сложно параметризовать: изменение формы кроны или высоты дерева потребует повторной работы художника.

Эти ограничения мотивируют переход к процедурным методам генерации, которые позволяют автоматически создавать сложные геометрические

структуры на основе набора правил и параметров. Процедурный подход решает проблемы масштабируемости и разнообразия, позволяя генерировать целые экосистемы уникальных растений.

1.2 Сравнительный анализ существующих методов процедурной генерации деревьев

1.2.1 Обзор основных подходов

Существует несколько базовых подходов к процедурной генерации трёхмерных деревьев, каждый из которых обладает своими преимуществами и ограничениями.

Фрактальные методы основаны на рекурсивном применении правил ветвления. Классический пример — построение дерева путём многократного деления ветвей на более мелкие с уменьшением длины и толщины. Такой подход позволяет создавать визуально правдоподобные структуры с минимальным набором параметров. Однако фрактальные деревья обладают излишней регулярностью и не всегда точно воспроизводят морфологию реальных растений [1].

Методы на основе пространственной колонизации (Space Colonization Algorithm) моделируют рост дерева в направлении источников питания. Алгоритм размещает в пространстве множество точек-аттракторов, к которым стремятся растущие ветви. Данный метод позволяет получать органичные формы с естественным распределением ветвей, однако требует сложной настройки параметров и вычислительно затратен [2].

Грамматические методы описывают структуру дерева через формальные правила подстановки символов. Наиболее известным представителем являются L-системы (Lindenmayer systems), предложенные биологом Аристидом Линденмайером в 1968 году для моделирования развития растений. L-системы представляют собой параллельные системы переписывания строк, где каждый символ одновременно заменяется согласно заданным правилам [3].

Методы на основе физического моделирования симулируют биологические процессы роста с учётом гравитации, направления источника света и конкуренции за ресурсы. Такие подходы дают наиболее реалистичные результаты, но требуют значительных вычислительных ресурсов и сложны в

реализации [4].

1.2.2 Сравнительный анализ методов

При выборе метода генерации для решения поставленной задачи необходимо учитывать следующие критерии: вычислительная сложность, число управляющих параметров, возможность интерактивного редактирования и гибкость управления формой. Результаты сравнительного анализа представлены в таблице 1.1.

Таблица 1.1 – Сравнение методов процедурной генерации деревьев

Критерий	**Метод**			
	Фракталы	**Space Colonization**	**L-системы**	**Физ. модели**
Вычислительная сложность	Линейная относительно числа сегментов	Квадратичная	Линейная относительно длины строки	Высокая: требуется итеративная симуляция временных шагов
Число управляющих параметров	3–5	8-12	5-10	>15
Инкрементальное обновление при изменении параметров	Возможно	Невозможно (полная регенерация)	Возможно	Невозможно (полная регенерация)
Возможность задания асимметричной формы кроны	Ограничена	Да	Да	Да
Разнообразие при одних параметрах	Нет	Ограниченная	Да (стохастические L-системы)	Да

Фрактальные методы имеют минимальное число параметров и линейную сложность, но ограничены в создании асимметричных форм; метод на основе пространственной колонизации обладает квадратичной сложностью и не поддерживает инкрементальное обновление, что делает метод непригодным для интерактивного редактирования; физические модели требуют большого числа параметров и полной пересимуляции при изменении любого из них, согласно данным, представленным в таблице 1.1.

L-системы сочетают линейную сложность, умеренное число параметров, возможность инкрементального обновления и высокую гибкость в задании

формы (таблица 1.1). Таким образом, для реализации системы визуализации с графическим интерфейсом выбран метод на основе L-системы.

1.2.3 L-системы как основа для генерации деревьев

L-система представляет собой формальную грамматику, которая определяется тройкой (1.1) [3].

$$G = (V, \omega, P), \tag{1.1}$$

где:

— V — алфавит (набор символов);

— ω — аксиома (начальная строка);

— P — набор правил продукции вида $a \to w$, где $a \in V$ и w — строка символов из V.

Генерация структуры дерева происходит итеративно. На каждом шаге все символы строки одновременно заменяются согласно правилам продукции. После n итераций получается строка, которая отражает геометрическую структуру дерева.

Для визуализации трёхмерных структур используется черепашья графика — метод интерпретации строки L-системы как последовательности команд для виртуального исполнителя. Черепаха имеет позицию в пространстве и ориентацию, определяемую тремя ортогональными векторами: направлением движения, левым направлением и направлением вверх.

Базовый набор команд включает:

— `F` — движение вперёд с рисованием линии (создание сегмента ветви);

— `f` — движение вперёд без рисования;

— `+` — поворот влево на заданный угол;

— `-` — поворот вправо на заданный угол;

— `&` — наклон вниз;

— `^` — наклон вверх;

— `/` — поворот по часовой стрелке относительно оси движения;

— `\` — поворот против часовой стрелки относительно оси движения;

— `[` — сохранение текущего состояния черепахи в стек;

— `]` — восстановление состояния из стека.

Команды «`[`» и «`]`» позволяют моделировать ветвление: при встрече символа «`[`» текущая позиция и ориентация сохраняются, система генерирует боковую ветвь, а затем при встрече «`]`» возвращается к сохранённой точке для продолжения роста основной ветви.

1.2.4 Типы L-систем

Детерминированные контекстно-свободные L-системы (D0L-системы) являются вариантом, где каждому символу соответствует ровно одно правило продукции, не зависящее от окружающих символов.

Пример D0L-системы, заданной тройкой (1.2):

$$\begin{aligned} &V = \{F, +, -, [,]\}; \\ &\omega = F; \\ &P: \ F \to F[+F]F[-F][F]. \end{aligned} \tag{1.2}$$

Визуализация результата работы L-системы (1.2) после пяти итераций при угле ветвления 20° приведена на рисунке 1.1.

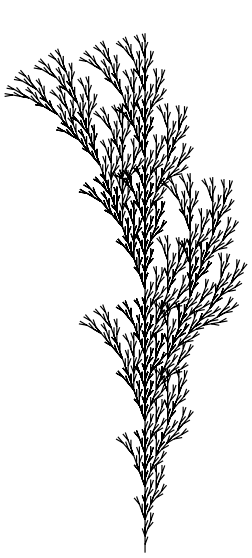

Рисунок 1.1 – Визуализация D0L-системы после 5 итераций (угол ветвления — 20°)

Стохастические L-системы вводят элемент случайности, позволяя каждому символу иметь несколько альтернативных правил продукции с заданными вероятностями. Это обеспечивает визуальное разнообразие генерируемых структур.

Пример стохастической L-системы приведён в выражении (1.3):

$$\begin{aligned} &V = \{F, +, -, [,]\}; \\ &\omega = F; \\ &P : \begin{cases} F \to F[+F]F[-F]F & p = 0{,}7, \\ F \to F[+F]F & p = 0{,}3. \end{cases} \end{aligned} \tag{1.3}$$

При каждом шаге подстановки для символа F случайным образом выбирается одно из правил в соответствии с заданными вероятностями. В результате при одинаковых параметрах система генерирует различные по форме деревья.

Контекстно-зависимые L-системы учитывают символы, предшествующие и следующие за заменяемым. Правила имеют вид $A{<}B{>}C \to D$, означающее, что символ B заменяется на D только при условии, что слева от него находится A, а справа — C.

Параметрические L-системы позволяют ассоциировать с каждым символом числовые параметры, изменяющиеся в процессе генерации. Это даёт возможность моделировать плавное уменьшение толщины ветвей, изменение углов ветвления в зависимости от порядка ветви и другие биологически

реалистичные эффекты.

Пример параметрической L-системы, заданной набором правил (1.4):

$$\begin{aligned} &\omega = F(1); \\ &P : \begin{cases} F(s) \to F(0{,}7s)[+F(0{,}7s)][-F(0{,}7s)], & s > s_{\min}, \\ F(s) \to F(s), & s \leqslant s_{\min}, \end{cases} \end{aligned} \tag{1.4}$$

где параметр s задаёт относительную длину сегмента, а порог $s_{\min}$ ограничивает глубину ветвления.

В рамках данной работы используется детерминированная контекстно-свободная L-система (D0L-система). Такой выбор обусловлен тем, что D0L-системы обладают линейной вычислительной сложностью по длине строки, просты в реализации и отладке, а также обеспечивают предсказуемый результат при фиксированных параметрах. Это важно для интерактивной визуализации, когда пользователь ожидает воспроизводимого эффекта при повторном запуске генерации с одинаковыми настройками.

Стохастические и контекстно-зависимые L-системы дают больше визуального разнообразия и позволяют учитывать взаимодействие между частями структуры, однако существенно усложняют алгоритм и интерпретацию результата. Для поставленной задачи — визуализации одиночного дерева с возможностью управления базовыми параметрами формы — возможностей D0L-системы оказывается достаточно.

1.3 Основные понятия и обозначения

Визуализация трёхмерной сцены представляет собой процесс преобразования математической модели объектов в двумерное растровое изображение. Данный процесс включает в себя геометрические преобразования, удаление невидимых поверхностей, расчёт освещённости и построение теней.

Для формализации описания алгоритмов введена следующая система обозначений:

— $P(x, y, z)$ — точка на поверхности объекта в мировых координатах;

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

— $\vec{L}$ — единичный вектор направления на источник света;

— $\vec{V}$ — единичный вектор направления на наблюдателя (камеру);

— $\vec{R}$ — единичный вектор отражённого света;

— I — итоговая интенсивность (цвет) пикселя.

1.4 Методы рендеринга (синтеза изображений)

Существует два фундаментальных подхода к синтезу изображений: трассировка лучей (ray-tracing) и растеризация (rasterization).

1.4.1 Трассировка лучей

Трассировка лучей моделирует физический путь распространения света. Для каждого пикселя экрана строится луч из точки наблюдения, и находится ближайшая точка пересечения с объектами сцены. Цвет пикселя определяется путём рекурсивного пускания вторичных лучей (отражение, преломление, тени). Вычислительная сложность: $O(N_{pixels} \cdot M_{objects})$ [5].

1.4.2 Растеризация

Растеризация основана на проекции геометрических примитивов (обычно треугольников) на плоскость экрана. Для каждого примитива определяется набор пикселей, который он покрывает. Процесс растеризации включает трансформацию вершин и интерполяцию атрибутов внутри примитива. Вычислительная сложность: $O(M_{objects} \cdot K_{pixels\_per\_object})$ [6].

1.4.3 Сравнительный анализ методов рендеринга

Сравнительный анализ методов представлен в таблице 1.2.

Таблица 1.2 – Сравнение методов рендеринга

Критерий	**Метод**	
	Растеризация	**Трассировка лучей**
Базовый принцип	Перебор объектов сцены	Перебор пикселей экрана
Вычислительная сложность	Линейная от числа полигонов	Логарифмическая от числа полигонов (при использовании ускоряющих структур), линейная от разрешения
Глобальное освещение	Требует сложных аппроксимаций	Реализуется естественно
Применимость для L-систем	Высокая (быстрый вывод множества мелких деталей)	Средняя (требует построения BVH-деревьев)

Для задачи интерактивной визуализации процедурных деревьев, содержащих большое количество полигонов, подходящим методом является растеризация, трассировка лучей, обеспечивая фотореалистичность, обладает избыточной вычислительной сложностью для поставленной задачи, согласно данным, представленным в таблице 1.2.

1.5 Алгоритмы удаления невидимых граней

Задача удаления невидимых граней заключается в определении видимых частей объектов сцены для заданной точки наблюдения. Существуют два основных подхода: алгоритмы пространства объектов, определяющие видимость геометрии до растеризации, и алгоритмы пространства изображения, работающие на уровне пикселей.

1.5.1 BSP-дерево

BSP-дерево (Binary Space Partitioning) является представителем алгоритмов пространства объектов и реализует метод художника — рисование объектов от дальних к ближним с перекрытием ранее нарисованных элементов [6].

Алгоритм использует предварительно построенное бинарное дерево, где каждый узел содержит треугольник и соответствующую плоскость разбиения. Плоскость делит пространство на две полуплоскости: положительную и отрицательную. Все треугольники распределяются по поддеревьям в зависимости от того, в какой полуплоскости они находятся. Если треугольник пересекает

плоскость разбиения, он разрезается на несколько частей.

Обход дерева для рендеринга выполняется рекурсивно: если камера находится в отрицательной полуплоскости узла, сначала рисуется положительное поддерево, затем треугольник узла, затем отрицательное поддерево. Если камера в положительной полуплоскости — порядок обратный. Этот порядок гарантирует корректную видимость независимо от положения камеры.

На рисунке 1.2 представлен пример построения BSP-дерева.

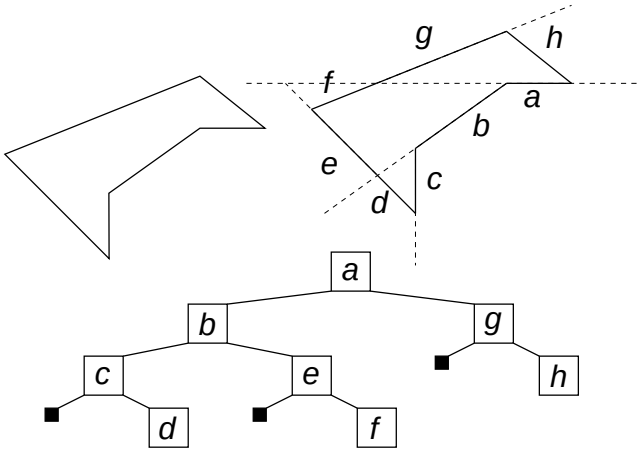

Рисунок 1.2 – Пример построения BSP-дерева

Ограничения: BSP-дерево строится как предобработка за время $O(N \log N)$, где N — число треугольников. Структура неизменна после построения, что делает метод непригодным для динамических сцен. Разрезание треугольников увеличивает число примитивов, а эффективность зависит от порядка добавления треугольников в дерево.

