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

ФАКУЛЬТЕТ ИУ «Информатика и системы управления»
КАФЕДРА ИУ-7 «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Конструирование и визуализация загородного посёлка»

Студент ИУ7-53Б (группа) _____ (подпись, дата) Звягин Д.О. (И.О. Фамилия)

Руководитель курсовой работы _____ (подпись, дата) Вишневская Т.И. (И.О. Фамилия)

2024 г.

СОДЕРЖАНИЕ

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В этой расчётно-пояснительной записке применяются следующие сокращения и обозначения.

ПО – Программное обеспечение.

Алгоритм QWFC (WFC) – Алгоритм квантового коллапса волновой функции.

ВВЕДЕНИЕ

Компьютерная графика – это область информатики, занимающаяся созданием, обработкой и отображением изображений с использованием вычислительных технологий. Она включает в себя как двумерную, так и трёхмерную графику, а также анимацию и визуализацию данных. Актуальность компьютерной графики возрастает с развитием технологий, таких как виртуальная и дополненная реальность, которые требуют высококачественной визуализации для создания реалистичных и интерактивных пользовательских опытов. В современных приложениях, от видеоигр до медицинской визуализации, компьютерная графика играет ключевую роль в представлении информации и взаимодействии с пользователями, что делает её важной областью для исследования и развития.

Целью данного курсового проекта является разработка ПО с пользовательским интерфейсом для генерации и визуализации загородного посёлка. Сцена содержит модели домов, источник света и камеру. Интерфейс пользователя должен позволять задавать параметры для генерации посёлка: размер домов, количество домов, ширина дорог, шаблоны расположения (кварталами или случайно).

Интерфейс приложения также должен предоставлять возможность для передвижения камеры и источника света.

Для достижения поставленной цели нужно решить следующие задачи:

— сравнение существующих алгоритмов процедурной генерации сцены;

— сравнение существующих алгоритмов компьютерной графики, использующихся для визуализации трёхмерной модели (сцены);

— выбор подходящих алгоритмов для решения поставленных задач;

— проектирование архитектуры и графического интерфейса ПО;

— выбор средств реализации ПО;

— разработка ПО;

— замер временных характеристик разработанного ПО.

1 Аналитическая часть

1.1 Формализация синтезируемой сцены

Сцена состоит из следующего набора объектов:

1) камера;
2) источник света;
3) модель загородного посёлка, состоящая из:
    - домов;
    - дорог;
    - деревьев.

Камера

Камера не является видимым объектом сцены. Она характеризуется только своим положением в пространстве и направлением просмотра. Изображение с камеры отображается в пользовательском интерфейсе ПО;

Источник света

Источник света отображает солнце, поэтому является всенаправленным и всегда находится на сравнительно большом расстоянии от других объектов сцены.

В пользовательском интерфейсе должна быть возможность задать положение источника света путём задания двух углов, которые задают направление, в котором будет находиться источник света относительно центра сцены.

Также в пользовательском интерфейсе должна быть возможность изменить расстояние, на котором находится источник света относительно центра сцены.

Модель загородного посёлка

Модель загородного посёлка является основной частью сцены. Сама модель должна генерироваться в соответствии с параметрами, заданными пользователем в пользовательском интерфейсе.

Далее формализуются объекты, входящие в сцену:

Ландшафт

В сцене не подразумевается использование сложного ландшафта, поэтому в качестве ландшафта используется плоское поле зелёного цвета (поле)

Дома

Дома, входящие в сцену состоят из геометрических примитивов, таких как:

— кубы;

— призмы.

Дома имеют простой прямоугольный фундамент разных размеров. В качестве крыши всегда используются призмы.

Дороги

Дороги должны проходить между домами и соединять их в улицы.

С точки зрения модели, дорогами являются прямоугольники, лежащие на плоскости ландшафта.

Деревья

Деревья также состоят из геометрических примитивов:

— параллелепипедов.

Все деревья будут представляться дубами, имеющими параллелепипед коричневого цвета в качестве ствола и тёмно-зелёную листву, отображаемую в виде куба.

Расположение объектов на модели

Основная плоскость является двумерной сеткой.

Пусть ячейка — это такой блок двумерной сетки который может:

— пустовать — в пределах ячейки нет объектов модели помимо основной плоскости;

— быть занят — в пределах ячейки содержится один или несколько объектов сцены.

В одной ячейке не может быть одновременно больше одного типа объектов (дом, дорога, дерево).

Объекты не обязательно должны занимать всю площадь ячейки.

Все дома должны стоять около хотя бы одной дороги.

Считается, что пространство за границей сцены может содержать ячейки любого типа, то есть находится в состоянии неопределённости.

1.2 Способы процедурной генерации сцены

Алгоритм квантового коллапса волновой функции

Алгоритм квантового коллапса волновой функции (далее - QWFC или WFC) — это метод генерации контента, который используется для создания двумерных и трёхмерных структур, таких как уровни в видеоиграх, текстуры и другие элементы. Он был разработан Максимом

Гуминым и основан на концепциях из квантовой механики, хотя и не имеет прямого отношения к физике [2].

Алгоритм работает путём заполнения сетки, каждая ячейка которой может принять одно из определённых состояний. Для заполнения сетки требуется предопределённый набор правил и приоритетов, который описывает то, какие ячейки могут быть "соседями"других ячеек. Изначально все ячейки находятся в состоянии суперпозиции – каждая может принять любое состояние (отсюда происходит слово "квантовый"в названии алгоритма). Затем, у случайной ячейки фиксируется одно из состояний (которое также выбирается случайно, но есть возможность задать приоритеты для тех или иных состояний). На основе этой зафиксированной ячейки, обновляется состояние остальной сетки, чтобы учесть появившиеся ограничения. Цикл повторяется, пока сетка не будет заполнена.

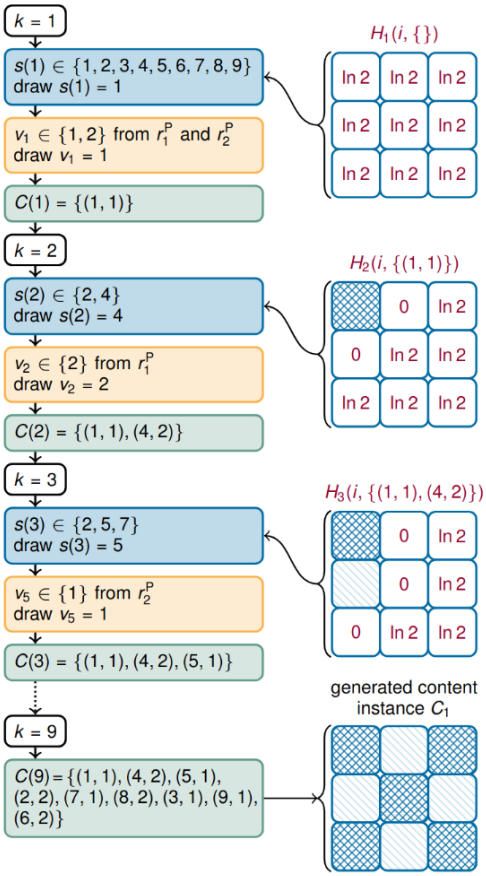

Рисунок 1.1 — Пример работы алгоритма QWFC

Этот алгоритм подходит для выполнения поставленной задачи (генерации загородного посёлка), так как он позволяет нам заполнить заданную плоскость в соответствии с ограничениями, а также предоставляет возможность влиять на результат посредством коэффициентов, которые можно задавать в пользовательском интерфейсе.

На рисунке 1.1 изображён пример работы алгоритма QWFC для создания изображения размера 3 на 3. В этом примере, возможны два типа ячеек — закрашенная или незакрашенная. Правило соседства для каждого типа ячеек гласит: ячейка не может соседствовать с ячейкой того же типа.

На рисунке число k означает итерацию алгоритма. Множество H — множество пар чисел, описывающих номер ячейки и зафиксированный тип. s — множество ячеек с минимальным количеством состояний. v — возможное фиксируемое состояние ячейки. C — выходной список, описывающий состояние матрицы.

1.3 Способы описания трёхмерных геометрических моделей на сцене

Существует три основных типа трёхмерных моделей [1]:

1) *каркасная модель* – описывает вершины и рёбра между ними;
2) *поверхностная модель* – описывает вершины, рёбра и поверхности;

3) *твердотельная модель* – описывает ещё и с какой стороны расположен материал;

Для решения поставленной задачи лучше всего подходит поверхностная модель. Каркасная модель не позволяет задать свойства плоскостей (такие, как цвет, параметры отражения и блеска), требуемые для правильного восприятия сцены. Твердотельная модель содержит дополнительные данные, не обязательные для достижения целей этой работы.

1.4 Сравнение алгоритмов удаления невидимых поверхностей

Алгоритмы удаления невидимых линий и поверхностей используются в машинной графике для удаления тех объектов (или частей объектов), которые перекрываются другими. Задача удаления невидимых линий и поверхностей является одной из наиболее сложных в машинной графике [3].

В этом разделе будут рассмотрены несколько алгоритмов удаления невидимых поверхностей.

Алгоритм Варнока

Алгоритм Варнока (или алгоритм художника) основывается на разбиении пространства на области, которые могут быть классифицированы, как сложные и простые.

Под сложной областью подразумевается такая область (часть пространства на экране), которая превышает по размеру один пиксель и в которой возникает любой из следующих случаев:

— в область попали несколько объектов;

— в область попал один объект, который занимает не всю область.

