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

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Программа визуализации лесного массива»

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

Руководитель _____ **Н. Б. Толпинская**
(Подпись, дата) (И.О.Фамилия)

2022 г.

Содержание

Введение

Уже многие годы одними из самых популярных компьютерных игр являются строительные игры, чаще всего они имеют большой функционал, но примитивную пиксельную графику, которая не только не недостаток, но, наоборот, визитная карточка таких игр. Самыми популярными примерами можно назвать «Terraria», с аудиторией более 35 миллионов, и «Minecraft», чье количество пользователей оценивают от 125 до 400 миллионов. Основным направлением развития данных игр является создание новых биомов – совокупностей экосистем одной природно-климатической зоны. Мною была выбрана тема, связанная с генерацией лесного массива. Однако при помощи материалов и алгоритмов, необходимых для выполнения курсовой работы, в последствии можно создать любой другой биом. Реалистичности играм также придает автоматическое переключение времен суток, которое является одной из задач выбранной темы.

Цель курсовой работы: реализация программного обеспечения для визуализации лесного массива.

Задачи курсовой работы:

- описание и формализация доступных моделей;
- анализ и выбор соответствующих алгоритмов компьютерной графики для визуализации сцены и объектов;
- анализ и выбор языка программирования;
- реализация выбранных алгоритмов визуализации;
- реализация программного обеспечения для визуализации и редактирования лесного массива.

1 Аналитическая часть

1.1 Описание и формализация объектов сцены

Сцена состоит из объектов, описанных ниже.

- **Время суток** — не имеет собственной модели, однако оказывает влияние на имеющиеся цветовые гаммы остальных объектов. В программе предусмотрена реализация плавного перехода от дня с яркой цветовой гаммой к ночи с приглушенной и более темной цветовой гаммой. Далее все цвета будут описаны в соответствии дневной цветовой гамме.

- **Площадка сцены** — прямоугольный параллелепипед, состоящий из определенного количества кубов константного размера, являющихся ячейками для размещения объектов сцены. Объекты сцены могут располагаться лишь по одну сторону площадки. Начальное положение — параллельно плоскости OXY. Цвет — светло-зеленый.

- **Объекты сцены** — модели, расположенные на площадке сцены, занимающие одну ячейку сцены по длине и ширине и отдельное для каждой модели количество клеток по высоте. Каждая модель — набор граней, описываемых координатами в трехмерном пространстве, соединенных ребрами. Программа предусматривает добавление объектов на сцену с возможностью выбора расположения объектов, но не добавление новых объектов. Цвет некоторых объектов не является предустановленным, а выбирается случайно из допустимых.

  В программе доступны объекты, описанные ниже.

- **Лиственное дерево** выглядит согласно модели, изображенной на рисунке 1.1, (крона может быть других оттенков зеленого).

- **Дерево хвойное** выглядит согласно модели, изображенной на рисунке 1.2.

Рисунок 1.1 – Модель лиственного дерева

Рисунок 1.2 – Модель хвойного дерева

- Кустарник выглядит согласно модели, изображенной на рисунке 1.3, (листва может быть других оттенков зеленого).

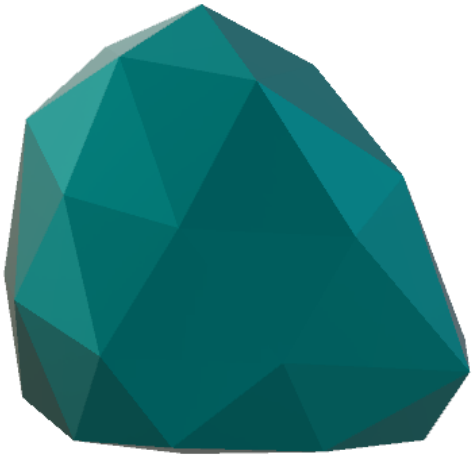

Рисунок 1.3 – Модель кустарника

- Водоем — куб, определенный так же, как и кубы земли (площадки сцены), однако имеющий синий цвет.
- Источник света – материальная точка, испускающая параллельные лучи света. Положение источника света задается координатами X, Y, Z. А в зависимости от расположения источника и направления распространения света определяется тень от объектов сцены.

1.2 Анализ и выбор способа задания моделей

Модели могут задаваться способами, описанными ниже.

- **Каркасная модель** представляет форму объектов в виде конечного множества линий. Для каждой линии известны координаты концевых точек и функция линии.
- **Поверхностная модель** представляет форму объектов с помощью ограничивающих ее поверхностей (данные о гранях, вершинах, ребрах, функции поверхностей).
- **Объемные твердотельные модели** дополнительно содержат в явной форме сведения о принадлежности элементов внутреннему или внешнему по отношению к объекту пространству [1].

Каркасная модель в данном случае не подходит, так как большинство сложных моделей невозможно воспринимать в таком виде, а совокупность таких моделей превращается в абстрактную и непонятную картину. Если опираться, на опыт продуктов, описанных во введении, то необходимо выбирать объемные твердотельные модели, так как функциональность игр дает возможность взаимодействия с внутренним пространством моделей. Однако тема курсовой работы предполагает только визуализацию объектов без взаимодействия с ними, по этой причине оптимальным способом задания моделей будет именно поверхностная модель.

1.3 Анализ и выбор способа задания поверхностных моделей

Поверхностные модели могут задаваться способами, следующими далее.

- **Параметрическое представление** — для получения поверхности необходимо вычислить функцию, зависящую от параметра.
- **Полигональная сетка** – совокупность вершин, ребер и граней, которые определяют форму объекта.
  - **Вершинное представление** хранит вершины, указывающие на другие вершины, с которыми они соединены.
  - **Список граней** представляет объект как множество граней и вершин.
  - **Таблица углов (веер треугольников)** хранит вершины в предопределенной таблице [2].

Наиболее важным фактором для выбора способа задания поверхностной модели в курсовой работе является скорость, по этой причине была выбрана модель, заданная полигональной сеткой, которая к тому же позволяет избежать проблем при описании сложных объектов сцены. Оптимальный способ, позволяющий эффективно преобразовывать модели, яв-

ляется способ хранения полигональной сетки при помощи списка граней, так как структура включает в себя список вершин.