1.5.2 Z-буффер

Z-буфер — двумерный массив, параллельный выходному изображению, в котором для каждого пикселя хранится скалярное значение глубины, представляющее расстояние от центра проекции до видимой поверхности. Это алгоритм пространства изображения, работающий на уровне пикселей [6].

При растеризации треугольника для каждого покрываемого пикселя вычисляется глубина z методом интерполяции. Новая поверхность может перекрыть пиксель только если её глубина меньше текущего значения в буфере. В этом случае обновляются как цвет в кадровом буфере, так и значение в Z-буфере. Это обеспечивает неявное определение видимости — корректный

порядок рисования определяется автоматически во время растеризации, без предварительной сортировки объектов.

После завершения рендеринга z-буфер описывает первое пересечение луча, проходящего из центра проекции через каждый пиксель. Это позволяет использовать z-буфер не только для определения видимости, но и для последующих проходов рендеринга.

На рисунке 1.3 представлен пример работы буфера глубины при растеризации двух треугольников.

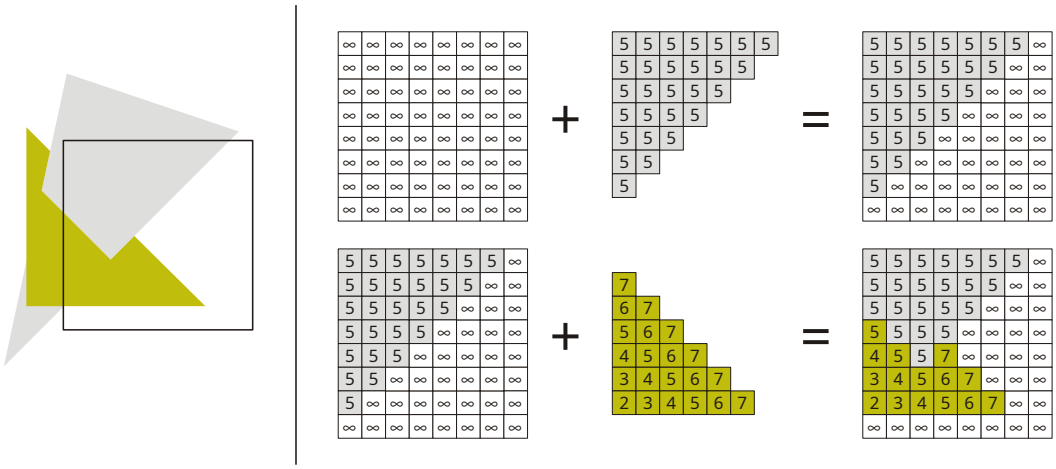

Рисунок 1.3 – Принцип работы буфера глубины при растеризации двух треугольников

Ограничения: если объекты перекрываются на уровне одного пикселя (например, на силуэтах), возникает ступенчатость. Кроме того, при ограниченной точности буфера глубины (обычно 24 бита) возможен эффект *z-fighting* — мерцание между двумя поверхностями с близкими значениями глубины.

Преимущества: z-буфер обеспечивает вычислительную сложность, пропорциональную только числу пикселей и независимую от числа объектов в сцене. Такой асимптотический выигрыш достигается за счёт постоянного расхода памяти на буфер глубины.

1.5.3 Сравнительный анализ

Результаты сравнения алгоритмов представлены в таблице 1.3.

Таблица 1.3 – Сравнение алгоритмов удаления невидимых граней

Критерий	**Алгоритм**	
	BSP-дерево	**Z-буфер**
Вычислительная сложность	$O(N)$ на кадр (обход дерева)	$O(N \times A)$, где A — площадь проекции
Память	$O(N \log N)$ (структура дерева)	$O(R^2)$ (буфер глубины)
Предобработка	Требуется $O(N \log N)$	Не требуется
Порядок рендеринга	Не зависит	Не зависит
Динамические сцены	Нет (требуется перестроение)	Полная поддержка

Для данной работы выбран z-буфер, поскольку метод поддерживает динамические сцены, не требует предобработки и позволяет вычислять видимость во время растеризации. BSP-дерево требует перестроения при изменении геометрии, что неприемлемо для интерактивного редактирования параметров L-системы.

1.6 Алгоритмы закраски

1.6.1 Модели освещения

Для расчёта цвета поверхности используется модель освещения, описывающая взаимодействие света с материалом.

Модель Ламберта учитывает только диффузное рассеивание света. Яркость, которая вычисляется по формуле (1.5), не зависит от угла обзора [5]:

$$I_{diff} = k_d I_L (\vec{N} \cdot \vec{L}), \tag{1.5}$$

где k_d — коэффициент диффузного отражения, I_L — интенсивность источника.

Модель Фонга добавляет зеркальный блик, зависящий от угла между вектором отражения $\vec{R}$ и вектором взгляда $\vec{V}$, и тогда интенсивность равна выражению (1.6) [5]:

$$I_{spec} = k_s I_L (\vec{R} \cdot \vec{V})^{\alpha}, \tag{1.6}$$

где $\vec{R} = 2(\vec{N} \cdot \vec{L})\vec{N} - \vec{L}$. Расчёт вектора $\vec{R}$ требует вычислительных затрат для каждого пикселя.

Модель Блинна-Фонга является модификацией модели Фонга. Вместо вектора отражения используется «серединный вектор» $\vec{H}$ (1.7), который является биссектрисой между $\vec{L}$ и $\vec{V}$ [5]:

$$\vec{H} = \frac{\vec{L} + \vec{V}}{||\vec{L} + \vec{V}||}. \tag{1.7}$$

Интенсивность блика рассчитывается как $(\vec{N} \cdot \vec{H})^{\alpha}$.

1.6.2 Методы затенения

Модель освещения определяет, как вычисляется яркость в отдельной точке поверхности. Однако для визуализации полигональных моделей необходимо распространить это значение по всему полигону. Существуют три основных подхода: плоское затенение, затенение по Гуро и затенение по Фонгу [7].

Плоское затенение использует одну нормаль (обычно усреднённую по грани) для расчёта освещённости всей поверхности полигона. Это быстро, но приводит к резким перепадам яркости на границах граней и визуально подчёркивает полигональную структуру [7].

Затенение по Гуро вычисляет освещённость в вершинах с использованием усреднённых нормалей, а затем линейно интерполирует полученную интенсивность по поверхности полигона. Такой подход даёт плавные переходы цвета и снижает вычислительную нагрузку, поскольку модель освещения применяется только в вершинах. Однако он плохо передаёт зеркальные блики, особенно если они находятся между вершинами — в таких случаях блик либо исчезает, либо размывается неестественно [6].

Затенение по Фонгу устраняет этот недостаток: вместо интерполяции интенсивности он интерполирует нормали от вершин к каждому пикселю, после чего вычисляет полную модель освещения непосредственно в фрагменте. Это обеспечивает точное отображение бликов и более реалистичную форму световых переходов, но требует значительных вычислительных ресурсов — нормализация нормали и расчёт освещения выполняются для каждого пикселя [7].

На рисунке 1.4 приведён пример визуализации основных методов затенения.

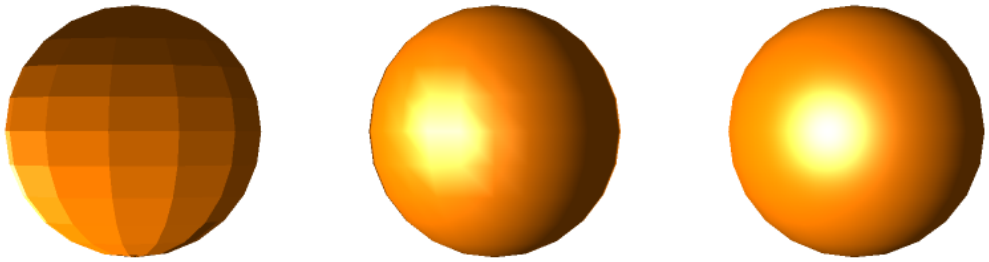

Рисунок 1.4 – Сравнение плоского затенения, затенения по Гуро и затенения по Фонгу

1.6.3 Выбор модели освещения для визуализации

В таблице 1.4 представлено сравнение моделей освещения.

Таблица 1.4 – Сравнение моделей освещения

Критерий	**Модель**		
	Ламберт	**Фонг**	**Блинн–Фонг**
Учитываемые компоненты	Только диффузное отражение	Диффузное + зеркальное	Диффузное + зеркальное
Зависимость от угла обзора	Нет	Да	Да
Вычислительная сложность	Низкая	Высокая	Умеренная
Визуальный реализм	Низкий	Высокий	Высокий
Подходящее применение	Базовое освещение, тени	Реалистичные блики при точном управлении	Реалистичные блики с улучшенной производительностью

Для визуализации растительности выбрана модель Блинна-Фонга в сочетании с методом затенения по Фонгу. Серединный вектор $\vec{H}$ модели Блинна-Фонга может быть вычислен один раз для удалённого источника света и наблюдателя, что существенно сокращает количество операций на пиксель по сравнению с классической моделью Фонга. Метод затенения по Фонгу обеспечивает высокое визуальное качество благодаря точному отображению зеркальных бликов. Несмотря на большие вычислительные затраты (интерполяция трёх компонент нормали и полный расчёт модели освещения для каждого пикселя), эта комбинация критически важна для достоверной визуализации сложных полигональных моделей растений, где детали освещения существенно влияют на реалистичность изображения.

1.7 Алгоритмы построения теней

Тени необходимы для восприятия пространственной глубины и структуры дерева.

Теневые объёмы — геометрический алгоритм, создающий объём пространства, находящийся в тени. Требует поиска силуэтных ребер модели и генерации дополнительной геометрии. Для высокополигональных моделей деревьев метод неприменим из-за высокой алгоритмической сложности [5].

Карты теней — алгоритм, работающий в пространстве изображения. Основан на идее, что освещены только те точки, которые «видны» из источника света. Сцена рендерится с позиции источника света, записываются только значения глубины z_{light}. При финальном рендеринге точка P переводится в систему координат источника света, её глубина d сравнивается со значением из карты теней. Если $d > z_{light}$, точка в тени [5].

В таблице 1.5 представлено сравнение методов построения теней.

Таблица 1.5 – Сравнение методов построения теней

Критерий	**Метод**	
	Теневые объёмы	**Карты теней**
Принцип работы	Геометрический: построение объёмов, блокирующих свет	На основе глубины: сравнение расстояний от источника света
Зависимость от геометрической сложности сцены	Да (требуется обработка всех силуэтных рёбер)	Нет (зависит только от разрешения карты теней)
Вычислительная сложность	Высокая для сложных моделей (например, деревьев с тысячами листьев)	Умеренная, фиксированная по геометрии (масштабируется с разрешением)
Требуемая дополнительная геометрия	Да (генерация теневых объёмов)	Нет
Подходящее применение	Простые сцены с чёткими границами теней	Сложные сцены, включая высокополигональную растительность

В работе используется метод построения карты теней, так как его вычислительная сложность не зависит от количества листьев и ветвей дерева, а определяется только разрешением карты теней, согласно данным, представленным в таблице 1.5.

1.8 Вывод

В аналитической части, было дано описание предметной области, рассмотрены существующие методы процедурной генерации деревьев, было дано обоснование выбора L-систем в качестве метода генерации дерева и описаны основные методы и алгоритмы рендеринга. В качестве алгоритма для рендеринга изображения была выбрана растеризация, в качестве алгоритма удаления невидимых граней — z-буфер, в качестве модели затенения — модель Блинна-Фонга, а для построения теней — алгоритм построения карты теней.

2 Конструкторская часть

2.1 Требования к программному обеспечению

Программа должна предоставлять следующий функционал:

— изменение параметров направления освещения;

— изменение положения камеры в пространстве;

— настройка параметров L-системы;

— настройка параметров геометрии дерева;

— генерация и визуализация дерева, при заданных параметрах L-системы и геометрии дерева.

2.2 Используемые структуры данных

Для визуализации трёхмерных деревьев используются следующие структуры данных.

— Вершина — структура для хранения атрибутов одной точки поверхности, содержащая следующие поля:

  – координаты положения — трёхмерный вектор вещественных чисел;

  – нормаль — вектор, задающий направление перпендикуляра к поверхности в данной точке.

— Треугольник — структура для описания грани поверхности, содержащая три целочисленных индекса, ссылающихся на вершины в общем списке.

— Полигональная сетка — структура для хранения объекта сцены, содержащая следующие поля:

  – список вершин — массив структур «вершина»;

  – список треугольников — массив структур «треугольник».

— Источник освещения — структура для хранения направления света в виде трёхмерного векотра и интенсивности.

— Камера — структура, отвечающая за формирование изображения трёхмерной модели. Содержит следующие поля:

  – положение камеры — тройка вещественных чисел, задающая координаты камеры;

  – точка наблюдения — тройка вещественных чисел, задающая центр сцены, на который направлен взгляд;

  – параметры проекции — угол обзора, соотношение сторон, расстояния до ближней и дальней плоскостей отсечения.

