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:

«Моделирование сцены улицы из библиотеки объектов»

Студент ИУ7-52Б (Группа) _____ (Подпись, дата) Козырнов А. Д. (И. О. Фамилия)

Руководитель курсовой работы _____ (Подпись, дата) Мартынюк Н. Н. (И. О. Фамилия)

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время компьютерное графическое моделирование используется для уменьшений недопониманий между заказчиком и исполнителем. С его помощью можно заранее определить детали проекта.

Цель курсовой работы — моделирование сцены улицы из разработанной библиотеки объектов (дома разной этажности, заправка, светофор) и предоставление возможности изменить положение камеры и источника света, а также сохранение и просмотр разработанных сцен. Для решения поставленной цели необходимо выполнить следующие задачи:

— выбрать алгоритмы компьютерной графики для визуализации трехмерной сцены;

— выбрать язык программирования и среду разработки;

— разработать программное обеспечение и реализовать выбранные алгоритмы визуализации;

— провести замеры временных характеристик разработанного программного обеспечения.

1 Аналитический раздел

В данном разделе проводится выбор и анализ существующих алгоритмов построения изображения трехмерной сцены.

1.1 Описание объектов сцены

Сцена состоит из набора следующих объектов:

— Площадка — правильный параллелипипед, состоящий из сетки ячеек, на которой размещаются объекты сцены. Размеры площадки задаются по ширине и длине одновременно. Ячейка — заранее определенный объект, определенный внутри программы;

— Объект сцены — модель, расположенная на ячейке сцены. Данные объекты могут занимать только одну ячейку сцены одновременно. Каждая такая модель описана массивом граней с нормалями. Все доступные объекты сцены определены заранее, и в программе не предусмотрена возможность изменения старых и добавления новых объектов сцены. Данные модели можно перемещать по сцене и вращать относительно их заданного центра.

— Источник света — точка в пространстве. Источник света имеет координаты, направление и интенсивность освещения. В зависимости от расположения и направления источника света относительно камеры определяются тени от объектов сцены.

— Камера — точка в пространстве. От ее положения и направления взгляда пользователь может наблюдать сцену с определенного ракурса. При изменении положения или направления камеры, обзор на сцену изменяется.

1.2 Анализ способов задания моделей

В компьютерной графике способов задания моделей три: каркасная, поверхностная и объемная [1].

Каркасная модель представляется набором вершин и ребер, соединяющих вершины. Данное представление не всегда точно передает информацию об объектах сцены, так как может неоднозначно определять его форму.

Поверхностная модель — это модель, которая каким-либо образом задана поверхностями. Описание прверхности может быть аналитическим или с помощью полигонов.

Твердотельная форма задания модели отличается от поверхностной тем, что в таких моделях задается информация о материале объекта.

1.2.1 Выбор способа задания модели

Для решения поставленной задачи необходимо правильное восприятие форм объекта, поэтому каркасная модель не подходит для решения поставленной задачи. Твердотельная модель также не подходит для решения задачи, так как неважно из чего состоят объекты сцены. Поверхностная модель точно определяет форму объекта сцены, поэтому в данной программе можноиспользовать поверхностную модель.

1.3 Анализ алгоритмов удаления невидимых ребер и граней

В компьюетрной графике важнейшей задачей является удаление невидимых ребер и граней. Данные алгоритмы определяют, какие ребра и грани видны наблюдателю, находящегося в определенной точке пространства относительно этих ребер и граней.

Алгоритмы делятся на способы решения. Первая группа алгоритмов решают задачу в объектом пространстве, другие — задачу в пространстве изображения. Алгоритмы, работающие с объектным пространством, дают более точные результаты, однако медленнее алгоритмов, работающих в пространстве изображения.

1.3.1 Алгоритм Робертса

Данный алгоритм работает в объектном пространтсве. Решает задачу только с выпуклыми телами. Алгоритм выполняется в 4 этапа [2]:

— подготовка исходных данных — разбиение невыпуклых объектов на выпуклые, составление матрицы тела каждого из объектов;

— удаление ребер, экранируемых самим телом;

— удаление ребер, экранируемых другими телами;

— удаление линий пересечения тел, экранируемых самими телами и другими телами, связанными отношением протыкания.

Преимущества:

— высокая точность определения видимости граней без приближений.

Недостатки:

— Тела сцены должны быть выпуклыми, что не всегда так.

— Высокая вычислительная сложность при большом количестве граней. Рост вычислительной сложности — квадрат количества граней.

1.3.2 Z-буфер

Данный подход работает в пространстве изображения.

Алгоритм Z-буфер [3] использует два буфера: буфер глубины и буфер кадра.

Буфер глубины определяет для каждого пикселя информацию о глубине ближайшего к наблюдателю объекта. В буфере кадра хранится цвет пикселей.

Алгоритм начинает работу с инициализации буфера глубины максимальным значением глубины. При растеризации (отрисовке) каждого полигона вычисляется глубина каждого из его пикселей. Если глубина очередного пикселя меньше соответствующей глубины в буфере глубины, то изменяется соответствующая информация в буфере кадра и значение глубины этого пикселя обновляется.

Преимущества:

— возможность отрисовки объектов любой сложности;

— Линейная зависимость от количества пикселей, отрисовываемых на изображении;

— простота реализации.

Недостатки:

— необходима дополнительная память для хранения буфера глубины;

— возможны артефакты при недостаточной точности буфера.

1.3.3 Алгоритм обратной трассировки лучей

Наблюдатель видит объект посредством испускаемого источником света, который падает на этот объект и согласно законам оптики некоторым путем доходит до глаза наблюдателя. Отслеживать пути лучей от источника к наблюдателю неэффективно с точки зрения вычислений, поэтому наилучшим способом будет отслеживание путей в обратном направлении, то есть от наблюдателя к объекту.

Предполагается, что сцена уже преобразована в пространство изображения, а точка, в которой находится наблюдатель, находится в бесконечности на положительной полуоси , и поэтому световые лучи параллельны этой же оси. При этом каждый луч проходит через центр пикселя растра до сцены. Траектория каждого луча отслеживается для определения факта пересечения определенных объектов сцены с этими лучами. При этом необходимо проверить пересечение каждого объекта сцены с каждым лучом, а пересечение с z_{min} представляет видимую поверхность для данного пикселя. Если же точка наблюдателя находится не в бесконечности, то есть в рас- смотрении фигурирует перспективная проекция, то предполагается, что сам наблюдатель по-прежнему находится на положительной полуоси , а сам растр при этом перпендикуляром оси . Задача будет состоять в том, чтобы построить одноточечную центральную проекцию на картинную плоскость.

Данный алгоритм позволяет создавать фотореалистичные изображения с отражениями и преломлениями лучей света.

Преимущества:

— возможность отрисовки объектов любой сложности;

— точное моделирование оптических свойств световых лучей.

Недостатки:

— высокая вычислительная сложность.

1.3.4 Выбор алгоритма удаления невидимых ребер и граней

Так как не требуется высокая реалистичность от объектов сцены, а форма ее объектов может быть как выпуклой, так и невыпуклой, алгоритм

Робертса не подходит для решения задачи. Алгоритм обратной трассировки лучей также не подходит, потому что не требуется фотореалистичные изображения с отражениями и преломлениями лучей света. Поэтому для решения поставленной задачи был выбран алгоритм Z-буфер, потому что в достаточной мере позволяет определить видимость граней и ребер и имеет приемлемую скорость вычислений, в основном не зависящую от количества объектов сцены.

1.4 Анализ методов закраски

Закраска определяет визуальную составляющую объектов сцены и влияет на их восприятие.

1.4.1 Плоская закраска

Плоская закраска подразумевает, что источник света находится в бесконечности. По этой причине считается, что угол падения лучей света на поверхность одинаков. Так как интенсивность зависит от угла падения, то это значит, что вся грань будет закрашена одним цветом.

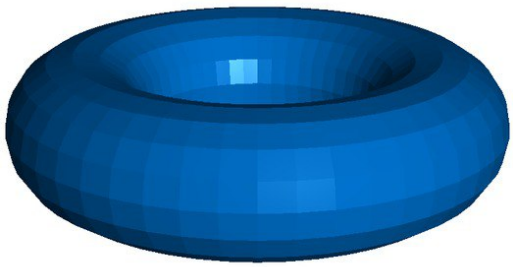


Рисунок 1.1 – Плоская закраска

Преимущества:

— быстрая скорость обработки;

— простота реализации.

Недостатки:

— возможны образования ребер между гранями, которых на самом деле нет;

— не учитывается плавный переход интенсивности освещения от источника света, находящегося на конечном расстоянии.

1.4.2 Закраска по Гуро

Закраска по Гуро [4] интерполирует интенсивность освещения между вершинами одной грани. Интенсивность в вершинах вычисляется по нормалям в этих вершинах. Интерполяция происходит в два этапа: интерполирование по ребрам и интерполирование между ребрами.

Данная закраска хорошо подходит для диффузиозного отражения.