1.4 Анализ и выбор алгоритма удаления невидимых линий и поверхностей

Алгоритмы удаления невидимых линий и поверхностей служат для определения линий ребер, поверхностей или объемов, которые видимы или невидимы для наблюдателя, находящегося в заданной точке пространства. Алгоритмы удаления невидимых линий или поверхностей можно классифицировать по способу выбора системы координат или пространства, в котором они работают. Выделяют три класса алгоритмов удаления невидимых линий или поверхностей:

- алгоритмы, работающие в объектном пространстве;
- алгоритмы, работающие в пространстве изображения;
- алгоритмы, формирующие список приоритетов.

Алгоритмы, работающие в объектном пространстве, имеют дело с физической системой координат, в которой описаны эти объекты. При этом получаются весьма точные результаты, ограниченные лишь точностью вычислений. Полученные изображения можно свободно увеличивать во много раз.

Алгоритмы же, работающие в пространстве изображения, имеют дело с системой координат того экрана, на котором объекты визуализируются. При этом точность вычислений ограничена разрешающей способностью экрана. Результаты, полученные в пространстве изображения, а затем увеличенные во много раз, не будут соответствовать исходной сцене.

Объем вычислений для любого алгоритма, работающего в объектном пространстве и сравнивающего каждый объект сцены со всеми остальными объектами этой сцены, растет теоретически, как квадрат числа объектов. Аналогично, объем вычислений любого алгоритма, работающего в

пространстве изображения и сравнивающего каждый объект сцены с позициями всех пикселей в системе координат экрана, растет теоретически как n^N. Здесь n – число объектов, N – число пикселей. Теоретически трудоемкость алгоритмов, работающих в объектном пространстве, меньше трудоемкости алгоритмов, работающих в пространстве изображения, при $n < N$. Однако на практике это не так. Дело в том, что алгоритмы, работающие в пространстве изображения, более эффективны потому, что для них легче воспользоваться преимуществом когерентности при растровой реализации [3].

В выбранной задаче наибольшее значение имеет быстродействие алгоритма, по этой причине при первичном анализе предпочтение отдается алгоритмам, работающим в пространстве изображения, однако стоит отдельно рассмотреть преимущества и недостатки наиболее популярных алгоритмов.

1. **Алгоритм обратной трассировки лучей.**

   Преимущества:

   - высокая реалистичность синтезируемого изображения;
   - работа с поверхностями в математической форме.

   Недостатки:

   - производительность.

2. **Алгоритм Робертса.**

   Преимущества:

   - высокая точность вычислений.

   Недостатки:

   - рост сложности алгоритма – квадрат числа объектов;
   - необходимость выпуклости тел.

3. **Алгоритм, использующий Z-буфер.**

   Преимущества:

- простота реализации;
- оценка вычислительной трудоемкости алгоритма линейна;
- экономия вычислительного времени (элементы сцены не сортируются).

Недостатки:

- большие затраты памяти;
- сложность реализации эффектов прозрачности [4].

Так как наибольшей скоростью отличается алгоритм, использующий Z-буфер, недостатки которого к тому же не оказывают большого влияния на выбранную задачу, для выполнения курсовой работы выбран именно он.

В данном алгоритме используется Z-буфер — буфер глубины, используемый для запоминания координаты z (глубины) каждого видимого пикселя в пространстве изображения. В процессе работы глубина или значение z каждого нового пикселя, который нужно занести в буфер кадра, сравнивается с глубиной того пикселя, который уже занесен в Z-буфер. Если это сравнение показывает, что новый пиксель расположен впереди пикселя, находящегося в буфере кадра, то новый пиксель заносится в этот буфер и, кроме того, производится корректировка Z-буфера новым значением z. Если же сравнение дает противоположный результат, то никаких действий не производится. По сути, алгоритм является поиском по x и y наибольшего значения функции $Z(x, y)$ [5].

1.5 Анализ и выбор алгоритма построения теней

Алгоритмом построения теней был выбран алгоритм карты теней, использующий алгоритм на основе Z-буфера и то, что все объекты в задаче разбиты на треугольники размером несопоставимым с размером одного блока земли. При отрисовке отдельно взятого треугольника изначально

считается общее количество пикселей во время алгоритма, использующего Z-буфер, а в самом алгоритме карты теней происходит проверка, есть ли в теневом буфере пиксель с координтами (x, y), в том случае если он уже есть, декрементируется количество пикселей, которые должны быть реально отрисованы, треугольника, которому принадлежит пиксель, находящийся в теневом буфере. Таким образом находится число всех пикселей какой-либо грани и перекрытых пикселей данной грани, что позволяет определять интенсивность тени по формуле 1.1.

$$u = \frac{DrawedPixels}{AllPixels} \tag{1.1}$$

Данный алгоритм позволяет строить тени наиболее быстро, а также накладывать мягкие тени.

1.6 Анализ и выбор способов оптимизации визуализации

Алгоритм отбраковки нелицевых граней (backface culling). Для определения того, является заданная грань лицевой или нет достаточно взять произвольную точку этой грани и проверить выполнение условия (I,n)<=0, где I – вектор, направленный к наблюдателю, n – вектор внешней нормали грани. Если грань является нелицевой, то не визуализируется.

Алгоритм оболочек (Bounding Volumes, Bounding-Volume-Hierarchy). Если оболочки не пересекаются, то и содержащиеся в них объекты тоже пересекаться не будут. Однако, если оболочки пересекаются, то сами объекты пересекаться не обязаны. В качестве ограничивающих тел чаще всего используются прямоугольные ограничивающие параллелепипеды.

Для данной работы был выбран метод отсечения нелицевых граней, так как наилучшим образом подходит для задачи, а именно в силу представления площадки сцены в виде кубов имеет заведомо большое количество нелицевых граней.

Вывод

В ходе аналитической части были рассмотрены и проанализированы как способы задания самих моделей, так и различные алгоритмы, необходимые для построения реалистичного изображения в заданной теме, а именно алгоритмы удаления невидимых линий и поверхностей, а также алгоритмы построения теней и способы оптимизации визуализации.