Простой областью, в свою очередь, называются такие области, которые либо не превышают по размеру один пиксель, либо не являются сложными.

Алгоритм рекурсивно делит экранное пространство на области, пока не достигает простых областей, которые могут быть закрашены тем или иным цветом.

Трассировка лучей

Трассировка лучей — это более сложный и точный метод, который использует физический принцип распространения света.

В этом методе лучи "выстреливаются"из камеры в сцену, и для каждого луча вычисляется, какие объекты он пересекает. При пересечении луча с объектом, луч может не только остановиться, но и отразиться или преломиться.

Трассировка лучей позволяет добиться высокой степени реализма, однако она требует значительных вычислительных ресурсов.

Алгоритм с Z-буфером

Алгоритм с Z-буфером (или глубинным буфером) — это один из простейших [3] методов удаления невидимых поверхностей.

Он использует дополнительный буфер для хранения информации о глубине каждого пикселя на экране. При отрисовке каждого объекта в сцене сравнивается его глубина с уже сохраненной в Z-буфере. Если новый объект ближе к камере, его цвет записывается в цветовой буфер, а его глубина — в Z-буфер.

Этот метод позволяет эффективно обрабатывать сложные сцены и обеспечивает высокую производительность, что позволяет использовать его и при отрисовке сцен в реальном времени.

Выбор подходящего алгоритма

Было принято решение использовать алгоритм с Z-буфером.

Алгоритм Варнока хоть и предлагает относительную простоту, он будет не совсем оптимален для отрисовки большого количества примитивов, из которых будет состоять модель загородного посёлка.

Алгоритм трассировки лучей, в свою очередь, позволяет достичь относительного реализма по сравнению с другими алгоритмами, но он делает это затрачивая большое количество вычислительных ресурсов. Поскольку целью работы не является реалистичная визуализация модели, затраты на алгоритм трассировки лучей не будут оправданными.

Алгоритм с Z-буфером, в свою очередь, предлагает и относительную скорость работы и достаточное для цели работы качество изображения.

1.5 Сравнение алгоритмов построения теней

В данной работе будет использоваться метод теневых карт для построения теней на изображении.

Метод теневых карт

Метод теневых карт основывается на построении карты теней методом заполнения Z-буфера с точки зрения источника света и сравнения этого буфера с точки зрения камеры для правильного затенения пикселей [4].

Процесс применения этого метода можно разбить на следующие этапы:

1) создание теневых карт;
2) сравнение глубин пикселей;
3) применение освещения.

Создание теневых карт

На этом этапе информация о сцене заносится в Z-буфер с точки зрения источника света. Вместо цвета и яркости пикселей в Z-буфер будет сохраняться значение глубины каждого фрагмента изображения. В результате такого заполнения, будет получена текстура, называемая теневой картой. Эта текстура будет использоваться для определения, находится ли фрагмент изображения в тени.

Сравнение глубин пикселей

На втором этапе информация о сцене заносится в Z-буфер с точки зрения камеры. Для каждого фрагмента изображения координаты преобразуются в координаты теневой карты и значение глубины фрагмента сравниваются с соответствующим значением в теневой карте. Если значение глубины фрагмента больше, чем значение в теневой карте, фрагмент находится в тени.

Применение освещения

На последнем этапе полученная информация о тенях применяется к изображению: те пиксели (фрагменты) которые не находятся в тени, становятся светлее, а остальные - затеняются, в зависимости от интенсивности света.

Преимущества и недостатки

Метод теневых карт в сочетании с алгоритмом Z-буфера имеет свои преимущества и недостатки. К преимуществам можно отнести:

— высокую производительность для динамических сцен;

— возможность создания реалистичных теней для сложных объектов.

Однако существуют и недостатки:

— при низком разрешении теневых карт, могут наблюдаться проблемы с качеством изображения;

— ограниченная точность теней для объектов: чем дальше расположен объект от источника света, тем менее точной будет его тень.

Вывод

Метод теневых карт достаточен для достижения целей работы. Поскольку сцена имеет всего один источник света, не придётся производить большое количество вычислений при визуализации сцены. Качество теней должно быть достаточным, так как источник света расположен на условно бесконечном расстоянии от сцены, а следовательно равноудалён от всех объектов.

2 Конструкторская часть

2.1 Требования к программному обеспечению

Программа должна предоставлять графический интерфейс со следующим функционалом:

— изменение параметров генерации модели загородного посёлка;

— возможность генерации загородного посёлка в соответствии с заданными параметрами;

— изменение положения источника света;

— изменение положения и направления камеры.

Программа должна соответствовать следующим требованиям:

— генерация посёлка происходит при нажатии на соответствующую кнопку в интерфейсе;

— генерация осуществляется в соответствии с параметрами, заданными пользователем;

— программа должна корректно обрабатывать ввод некорректных данных.

2.2 Описание структур данных

Структуры данных, описывающие содержимое сцены

1) Сцена представляет собой список, содержащий указатели на объекты сцены (дома, дороги и т.п.);

2) Объекты сцены включают в себя следующие данные:

   — массив поверхностей;

   — матрица преобразований.

3) Поверхность включает в себя следующие данные:

   — массив вершин;

   — вектор нормали;

   — спектральные характеристики.

4) Вершины включают в себя следующие данные:

   — координаты вершины.

5) Источник света включает в себя следующие данные:

   — координаты источника.

6) Камера включает в себя следующие данные:

   — координаты камеры;

   — матрица преобразований направления камеры.

Следует отметить, что матрица преобразований камеры описывает как её координаты, так и её направление.

Камера с единичной матрицей трансформации находится в точке с координатами $(0, 0, 0)$

и направлена вдоль оси X.

Структуры данных, необходимые для генерации сцены

В процессе генерации, содержимое сцены описывается матрицей (двумерным массивом), каждая ячейка которой означает нахождение определённого объекта на координатах, соответствующих данной ячейке матрицы.

Объекты матрицы содержат следующие данные:

— идентификатор типа объекта;

— таблица возможных идентификаторов типов соседних клеток (по одной записи на каждую соседнюю ячейку).

2.3 Алгоритм построения изображения

К алгоритму построения алгоритма приведены следующие требования:

— на вход подаётся список объектов сцены, описание источника света и камеры;

— на выход подаётся построенное изображение.

Алгоритм построения изображения представлен на рисунке 2.1.

В следующих пунктах будут подробнее описаны этапы данного алгоритма.

2.3.1 Матрицы преобразования

В данном алгоритме (и структурах данных) под словосочетанием “матрица преобразования” подразумеваются такая матрица, при умножении всех точек объекта на которую, будет получен преобразованный объект [1].

Данная матрица, как правило, получается как результат произведения нескольких матриц аффинных преобразований.

В данной работе используются следующие аффинные преобразования:

— перенос;

— масштабирование;

— поворот.

Матрица переноса представляется следующим образом:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ dx & dy & dz & 1 \end{pmatrix}$$

здесь, dx, dy, dz — это смещения по осям Ox, Oy и Oz соответственно.

Матрица масштабирования представляется следующим образом:

Рисунок 2.1 — Алгоритм построения изображения

$$\begin{pmatrix} kx & 0 & 0 & 0 \\ 0 & ky & 0 & 0 \\ 0 & 0 & kz & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

здесь, kx, ky, kz — это коэффициенты масштабирования по осям Ox, Oy и Oz соответственно.

Матрицы поворота представляются по-разному для каждой оси.

Матрица поворота вокруг оси Ox представляется следующим образом:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\alpha & sin\alpha & 0 \\ 0 & -sin\alpha & cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Матрица поворота вокруг оси Oy представляется следующим образом:

$$\begin{pmatrix} cos\alpha & 0 & -sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ sin\alpha & 0 & cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Матрица поворота вокруг оси Oz представляется следующим образом:

$$\begin{pmatrix} cos\alpha & sin\alpha & 0 & 0 \\ -sin\alpha & cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Во всех матрицах вращения, α — угол поворота вокруг соответствующей оси в радианах.

2.3.2 Приведение к пространству камеры

Поскольку положение и направление камеры описывается матрицей преобразований, приведение точки к пространству координат камеры равносильно применению к данной точке обратных преобразований, применённых к камере.

Например при смещении камеры на 10 точек по оси Ox, относительно камеры объекты сцены сместятся на -10 точек по той же оси. При повороте камеры на 10 градусов вокруг оси Ox, объекты сцены повернутся на -10 градусов вокруг той же оси относительно координат камеры.

Пользуясь этим свойством, для упрощения перевода координат точек в пространство камеры, можно воспользоваться следующим приёмом: при преобразовании камеры (координат и направления) можно записывать в её матрицу преобразования обратные значения.

Таким образом, для приведения точки к пространству камеры, достаточно применить к этой точке матрицу преобразования камеры.

2.3.3 Удаление невидимых поверхностей

Удаление невидимых поверхностей для этой работы будет проводиться по двум критериям:

— скалярное произведение нормали поверхности и вектора направления камеры;

— попадание поверхности в пирамиду видимости камеры.

Первый критерий позволяет удалить нелицевые (с точки зрения камеры) поверхности. Это позволяет существенно уменьшить количество поверхностей, которые придётся обрабатывать при построении буфера кадра.

Для определения видимости поверхности, достаточно вычислить значения скалярного произведения $(\vec{N}, \vec{V})$, где $\vec{N}$ — вектор нормали поверхности, а $\vec{V}$ — вектор направления камеры. Тогда, при значении $(\vec{N}, \vec{V})$ меньше нуля, поверхность будет лицевой, а во всех остальных случаях — нелицевой.