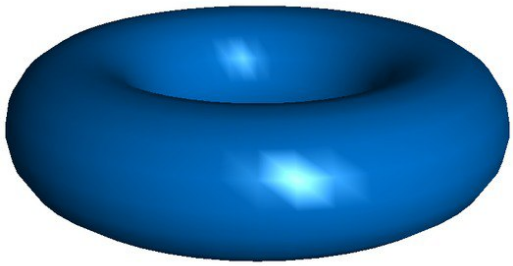


Рисунок 1.2 – Закраска по Гуро

Преимущества:

— градиент интенсивности освещения между вершинами грани;

Недостатки:

— возможна потеря бликов света, так как они могут не попадать на вершины граней.

1.4.3 Закраска по Фонгу

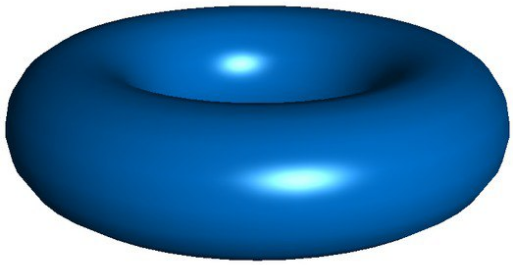


Рисунок 1.3 – Закраска по Фонгу

Закраска по Фонгу [5] работает аналогично закраске по Гуро, однако вместо интерполирование интенсивности света на вершинах граней, интерполируются нормали этих граней. Поэтому для рассчета интенсивности освещения

потребуется в каждом пикселе считать интенсивность относительно интерполированной нормали.

Данная закраска хорошо подходит для зеркального отражения.

Преимущества:

— высокая реалистичность;

— блики света;

— плавные переходы света и тени.

Недостатки:

— более высокая вычислительная сложность относительно других алгоритмов;

1.4.4 Выбор алгоритма закраски

Так как не требуется высокая реалистичность и модели представлены в виде граней с вершинами, которые не имеют собственных нормалей, закраска по Фонгу не подходит для решения задачи. Так как объекты сцены будут достаточно простыми, можно сделать выбор в пользу более производительного алгоритма закраски. Поэтому выбираем алгоритм простой закраски, так как работает быстрее остальных и в достаточной мере обеспечивает визуализацию объектов сцены.

1.5 Вывод

В разделе были проанализированы существующие алгоритмы построения трехмерной сцены и выбраны методы для решения поставленной задачи. Был выбран алгоритм Z-буфер с плоской закраской.

2 Конструкторский раздел

В данном разделе представлены алгоритмы, выбранные для решения задачи, рассмотрены структуры данных.

2.1 Требования к программному обеспечению

Программа должна выполнять следующую функции:

— создание сцены с заданным размером полотна;

— перемещение и поворот камеры;

— перемещение источника света;

— добавление и удаление объектов сцены;

— перемещение объектов сцены.

2.2 Общий алгоритм построения сцены

На рисунках (2.1) — (2.4) представлена IDEF0 диаграмма алгоритма построения сцены.

Рисунок 2.1 – Схема алгоритма уровня A0

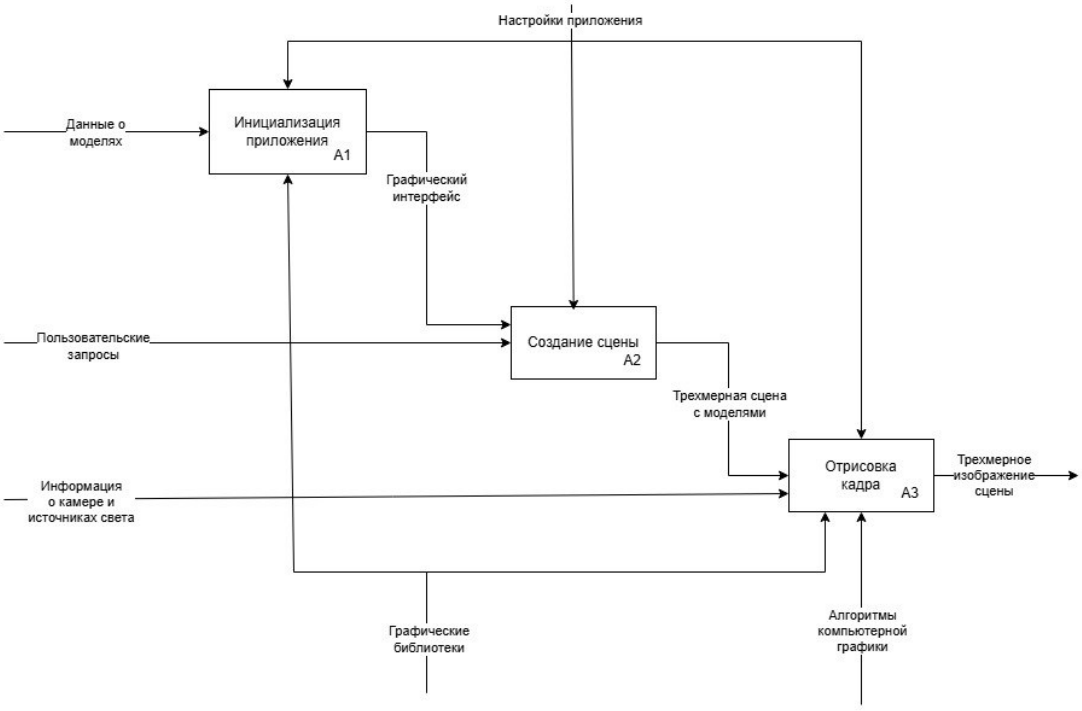


Рисунок 2.2 – Декомпозиция уровня A0

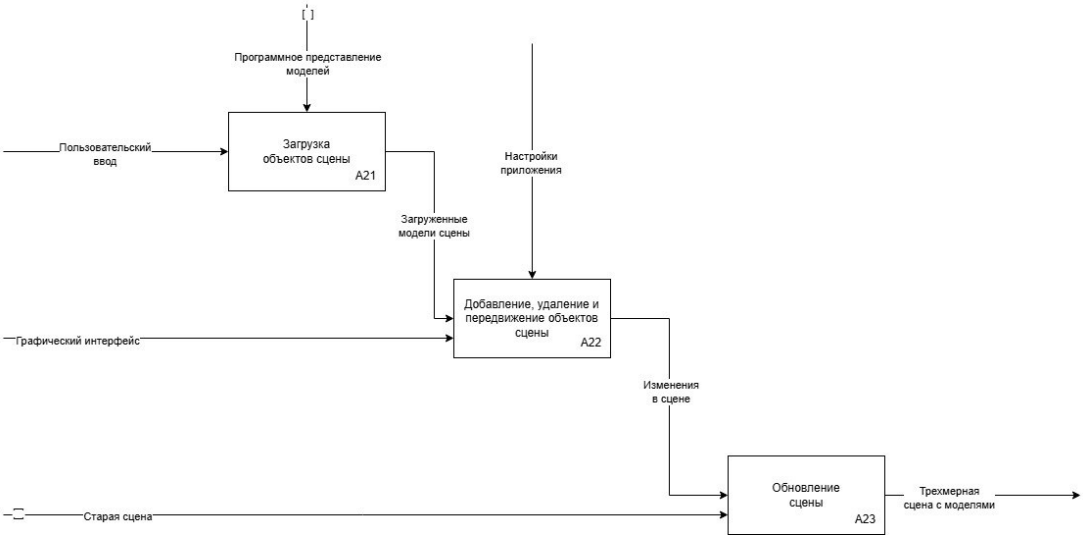


Рисунок 2.3 – Декомпозиция уровня A2

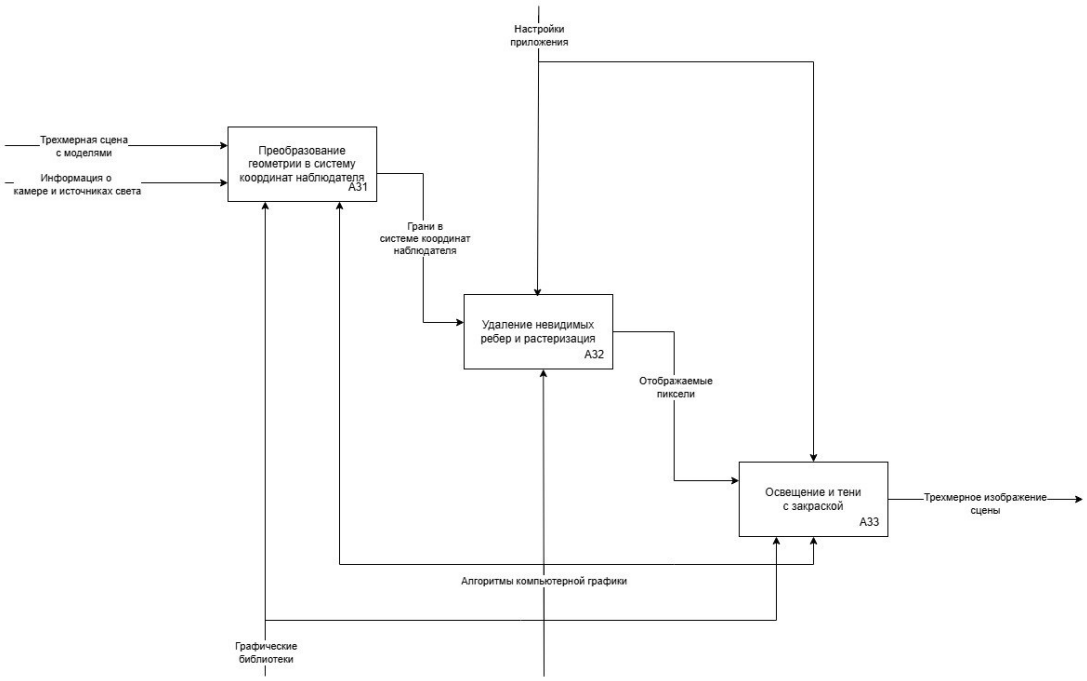


Рисунок 2.4 – Декомпозиция уровня A3

2.3 Модифицированный алгоритм Z-буфер

Для каждого источника света нужно инициализировать теневой Z-буфер. Далее определить глубину для каждого пикселя теневого буфера источника света для точки наблюдения из источника света.