Из всех способов задания моделей и алгоритмов были выбраны следующие:

- поверхностная модель (путем хранения списка граней);
- алгоритм, использующий Z-буфер;
- алгоритм карты теней;
- алгоритм отбраковки нелицевых граней.

2 Конструкторская часть

2.1 Требования к программному обеспечению

Программа должна предоставлять следующие функции:

- создание сцены определенного размера;
- добавление объектов сцены;
- создание источника света на сцене;
- плавное переключение времен суток;
- поворот визуализируемой сцены;

2.2 Общий алгоритм визуализации сцены

1. Создать сцену по определенным в программе размерам, на которой будут размещаться объекты.
2. Создать источник света в определенном в программе положении.
3. Выбрать одну из моделей при помощи нажатия определенной клавиши клавиатуры.
4. Разместить модели путем выбора мышью одной из клеток на сцене.
5. Учитывая нынешнее положение источника света выбрать соответствующую цветовую гамму.
6. Учитывая нынешнее положении наблюдателя и источника света, определить падающие от объектов сцены тени.
7. Визуализировать сцену.

2.3 Алгоритм Z-буфера

На рисунке 2.1 представлена схема алгоритма Z-буфера.

Рисунок 2.1 – Схема алгоритма Z-буфера

2.4 Алгоритм карты теней

Данный алгоритм модифицирует алгоритм Z-буфера. На рисунке 2.2 представлена схема теневого алгоритма на основе схемы алгоритма Z-буфера (жирным выделены элементы, относящиеся непосредственно к алгоритму карты теней).

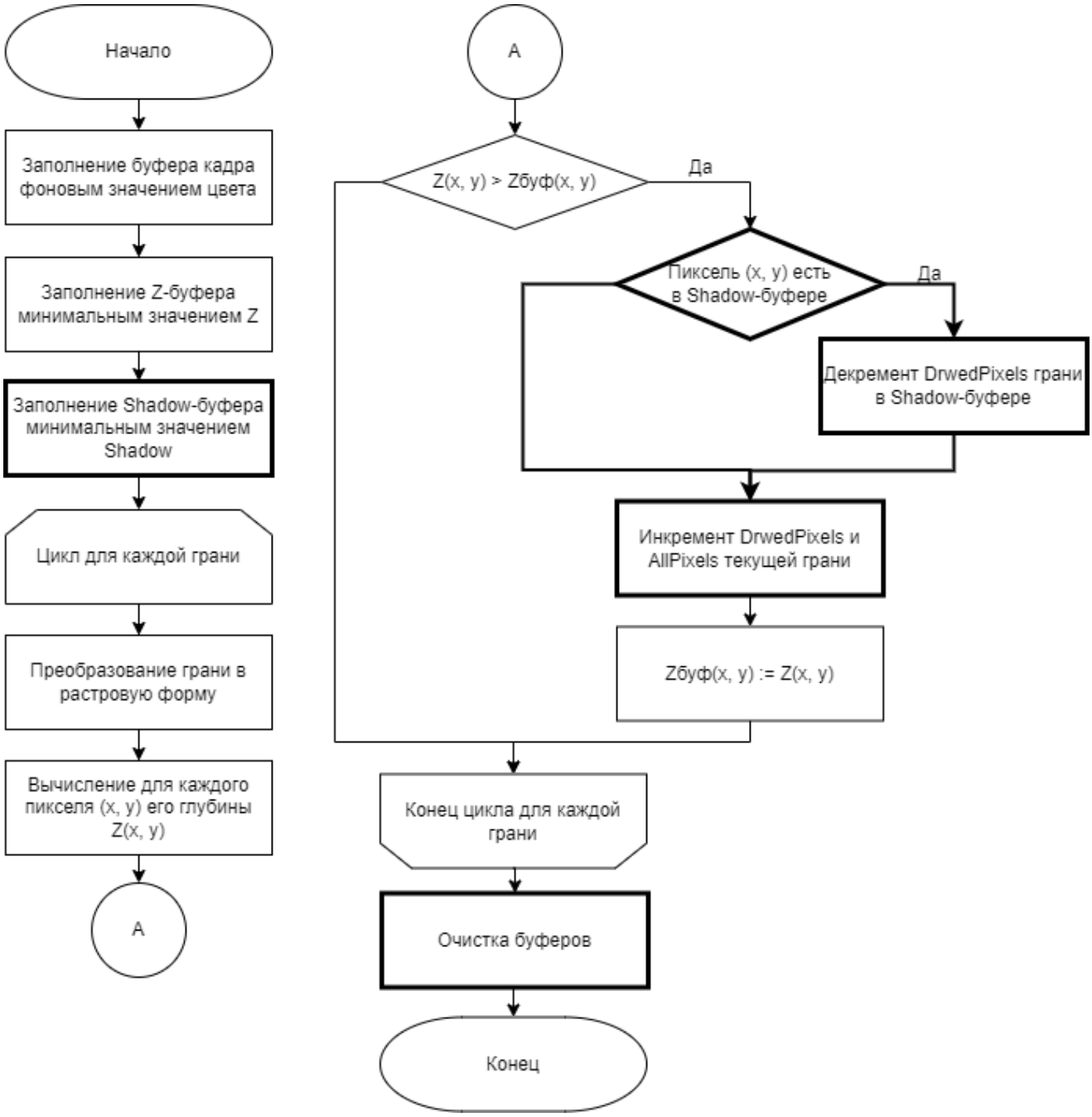

Рисунок 2.2 – Схема алгоритма карты теней

2.5 Метод отбраковки нелицевых граней

На рисунке 2.3 представлена схема алгоритма отбраковки нелицевых граней.