Следует отметить, что для правильного определения нелицевых граней, их необходимо также переводить в пространство перспективной проекции, если оно используется при построении сцены.

Второй критерий позволяет удалить те поверхности, которые не попадают в поле зрения камеры. Часто для упрощения вычислений этого критерия, используется не сама поверхность (которая может задаваться сложной фигурой), а параллелепипед, вписывающий данную поверхность в себя.

Удобно использовать такой параллелепипед, стороны которого параллельны осям системы координат камеры. Тогда, для вычисления его параметров будет достаточно вычислить максимальные и минимальные значения каждой из координат.

После этого, выполняется определение видимости этого параллелепипеда путём пересечения его с пирамидой видимости камеры [3].

Определение пересечения бесконечной пирамиды и прямоугольника тривиально, и сводится к определению определению пересечения прямых.

2.3.4 Алгоритм Z-буфера

Алгоритм Z-буфера используется для растеризации поверхностей объектов. Поскольку под поверхностью в рамках этой работы подразумевается часть плоскости, ограниченная некоторым многоугольником, задача растеризации многоугольника и внесения его в буфер сводится к определению принадлежности точки этому многоугольнику и поиску значения координаты z этой точки.

Поскольку плоскость можно задать тремя точками, достаточно взять любые три точки из многоугольника, которым задаётся поверхность объекта. Сделать это можно следующим образом:

Пусть заданы точки A, B, C, задающие плоскость в пространстве камеры. Тогда вектор нормали $\vec{N}$ к этой плоскости можно найти по формуле

$$\vec{N} = \vec{AB} \times \vec{AC}$$

Теперь, можно записать уравнение плоскости в виде

$$N_x(x - x_a) + N_y(y - y_a) + N_z(z - z_a) = 0$$

Где N_x, N_y, N_z — компоненты вектора $\vec{N}$ по осям x, y и z, а x_a, y_a и z_a — координаты произвольной точки этой плоскости. Можно выбрать любую точку, например A.

Раскрыв скобки, можно перейти к следующему виду:

$$N_x x + N_y y + N_z z - (N_x x_a + N_y y_a + N_z z_a) = 0 \tag{2.1}$$

Поскольку занесение координат в Z-буфер можно производить построчно (последовательно для каждой строки буфера), вычисление координаты z можно производить пошаговым способом (в пределах одной строки) [3].

Поскольку в пределах одной строки, $y = const$, глубина пикселя данной строки с координатой $x_1 = x + \Delta x$ можно вычислить по формуле 2.2.

$$z_1 = z - \frac{a}{c} \tag{2.2}$$

Где a и c — коэффициенты уравнения плоскости при x и z из уравнения 2.1.

Для первой точки на каждой из строк, значение координаты z придётся вычислять из уравнения плоскости.

Для определения точек, принадлежащих многоугольникам, можно использовать алгоритмы растровой развёртки многоугольников.

На рисунке 2.2 изображён алгоритм Z-буфера.

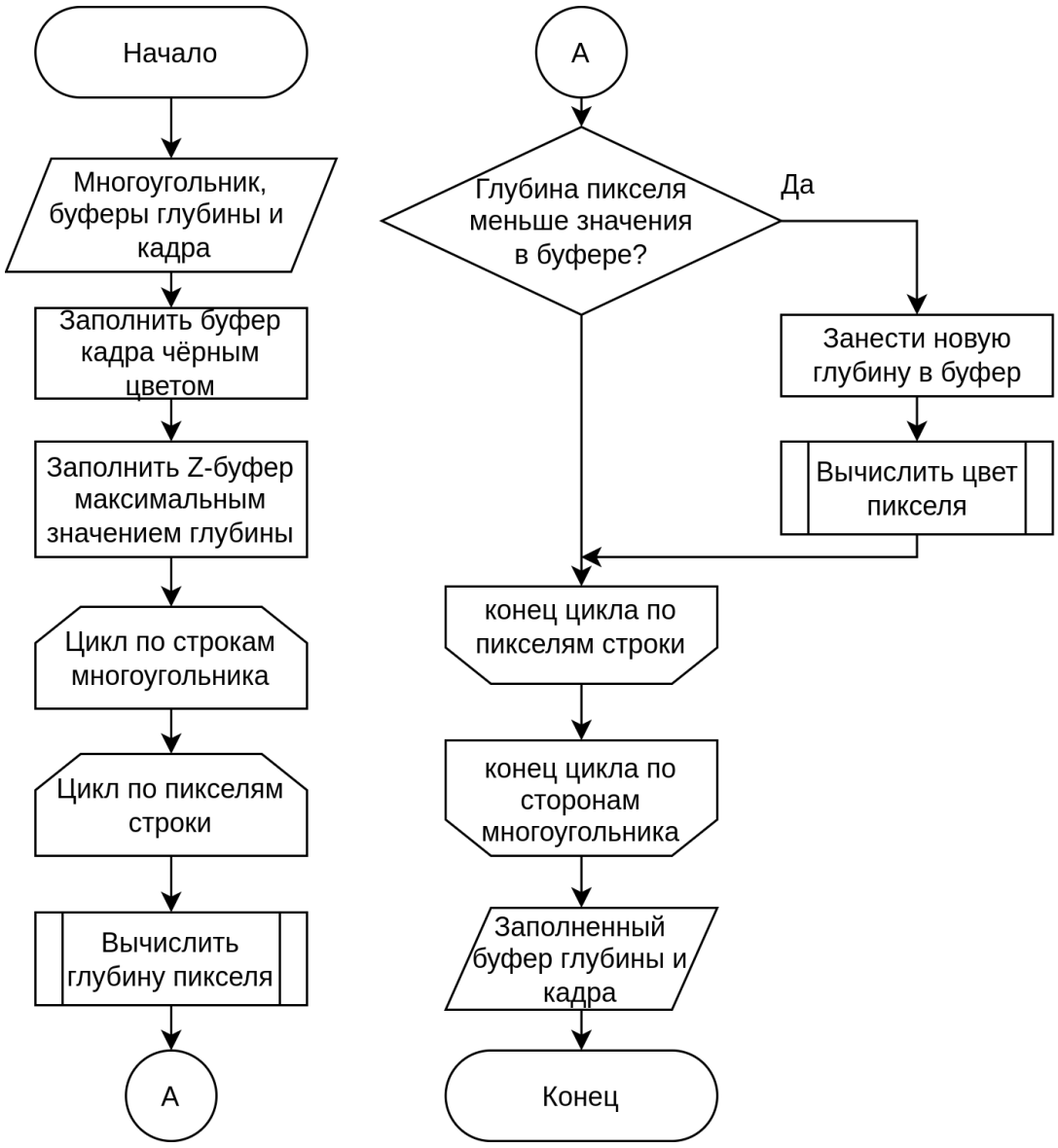

Рисунок 2.2 — Алгоритм Z-буфера для многоугольника

2.4 Алгоритм генерации сцены

Для генерации сцены будет использован алгоритм квантового коллапса волновой функции.

Для корректной работы алгоритма крайне важно правильное задание начальных ограничений. Под ограничением подразумевается возможность или невозможность появления определённой ячейки вплотную к другой ячейке с определённой стороны [2].

Например если в одной ячейке содержится прямой участок дороги, то к тем сторонам ячейки, которые пересекает дорога, могут быть “присоединены” только такие ячейки, в которых к этим же сторонам будут присоединены другие участки дороги. Это ограничение изображено на рисунке 2.3.

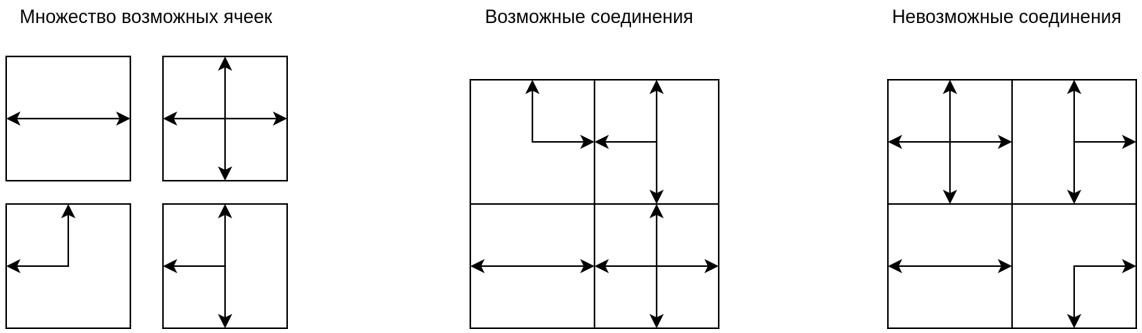

Рисунок 2.3 — Визуализация ограничений генерации

Описание ограничений выполняется вручную для каждой возможной ячейки и её поворота, однако этот процесс можно значительно упростить, если учитывать симметрию и возможные повороты ячеек на уровне программного кода.

Пусть необходимо заполнить матрицу генерации M размером 50 на 50 ячеек. Изначально все ячейки находятся в состоянии суперпозиции, то есть каждая ячейка может принять любое возможное значение. Возможные значения, в свою очередь, определяются пересечением ограничений соседних ячеек. Если какой-либо из соседей не находится в строго определённом состоянии, его ограничениями будет являться объединение ограничений каждого из возможных состояний.

Пока все ячейки матрицы не примут строго определённое состояние, выполняется цикл, который выбирает ячейку с наименьшим количеством возможных состояний и фиксирует её значение, выбирая случайным образом одно из возможных значений.