После заполнения всех теневых буферов, следует выполнять основной алгоритм Z-буфера относительно положения наблюдателя. При этом, если очередной пиксель виден, нужно проверить, видим ли этот пиксель относительно какого-либо источника света.

Для определения видимости точки из буфера кадра, нужно рассматриваемую точку (x, y, z) преобразовать из системы координат наблюдателя в систему координат источника света (x', y', z'). Если в теневом буфере $z'(x', y') < z_{shadow}(x', y')$, то точка видна. Иначе она находится в тени.

На рисунке (2.1) показан модифицированный алгоритм Z-буфера.

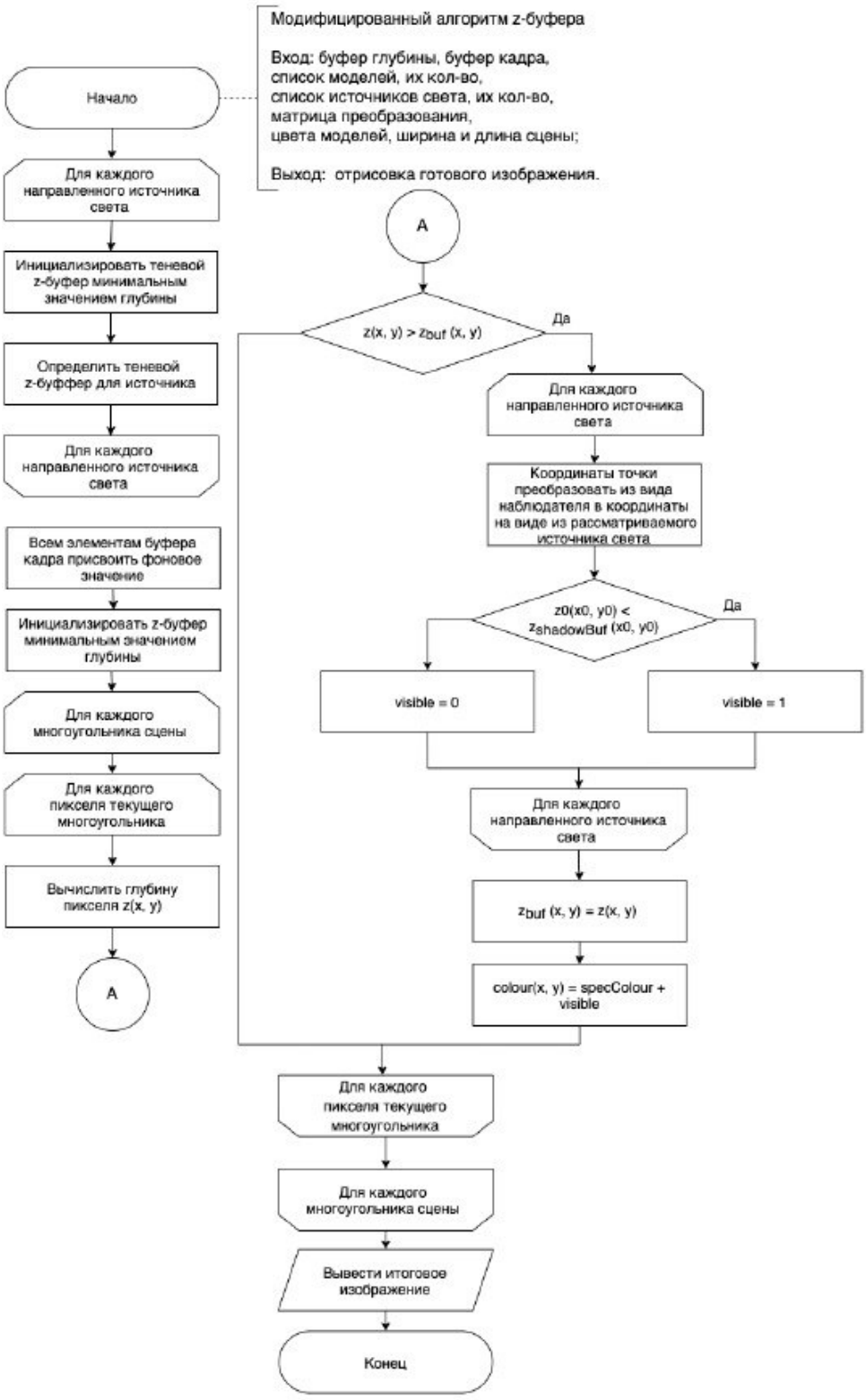


Рисунок 2.5 – Модифицированный алгоритм Z-буфера

2.4 Используемые структуры данных и классы

Для реализации работы программы используются следующие структуры данных:

1. Сцена
    - Массив объектов сцены;
    - Камера и источник света;
    - Методы добавления и удаления объектов сцены.
2. Составная трехмерная модель
    - Массив объектов сцены, из которых состоит объект.
3. Объект сцены
    - Матрица преобразования;
    - Массив граней.
4. Грань
    - 3 вершины;
    - Нормаль.
    - Цвет;
5. Вершина
    - Координаты в пространстве;
6. Камера
    - Координаты в пространстве;
    - 3 вектора, определяющих направление камеры.
7. Источник света
    - Координаты в пространстве;
    - 3 вектора, определяющих направление источника света;
    - Интенсивность света.

2.5 Вывод

В данном разделе были представлены алгоритмы, выбранные для решения задачи, были рассмотрены структуры данных.

3 Технологический раздел

В данном разделе рассматривается выбор средств реализации, описывается структура классов программы и приводится интерфейс программного обеспечения.

3.1 Средства реализации

Для реализации программного обеспечения был выбран язык C++ [6]. Это обусловлено следующими причинами:

— C++ обладает высокой вычислительной производительностью;

— обладает большим количеством литературы и примеров;

При написании программного обеспечения были задействованы библиотеки SDL2 [7], GLM [8] и Dear ImGui [9]. Библиотека SDL2 представляет из себя библиотеку, представляющую из себя окно и дающая исполнителю функции для попиксельной манипуляции с изображением. GLM библоитека используется по причине наличия математических векторов и функций, полезных для компьютерной графики. Dear ImGui библиотека используется для реализации пользовательского интерфейса.

3.2 Структура классов программного обеспечения

В данной диаграмме на рисунке (3.1) показаны основные классы программного обеспечения.

Описание некоторых классов, которые решают задачу программного обеспечения:

— Model — модель сцены, содержащая некоторое описание поверхностной модели объекта;

— SurfaceModel — некоторая реализация поверхностной модели объекта. Состоит из массива граней Facet.

— CompositeObject — класс, позволяющий группировать объекты в группы.

— Scene — класс, представляющий из себя набор массив объектов и методов для добавления и удаления объектов.