— Сцена — структура, содержащая в себе набор объектов, источник освещения и камеру.

2.3 Разработка алгоритмов

2.3.1 Функциональная модель процесса построения изображения

На рисунке 2.1 показана диаграмма IDEF0 верхнего уровня, описывающая построение изображения.

На рисунке 2.2 представлена диаграмма IDEF0 декомпозиции уровня A0.

На рисунке 2.3 представлена диаграмма IDEF0 декомпозиции уровня A1.

На рисунке 2.4 представлена диаграмма IDEF0 декомпозиции уровня A2.

На рисунке 2.5 представлена диаграмма IDEF0 декомпозиции уровня A3.

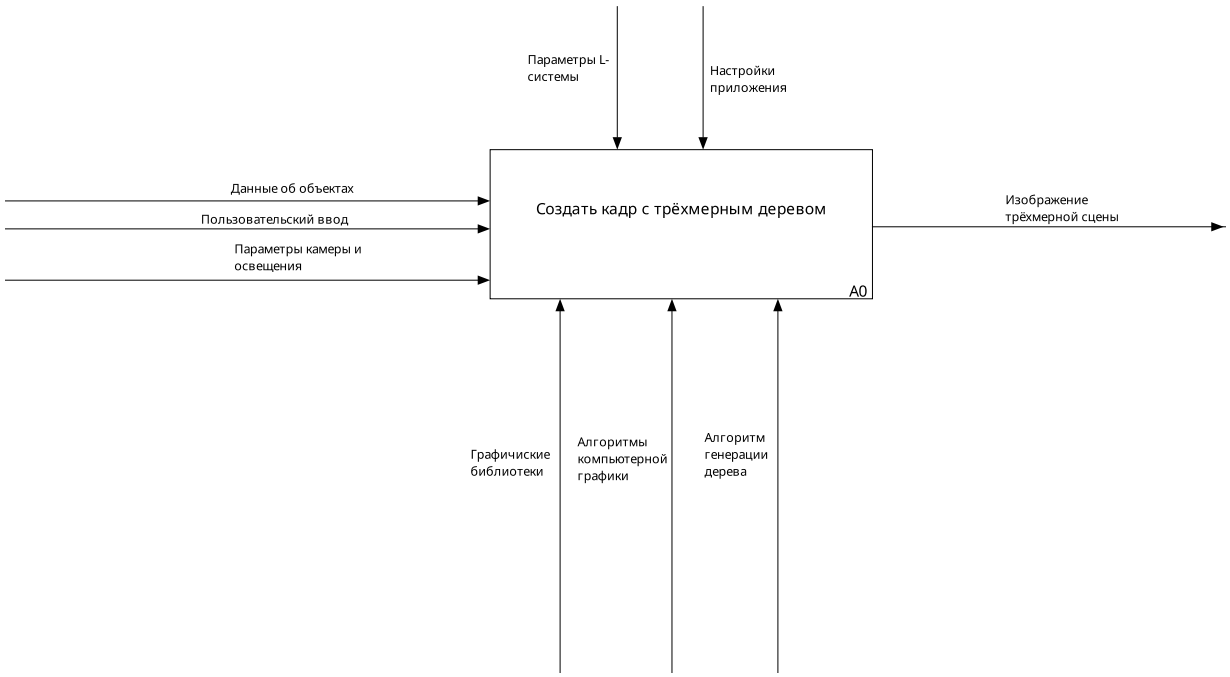

Рисунок 2.1 – Функциональная схема алгоритма построения изображения, декомпозиция верхнего уровня

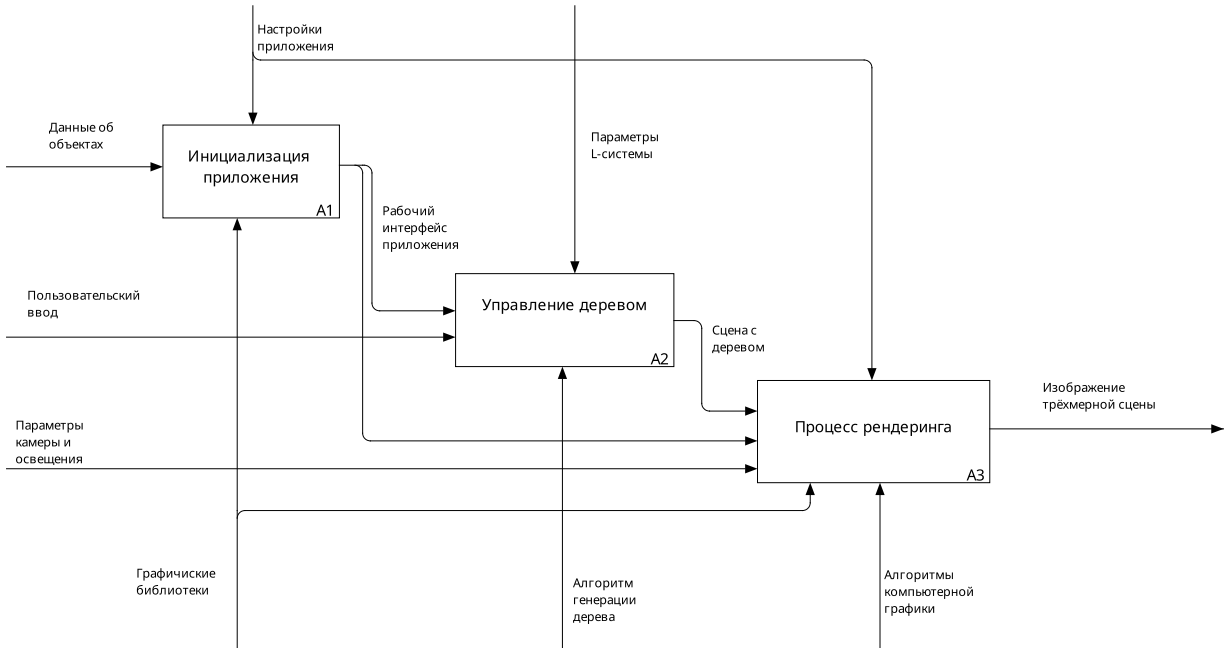

Рисунок 2.2 – Функциональная схема алгоритма построения изображения, декомпозиция уровня A0

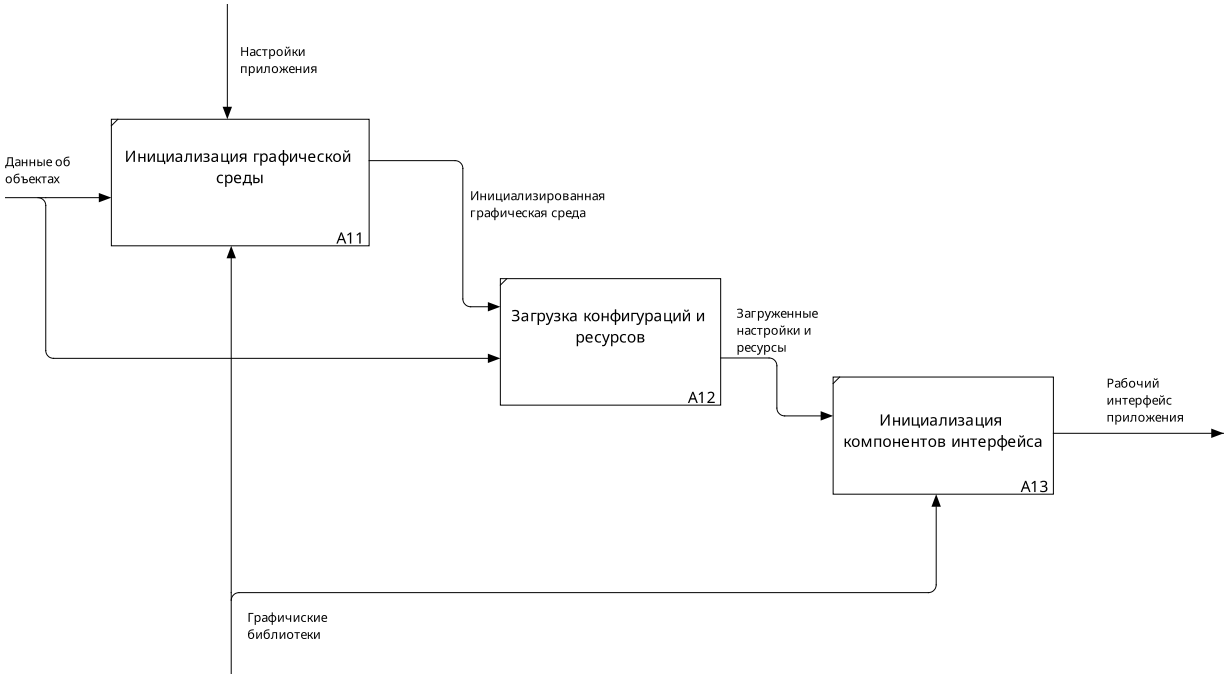

Рисунок 2.3 – Функциональная схема алгоритма построения изображения, декомпозиция уровня A1

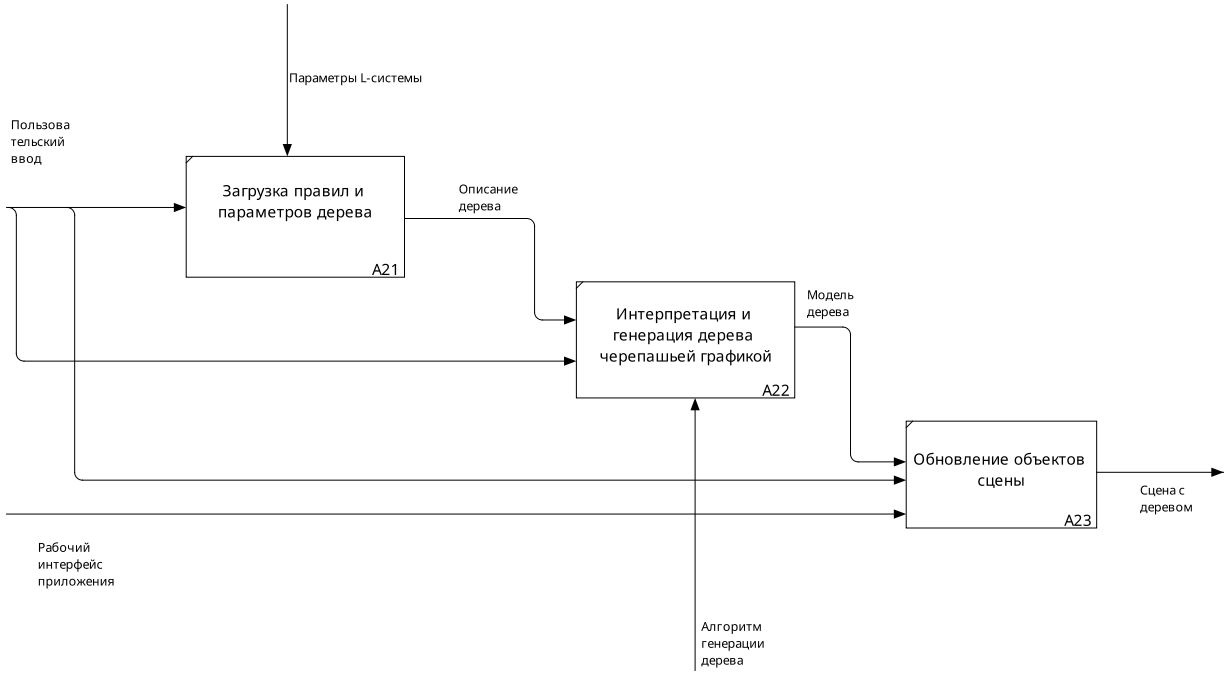

Рисунок 2.4 – Функциональная схема алгоритма построения изображения, декомпозиция уровня A2

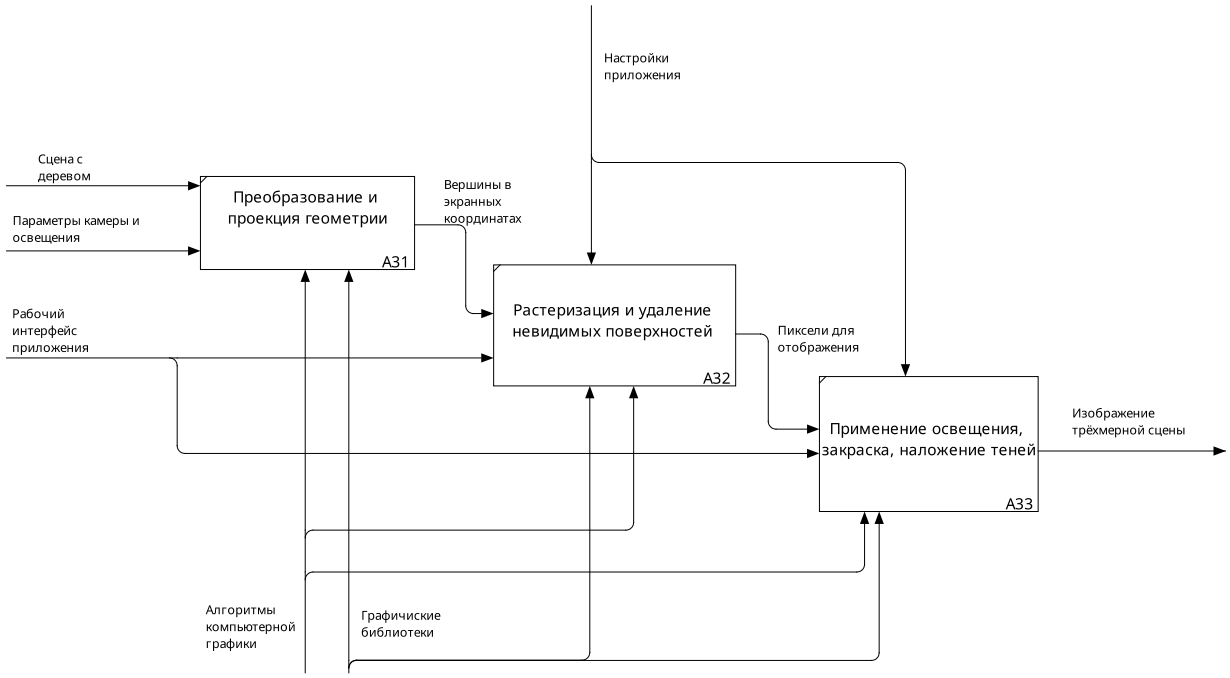

Рисунок 2.5 – Функциональная схема алгоритма построения изображения, декомпозиция уровня A3

2.3.2 Алгоритм построения изображения

Для отображения трехмерной сцены на двумерный экран координаты вершин каждого объекта должны пройти последовательность преобразований из локальной системы координат в систему координат экрана.