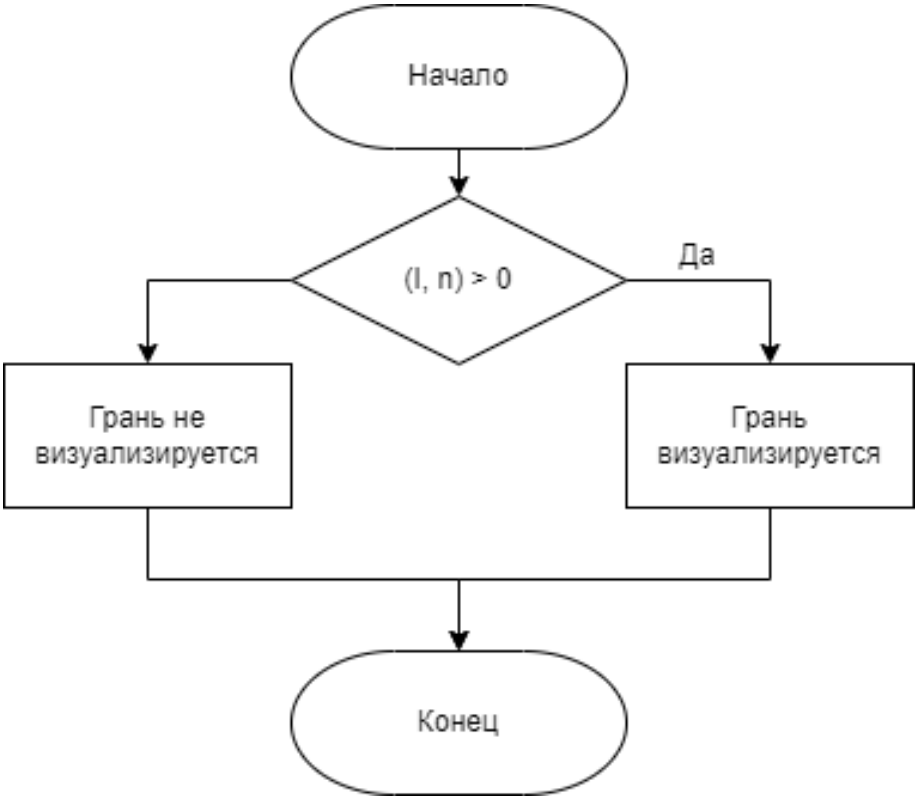

Рисунок 2.3 – Схема алгоритма отбраковки нелицевых граней

2.6 Представление данных в программном обеспечении

В таблице 2.1 описано представление данных в программном обеспечении.

Таблица 2.1 – Представление данных в программном обеспечении

Данные	**Представление**
Точка трехмерного пространства	Координаты по осям x, y, z
Полигон	Три точки трехмерного пространства
Объект сцены	Список полигонов
Источник света	Координаты по осям x, y, z

Вывод

На основе теоретических данных, полученных из аналитического раздела, были описаны требования к программному обеспечению, общий алгоритм решения задачи, алгоритм Z-буфера, алгоритм карты теней и алгоритм отбраковки нелицевых граней, а также было обобщено представление данных в программном обеспечении.

3 Технологическая часть

3.1 Выбор языка программирования

При написании программного продукта был выбран язык C#. Это обусловлено следующими далее факторами.

- C# обладает богатой стандартной библиотекой, что позволяет решать некоторые задачи легче.
- Данный язык поддерживает объектно-ориентированную парадигму программирования. Благодаря чему можно приводить объекты сцены к объектам классов, а также пользоваться шаблонами проектирования. Это дает возможность эффективного написания кода.
- Имеется опыт разработки на данном языке программирования.

3.2 Структура реализуемых классов

На рисунке 3.1 представлены классы модуля **Core** — является базой классов и отвечает за рендеринг изображения.