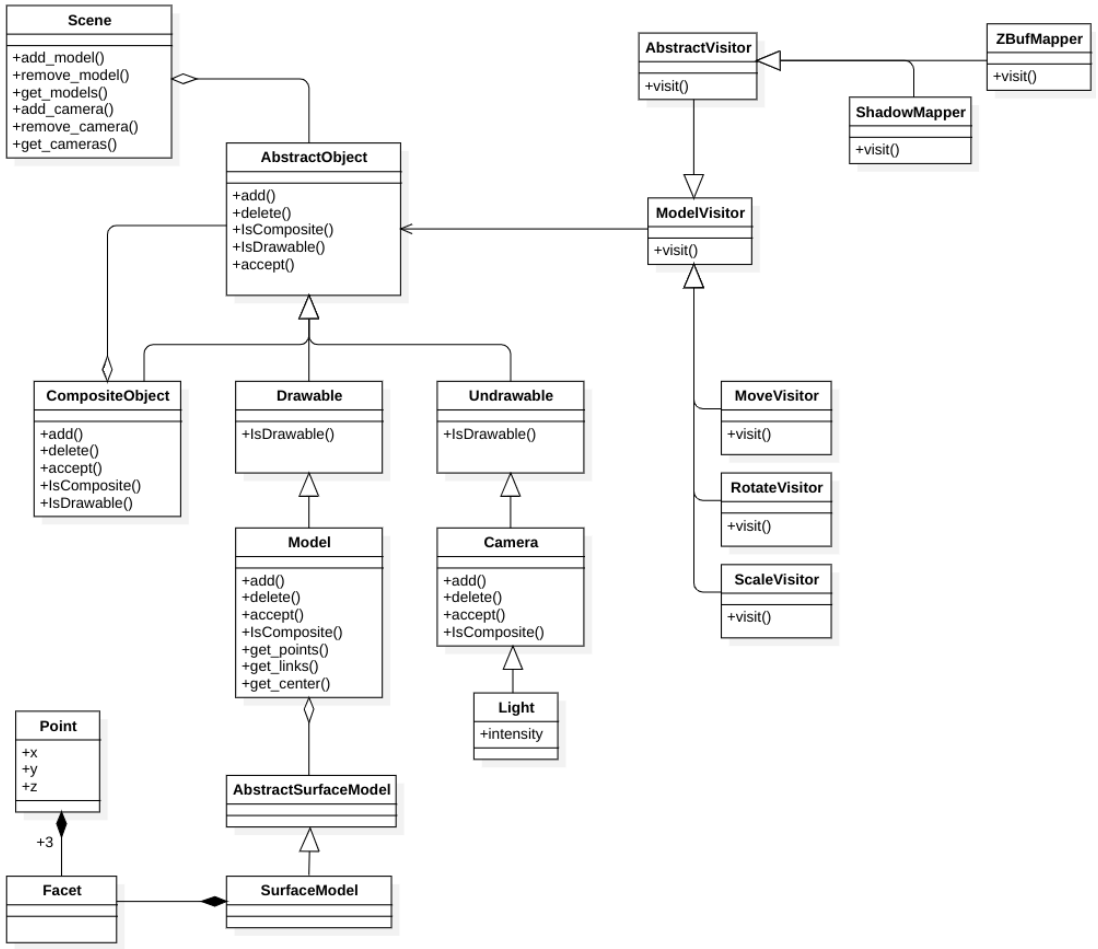


Рисунок 3.1 – Структура основных классов программного обеспечения

— Camera — класс, представляющий из себя точку наблюдения.

— Light — класс, представляющий из себя точечный направленный источник света.

— ZBufMapper — класс, представляющий из себя модифицированный алгоритм Z буффером.

— ShadowMapper — класс, представляющий из себя алгоритм заполнения теневого буфера.

3.3 Реализация заполнения теневого буфера

В листинге 3.1 представлен алгоритм заполнения теневого буффера источника света.

Листинг 3.1 – Реализация заполнения теневого буфера

```
1  void NewDrawVisitor::shadow_for_model(std::list<Facet>& facets,
       glm::mat4 transform, std::shared_ptr<Light>& light)
2  {
3      std::array<glm::vec3, 3> points;
4
5      glm::mat4x4 model = transform;
6      glm::mat4x4 projection = light->get_perspective_matrix();
7      glm::mat4x4 view = light->get_view_matrix();
8      glm::vec4 viewport(0.0f, 0.0f,
           ControlSystem::Buffer::shadow_res,
           ControlSystem::Buffer::shadow_res);
9      glm::mat4 proj = projection * view;
10
11     auto& shadowMap = light->shadow_buffer;
12
13
14     for (Facet& facet : facets)
15     {
16         glm::vec3 p0 = facet.A;
17         glm::vec3 p1 = facet.B;
18         glm::vec3 p2 = facet.C;
19
20
21         p0 = glm::project(p0, model, proj, viewport);
22         p1 = glm::project(p1, model, proj, viewport);
23         p2 = glm::project(p2, model, proj, viewport);
24
25         if (glm::isnan(p0.x) || glm::isnan(p0.y) ||
               glm::isnan(p0.z) || glm::isnan(p1.x) || glm::isnan(p1.y)
               || glm::isnan(p1.z) || glm::isnan(p2.x) ||
               glm::isnan(p2.y) || glm::isnan(p2.z))
26         {
27             continue;
28         }
29         if (p0.z < 1.0f || p1.z < 1.0f || p2.z < 1.0f)
30         {
31             continue;
32         }
33
34         std::array<glm::vec3, 3> transformedDots = {
```

```
35              p0, p1, p2
36          };
37          if (transformedDots[0].y > transformedDots[1].y)
38              std::swap(transformedDots[0], transformedDots[1]);
39          if (transformedDots[0].y > transformedDots[2].y)
40              std::swap(transformedDots[0], transformedDots[2]);
41          if (transformedDots[1].y > transformedDots[2].y)
42              std::swap(transformedDots[1], transformedDots[2]);
43          if (transformedDots[2].y - transformedDots[0].y <= 1)
44          {
45              continue;
46          }
47          int x1 = round(transformedDots[0].x);
48          int x2 = round(transformedDots[1].x);
49          int x3 = round(transformedDots[2].x);
50          double z1 = transformedDots[0].z;
51          double z2 = transformedDots[1].z;
52          double z3 = transformedDots[2].z;
53          int y1 = round(transformedDots[0].y);
54          int y2 = round(transformedDots[1].y);
55          int y3 = round(transformedDots[2].y);
56  #pragma omp parallel for
57          for (int curY = (y1 < 0) ? 0 : y1;
58               curY < ((y2 >= (int) ControlSystem::Buffer::height) ?
                     (int) ControlSystem::Buffer::height - 1 : y2);
                     curY++)
59          {
60              double alnc = 0;
61              if (y1 != y2)
62                  alnc = (double)(curY - y1) / (y2 - y1);
63              double blnc = 0;
64              if (y1 != y3)
65                  blnc = (double)(curY - y1) / (y3 - y1);
66              int xA = round(x1 + (x2 - x1) * alnc);
67              int xB = round(x1 + (x3 - x1) * blnc);
68              double zA = z1 + (z2 - z1) * alnc;
69              double zB = z1 + (z3 - z1) * blnc;
70              if (xA > xB)
71              {
72                  std::swap(xA, xB);
73                  std::swap(zA, zB);
```

```
74              }
75              if (xA < 0)
76                  xA = 0;
77              if (xB >= (int) ControlSystem::Buffer::width)
78                  xB = (int) ControlSystem::Buffer ::width - 1;
79              NewDrawVisitor::interpolateRowIntoShadowMap(shadowMap,
                    xA, xB, zA, zB, curY);
80          }
81
82  #pragma omp parallel for
83          for (int curY = (y2 < 0) ? 0 : y2;
84               curY <= ((y3 >= (int) ControlSystem::Buffer::height) ?
                     (int) ControlSystem::Buffer::height - 1 : y3);
                     curY++)
85          {
86              double alnc = 0;
87              if (y2 != y3)
88                  alnc = (double)(curY - y2) / (y3 - y2);
89              double blnc = 0;
90              if (y1 != y3)
91                  blnc = (double)(curY - y1) / (y3 - y1);
92              int xA = round(x2 + (x3 - x2) * alnc);
93              int xB = round(x1 + (x3 - x1) * blnc);
94              double zA = z2 + (z3 - z2) * alnc;
95              double zB = z1 + (z3 - z1) * blnc;
96              if (xA > xB)
97              {
98                  std::swap(xA, xB);
99                  std::swap(zA, zB);
100             }
101             if (xA < 0)
102                 xA = 0;
103             if (xB >= (int) ControlSystem::Buffer::width)
104                 xB = (int) ControlSystem::Buffer ::width - 1;
105             NewDrawVisitor::interpolateRowIntoShadowMap(shadowMap,
                    xA, xB, zA, zB, curY);
106         }
107     }
108 }
```

3.4 Интерфейс программного обеспечения

На рисунках (3.4) — (3.11) представлены элементы интерфейса программного обеспечения.

Рисунок 3.2 – Интерфейс программного обеспечения в виде панели задач

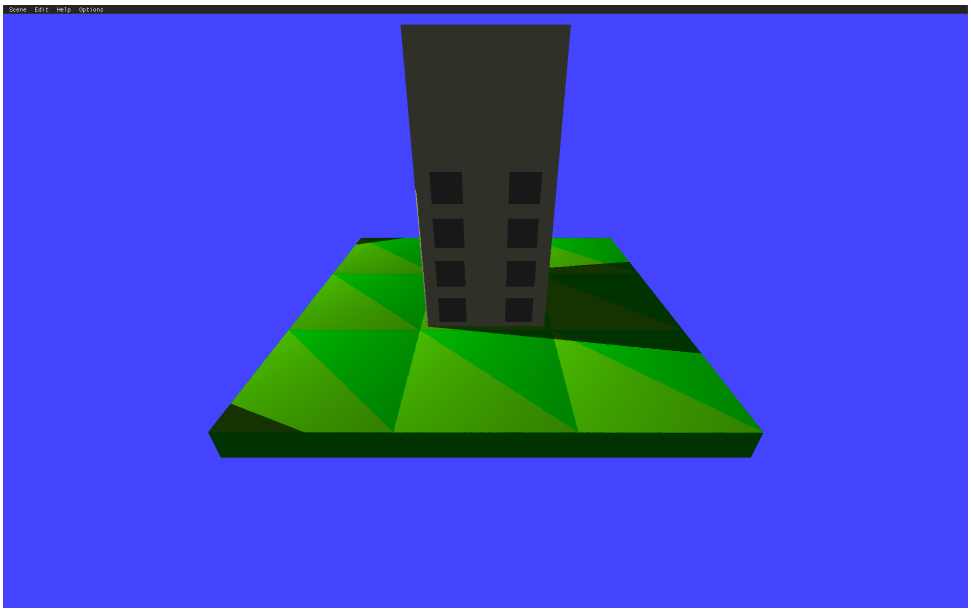


Рисунок 3.3 – Весь интерфейс программного обеспечения, включая выводимое изображение

На рисунках (3.7) — (3.8) зеленые ячейки означают, что в данное место можно поставить объект сцены, когда синие — нельзя. Количество ячеек зависит от заданного размера полотна сцены.