Предусмотрена возможность повлиять на вероятность выбора того или иного состояния ячейки путём задания некоторого веса. Например, в случае с дорогами, можно повысить вероятность появления поворота и уменьшить вероятность появления перекрёстка.

После обновления ячейки в матрице, ограничения её соседей обновляются.

Алгоритм коллапса волновой функции представлен на рисунке 2.4.

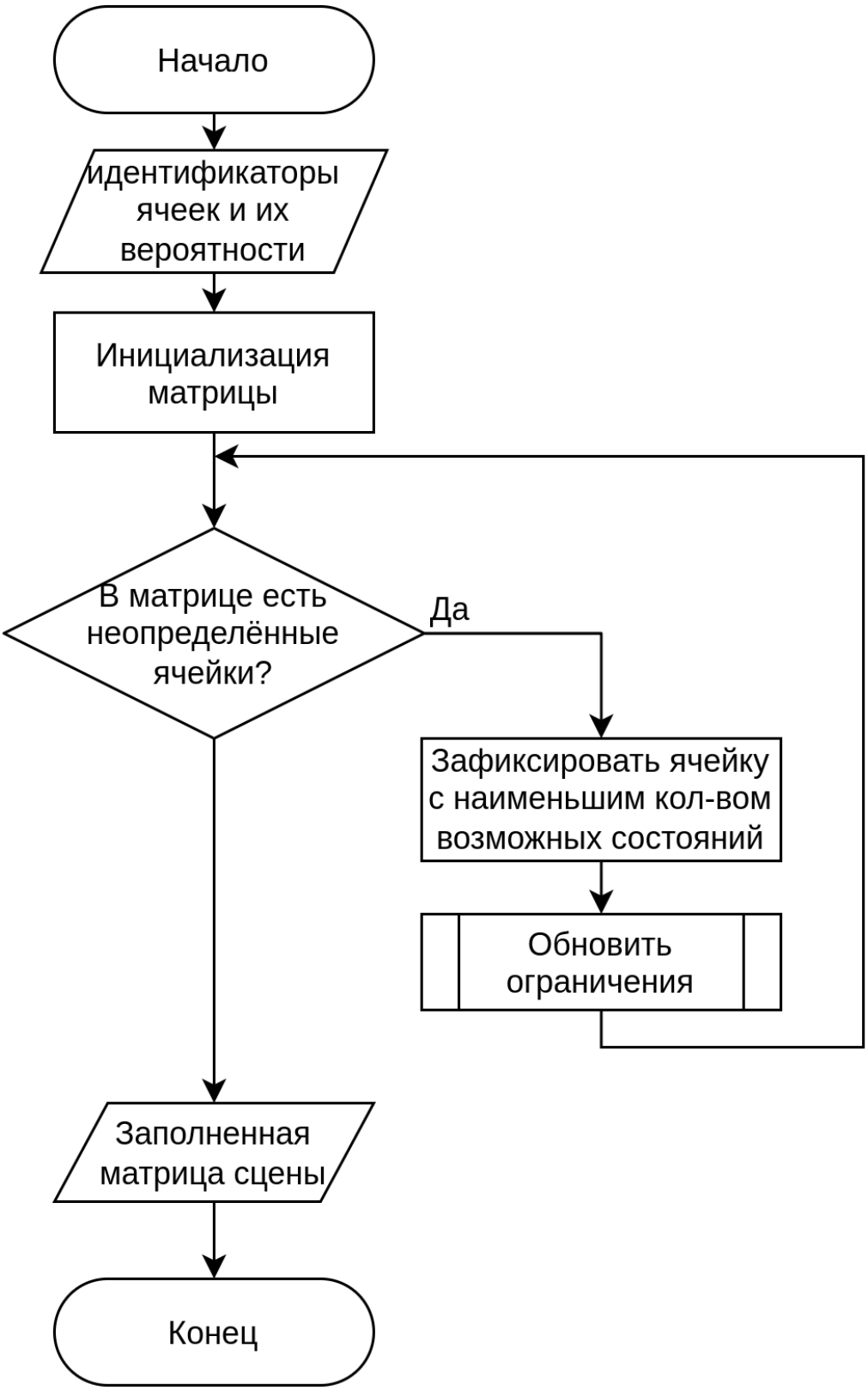

Рисунок 2.4 — Алгоритм квантового коллапса волновой функции

В рамках данной работы будут использованы 10 типов ячеек:

— дорожный перекрёсток;

— горизонтальный прямой участок дороги;

— вертикальный прямой участок дороги;

— 4 типа угловых участков дороги;

— пустырь;

— дерево;

— дом.

Поскольку ячеек для описания разных типов дорог семь, а остальных ячеек всего три, необходимо снизить вероятность появления каждого определённого типа дороги в семь раз. Такое снижение вероятности сравняло бы шансы появления дороги или другого типа объекта в той или иной ячейке матрицы, однако вероятности дорог снижены неравномерно для уменьшения количества перекрёстков и поворотов.

Таким образом, получаем скорректированные вероятности появления:

— перекрёстка — 10%;

— прямых участков — 25%;

— угловых участков — 10%.

Соотношение других типов объектов (и дорог) друг к другу также можно корректировать в интерфейсе приложения.

3 Технологическая часть

3.1 Средства реализации

Язык программирования

Для реализации ПО был выбран язык программирования C++ [6] стандарта “C++20” по следующим причинам:

— данный язык является объектно-ориентированным, что позволяет проектировать сложные системы взаимодействия объектов для обработки и визуализации;

— стандартная библиотека языка достаточна для осуществления всех спроектированных алгоритмов;

— стандартная библиотека языка предоставляет средства для профилирования и исследования временных затрат;

— для языка разработано большое количество библиотек, расширяющих возможности языка.

В процессе разработки ПО были использованы внешние библиотеки:

— Qt6 [7] для осуществления интерфейса программы;

— OneApi TBB [8] для параллелизации процессов.

Система сборки

В качестве системы сборки проекта была выбрана система CMake [10], так как она предназначена для работы с языками C и C++ и предоставляет возможность контроля над подключаемыми файлами и библиотеками.

Среда разработки

Для разработки была выбрана Visual Studio Code [9], так как она предоставляет достаточный интерфейс для написания и отладки кода, а также обеспечивает возможность работы с избранной системой сборки.

Для редактирования интерфейса была использована программа QtDesigner, поставляемая вместе с библиотекой Qt.

3.2 Структура классов

Были разработаны следующие классы:

— *Point2D* — для работы с точками на плоскости;

— *Point3D* — для работы с точками в пространстве;

— *Triangle* — для работы с треугольниками в пространстве;

— *Plane* — для работы с плоскостями;

— *Color* — описывающий цвет;

— *Face* — для работы с гранями объектов;

— *Square* — класс, описывающий квадрат в пространстве;

— *Cube* — класс, описывающий куб в пространстве;

— *BaseModel* — базовый класс модели на сцене;

— *FaceModel* — класс для работы с поверхностными моделями;

— *House* — класс, описывающий модель дома;

— *Tree* — класс, описывающий модель дерева;

— *RoadXX* — классы, описывающие модели дорог разных видов;

— *TransformationMatrix* — класс, описывающий матрицу трансформации моделей;

— *Scene* — класс, хранящий объекты сцены;

— *BaseCamera* — базовый класс, описывающий камеры;

— *PerspectiveCamera* — камера, отображающая точки по принципу перспективной проекции;

— *ShadowMap* — класс, предоставляющий доступ к карте теней;

— *BaseVisitor* — базовый класс для паттерна программирования “посетитель”;

— *ShadowMapVisitor* — “посетитель”, производящий вычисление карты теней;

— *RenderVisitor* — “посетитель”, производящий вычисление буфера кадра;

— *QWFC* — класс, осуществляющий генерацию матрицы сцены;

— *Cell* — класс, описывающий ячейку для алгоритма QWFC;

— *Rule* — класс, описывающий правило для алгоритма QWFC.

Схема разработанных классов изображена на рисунке 3.1.

Из перечисления выше исключены те классы и структуры, которые используются только для связи с интерфейсом и библиотекой Qt во избежание загромождения.

3.3 Реализация ПО

Процесс отрисовки изображения с помощью алгоритма Z-буфера происходит при использовании класса RenderVisitor на сцену. Эта функция представлена на листинге А.1.

На этом этапе происходит инициализация буфера глубины и кадра. Затем RenderVisitor обрабатывает каждый объект на сцене. Этот процесс происходит с использованием библиотеки tbb, поэтому каждый объект обрабатывается параллельно, что позволяет сильно сократить время создания кадра (см. исследовательскую часть).

Поскольку алгоритм Z-буфера не рассчитан на одновременную обработку несколькими потоками, каждый объект создаёт набор буферов кадра, которые затем объединяются для получения выходного изображения.

Функция обработки модели объектом RenderVisitor представлена на листинге А.2.