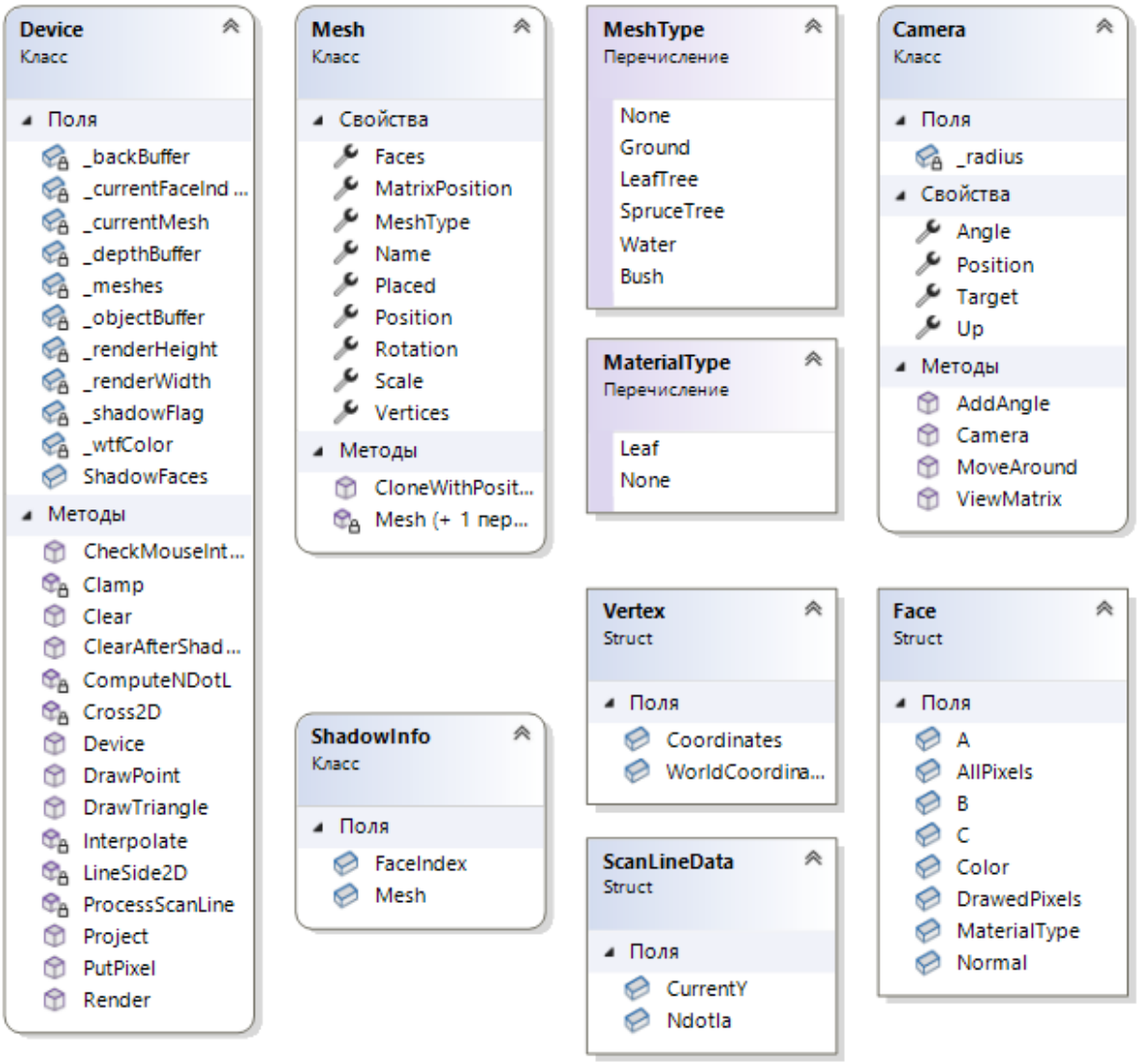

Рисунок 3.1 – Структура классов модуля Core

Описание реализуемых классов:

- **Mesh** — класс, предоставляющий функции работы с объектами;
- **ShadowInfo** — класс, содержащий информацию о тенях каждой грани каждого объекта;
- **Device** — класс, отвечающий за рендеринг изображения;
- **Camera** — клас, предоставляющий функции работы с камерой.

На рисунке 3.2 представлены классы модуля **Database** — загружает модели объектов.

Описание реализуемых классов:

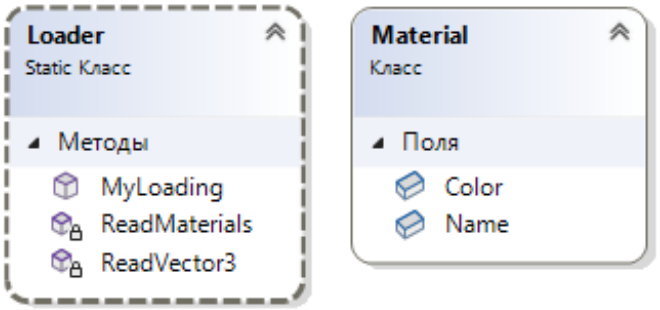

Рисунок 3.2 – Структура классов модуля Core

- **Loader** — класс-загрузчик моделей;
- **Material** — класс, содержащий информацию материале моделей.

На рисунке 3.3 представлены классы модуля **GameLogic** — отвечает за взаимодействие с программой, то есть объединяет модули Core и Database.

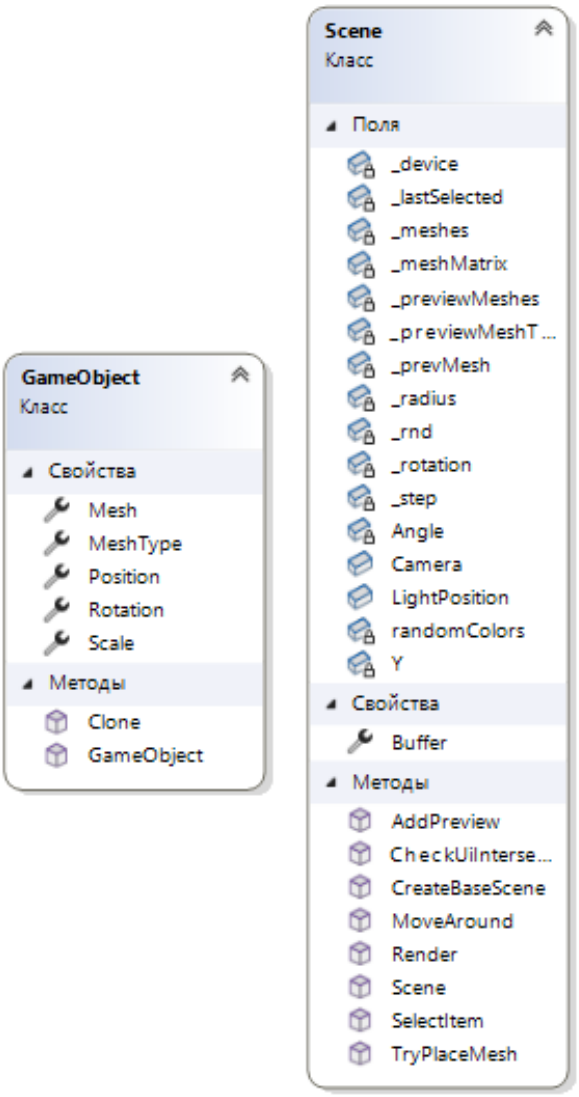

Рисунок 3.3 – Структура классов модуля GameLogic

Описание реализуемых классов:

- **GameObject** — класс, предоставляющий информацию об объектах и функции работы с ним;
- **Scene** — класс, предоставляющий информацию о сцене и функции работы с ней.

На рисунке 3.4 представлены классы модуля **Presentation** — отвечает за создание окна, сцены, интерфейс.

Описание реализуемых классов:

- **MainWindow** — выполняет все функции модуля.