На рисунке (3.8) показано всего 3 опции в каждой ячейке: создание нового объекта, удаление объекта (если имеется) и изменение его положения на полотне.

На рисунке (3.9) предоставляется список из заранее спроектированных объектов сцены, которые можно поставить на полотно сцены.

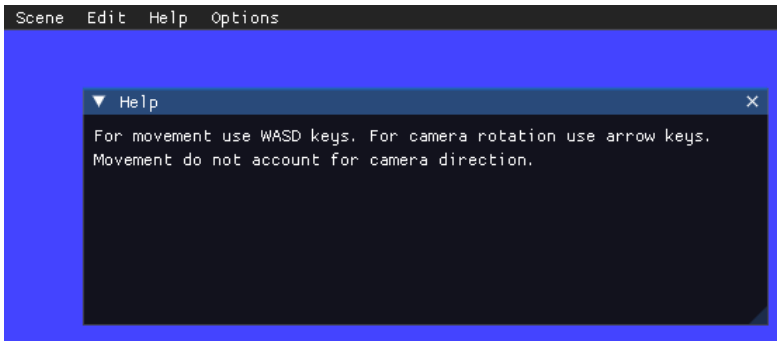


Рисунок 3.4 – Интерфейс окна с помощью

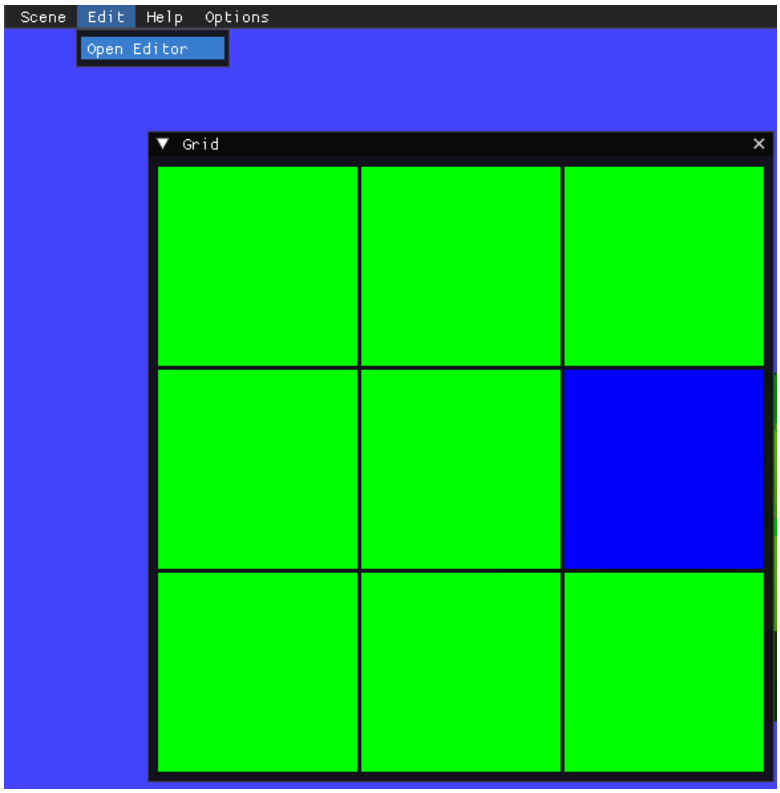


Рисунок 3.5 – Интерфейс окна редактирования

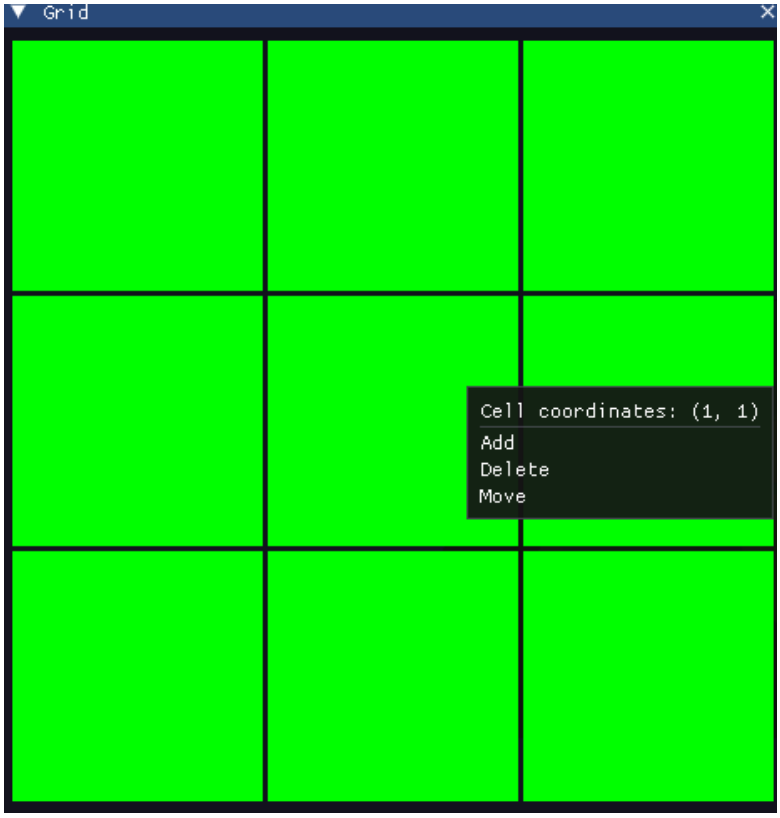


Рисунок 3.6 – Интерфейс окна редактирования с опциями

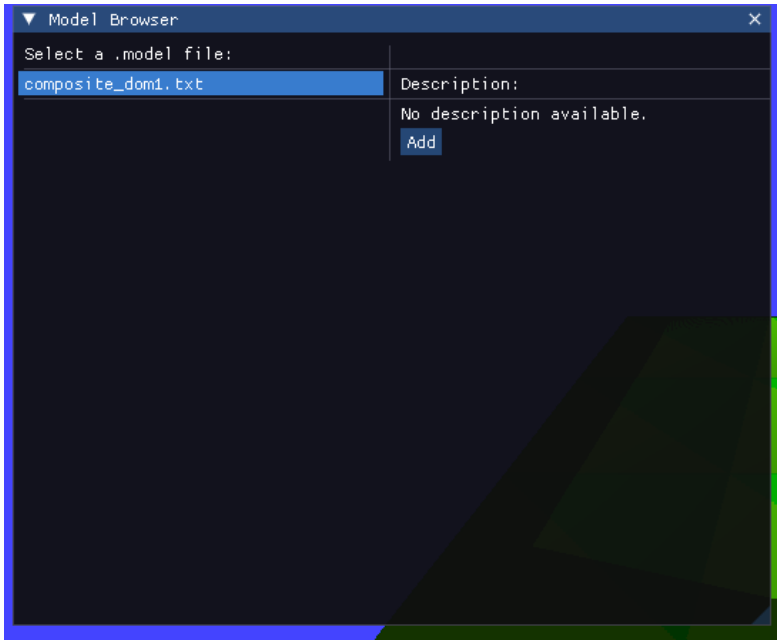


Рисунок 3.7 – Интерфейс окна выбора моделей

Рисунок 3.8 – Интерфейс опций работы со сценой

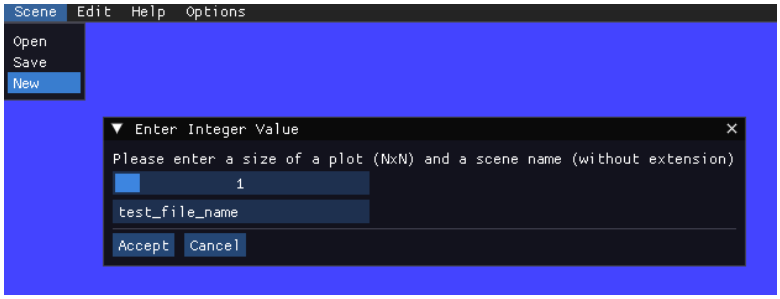


Рисунок 3.9 – Интерфейс окна создания новой сцены

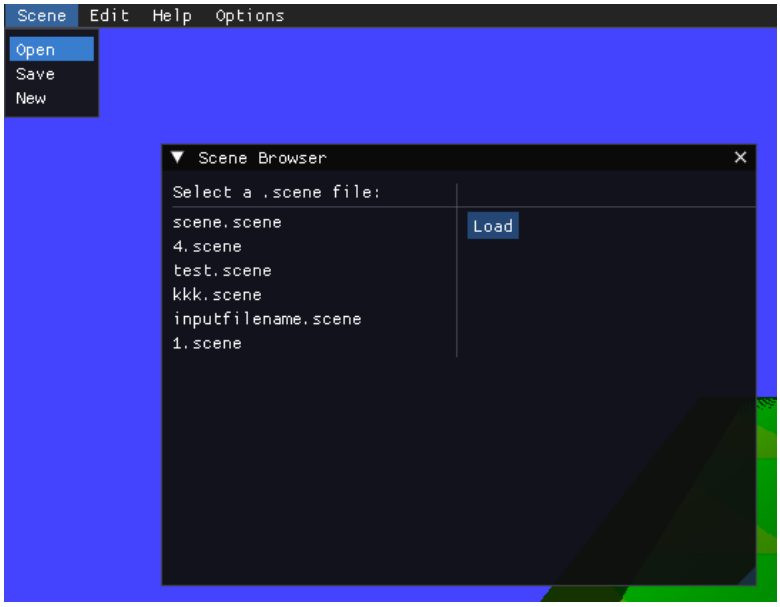


Рисунок 3.10 – Интерфейс окна открытия ранее созданной сцены

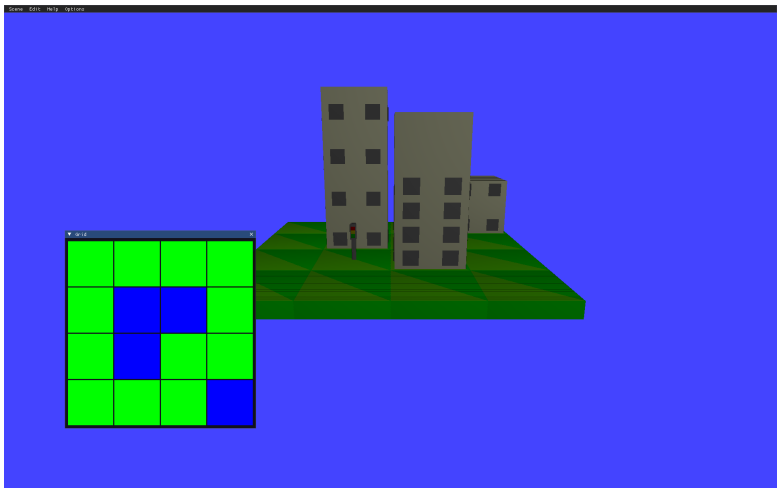


Рисунок 3.11 – Выводимое изображение при одинаковом направлении и расположении источника света и наблюдателя

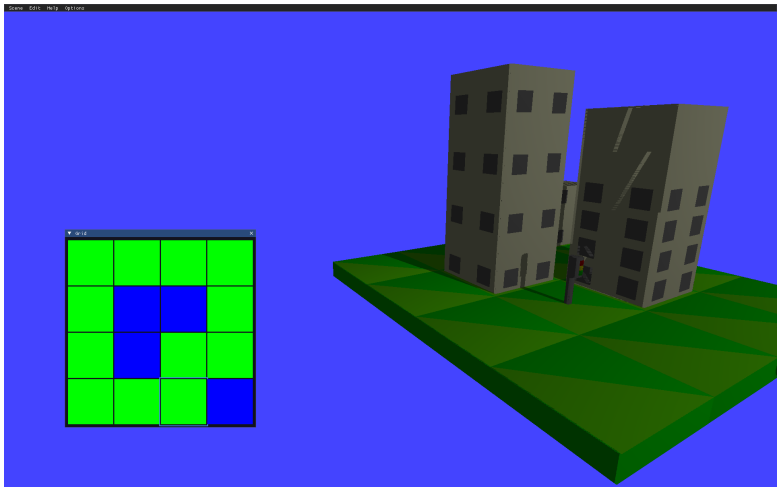


Рисунок 3.12 – Выводимое изображение при измененном положении наблюдателя

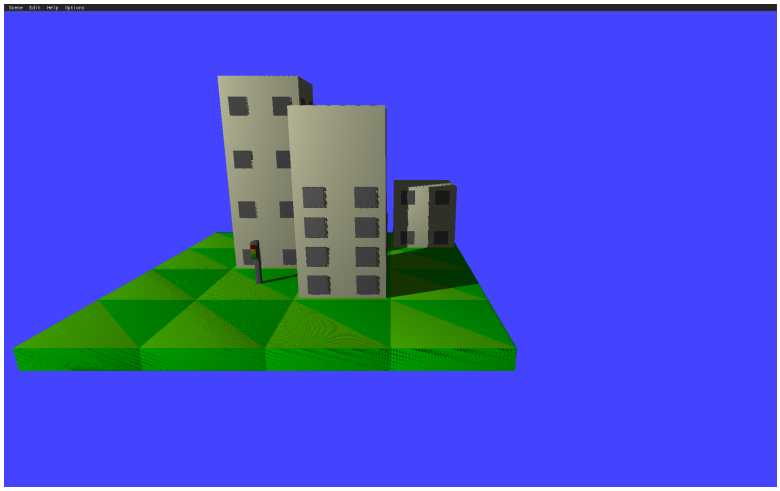


Рисунок 3.13 – Выводимое изображение при измененном положении источника света

3.5 Вывод

В данном разделе был рассмотрен выбор средств реализации, описана структура классов программы и приведен интерфейс программного обеспечения.

4 Исследовательский раздел

В данном разделе приведены технические характеристики устройства, на котором проводилось измерение времени работы программного обеспечения, а также результаты замеров времени.

4.1 Технические характеристики

Технические характеристики оборудования, на котором проводилось измерение:

— операционная система — Linux;

— процессор — AMD Ryzen 7 5800H;

— ОЗУ — 16 Гб;

4.2 Цель эксперимента

Целью эксперимента является сравнение скорости отрисовки кадра в зависимости от количества граней на сцене и присутствии источника света модифицированным алгоритмом Z-буфер.

4.3 Результаты эксперимента

В каждом эксперименте создается полотно сцены некоторого размера без объектов, расположенных на ней. Для каждого такого замера времени позиция наблюдателя и источника света находятся относительно замеряемой сцены в одном и той же позиции. Эксперименты проводились 20 раз и усреднялись. На сцене находился только 1 источник света. На рисунке 4.1 и на таблице 4.1 представлена зависимость времени отрисовки кадра от количества граней на сцене.