Из каждого объекта извлекаются грани, затем происходит фильтрация алгоритмами отсечения невидимых поверхностей невидимых граней. Оставшиеся грани создают буферы кадра

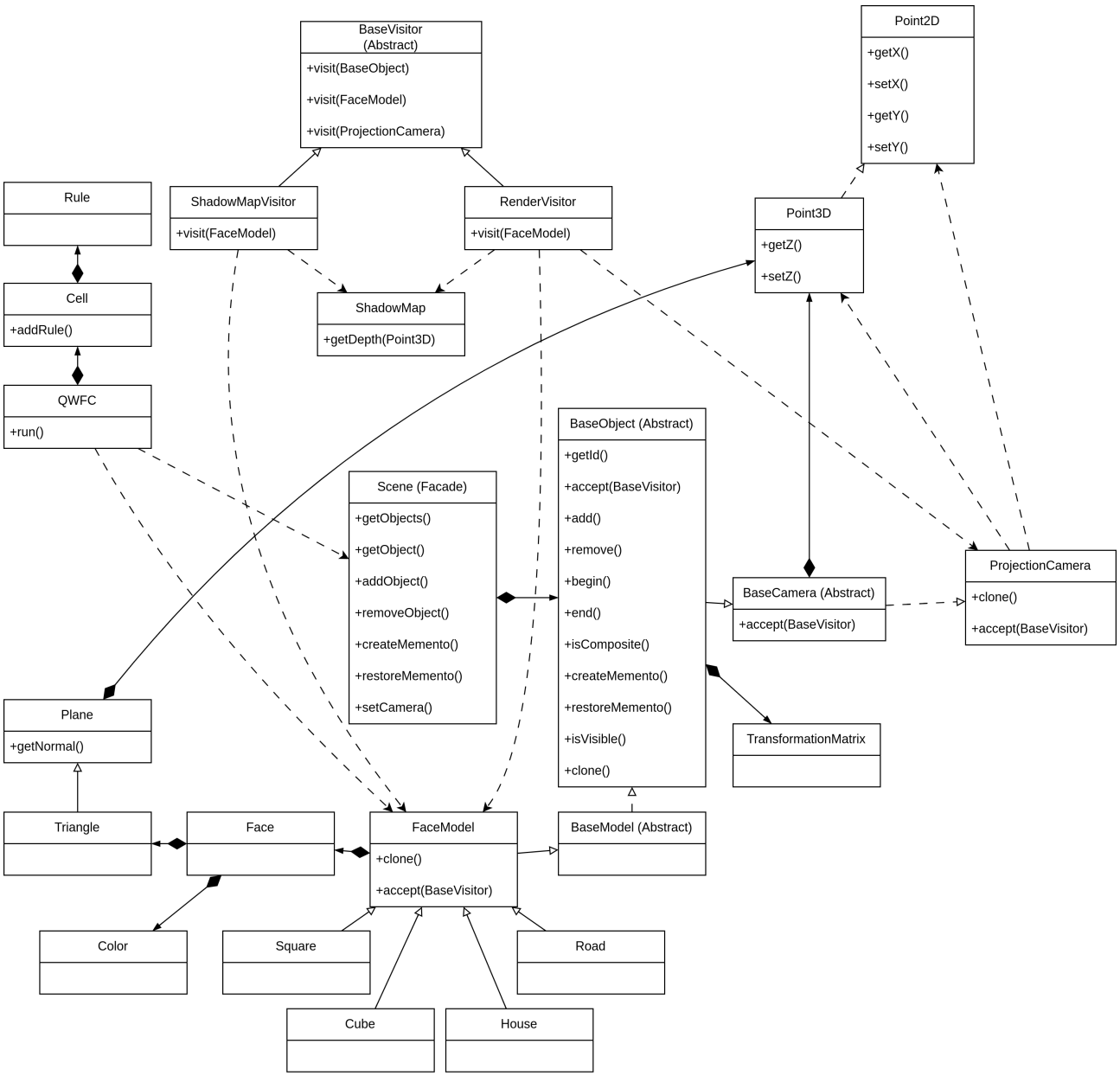

Рисунок 3.1 — Диаграмма классов

и сохраняют их в общий массив. Обработка каждой грани также происходит одновременно. Поскольку все грани представляются треугольниками, процесс растеризации был оптимизирован для них. Координаты глубины и мировых координат вычисляются итерационно для каждой сканирующей строки треугольника в соответствии с алгоритмом, разработанным в аналитической части. Также в этом методе происходит сравнение координат точек с координатами в карте теней.

Вычисление карты теней выполняется аналогично заполнению глубины буфера кадра, но вместо матрицы преобразования камеры используется матрица преобразования источника света.

Заполнение буфера кадра сцены происходит когда изменяется сцена, то есть меняется положение и/или направление камеры, изменяется положение источника света или изменяются объекты на сцене.

Расчёт карты теней происходит только при изменении положения источника света или изменении объектов на сцене.

Метод collapseCell класса QWFC представлен на листинге А.3. Фиксирование состояния ячейки производится в соответствии с приоритетами, их учёт можно увидеть в этой функции.

После фиксирования состояния ячейки состояние соседних ячеек рекурсивно обновляется.

Для тестирования алгоритма QWFC, вместе с основным файлом программы “app”, собирается файл “qwf”, производящий замеры времени генерации сцены.

Пользовательский интерфейс

Пользовательский интерфейс состоит из описанных ниже групп управления.

Группа управления генерацией сцены

В данной группе управления пользователю доступны:

— два поля ввода для задания размеров матрицы сцены;

— четыре поля ввода для задания вероятностей появления разных типов объектов;

— кнопка “создать”, осуществляющая генерацию новой сцены.

Группа управления положения источника света

В данной группе управления пользователю доступны два ползунка для управления поворотом источника вокруг осей Ox и Oy.

Управление камерой производится с помощью клавиатуры:

— клавиши “W”, “A”, “S”, “D” используются для движения камеры вперёд, влево, назад и вправо соответственно;

— клавиши “I”, “J”, “K”, “L” используются для поворотов камеры вверх, влево, вниз и вправо соответственно;

— клавиши “Пробел” и “B” используются для движения камеры вверх и вниз вдоль оси Oy соответственно.

3.4 Демонстрация работы программы

На рисунках 3.2-3.6 представлены примеры работы программы.