Результирующее положение вершины V_{clip} в пространстве отсечения вычисляется по формуле (2.1) путем умножения исходных локальных координат вершины V_{local} на матрицу модели (M), матрицу вида (V) и матрицу проекции (P):

$$V_{clip} = P \cdot V \cdot M \cdot V_{local} \tag{2.1}$$

Матрица модели (M) преобразует вершины из локальных координат объекта в мировые координаты. Она включает в себя перенос, поворот и масштабирование.

Матрица вида (V) преобразует мировые координаты в систему координат камеры. Она определяется положением наблюдателя и направлением взгляда.

Матрица проекции (P) преобразует координаты из системы камеры в пространство отсечения, отображая видимую усечённую пирамиду в

канонический объём в однородных координатах.

На рисунке 2.6 представлена схема алгоритма построения изображения.

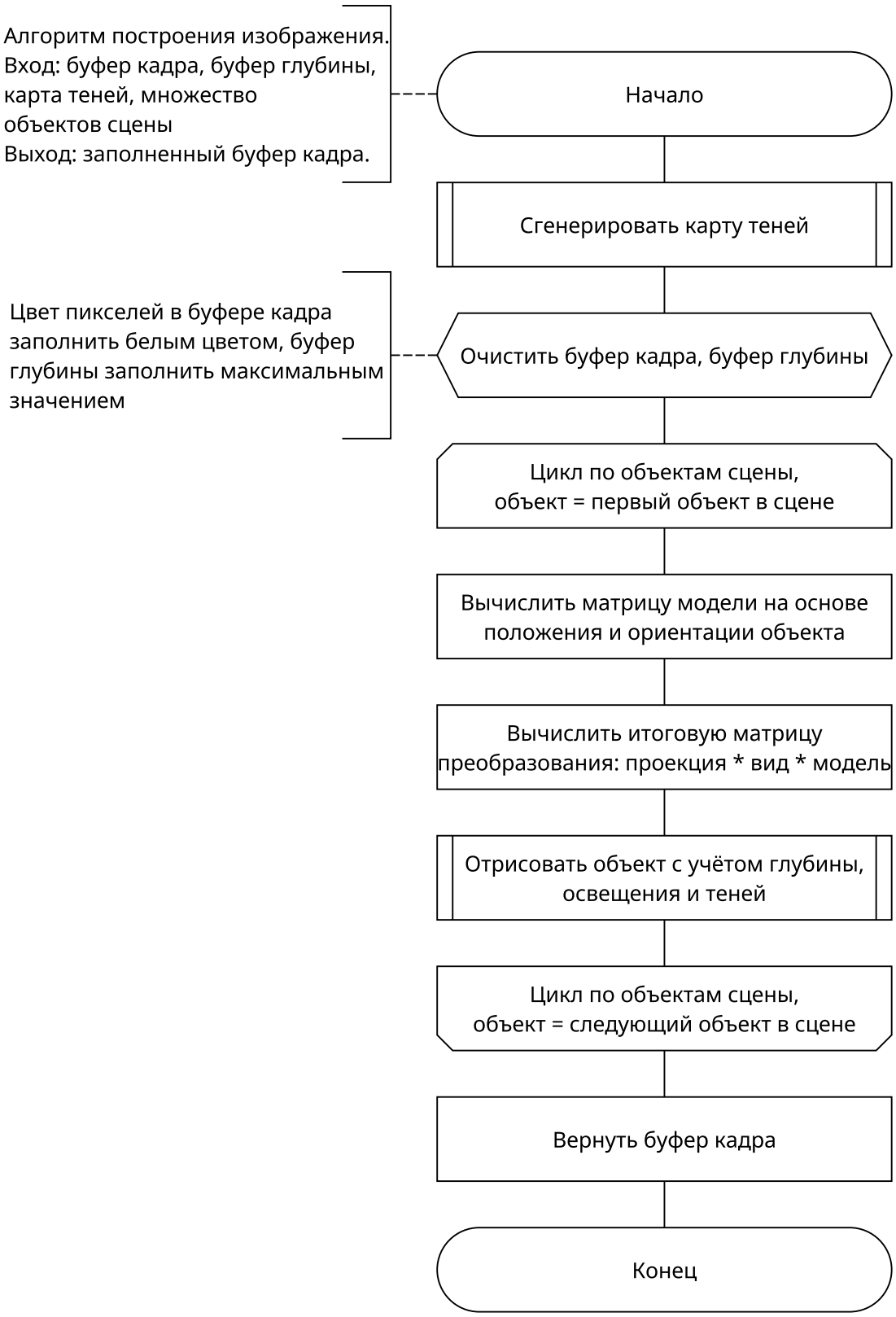

Рисунок 2.6 – Схема алгоритма построения кадра

2.3.3 Алгоритм построения карты теней

На рисунке 2.8 представлена схема алгоритма построения теневой карты.

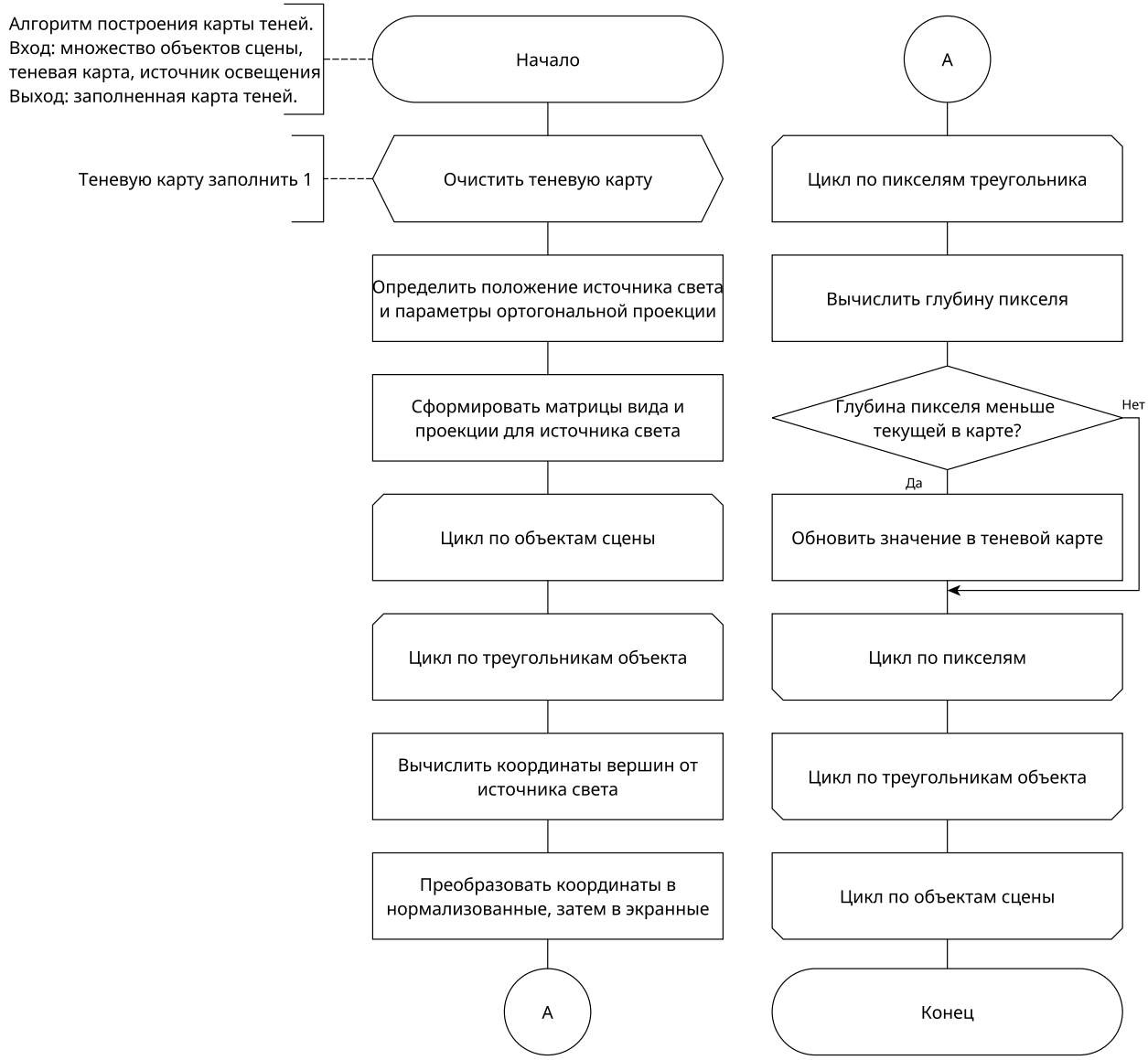

Рисунок 2.7 – Схема алгоритма построения теневой карты

2.3.4 Алгоритм растеризации с учётом z-буффера

Проверка глубины выполняется непосредственно при заполнении каждого треугольника. Для каждого пикселя, покрываемого треугольником, вычисляется интерполированное значение глубины. Если оно меньше текущего значения в z-буфере, обновляются как цвет, так и глубина в соответствующих буферах. Это обеспечивает корректное отображение перекрывающихся ветвей и листьев без предварительной сортировки объектов.

На рисунке 2.8 представлена схема алгоритма растеризации с учётом z-буффера.

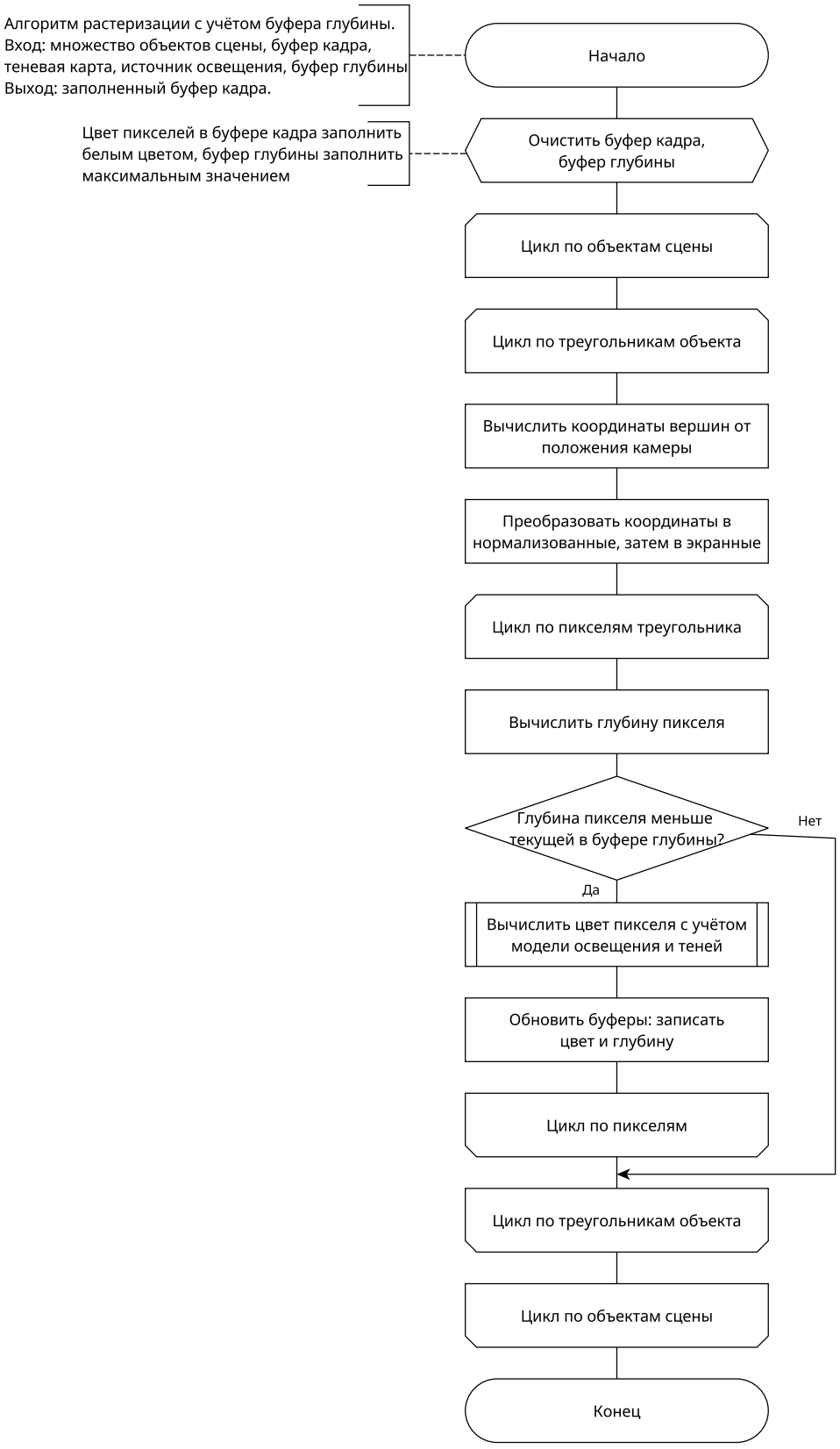

Рисунок 2.8 – Схема алгоритма растеризации с учётом z-буффера

2.3.5 Алгоритм модели освещения Блинна-Фонга

Разработанная модель освещения включает три компонента:

— фоновое (ambient) — равномерное освещение, имитирующее рассеянный свет в сцене;

— диффузное (diffuse) — основной вклад, зависящий от угла между нормалью поверхности и направлением света;

— зеркальное (specular) — блик, моделирующий отражение от глянцевых участков.