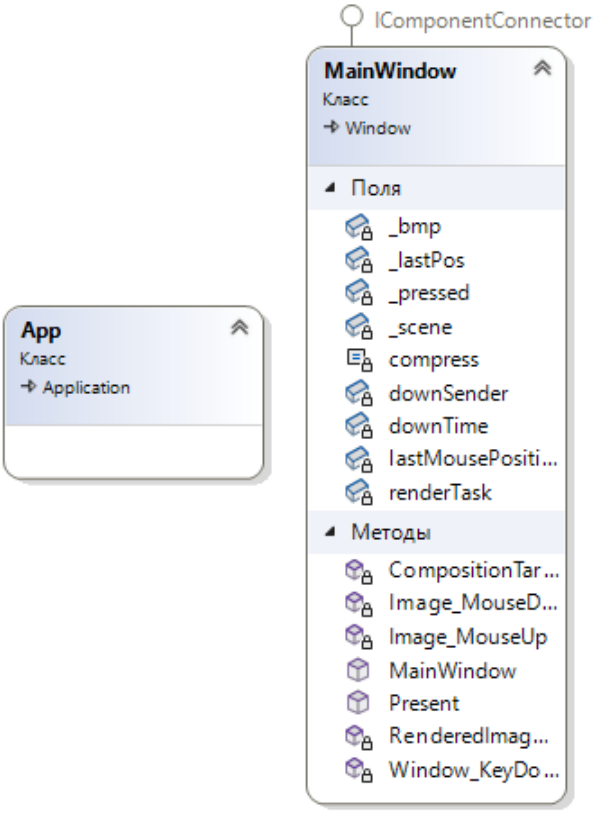

Рисунок 3.4 – Структура классов модуля Database

3.3 Реализации алгоритмов

На листинге 3.1 программная реализация алгоритма Z-буфера и карты теней.

Листинг 3.1 – Модифицированный алгоритм Z-буфера

```
1  public void PutPixel(int x, int y, float z, Color4 color)
2  {
3      var index = (x + y * _renderWidth);
4      var index4 = index * 4;
5
6      if (_depthBuffer[index] < z)
7          return;
8
9     if (_currentMesh.MeshType is MeshType.Ground or MeshType.Water)
10         _objectBuffer[index] = _currentMesh;
11
12     _depthBuffer[index] = z;
13
14     if (_shadowFlag)
15     {
16         if (ShadowFaces[index].FaceIndex >= 0)
17             ShadowFaces[index].Mesh.Faces[ShadowFaces[index].
                   FaceIndex].DrawedPixels -= 1;
18
19         _currentMesh.Faces[_currentFaceIndex].AllPixels += 1;
20         _currentMesh.Faces[_currentFaceIndex].DrawedPixels += 1;
21         ShadowFaces[index].FaceIndex = _currentFaceIndex;
22         ShadowFaces[index].Mesh = _currentMesh;
23     }
24     else
25     {
26         _backBuffer[index4 + 0] = (byte)(color.Blue * 255);
27         _backBuffer[index4 + 1] = (byte)(color.Green * 255);
28         _backBuffer[index4 + 2] = (byte)(color.Red * 255);
29         _backBuffer[index4 + 3] = (byte)(color.Alpha * 255);
30     }
31 }
```

На листинге 3.2 программная реализация алгоритма отбраковки нелицевых граней.

Листинг 3.2 – Алгоритм отбраковки нелицевых граней

```
1  float ComputeNDotL(Vector3 normal, Vector3 lightPosition)
2  {
3      var lightDirection = lightPosition;
4
5      normal.Normalize();
6      lightDirection.Normalize();
7
8      return Vector3.Dot(normal, lightDirection);
9  }
10
11 float cameraNDotL = ComputeNDotL(vnFace, cameraPos);
12
13 if (cameraNDotL < 0)
14     return;
```

3.4 Интерфейс программного обеспечения

Было решено сделать насколько возможно естественный интерфейс, напоминающий приведенные во введении игры.

На рисунке 3.5 представлена программная инструкция использования.

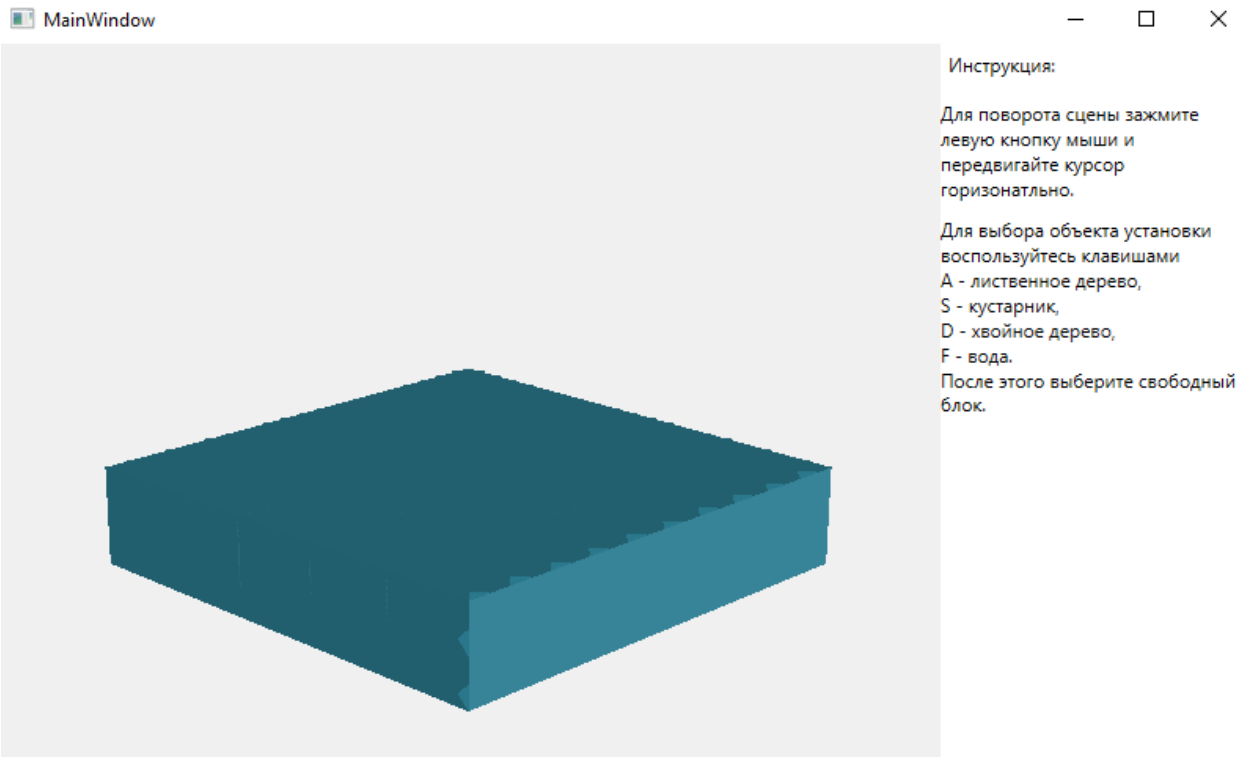

Рисунок 3.5 – Инструкция использования программного обеспечения

Вывод

В данном разделе был выбран язык программирования, описаны структуры реализуемых классов, сведения о модулях программы, а также рассмотрена реализация некоторых алгоритмов и интерфейс программы.