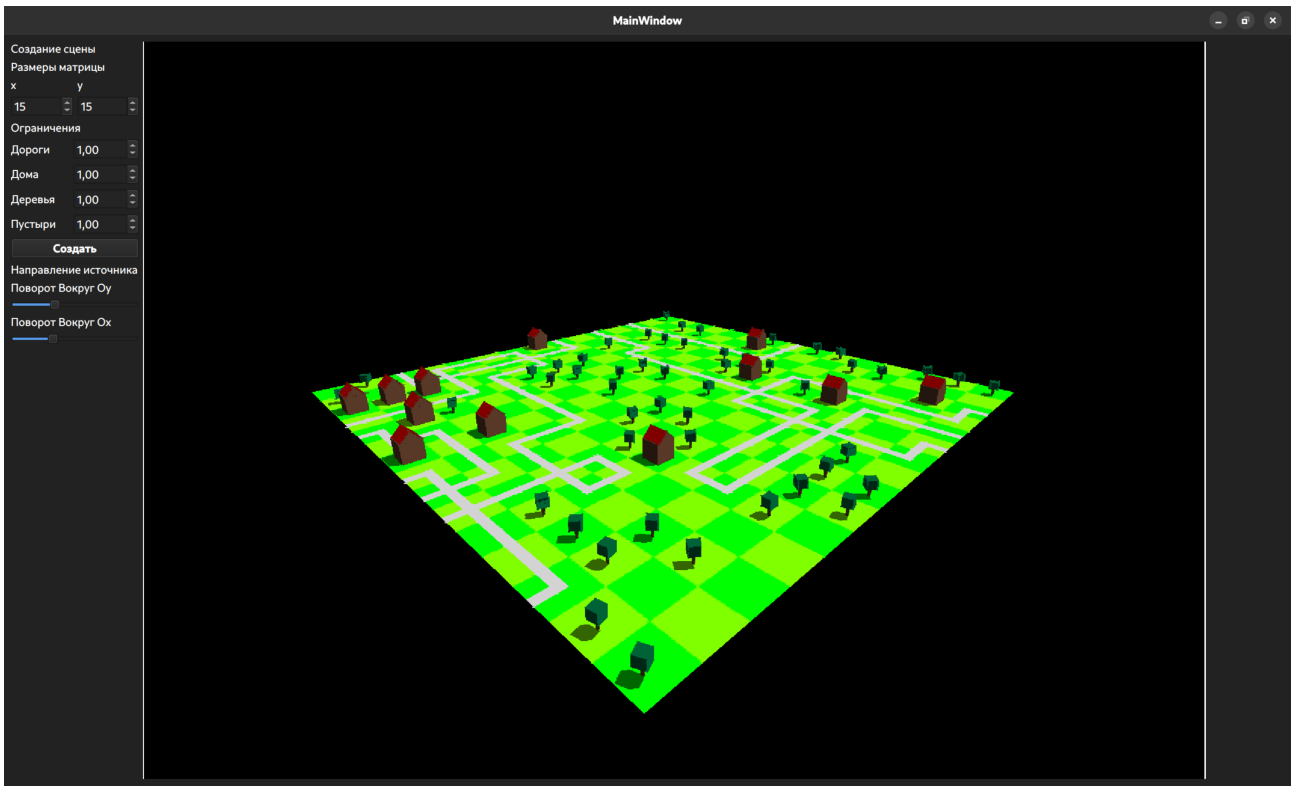

Рисунок 3.2 — Пример работы программы — генерация посёлка 15 на 15 ячеек

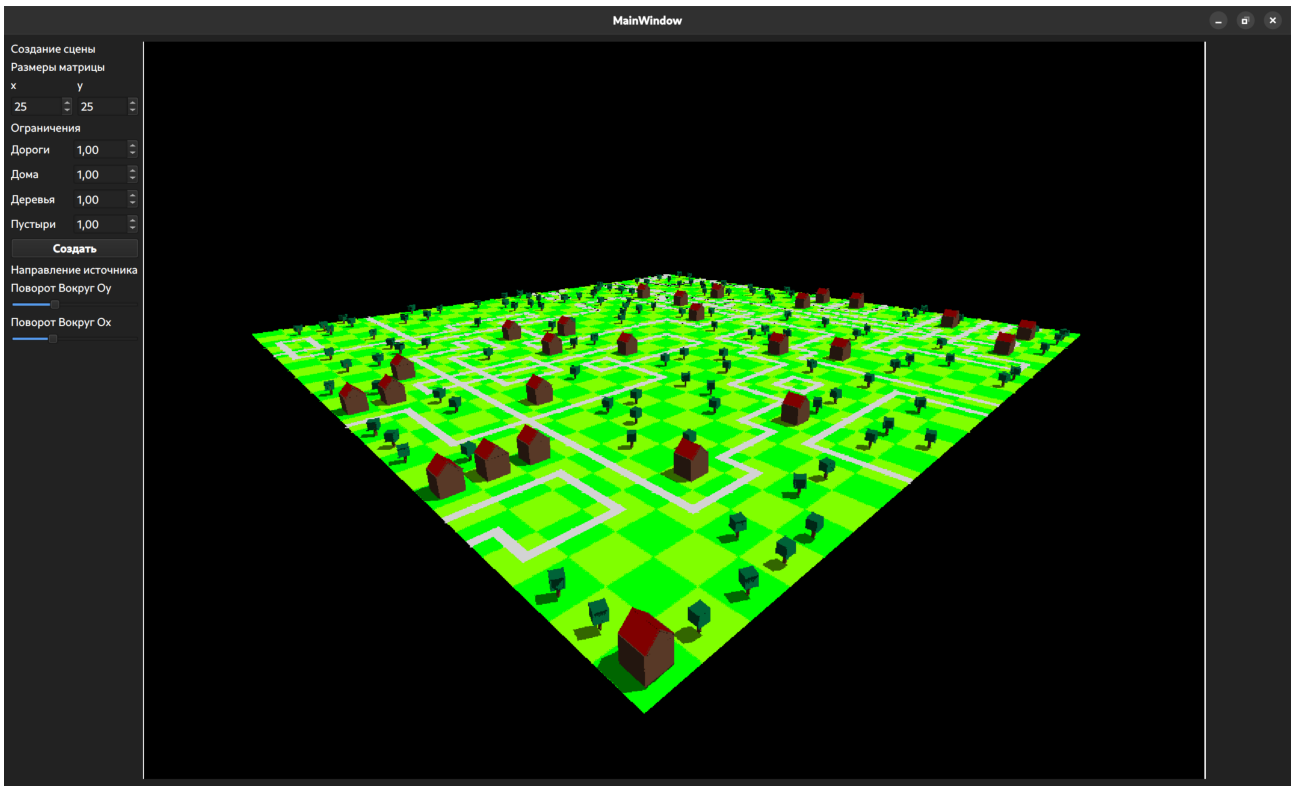

Рисунок 3.3 — Пример работы программы — генерация посёлка 25 на 25 ячеек

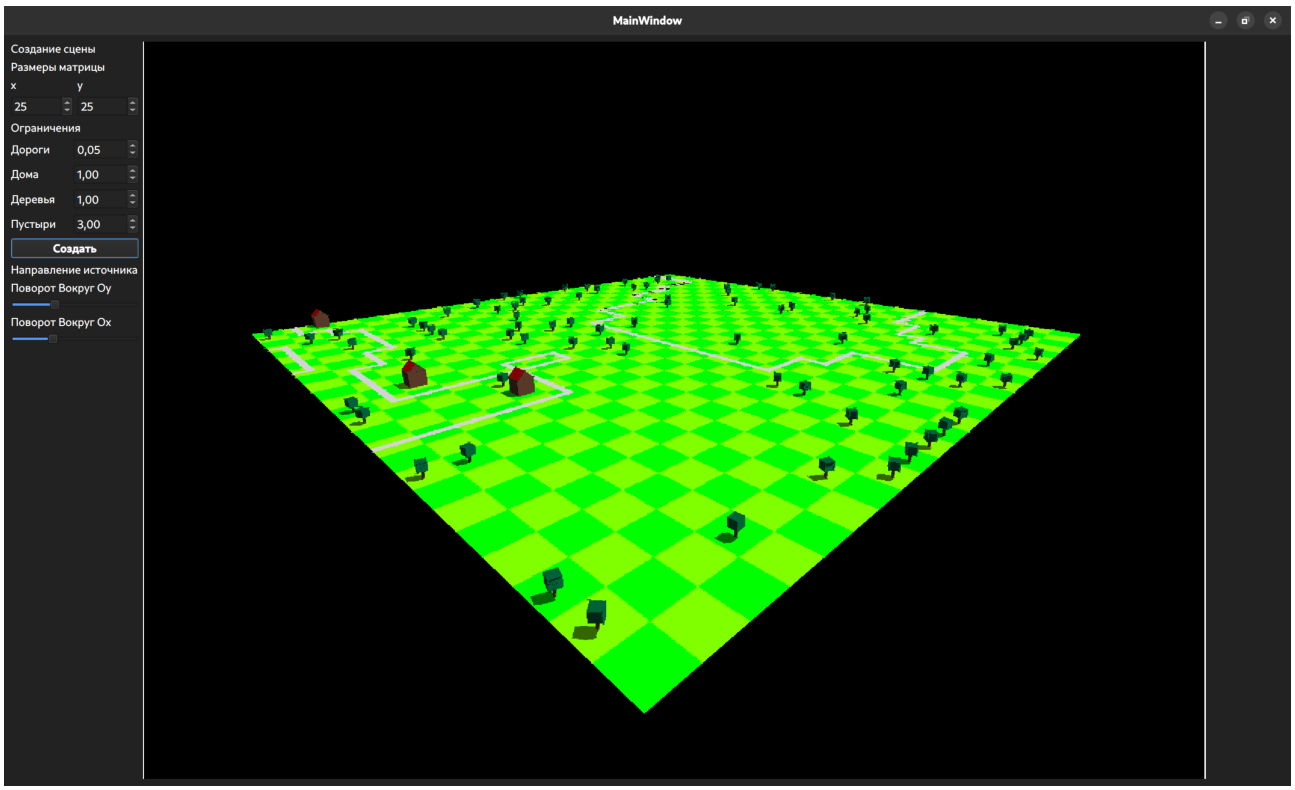

Рисунок 3.4 — Пример работы программы — генерация посёлка с уменьшенным количеством дорог и увеличенным колчичеством пустырей

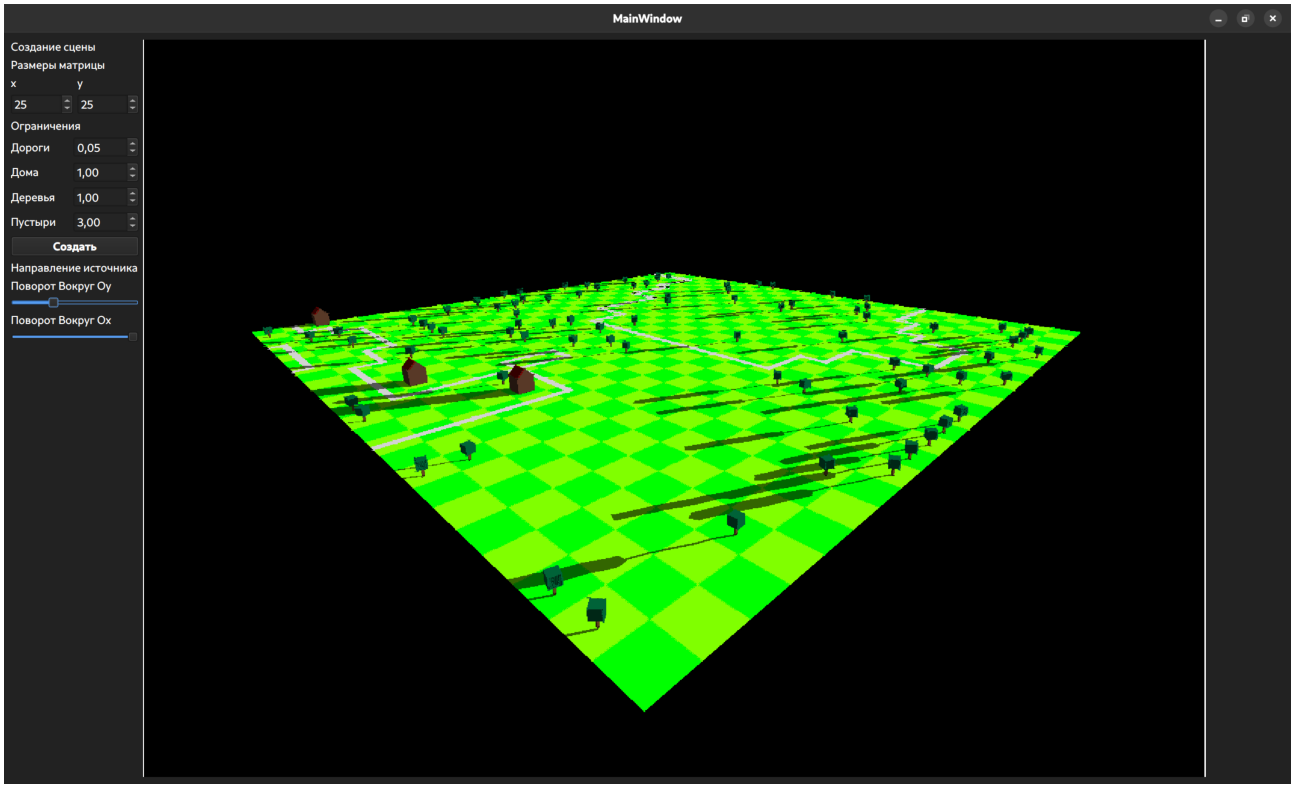

Рисунок 3.5 — Пример работы программы — Источник света приближен к земле, тени становятся длиннее