Итоговая формула (2.2) расчета освещенности в точке:

$$I_{total} = I_{ambient} + I_{diffuse} + I_{specular} \tag{2.2}$$

На рисунке 2.9 представлена схема алгоритма модели освещения Блинна-Фонга.

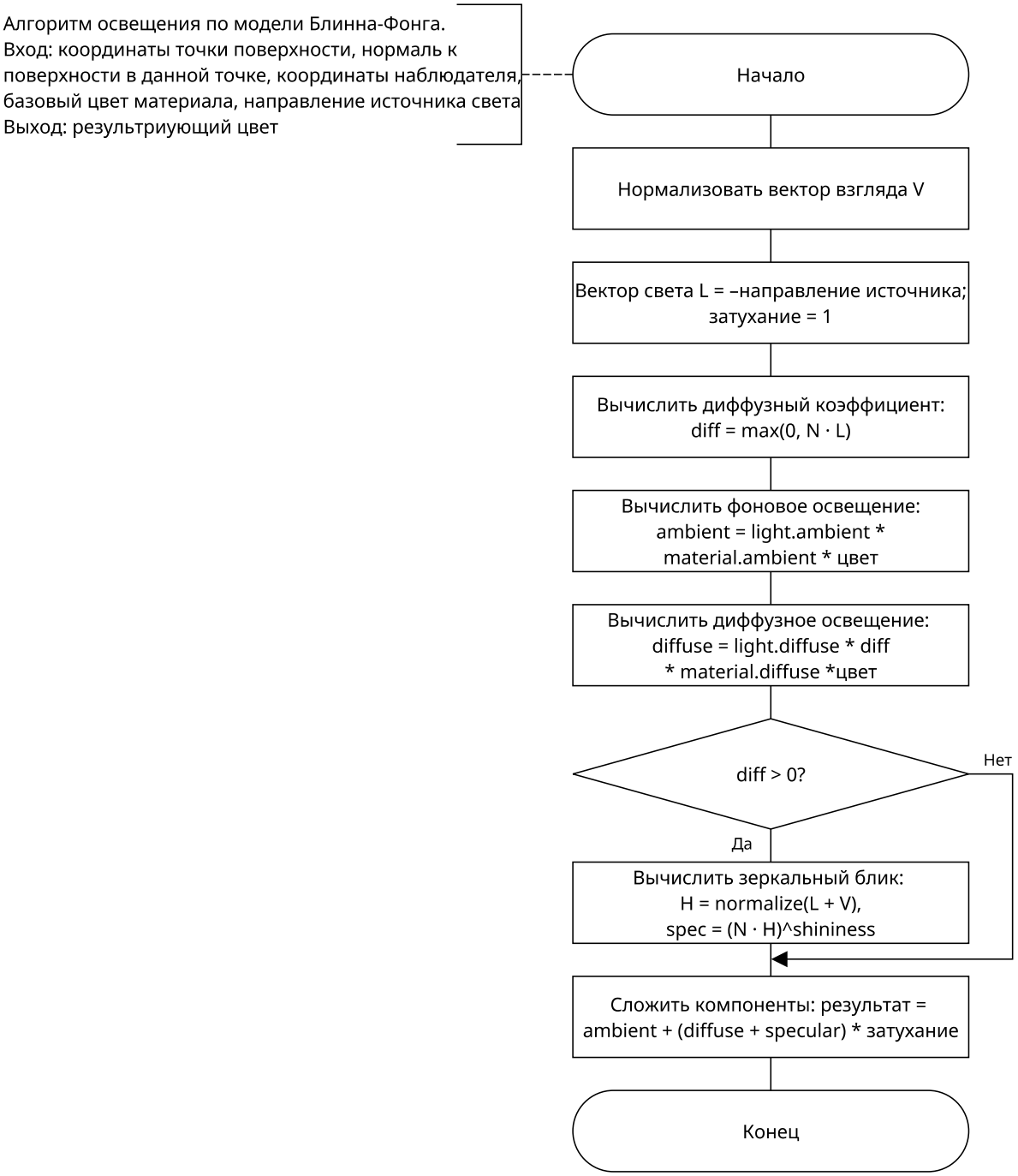

Рисунок 2.9 – Схема алгоритма модели освещения Блинна-Фонга

2.3.6 Алгоритм генерации структуры дерева на основе L-системы

Алгоритм принимает на вход аксиому, набор правил продукции и число итераций, после чего последовательно применяет правила к каждому символу строки.

На выходе формируется последовательность команд (таких как `F`, `[`, `+`,

& и др.), которая в дальнейшем интерпретируется трёхмерной черепашьей графикой для построения геометрической модели.

На рисунке 2.10 представлена схема данного алгоритма.

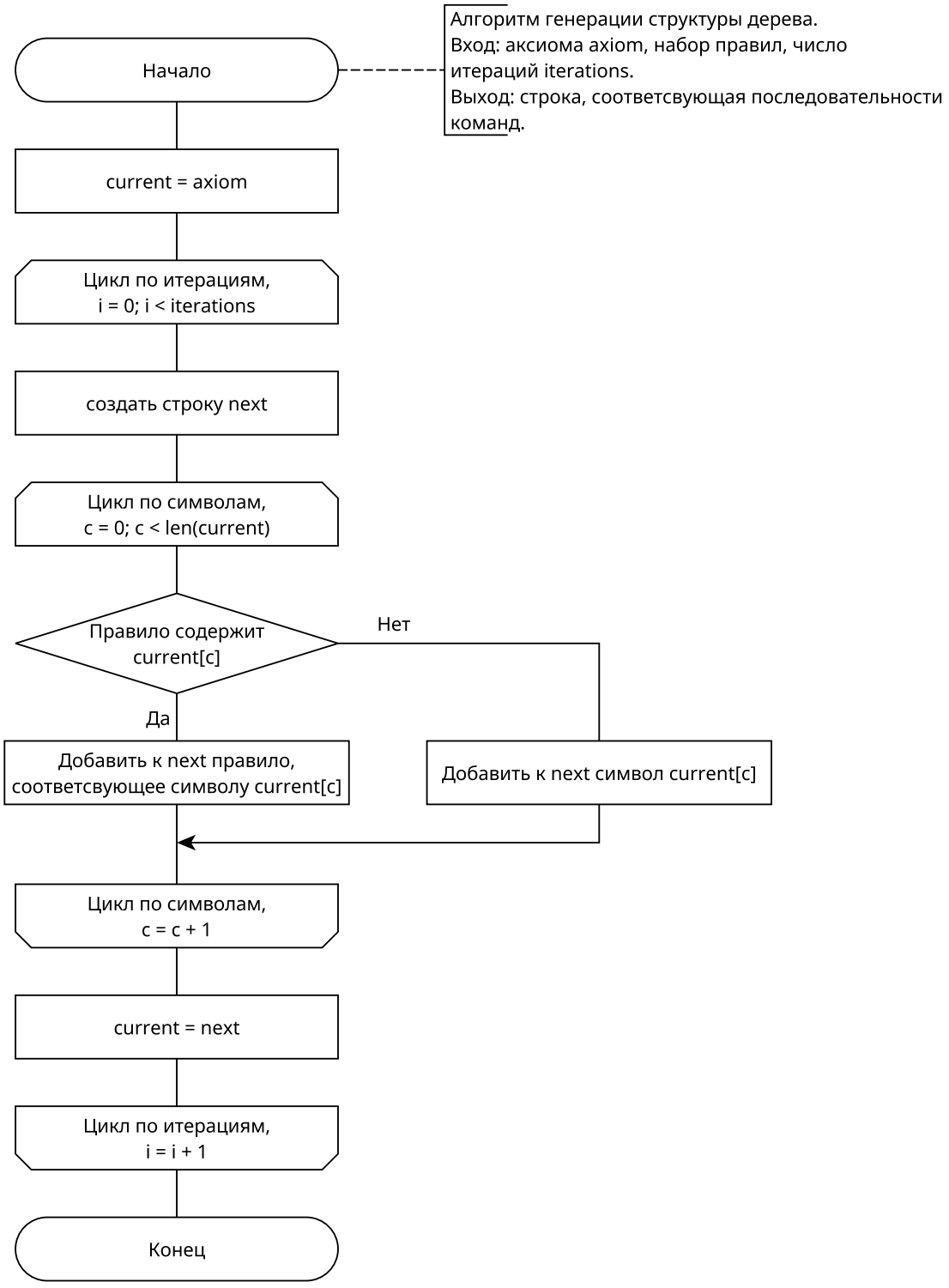

Рисунок 2.10 – Схема алгоритма генерации структуры дерева на основе заданной L-системы

2.4 Вывод

В конструкторской части было спроектировано программное обеспечение, описаны схемы основных алгоритмов и используемые структуры данных.

3 Технологическая часть

3.1 Средства реализации

Для реализации программного обеспечения был выбран язык C++ [8]. Выбор C++ обусловлен наличием всех необходимых компонентов в стандартной библиотеке для реализации алгоритмов, разработанных в результате проектирования. В частности, были использованы следующие компоненты:

— контейнер `std::vector` для хранения вершин, нормалей, треугольников и листьев сцены;

— строки `std::string` и алгоритмы работы с ними — для представления и итеративного расширения последовательности символов L-системы;

— `std::stack` — для реализации состояния черепашьей-интерпретации (сохранение и восстановление позиции и ориентации при обходе ветвей);

— средства генерации псевдослучайных чисел (`std::default_random_engine` и `std::uniform_real_distribution`) — для внесения естественной вариативности в углы ветвления и длины сегментов;

— математические функции стандартной библиотеки (<cmath>).

В качестве сторонней библиотеки для реализации графического пользовательского интерфейса был выбран фреймворк Qt [9]. Выбор обусловлен его кроссплатформенностью, нативной поддержкой языка C++ и наличием визуального редактора интерфейсов (Qt Designer) [10].

В качестве системы управления процессом сборки был выбран CMake [11]. CMake представляет собой кроссплатформенную систему автоматизации сборки, которая генерирует файлы конфигурации для различных систем сборки.

В качестве среды разработки использовался CLion [12], обеспечивающий все необходимые инструменты для работы с проектами на C++.

3.2 Структура программы

Программа построена по объектно-ориентированному принципу и состоит из нескольких взаимосвязанных модулей, каждый из которых отвечает за

определённую функциональную область.

Модуль генерации геометрии отвечает за синтез структуры дерева и построение полигональной модели. Содержит:

— `LSystemGenerator` — реализует итеративную подстановку правил L-системы и возвращает строку команд;

— `TurtleInterpreter3D` — интерпретирует строку команд с помощью трёхмерной черепашьей графики и генерирует полигональную сетку ствола и ветвей, а также позиции листьев.

Модуль представления сцены определяет структуру виртуального мира и объектов в нём. Содержит:

— `Scene` — контейнер, хранящий все объекты сцены, источник света и камеру;

— `SceneObject` — абстрактный базовый класс для всех объектов сцены;

— `MeshObject` — объект сцены, содержащий одну полигональную сетку (используется для ствола и ветвей);

— `InstancedMeshObject` — объект для эффективного отображения множества копий одной геометрии (листья).

Модуль рендеринга осуществляет программную визуализацию сцены. Содержит:

— `Rasterizer` — основной класс растеризатора, выполняющий отрисовку треугольников с учётом z-буфера, освещения и теней;

— `ShadowMapRenderer` — специализированный рендерер для генерации карты теней;

— `BaseVisitor` — абстрактный класс, реализующий паттерн «Посетитель».

— `DrawVisitor`, `ShadowVisitor` — реализуют паттерн «Посетитель» для обхода объектов сцены с разными целями (основной рендеринг/теневой проход).

Модуль управления освещением выполняет расчёт цвета фрагментов на основе выбранной модели. Содержит:

— `Lighting` — статический класс с методом `calculatePhong` для расчёта освещения по модели Блинна–Фонга.