Таблица 4.1 – Зависимость времени отрисовки кадра от количества граней на сцене

Количество граней	С иточником света, сек.	Без источника света, сек.
12	0.4	0.34
48	1.25	0.41
108	1.45	0.42
192	1.7	0.46
300	1.76	0.48
432	1.91	0.49
588	2.13	0.5
768	2.34	0.54
972	2.41	0.57
1200	2.55	0.59

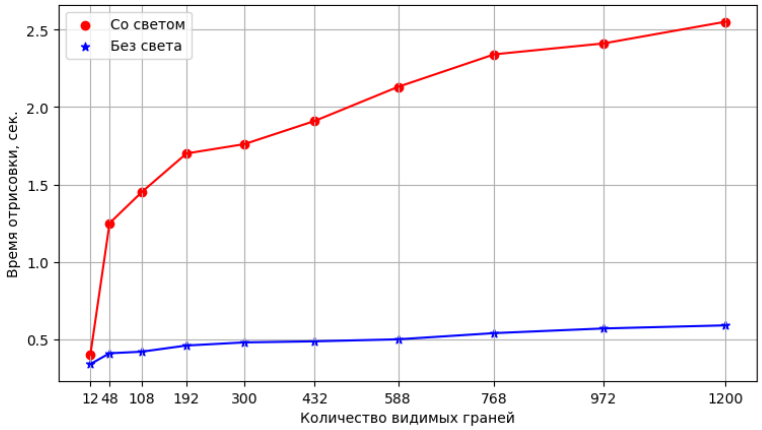


Рисунок 4.1 – Зависимость времени отрисовки кадра от количества граней на сцене

4.4 Вывод

В данном разделе были приведены технические характеристики устройства, на котором проводилось измерение времени работы программного обеспечения, а также результаты замеров времени.

Без учета освещенности алгоритм Z-буфер работает быстрее, чем с учетом освещенности. Это связано с тем, что для каждого источника света необходимо проинициализировать и заполнить теневой Z-буфер, а потом для каждого пикселя при растеризации грани выполнять преобразование координат в координаты в теневом Z-буфере для определения теней.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы цель курсовой работы была достигнута: было разработано программное обеспечение для моделирования сцены улицы из разработанной библиотеки объектов с возможностью изменять положение камеры и источника света.

Для достижения цели курсовой работы были решены следующие задачи:

— выбраны алгоритмы компьютерной графики для визуализации трехмерной сцены;

— выбран язык программирования и среда разработки;

— разработано программное обеспечение и реализованы выбранные алгоритмы визуализации;

— проведены замеры временных характеристик разработанного программного обеспечения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Прилипенко М. А.* Математические модели представления компьютерной графикой. — 2005.
2. *Робертс Г. А.* Алгоритм Робертса удаления невидимых линий и граней. — 1964.
3. *Catmull E. E.* A subdivision algorithm for computer display of curved surfaces. — 1974.
4. *Gouraud H.* Computer display of curved surfaces. — 1971.
5. *Phong F. T.* Illumination for computer generated pictures. — 1998.
6. Стандарт языка C++, Дата обращения: 24.09.2024. — 2024. — URL: `https://isocpp.org/std/the-standard`.
7. Документация библиотеки SDL, Дата обращения: 25.09.2024. — 2024. — URL: `https://wiki.libsdl.org/SDL3/FrontPage`.
8. Документация к OpenGL Mathematics (GLM), Дата обращения: 25.09.2024. — 2024. — URL: `http://glm.g-truc.net/glm.pdf`.
9. Мануал библиотеки ImGui, Дата обращения: 25.09.2024. — 2024. — URL: `https://pthom.github.io/imgui_manual_online/manual/imgui_manual.html`.