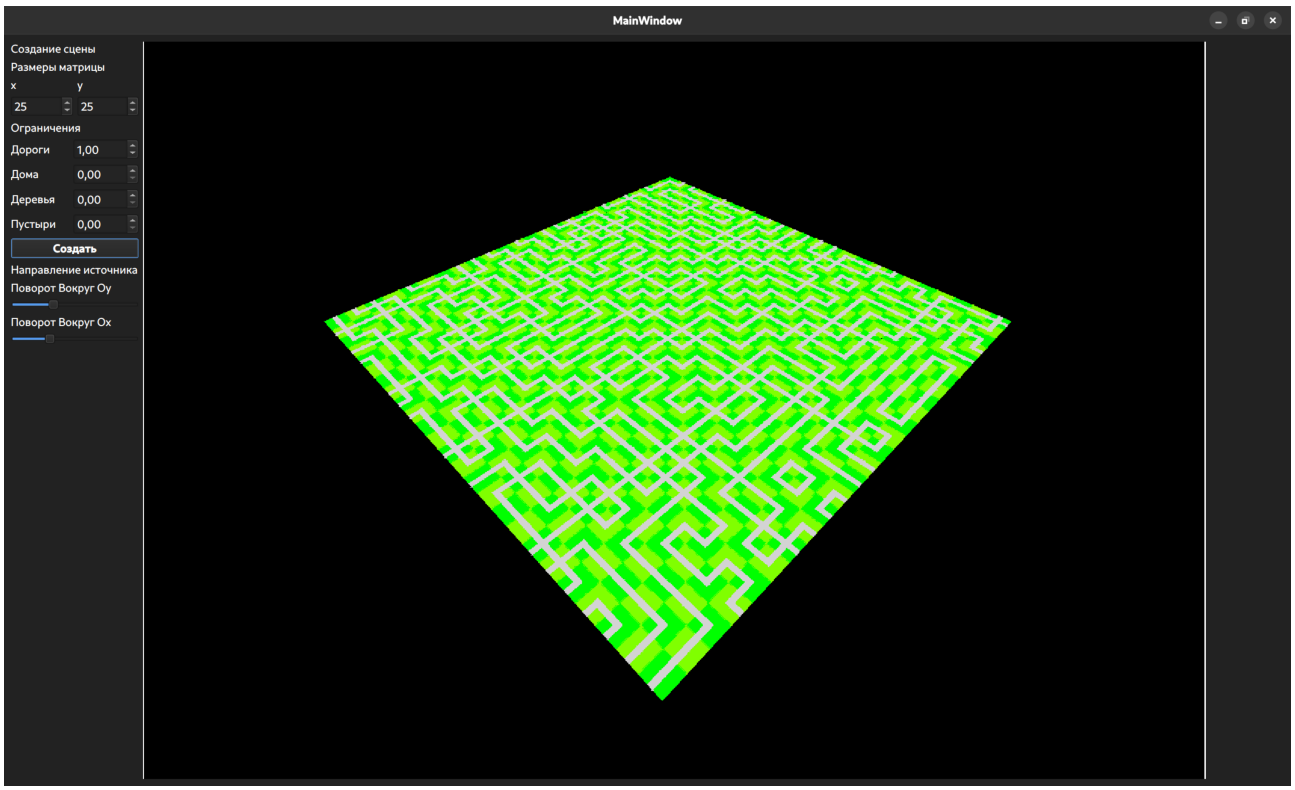

Рисунок 3.6 — Пример работы программы — запрещено появление любых объектов кроме дорог

3.5 Вывод

Была осуществлена реализация программного обеспечения для создания конструирования и визуализации загородного посёлка.

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось исследование:

— операционная система EndeavourOS 64бит;

— версия ядра Linux Linux 6.12.3-arch1-1;

— 13th Gen Intel(R) Core(TM) i5-13500H 4.70 ГГц 12 ядер (4 производительных + 8 энергоэффективных) 16 логических процессоров;

— оперативная память 16ГБ с частотой 5200МГц.

4.2 Описание исследования

Было произведено несколько исследований.

Исследование времени генерации сцены

Было проведено исследования времени генерации сцены от размеров матрицы. В ходе исследования производилась генерация квадратных сцен.

Для каждого размера матрицы, замер времени проводился 10 раз. В качестве результата сохранялось среднее значение затраченного времени.

Для замеров времени использовалась библиотека chrono, являющаяся частью стандартной библиотеки языка c++ [6].

Результаты временных замеров представлены в виде графика на рисунке 4.1.

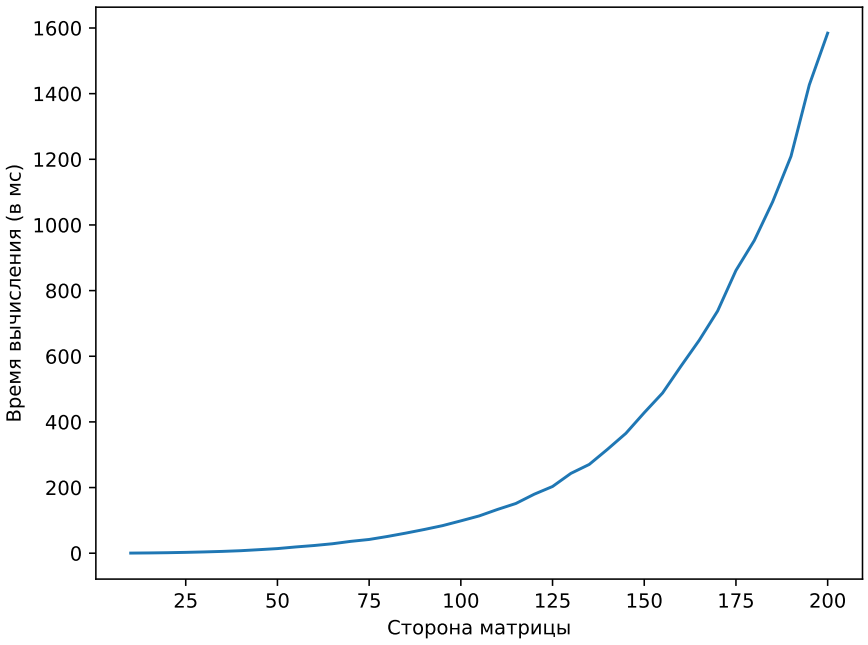

Рисунок 4.1 — График зависимости времени генерации сцены от размеров сцены

Исследование времени создания карты теней

Было проведено исследования зависимости времени создания карты теней от размеров буфера глубины. Размеры буфера по оси Ox и Oy в процессе исследования совпадали, то есть буфер был квадратным.

Количество моделей на сцене во время временных замеров оставалось постоянным.

Для замеров времени использовалась библиотека chrono, являющаяся частью стандартной библиотеки языка c++ [6].

Результаты временных замеров представлены в виде графика на рисунке 4.2.

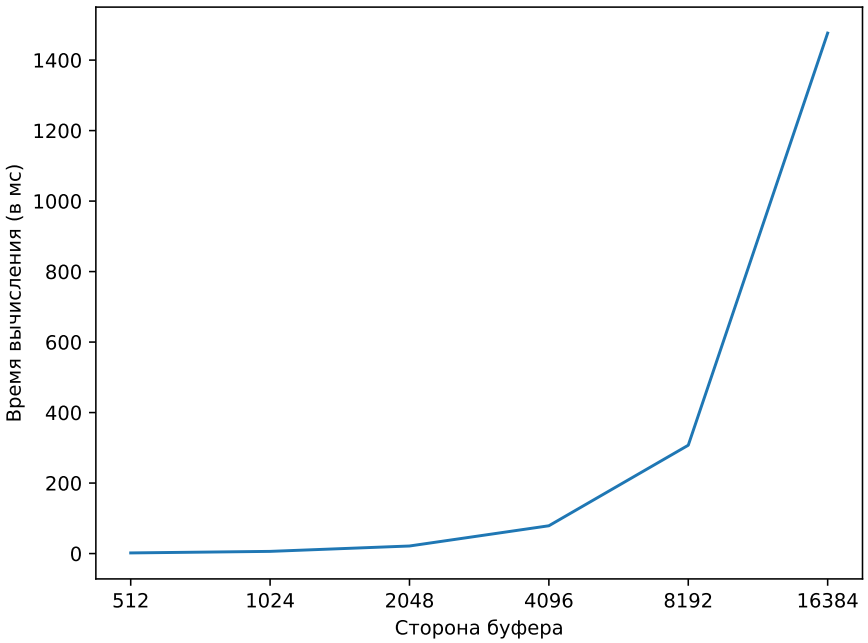

Рисунок 4.2 — График зависимости времени создания карты теней от её разрешения

Исследование времени генерации кадра

Было проведено исследования зависимости времени генерации от размеров матрицы сцены. В процессе исследования рассматривались только квадратные сцены.

Разрешение экрана (то есть размеры буфера кадра) в процессе исследования оставались постоянными.

Для замеров времени использовалась библиотека chrono, являющаяся частью стандартной библиотеки языка c++ [6].

Результаты временных замеров представлены в виде графика на рисунке 4.3.