Модуль пользовательского интерфейса обеспечивает взаимодействие пользователя с системой. Содержит:

— `MainWindow` — главное окно приложения на основе Qt, содержащее элементы управления (поля ввода, слайдеры) и виджет отображения;

— `SceneRenderer` — связующий компонент, координирующий генерацию, настройку камеры, источников света и вызов рендеринга.

К вспомогательным компонентам относятся:

— `Camera`, `OrbitCamera`, `FreeCamera` — управление точкой зрения;

— `Mesh`, `Vertex`, `Triangle` — представление геометрии;

— `Light`, `Material` — параметры освещения и материалов.

Взаимодействие модулей осуществляется через чётко определённые интерфейсы:

— модуль генерации выдаёт `Mesh` и позиции листьев;

— модуль сцены инкапсулирует данные;

— модуль рендеринга принимает сцену, возвращает изображение;

— интерфейс инициирует генерацию и отображает результат.

На рисунке 3.1 показана диаграмма классов программы.

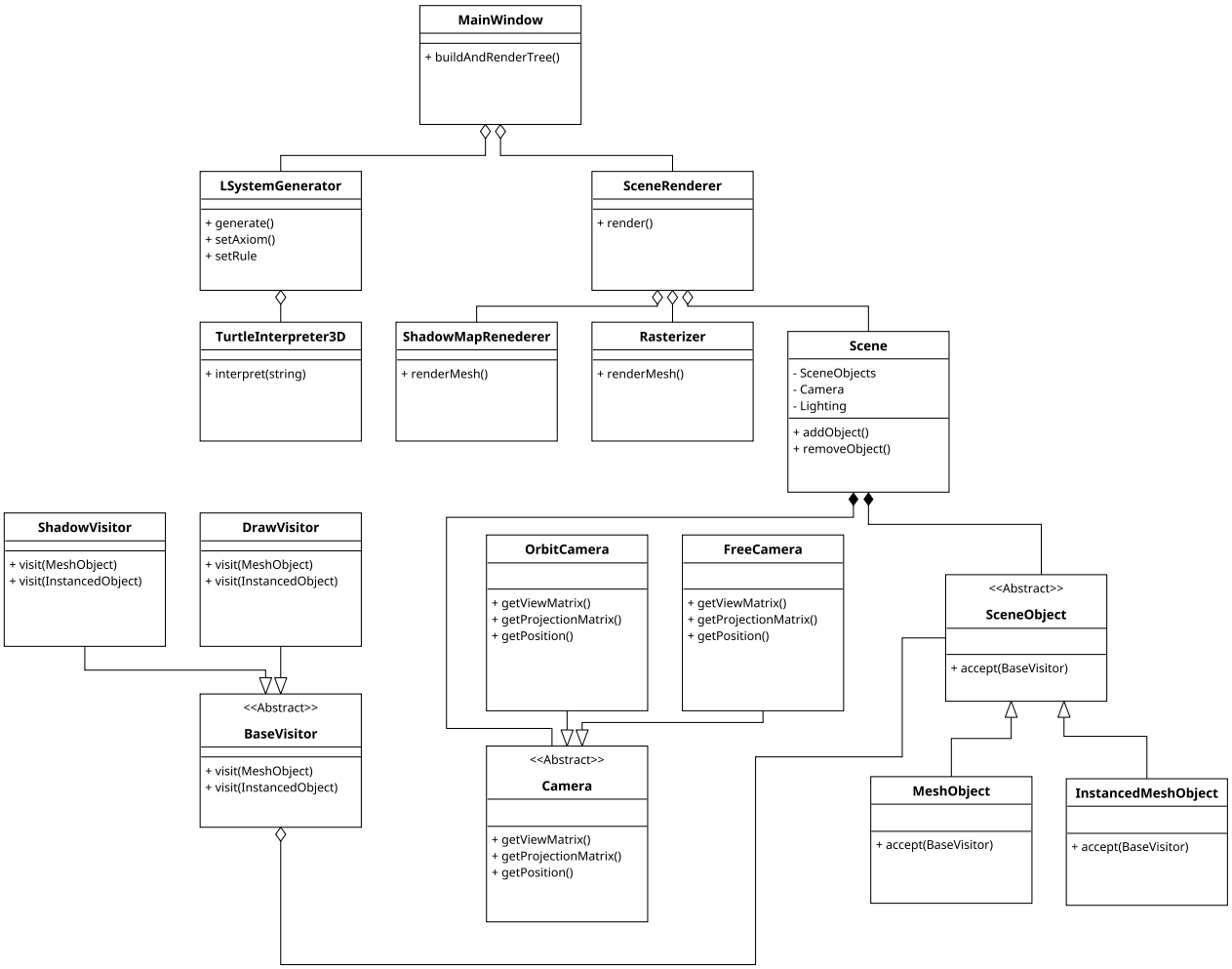

Рисунок 3.1 – UML-диаграмма классов программы

3.3 Реализация алгоритмов

В листинге Б.1 приведена реализация алгоритма растеризации с учётом буфера глубины.

В листинге Б.2 приведена реализация алгоритма построения карты теней.

В листинге Б.3 приведена реализация модели освещения по Блинну-Фонгу.

В листинге Б.4 приведена реализация алгоритма построения структуры дерева.

3.4 Описание пользовательского интерфейса

Пользовательский интерфейс состоит из панели управления параметрами и области визуализации трёхмерной сцены.

3.4.1 Структура интерфейса

Главное окно приложения разделено на две области:

— левая панель управления с элементами настройки параметров генерации;

— правая область визуализации с отображением трёхмерной модели дерева.

Панель управления организована в виде групповых элементов, каждый из которых отвечает за определённую категорию параметров.

3.4.2 Основные элементы управления

Группа «Готовые деревья» содержит выпадающий список с предустановленными конфигурациями деревьев различных типов.

Группа «Параметры L-системы» включает:

— поле ввода аксиомы L-системы;

— таблицу правил подстановки с возможностью добавления и редактирования;

— выпадающий список символов (A–Z) для выбора символа правила;

— поля настройки количества итераций, угла поворота и длины шага.

Система поддерживает стандартные операторы L-системы: F/G (рисование), f (перемещение), +/- (повороты), &/^ (наклоны), [ ] (ветвление), L (листья).

Группа «Геометрия дерева» содержит параметры:

— базовый радиус ствола (0,01–2,0);

— коэффициент уменьшения радиуса ветвей (0,1–0,99);

— минимальный радиус конечных ветвей (0,001–0,1);

— фактор гравитации для провисания ветвей (0,0–0,5);

— количество сегментов в поперечном сечении (3–20).

Группа «Параметры освещения» позволяет настроить интенсивность и направление источника света (координаты x, y, z).

Группа «Камеры» предоставляет выбор между орбитальной камерой (вращение вокруг объекта) и свободной камерой (перемещение в пространстве).

Группа «Тени» содержит переключатель для включения/отключения расчёта теней методом shadow mapping.

3.4.3 Управление и визуализация

Генерация дерева выполняется по нажатию кнопки «Сгенерировать дерево».

Управление камерой осуществляется с помощью мыши (перетаскивание, колесо прокрутки) и клавиатуры (W/A/S/D для перемещения, Q/E для вертикального движения, 1/2/3 для предустановленных ракурсов).

Область визуализации отображает результат рендеринга. Размер области автоматически адаптируется при изменении размера окна.

3.5 Демонстрация работы программы

На рисунках 3.2 и 3.3 представлены примеры работы программы.

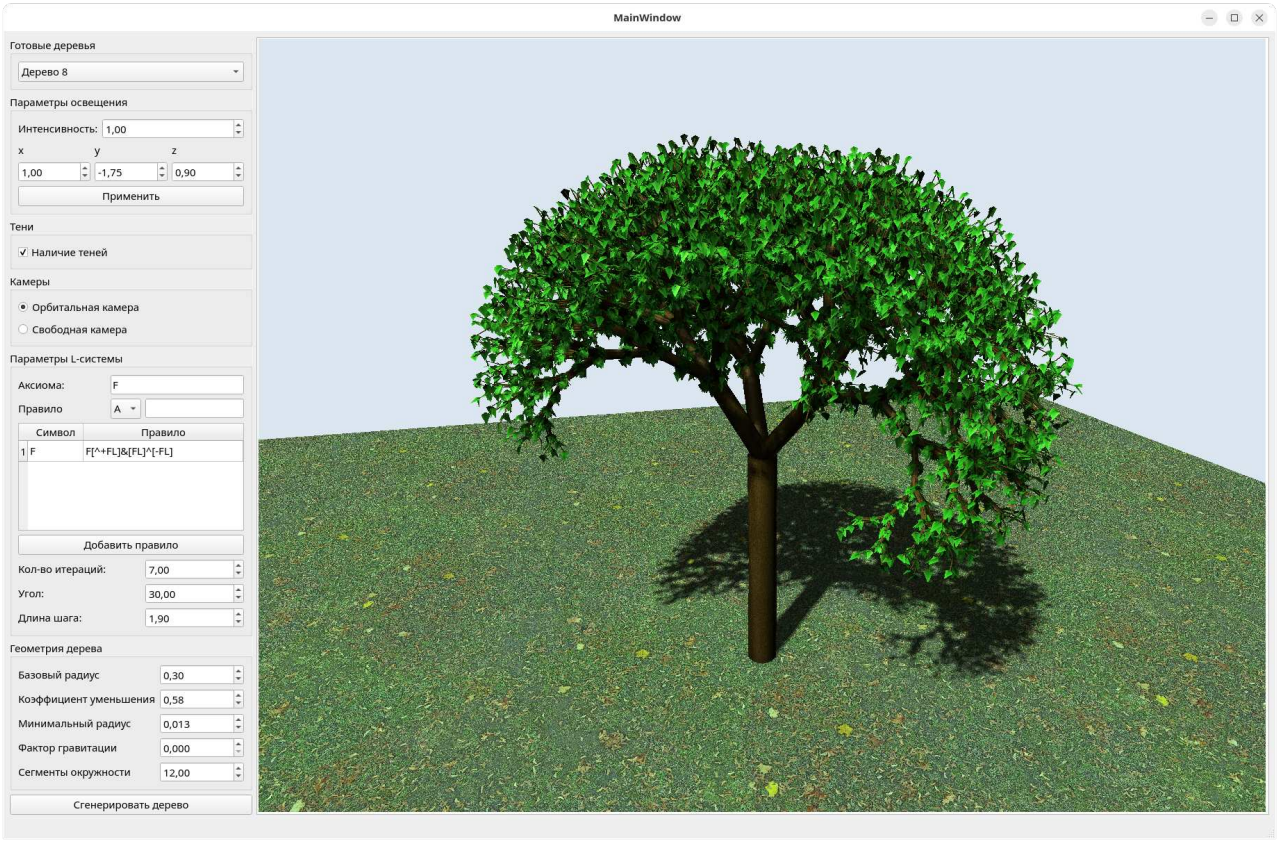

Рисунок 3.2 – Пример работы программы для дерева с листьями

Рисунок 3.3 – Пример работы программы для дерева без листьев

3.6 Вывод

В технологической части были описаны средства реализации программного обеспечения, описана его структура, и была приведена демонстрация работы программы.

4 Исследовательская часть

4.1 Технические характеристики ЭВМ

Замеры выполнялись на следующей машине:

— процессор 11th Gen Intel Core i5-1135G7 4.20 ГГц [13];

— оперативная память 8 Гб;

— операционная система EndeavourOS [14] Linux 6.17.9-arch1-1 x86_64.

Указанный процессор имеет 4 физических ядра и 8 логических [13]. Исследование проводилось на ноутбуке, подключённом в сеть, при включённом режиме производительности. При выполнении замеров были запущены лишь системные процессы.

4.2 Алгоритм проведения исследования

Исследование построено по следующему алгоритму.
В качестве варьируемых параметров выбраны:

— Глубина L-системы (число итераций) — $n \in \{2, 3, 4, 5, 6\}$;

— Разрешение финального изображения — $r \in \{256 \times 144, 426 \times 240, 640 \times 360, 854 \times 480, 1280 \times 720, 1920 \times 1080, 2560 \times 1440\}$;

— Разрешение карты теней — $s \in \{128 \times 128, 256 \times 256, 512 \times 512, 1024 \times 1024, 2048 \times 2048, 4096 \times 4096\}$.

Остальные параметры зафиксированы:

— аксиома: `"F"`;

— правило генерации: `"F[+FL][-FL][&FL][^FL]"`;

— угол ветвления: 25°;

— длина шага: 1.2;

— базовый радиус ствола: 0.25;

— коэффициент затухания радиуса: 0.6;

— минимальный радиус листа: 0.02;

— сегменты окружности: 8;

— источник света: направленный, направление $(-1, -2, -1)$;

— камера: орбитальная, фокус на центре сцены;

— тени: включены.

Все замеры времени выполнялись с помощью функции `std::chrono::high_resolution_clock` [8] и усреднялись по 10 запускам.

4.3 Зависимость времени генерации и рендеринга от глубины L-системы

В таблице 4.1 представлены результаты замеров времени генерации геометрии, времени рендеринга, количества треугольников и числа листьев от количества итераций.