ПРИЛОЖЕНИЕ А

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)

КУРСОВАЯ РАБОТА НА ТЕМУ:
«Моделирование сцены улицы из библиотеки объектов»

Студент: Козырнов А.Д.
Руководитель:Мартынюк Н.Н.

Москва, 2024 г.

Цель и задачи

Цель – моделирование сцены улицы из разработанной библиотеки объектов (дома разной этажности, заправка, светофор) и предоставление возможности изменить положение камеры и источника света, а также сохранение и просмотр разработанных сцен.

Задачи:

- выбрать алгоритмы компьютерной графики для визуализации трехмерной сцены;
- выбрать язык программирования и среду разработки;
- разработать программное обеспечение и реализовать выбранные алгоритмы визуализации;
- провести замеры временных характеристик разработанного программного обеспечения.

Объектов сцены и их описание

- Площадка
- Объекты сцены
- Источник света
- Камера

Используемые алгоритмы и подходы

- Поверхностная модель
- Модифицированный алгоритм, использующий Z-буфер
- Плоская закраска

Алгоритм построения изображения

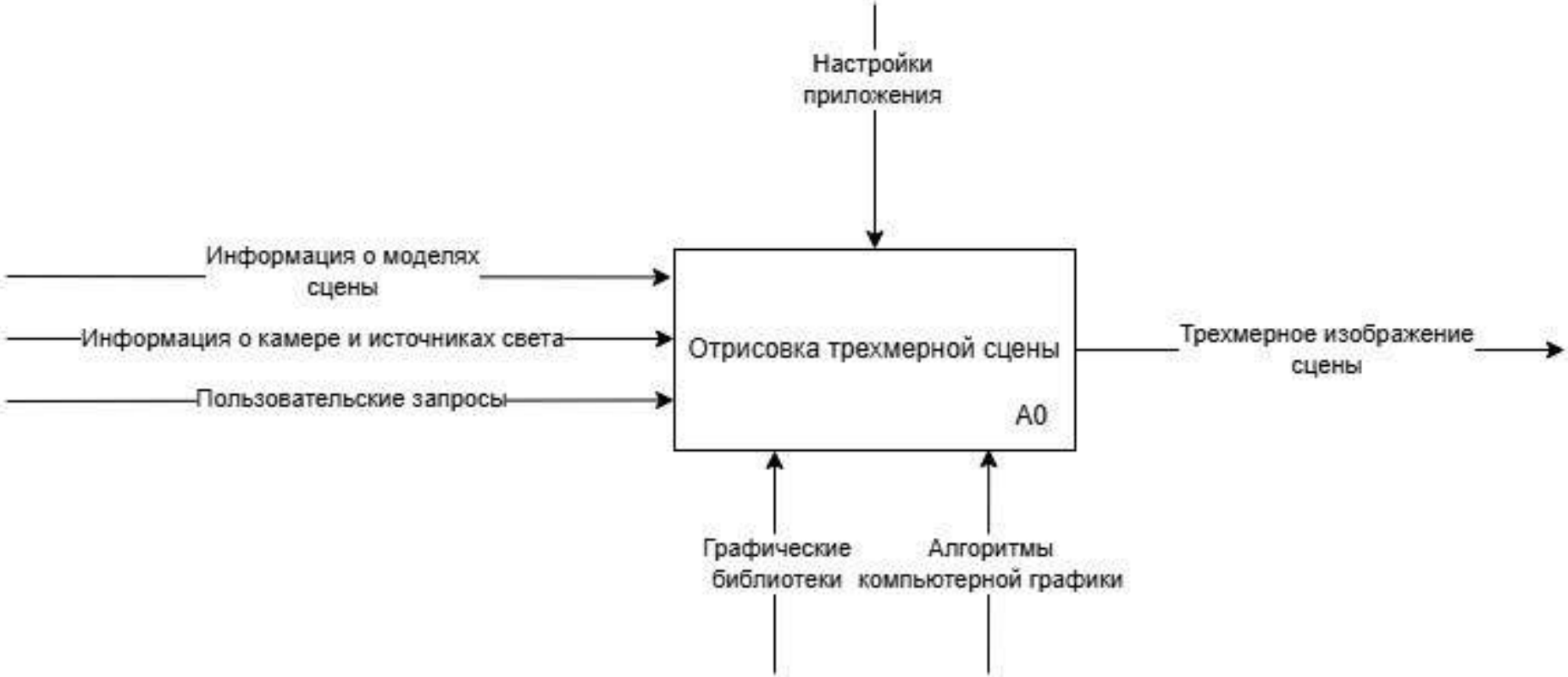
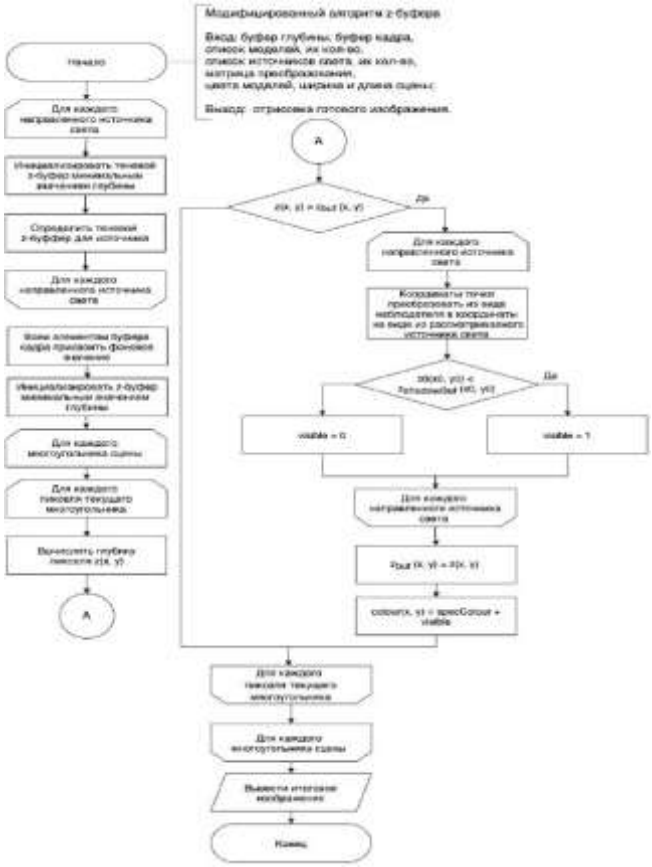


Схема алгоритма, использующая модифицированный Z-буфер



Выбор языка программирования и среды разработки

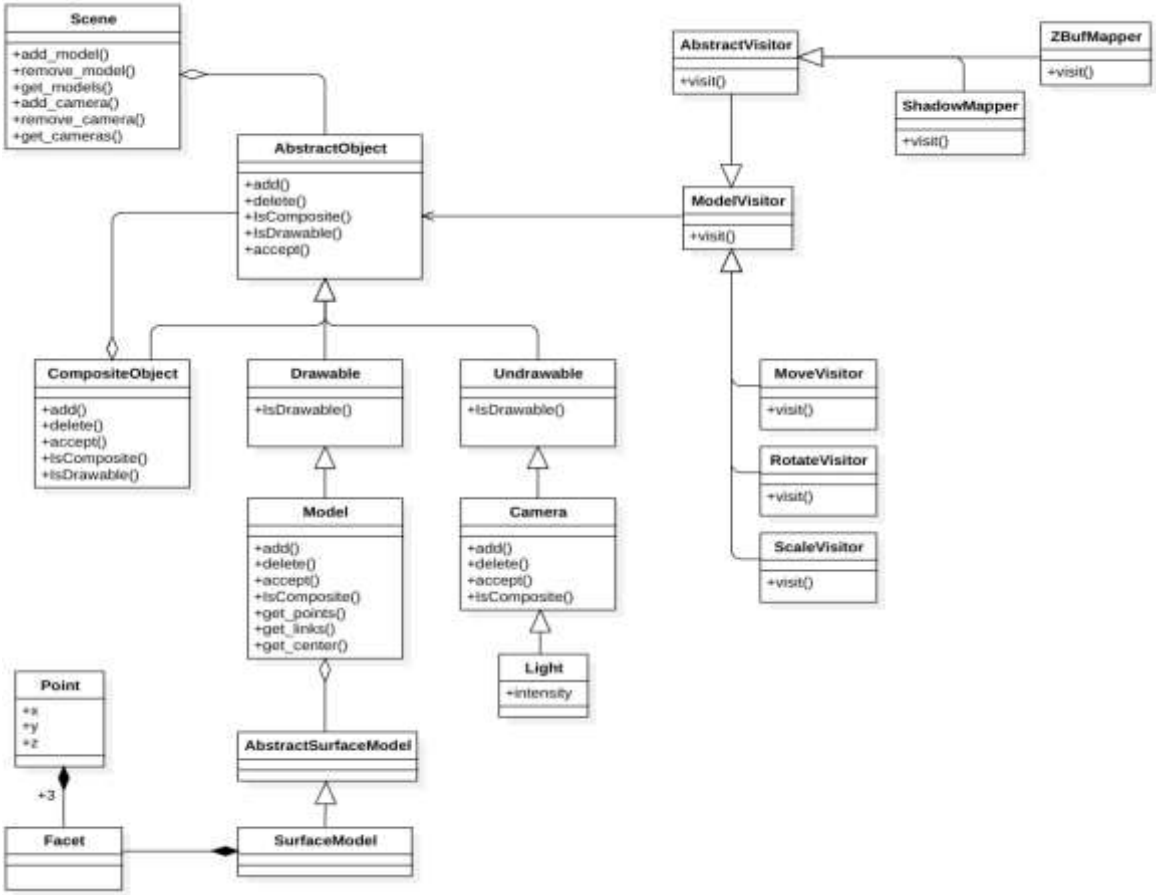
Для разработки программного обеспечения был выбран язык C++.