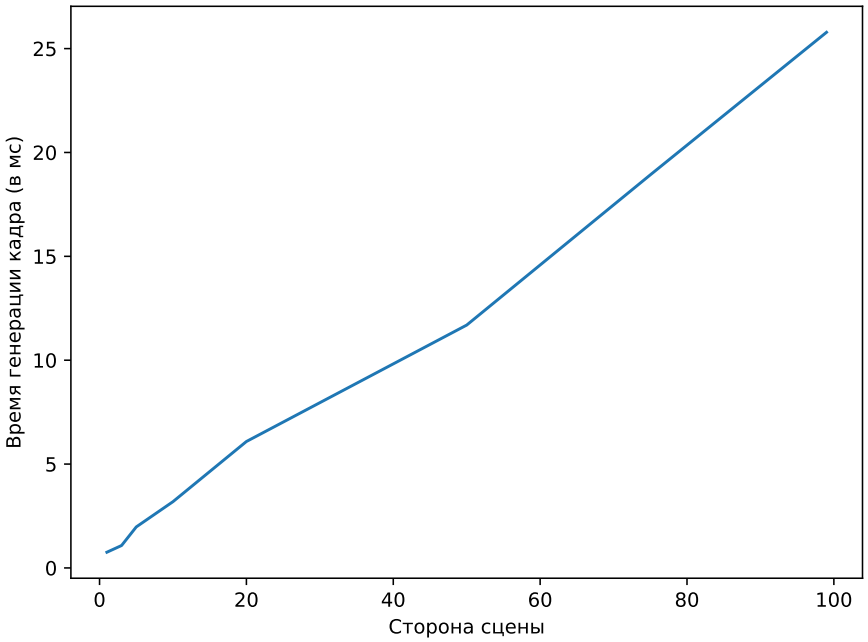

Рисунок 4.3 — График зависимости времени генерации кадра от размеров сцены

4.3 Вывод

Проанализировав графики, можно увидеть, что:

— зависимость времени генерации квадратной сцены от размера стороны матрицы близка к степенной функции;

— зависимость времени создания квадратной карты теней от размера стороны её буфера глубины близка к степенной функции;

— зависимость времени генерации кадра от размера сцены (а соответственно и количества объектов на ней) близка к линейной.

5 ЗАКЛЮЧЕНИЕ

Поставленная цель была достигнута и были выполнены все поставленные задачи:

— были сравнены существующие алгоритмы процедурной генерации сцены;

— были сравнены существующие алгоритмы компьютерной графики, использующихся для визуализации трёхмерной модели (сцены);

— были выбраны подходящие алгоритмы для решения поставленных задач;

— были спроектированы архитектура и графический интерфейс ПО

— были выбраны средства реализации ПО;

— было разработано ПО;

— были проведены замеры временных характеристик разработанного ПО;

В ходе проведенного исследования была проанализирована зависимость времени генерации квадратной сцены от размера стороны матрицы. Эта зависимость оказалась близка к степенной функции.

зависимость времени создания квадратной карты теней от размера стороны её буфера глубины. Эта зависимость оказалась близка к степенной функции.

В ходе проведенного исследования была проанализирована зависимость времени генерации кадра от размера сцены (а соответственно и количества объектов на ней). Эта зависимость оказалась близка к линейной.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. John F. Hugens Computer Graphics Principles and Practice / John F. Hugens // Бостон: Addison Wesley Professional 3-е издание, 1982, 1263с.

2. Heese, Raoul. Quantum Wave Function Collapse for Procedural Content Generation [Текст] / Heese, Raoul, Institute of Electrical and Electronics Engineers (IEEE), 2024, 13 с.

3. Д. Роджерс Алгоритмические основы машинной графики [Текст] / Д. Роджерс, Перевод С.А. Вичес, Москва: Мир, 1989, 512 страниц

4. А.В. Мальцев Моделирование теней в 3D сценах с помощью каскадных теневых карт в режиме реального времени // статья // Журнал ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ И ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ // Москва: Российская Академия Наук 2014г. // 52 страницы

5. Michael Abrash. Quake's lighting model: Surface caching. // Онлайн-ресурс: https://www.bluesnews.com/abrash/chap68.shtml // (дата обращения: 15 декабря 2024 г.).

6. ISO International Standard ISO/IEC 14882:2020(E) [Working Draft] – Programming Language C++ [Текст] / Geneva, Switzerland: International Organization for Standardization (ISO), 2020, 1900с.

7. Документация библиотеки Qt6 [Электронный ресурс] // Qt Gui, [сайт]. — URL: `https://doc.qt.io/qt-6/` (дата обращения: 15 декабря 2024 г.).

8. Документация библиотеки OneApi TBB [Электронный ресурс] // parallel_for, concurrent_vector, [сайт]. - URL: `https://www.intel.com/content/www/us/en/developer/tools/oneapi/onetbb-documentation.html` (дата обращения: 15 декабря 2024 г.).

9. Официальный сайт среды разработки Visual Studio Code [Электронный ресурс] / [сайт]. - URL: `https://code.visualstudio.com/` (дата обращения: 15 декабря 2024 г.).

10. Документация системы сборки CMake [Электронный ресурс] / [сайт]. - URL: `https://cmake.org/cmake/help/latest/index.html` (дата обращения: 15 декабря 2024 г.).

ПРИЛОЖЕНИЕ А

Листинг А.1 — Использование RenderVisitor на сцену

```
void RenderVisitor::visit(Scene &ref) {
  ShadowMap &smap = Singleton<ShadowMap>::instance();
  auto resolution = ctx->getResolution();
  colors.resize(resolution.second, std::vector<Color>(resolution.first));
  depth.resize(resolution.second,
               std::vector<double>(resolution.first,
                                   std::numeric_limits<double>::infinity()
                                   ));
  pixmaps.clear();

  tbb::parallel_for_each(ref.begin(), ref.end(), [this](const auto &newref
     ) {
    newref.second->accept(*this);
  });

  for (const MyPixmap &pmap :
       pixmaps | std::views::filter([](const MyPixmap &pmap) {
         return !pmap.getDepth().empty() && !pmap.getDepth().front().empty
            ();
       })) {
    auto &pdepth = pmap.getDepth(); auto offset = pmap.getOffset();
    auto &pcolor = pmap.getColor(); auto &pbright = pmap.getBrightness();

    for (int x = 0; x < pdepth.front().size(); ++x)
      for (int y = 0; y < pdepth.size(); ++y) {
        if (pdepth[y][x] < depth[y + offset.second][x + offset.first]) {
          depth[y + offset.second][x + offset.first] = pdepth[y][x];
          if (smap.isValid())
            colors[y + offset.second][x + offset.first] =
                pcolor * pbright[y][x];
          else
            colors[y + offset.second][x + offset.first] = pcolor;
        }
      }
  }

  ctx->setImage(colors);
}
```

Листинг А.2 — Использование RenderVisitor на модель

```
void RenderVisitor::visit(FaceModel &ref) {
  ShadowMap &smap = Singleton<ShadowMap>::instance();

  std::shared_ptr<TransformationMatrix> transf = ref.getTransformation();

  CameraManager &cm = Singleton<CameraManager>::instance();

  std::shared_ptr<BaseCamera> cam =
      std::dynamic_pointer_cast<BaseCamera>(cm.getCamera());
  std::shared_ptr<TransformationMatrix> camtransf = cam->getTransformation
     ();

  PTSCAdapter->setCamera(cm.getCamera());
  Point3D cam_position = cam->getPosition();
  auto resolution = ctx->getResolution();

  auto range =
      ref.getFaces() |
    std::views::filter(std::bind(&Face::isVisible, std::placeholders::_1,
                                 cam_position, *transf)) |
      std::views::filter([&transf, &camtransf](const Face &f) {
        return std::ranges::any_of(f.getPoints(), [&transf, &camtransf](
                                                      const Point3D &pt) {
    return camtransf->apply(transf->apply(pt)).z >= 2 * MyMath::EPS;
        });
      });

  tbb::parallel_for_each(
      range.begin(), range.end(),
      [this, &camtransf, &transf, &resolution, &smap](const Face &face) {
        pixmaps.emplace_back(face.getPixmap(
            [transf](const Point3D &pt) { return transf->apply(pt); },
            [&camtransf](const Point3D &pt) { return camtransf->apply(pt);
                },
            [this, &smap](const Point3D &pt) {
              return PTSCAdapter->convert(pt);
            },
            resolution.first, resolution.second, smap, true));
      });
}
```

Листинг А.3 — Метод фиксирования состояния ячейки класса QWFC

```
void QWFC::collapseCell(int x, int y) {
  if (x < 0 || y < 0 || x >= width || y >= height)
    return;
  auto &possibleValues = matrix[y][x];
  if (possibleValues.size() == 1)
    return;

  double totalProbability = 0.0;
  for (const auto &value : possibleValues) {
    totalProbability += cells.at(value).getProbability();
  }

  if (totalProbability < MyMath::EPS) {
    throw std::runtime_error("Can not continue.");
  }

  double random = static_cast<double>(rand()) / static_cast<double>(
     RAND_MAX) *
                  totalProbability;
  double cumulativeProbability = 0.0;
  int chosenValue = *possibleValues.begin();

  for (const auto &value : possibleValues) {
    cumulativeProbability += cells.at(value).getProbability();
    if (random <= cumulativeProbability) {
      chosenValue = value;
      break;
    }
  }

  possibleValues.clear();
  possibleValues.insert(chosenValue);
}
```

ПРИЛОЖЕНИЕ В

Презентация состоит из N слайдов.