Таблица 4.1 – Замеры времени характеристик сцены от количества итераций L-системы

Итерации	**Время генерации, мс**	**Время рендеринга, мс**	**Треуголь-ники**	**Листья**
2	0.407	171.014	1 184	74
3	1.841	217.264	5 746	508
4	8.146	304.548	25 317	2 763
5	40.899	535.007	112 864	13 951
6	206.739	1 319.613	525 921	70 140

На рисунке 4.1 представлен график зависимости времени генерации дерева от количества итераций L-системы.

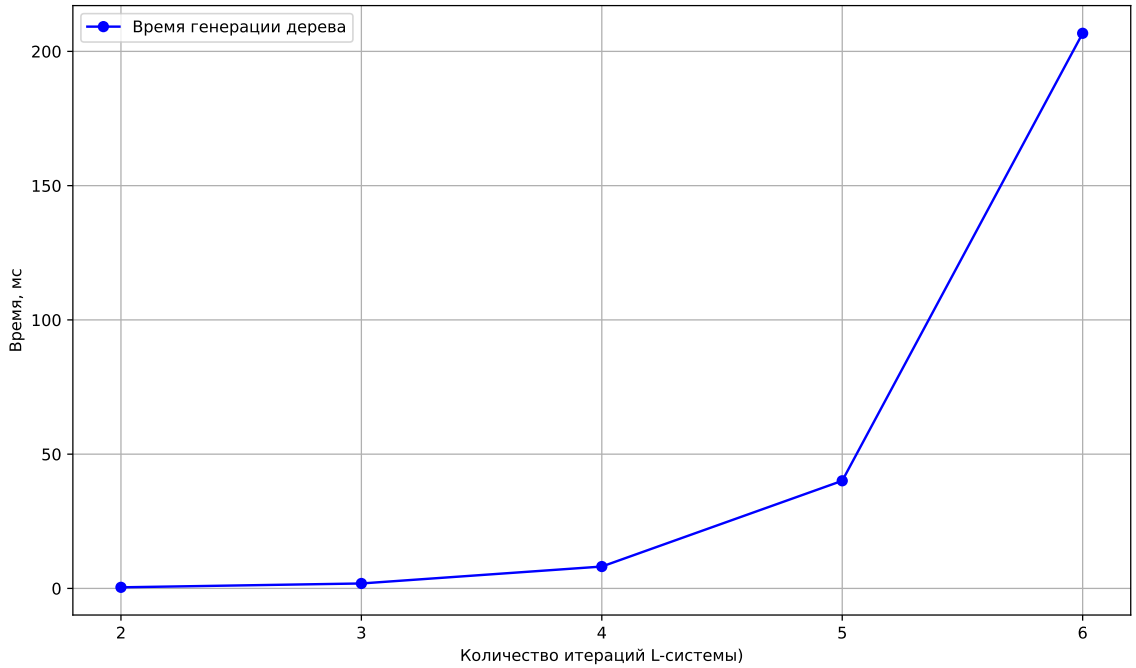

Рисунок 4.1 – График зависимости времени генерации дерева от количества итераций L-системы

На рисунке 4.2 представлен график зависимости времени рендеринга кадра от количества итераций L-системы.

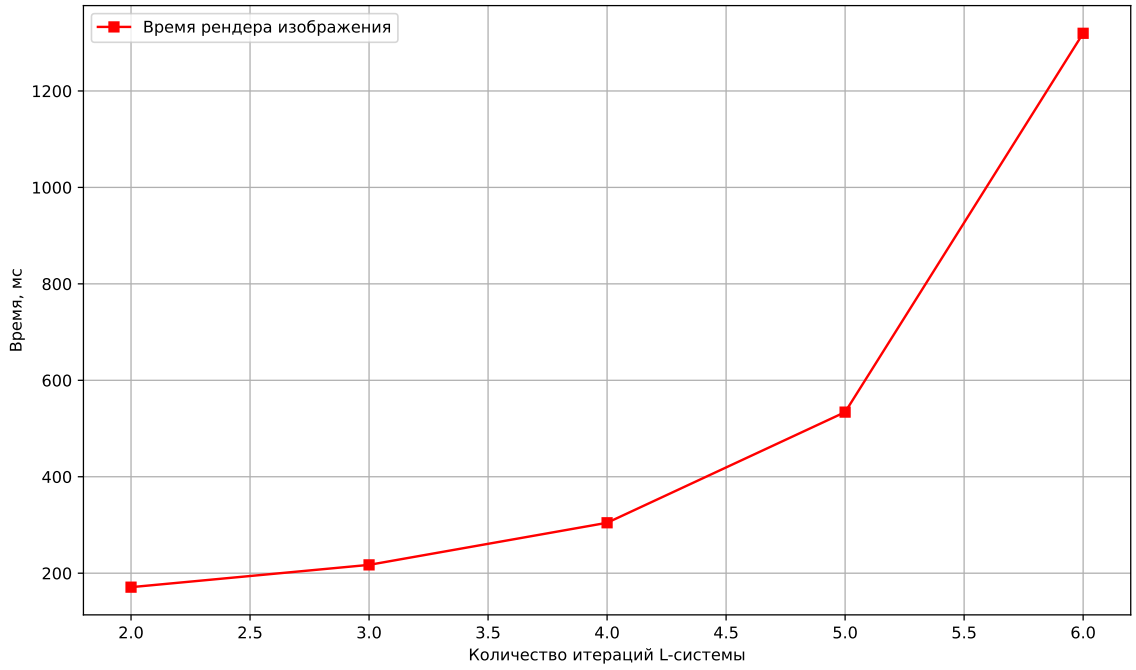

Рисунок 4.2 – График зависимости времени рендеринга кадра от количества итераций L-системы

Время генерации геометрии растёт по степенной зависимости: при увеличении глубины с 5 до 6 оно возрастает более чем в 5 раз.

Время рендеринга также следует степенной зависимости, но растёт медленнее, чем время генерации. Начиная с n=5, темп роста времени рендеринга заметно увеличивается.

4.4 Зависимость времени рендеринга при различных размерах изображения

В таблице 4.2 представлены результаты замеров времени рендеринга от разрешения изображения. При этом число глубины L-системы фиксировано и равно 5.

Таблица 4.2 – Результаты замеров времени рендеринга от разрешения изображения (при фиксированной глубине L-системы $n = 5$)

Разрешение изображения	**Время рендеринга, мс**
256×144	188.138
426×240	210.094
640×360	240.258
854×480	277.380
1280×720	363.993
1920×1080	533.528
2560×1440	757.984

На рисунке 4.4 представлен график зависимости времени рендеринга от разрешения изображения.

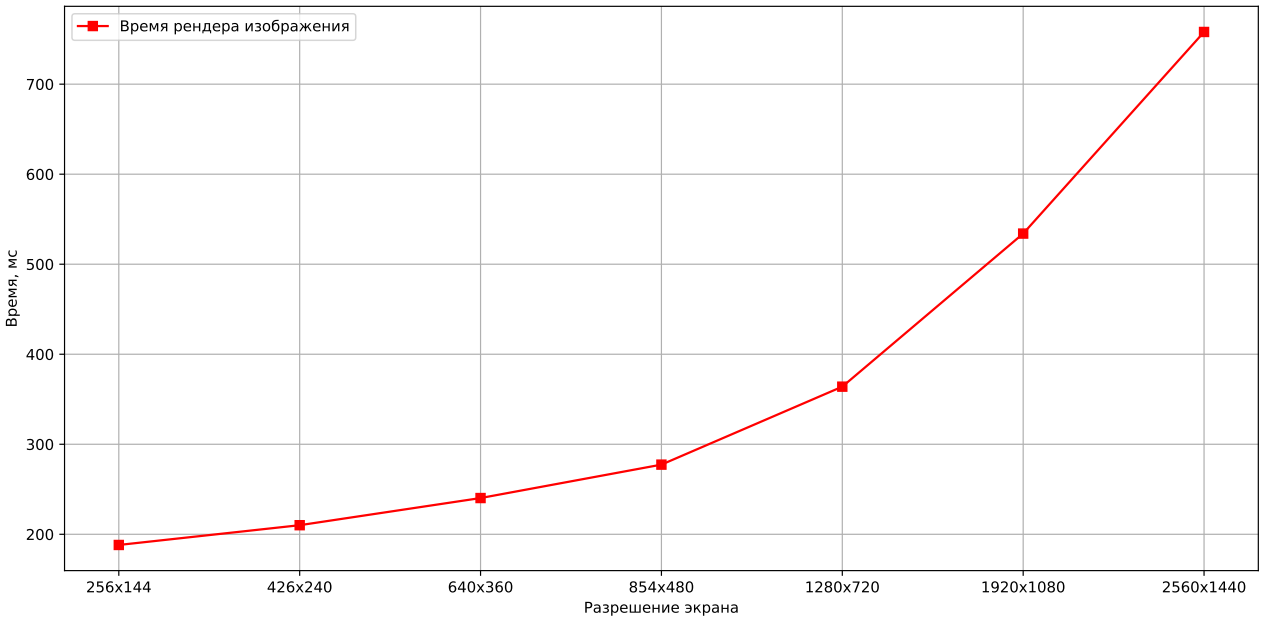

Рисунок 4.3 – График зависимости времени рендеринга от разрешения изображения

Время рендеринга растёт нелинейно с увеличением разрешения изображения, что соответствует степенной зависимости.

4.5 Зависимость времени рендеринга изображения при различных размеров теневой карты

В таблице 4.3 представлены результаты замеров времени рендеринга от разрешения теневой карты. При этом размер изображения фиксирован и равен 1920×1080, а количество итераций L-системы равно 5.

Таблица 4.3 – Результаты замеров времени рендеринга от разрешения изображения (при фиксированной глубине L-системы $n = 5$)

Разрешение теневой карты	**Время рендеринга, мс**
128	430.055
256	434.065
512	443.957
1024	466.171
2048	533.541
4096	763.706

На рисунке 4.4 представлен график зависимости времени рендеринга от

разрешения теневой карты.

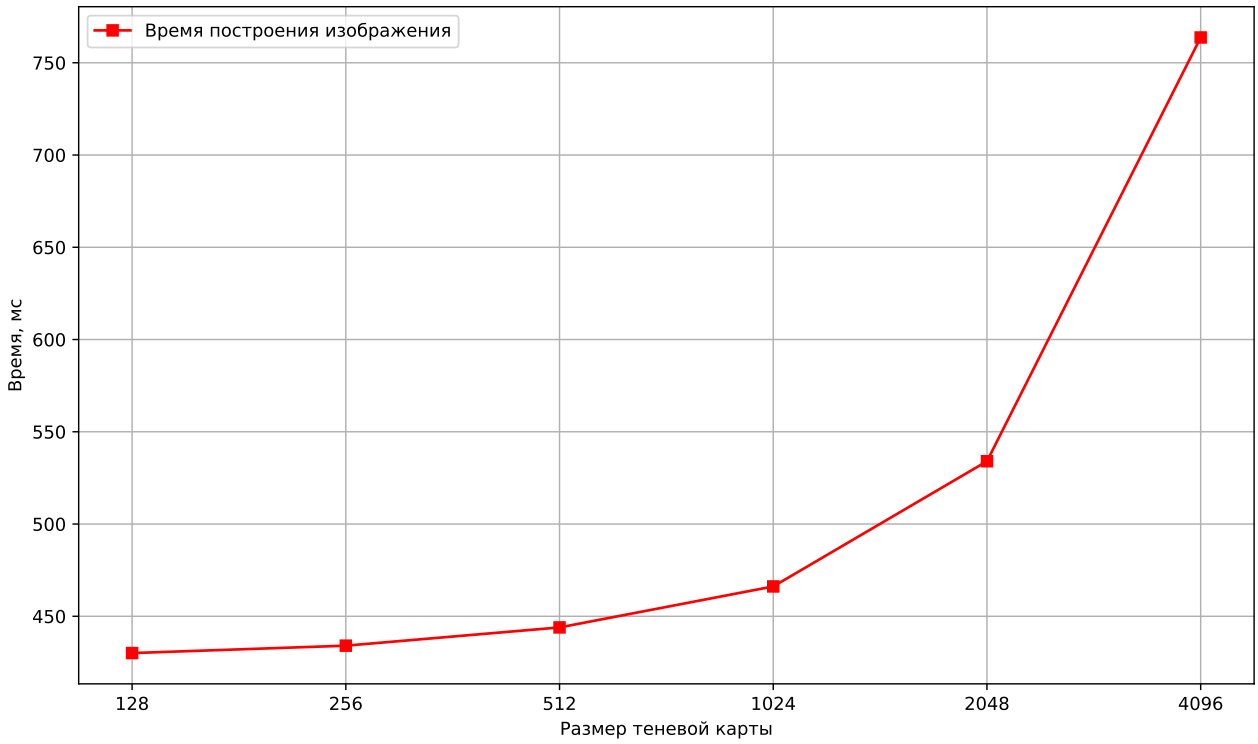

Рисунок 4.4 – График зависимости времени рендеринга от разрешения теневой карты

До разрешения 1024 × 1024 влияние теневой карты на производительность минимально: рост времени менее 9%. Однако начиная с 2048 × 2048 наблюдается резкое увеличение нагрузки — переход к 4096 × 4096 замедляет рендеринг на 43%. Это связано с квадратичным ростом объёма теневого буфера и, как следствие, количества операций сравнения глубины при затенении каждого пикселя.

4.6 Зависимость времени рендеринга изображения от угла обзора сцены

В таблице 4.4 представлены результаты замеров времени рендеринга в зависимости от угла поворота камеры. При этом разрешение изображения фиксировано (1920 × 1080), количество итераций L-системы равно 5, а разрешение теневой карты — 2048.