4 Экспериментальная часть

4.1 Тестирование программного обеспечения

Протестируем определенные в конструкторской части требования к программному обеспечению, а именно:

- создание сцены определенного размера (рисунок 4.1);
- добавление объектов сцены (рисунок 4.2);
- создание источника света на сцене (рисунки 4.2 и 4.3);
- плавное переключение времен суток (рисунки 4.2 и 4.3);
- поворот визуализируемой сцены (рисунок 4.4);

Рисунок 4.1 – Создание сцены определенного размера

Рисунок 4.2 – Визаулизация объектов сцены в дневное время суток

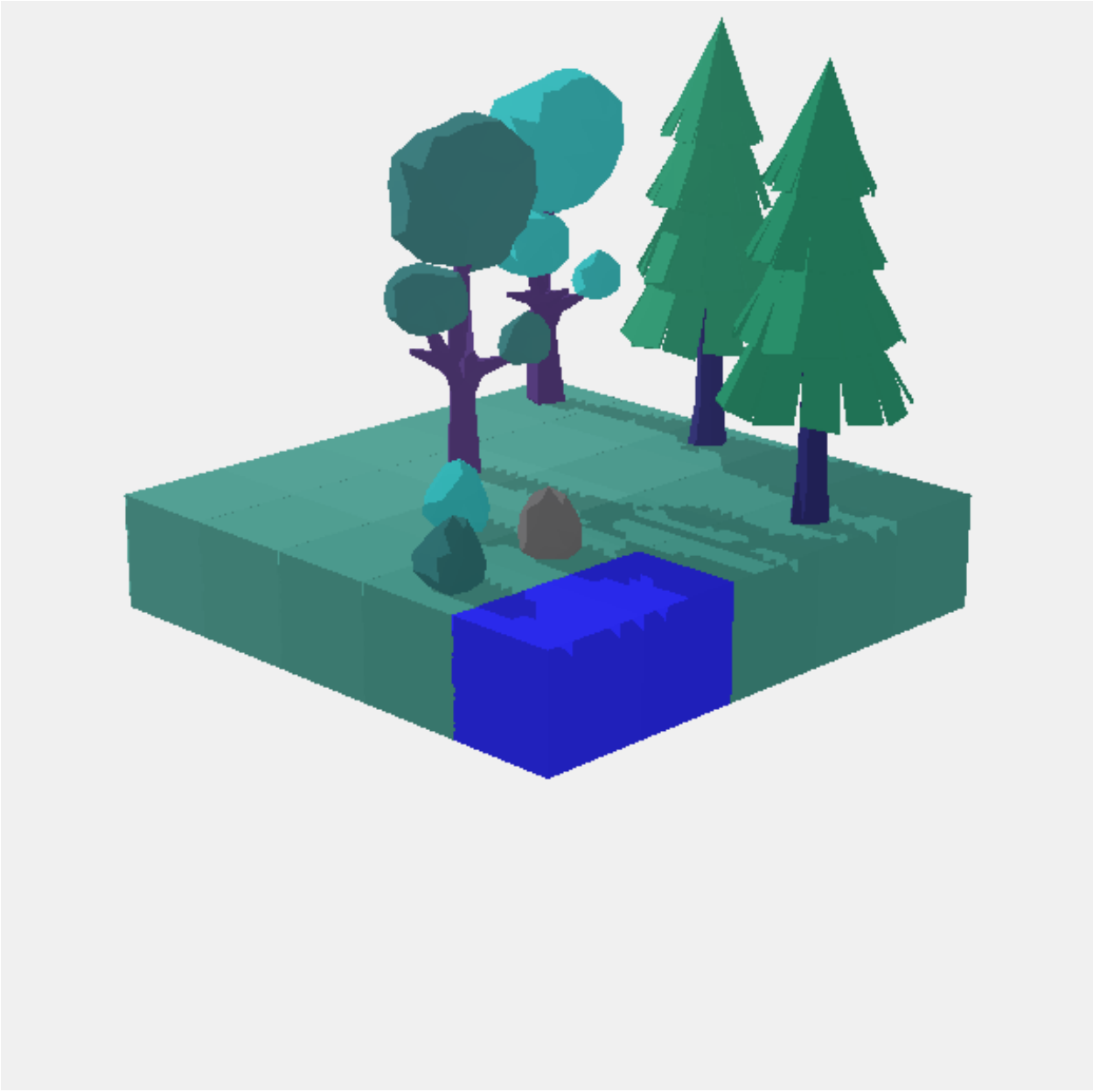

Рисунок 4.3 – Визаулизация объектов сцены в ночное время суток

Рисунок 4.4 – Поворот сцены

4.2 Цель эксперимента

Целью эксперимента является оценка эффективности реализованной программы.

4.3 Описание эксперимента

Ниже приведены технические характеристики устройства, на котором было проведен эксперимент:

1) операционная система Windows-10, 64-bit;
2) оперативная память 8 ГБ;
3) процессор Intel(R) Core(TM) i3-7020U CPU @ 2.30GHz, 2304 МГц, ядер 2, логических процессоров 4.

Для проведения эксперимента замерим время выполнения программы, реализующей способ оптимизации визуализации (алгоритм отбраковки нелицевых граней) и не реализующей данный алгоритм.

Таблица 4.1 – Замеры времени выполнения программы с реализацией алгоритма отбраковки нелицевых граней и без в миллисекундах.

Кол-во объектов сцены	Backface culling	Обычный рендеринг
0	225	456
5	247	536
10	308	590
15	420	673

На рисунке 4.5 представлена зависимость времени работы алгоритмов от количества объектов сцены.