На рисунке 4.5 представлена столбчатая диаграмма, отражающая зависимости времени рендеринга изображения от угла обзора сцены.

Таблица 4.4 – Зависимость времени рендеринга от угла обзора камеры

Угол обзора, градусы	**Время рендеринга, мс**
0	1288.324
45	1282.183
90	1332.171
135	1313.475
180	1238.220
225	1289.903
270	1276.557
315	1226.312

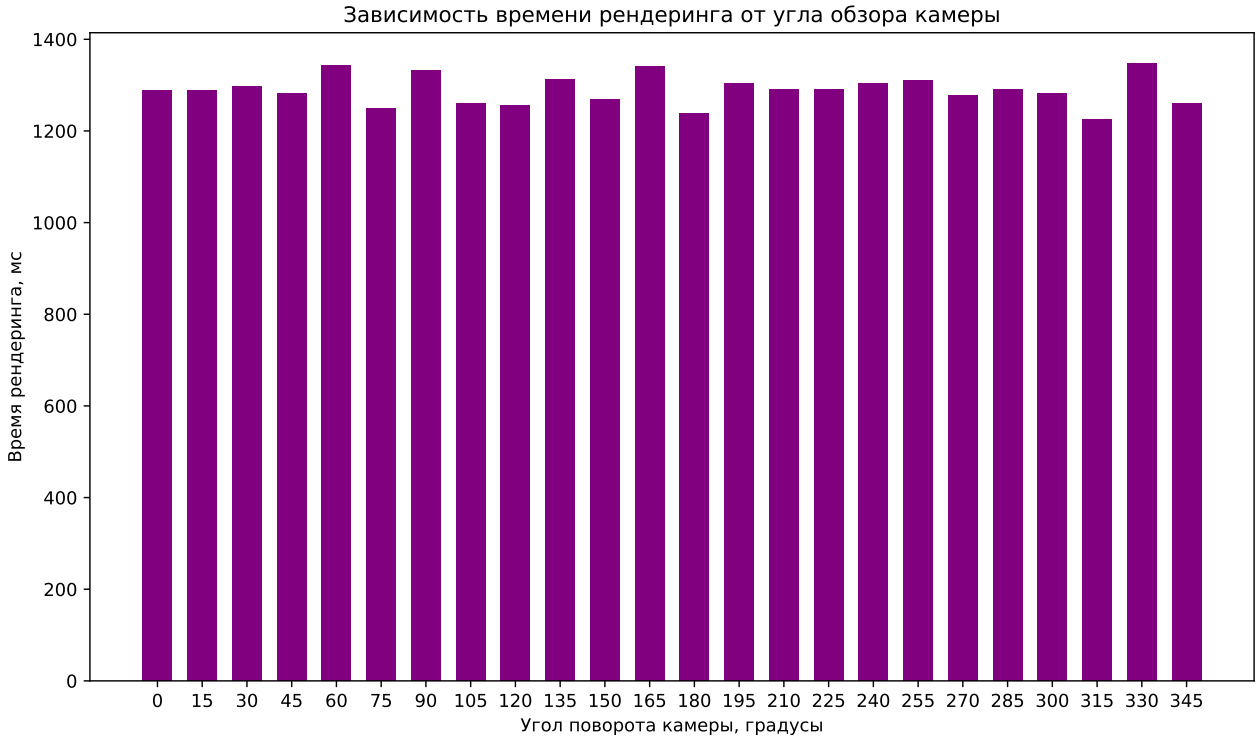

Рисунок 4.5 – Столбчатая диаграмма зависимости времени рендеринга от угла обзора сцены

По полученным таблице и гистограмме видно, что разница между минимальным и максимальным временем рендеринга составляет около 8.6%

4.7 Результаты исследования

Анализ полученных данных показал, что все исследуемые зависимости (время генерации от глубины L-системы, время рендеринга от разрешения изображения и от разрешения карты теней) имеют степенной характер.

Наиболее резкий рост наблюдается для времени генерации геометрии:

при увеличении числа итераций с 5 до 6 оно возрастает более чем в 5 раз, что связано с экспоненциальным ростом количества геометрических примитивов.

Время рендеринга также заметно увеличивается с ростом разрешения, особенно при переходе к высоким значениям 1920×1080 и выше для изображения, 2048×2048 и выше — для теневой карты. Это объясняется увеличением объёма данных, обрабатываемых на этапах растеризации и обработки теней.

В отличие от глубины L-системы или разрешения, угол обзора не влияет на объём генерируемой геометрии, поэтому его вклад в общее время рендеринга остаётся второстепенным.

4.8 Вывод

В данной части были приведены технический характеристики устройства, введена методология исследования, в соответствии с которой были проведены замеры, и проведён анализ полученных значений.

ЗАКЛЮЧЕНИЕ

Цель курсовой работы достигнута: разработано программное обеспечение, с графическим пользовательским интерфейсом для визуализации трёхмерных деревьев на основе L-систем с учётом параметров освещения, правил L-системы, углов ветвления и количества итераций генерации.

При этом в ходе работы были выполнены все поставленные задачи:

— проведён сравнительный анализ существующих методов генерации трёхмерных деревьев и сделаны выводы о применении L-систем для решения поставленной задачи;

— спроектирован и описан метод генерации трёхмерных деревьев на основе L-систем;

— выбраны средства для разработки ПО и реализована система визуализации;

— проведено исследование работы программы.

Разработанная система позволяет гибко настраивать параметры генерации и визуализации, а также оценивать их влияние на производительность и визуальный результат. Это делает её пригодной для дальнейшего развития — например, путём добавления анимации роста дерева или оптимизации с использованием графического процессора.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Mandelbrot B. B.* The Fractal Geometry of Nature. — San Francisco : W. H. Freeman, 1982. — 468 с.
2. *Runions A.*, *Lane B.*, *Prusinkiewicz P.* Modeling Trees with a Space Colonization Algorithm // Eurographics Workshop on Natural Phenomena. — Prague, 2007. — С. 63—70.
3. *Prusinkiewicz P.*, *Lindenmayer A.* The Algorithmic Beauty of Plants. — New York : Springer-Verlag, 1990. — 228 с.
4. *Prusinkiewicz P.*, *Rolland-Lagan A.-G.* Modeling Plant Growth and Development // Current Opinion in Plant Biology. — 2006. — Т. 9, № 5. — С. 580—586.
5. Computer Graphics: Principles and Practice / J. F. Hughes [и др.]. — 3-е изд. — Boston : Pearson Education, 2018. — 1264 с.
6. *Marschner S.*, *Shirley P.* Fundamentals of Computer Graphics. — 4-е изд. — Boca Raton : CRC Press, 2021. — 740 с.
7. *Гамбетта Г.* Компьютерная графика. — Санкт-Петербург : Питер, 2022. — 272 с.
8. *International Organization for Standardization.* ISO/IEC 14882:2020. Programming Languages — C++. — Geneva, Switzerland, 2020. — Standard.
9. Qt Documentation [Электронный ресурс]. — Режим доступа: `https://doc.qt.io/` (дата обращения: 10.12.2025).
10. Qt Widgets Designer Manual [Электронный ресурс]. — Режим доступа: `https://doc.qt.io/qt-6/qtdesigner-manual.html` (дата обращения: 10.12.2025).
11. CMake [Электронный ресурс]. — Режим доступа: `https://cmake.org/` (дата обращения: 10.12.2025).
12. CLion [Электронный ресурс]. — Режим доступа: `https://www.jetbrains.com/clion/` (дата обращения: 10.12.2025).

13. Intel Core i5-1135G7 Processor [Электронный ресурс]. — Режим доступа: https://www.intel.com/content/www/us/en/products/sku/208922/intel-core-i51135g7-processor8m-cache-up-to-4-20-ghz-with-ipu/specifications.html (дата обращения: 10.12.2025).

14. EndeavourOS [Электронный ресурс]. — Режим доступа: https://endeavouros.com/ (дата обращения: 10.12.2025).

ПРИЛОЖЕНИЕ А

Презентация к курсовой работе

Презентация содержит 21 слайд.

ПРИЛОЖЕНИЕ Б

```
void Rasterizer::drawScanline(int y, const ScreenVertex &left,
   const ScreenVertex &right, const glm::vec3 &cameraPos) {
  if (left.position.x > right.position.x)
    return;
  float dx = right.position.x - left.position.x;

  if (dx < 0.1f) {
    int x = (int)(left.position.x + 0.5f);
    if (x >= 0 && x < image.width())
      if (zBuffer->testAndSet(x, y, left.depth)) {
        QColor color = calculateColor(left, cameraPos);
        image.setPixel(x, y, color.rgb());
        pixelsDrawn++;
      }
    return;
  }
  int xStart = std::max(0, static_cast<int>(std::floor(left.
     position.x + 0.5f)));
  int xEnd = std::min(image.width() - 1, static_cast<int>(std::
     floor(right.position.x + 0.5f)));
  float invDx = 1.0f / dx;
  QRgb* scanline = reinterpret_cast<QRgb*>(image.scanLine(y));

  for (int x = xStart; x <= xEnd; x++) {
    float px = x + 0.5f;
    float t = (px - left.position.x) * invDx;
    t = glm::clamp(t, 0.0f, 1.0f);
    ScreenVertex pixel = interpolate(left, right, t);
    if (pixel.depth >= 0.0f && pixel.depth <= 1.0f) {
      if (pixel.depth < 0.01f)
        continue;
      if (zBuffer->testAndSet(x, y, pixel.depth)) {
        QColor color = calculateColor(pixel, cameraPos);
        scanline[x] = color.rgb();
        pixelsDrawn++;
      }
    }
  }
}
```

Листинг Б.1 – Реализация растеризации с учётом z-буффера

```
void ShadowMapRenderer::fillTriangle(glm::vec2 v0, glm::vec2 v1,
   glm::vec2 v2, float z0, float z1, float z2) {
  int minX = std::max(0, (int)std::floor(std::min({v0.x, v1.x, v2
     .x})));
  int maxX = std::min(width - 1, (int)std::ceil(std::max({v0.x,
     v1.x, v2.x})));
  int minY = std::max(0, (int)std::floor(std::min({v0.y, v1.y, v2
     .y})));
  int maxY = std::min(height - 1, (int)std::ceil(std::max({v0.y,
     v1.y, v2.y})));

  for (int y = minY; y <= maxY; ++y) {
    for (int x = minX; x <= maxX; ++x) {
      glm::vec2 p(x + 0.5f, y + 0.5f);

      float det = (v1.y - v2.y) * (v0.x - v2.x) + (v2.x - v1.x) *
          (v0.y - v2.y);
      if (std::abs(det) < 1e-8f) continue;

      float w0 = ((v1.y - v2.y) * (p.x - v2.x) + (v2.x - v1.x) *
         (p.y - v2.y)) / det;
      float w1 = ((v2.y - v0.y) * (p.x - v2.x) + (v0.x - v2.x) *
         (p.y - v2.y)) / det;
      float w2 = 1.0f - w0 - w1;

      if (w0 >= 0 && w1 >= 0 && w2 >= 0) {
        float z = w0 * z0 + w1 * z1 + w2 * z2;
        z = glm::clamp(z, 0.0f, 1.0f);

        int idx = y * width + x;
        if (idx >= 0 && idx < static_cast<int>(shadowMap.size()))
          if (zBuffer->testAndSet(x, y, z))
            shadowMap[idx] = z;
      }
    }
  }
}
```

Листинг Б.2 – Реализация построения карты теней

```
glm::vec3 Lighting::calculatePhong(const glm::vec3 &fragPos,
   const glm::vec3 &normal, const glm::vec3 &viewPos,
                                    const glm::vec3 &baseColor,
                                       const Light &light, const
                                       Material &material) {
  glm::vec3 N = glm::normalize(normal);
  glm::vec3 V = glm::normalize(viewPos - fragPos);
  glm::vec3 ambient = light.ambient * material.ambient *
     baseColor;
  glm::vec3 L;
  float attenuation = 1.0f;

  if (light.type == LightType::Directional) {
    L = -glm::normalize(light.direction);
  } else {
    L = glm::normalize(light.position - fragPos);
    float distance = glm::length(light.position - fragPos);
    attenuation = 1.0f / (
      light.constantAttenuation +
      light.linearAttenuation * distance +
      light.quadraticAttenuation * distance * distance
    );
  }
  float diff = glm::dot(N, L);
  glm::vec3 diffuse = light.diffuse * diff * material.diffuse *
     baseColor;
  glm::vec3 specular(0.0f);

  if (diff > 0.0f) {
    glm::vec3 H = glm::normalize(L + V);
    float spec = std::pow(std::max(0.0f, glm::dot(N, H)),
       material.shininess);
    specular = light.specular * spec * material.specular;
  }

  glm::vec3 result = ambient + (diffuse + specular) * attenuation
     ;
  return glm::clamp(result, 0.0f, 1.0f);
}
```

Листинг Б.3 – Реализация модели освещения по Блинну-Фонгу

```
QString LSystemGenerator::generate() const {
  QString current = axiom;

  for (int i = 0; i < iterations; i++) {
    QString next;

    for (const QChar &c : current) {
      if (rules.contains(c))
        next += rules[c];
      else
        next += c;
    }
    current = next;
  }

  return current;
}
```

Листинг Б.4 – Реализация алгоритма построения структуры дерева