Таким образом, можно сделать вывод, что скорость работы программы линейно зависит от количества объектов сцены, однако благодаря реализованому алгоритму отбраковки нелицевых граней удалось добиться скорости работы программы в 2 раза меньшей, чем скорость работы тех же алгоритмов программы, но без данного способа оптимизации.

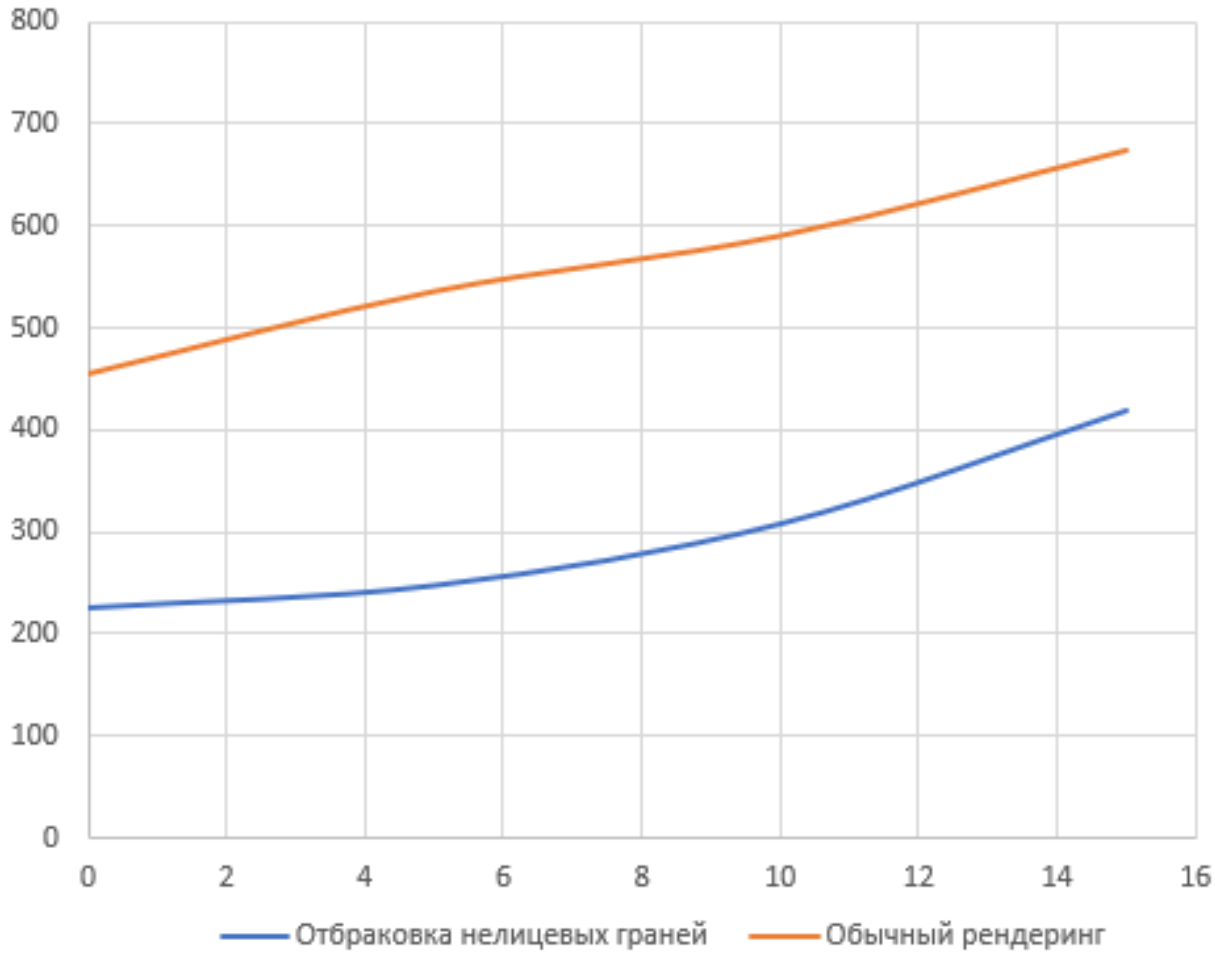

Рисунок 4.5 – Зависимость времени работы алгоритмов от количества объектов сцены

Вывод

В данном разделе было проведено тестирование разработанного программного обеспечения, а также проведена оценка эффективности реализованной программы посредством удаления алгоритма отбраковки нелицевых граней, оптимизирующего визуализацию.

Заключение

В ходе выполнения курсовой работы была достигнута поставленная **цель**: реализовано программное обеспечения для визуализации лесного массива.

Все **задачи курсовой работы** выполнены:

- описанны и формализованы доступные модели;
- проанализированы и выбраны соответствующие алгоритмы компьютерной графики для визуализации сцены и объектов;
- проанализированы доступные языки программирования и выбран один из возможных для реализации курсовой работы;
- реализованы выбранные алгоритмы визуализации;
- реализовано программное обеспечение для визуализации и редактирования лесного массива.

А также в курсовой работе был проведен эксперимент по оценке эффективности работоспособности программы. Для чего из программного обеспечения был удален backface culling, что показало ухудшение работоспособности. По этой причине можно сделать вывод, что данный алгоритм оптимизации улцчшает эффективность программы (приблизительно в 2 раза).

Список использованных источников

1. Муленко В. В. Компьютерные технологии и автоматизированные системы в машиностроении. —М.: РГУ нефти и газа им. И.М.Губкина, 2015. — С. 9.

2. Косников Ю.Н. Поверхностные модели в системах трехмерной компьютерной графики. Учебное пособие. – Пенза: Пензенский государственный университет, 2007. – 60 с.

3. Удаление невидимых линий и поверхностей [Электронный ресурс]. URL: `http://compgraph.tpu.ru/Del_hide_line.htm` (дата обращения: 10.07.2022).

4. Брундасов, С.М. Компьютерная графика: учеб. пособие / С.М. Брундасов. — Брянск: БГТУ, 2004. — С. 172–209.

5. Основы компьютерной графики: учебное пособие / А.Ю. Дёмин; Томский политехнический университет. — Томск: Изд-во Томского политехнического университета, 2011. — С. 137-143.