- обладает высокой вычислительной производительностью;
- обладает большим количеством литературы и примеров;

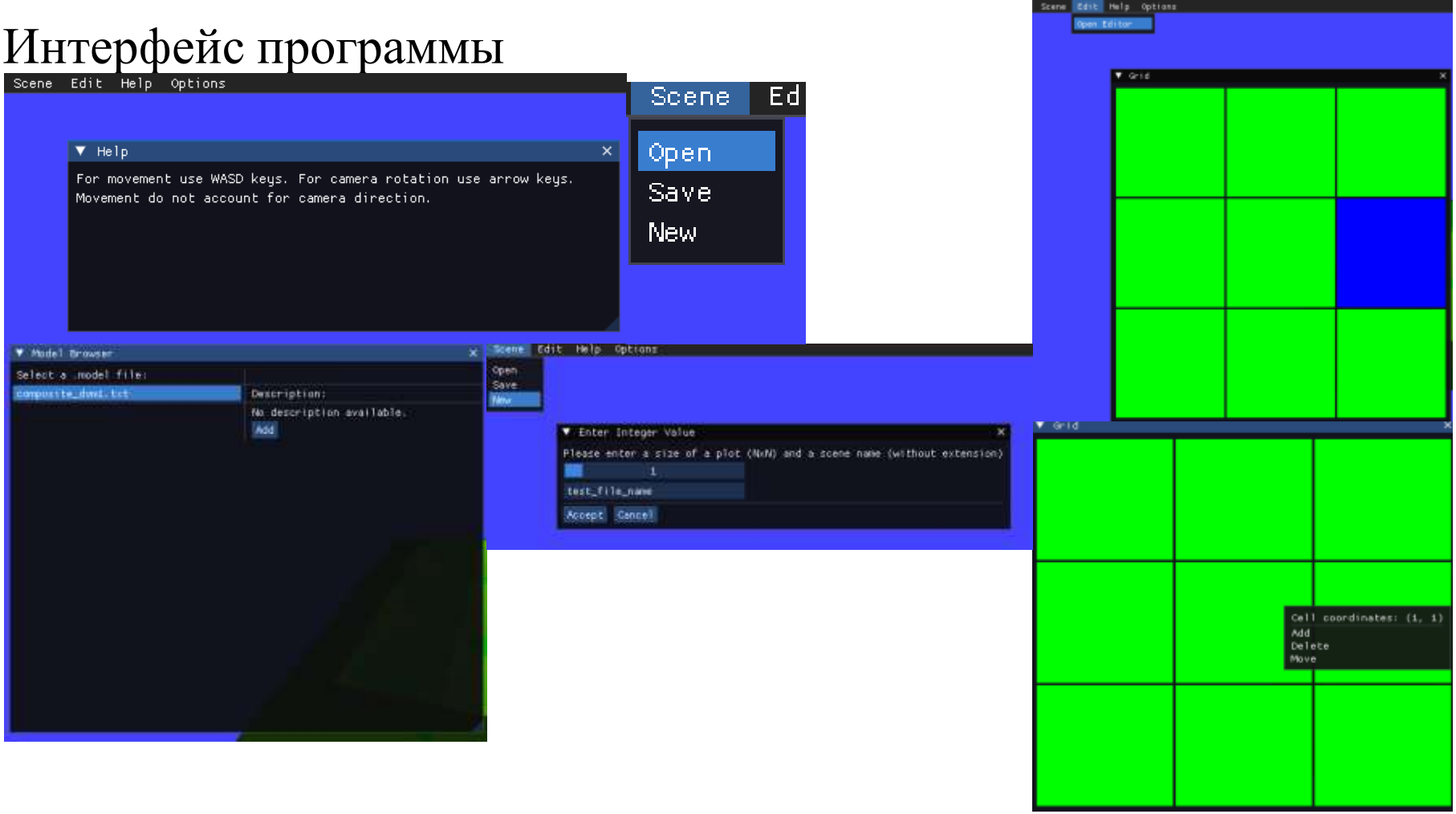
Библиотеки:

- SDL2
- Dear ImGui
- GLM

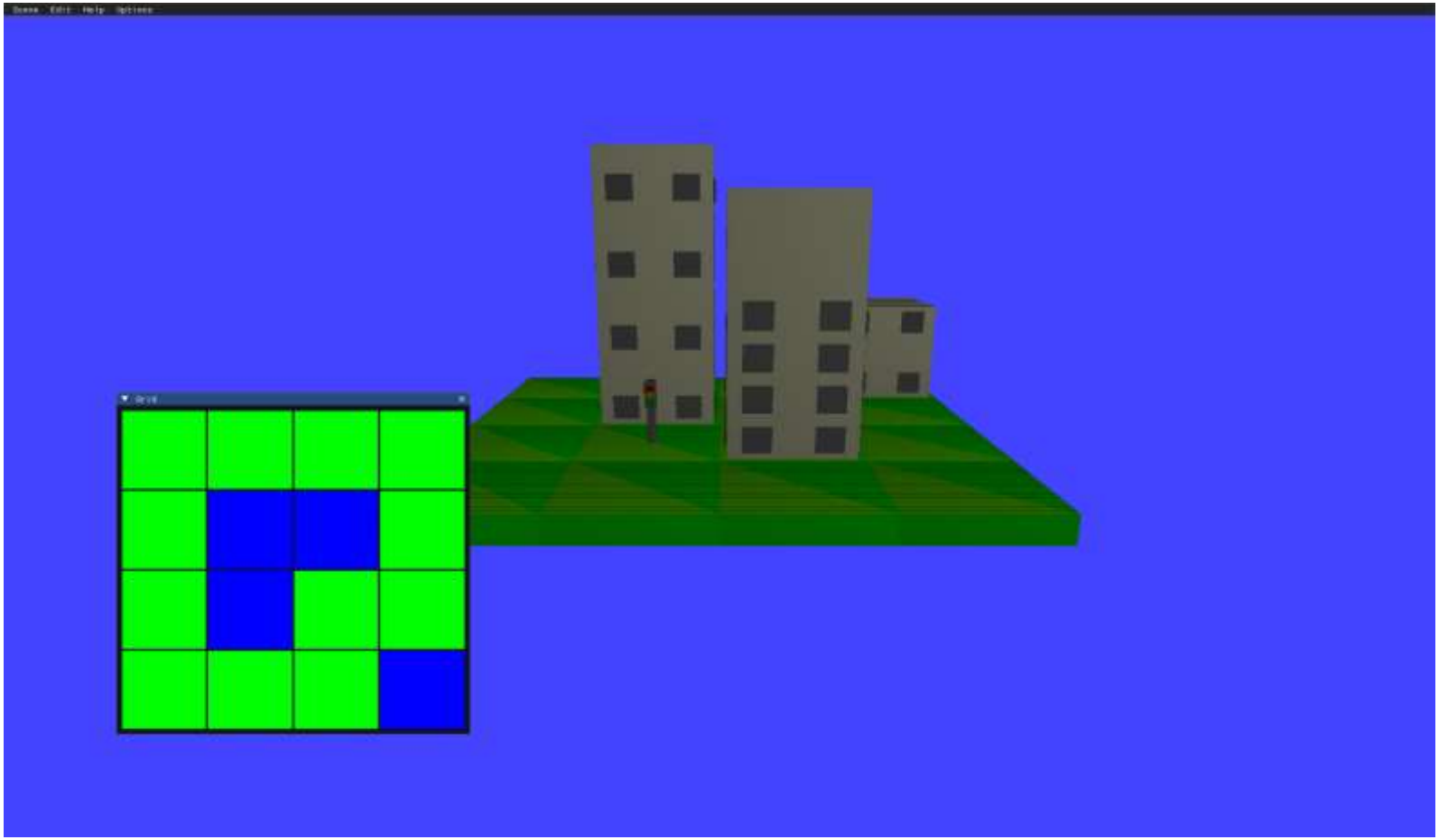
Схема основных классов программы



Интерфейс программы

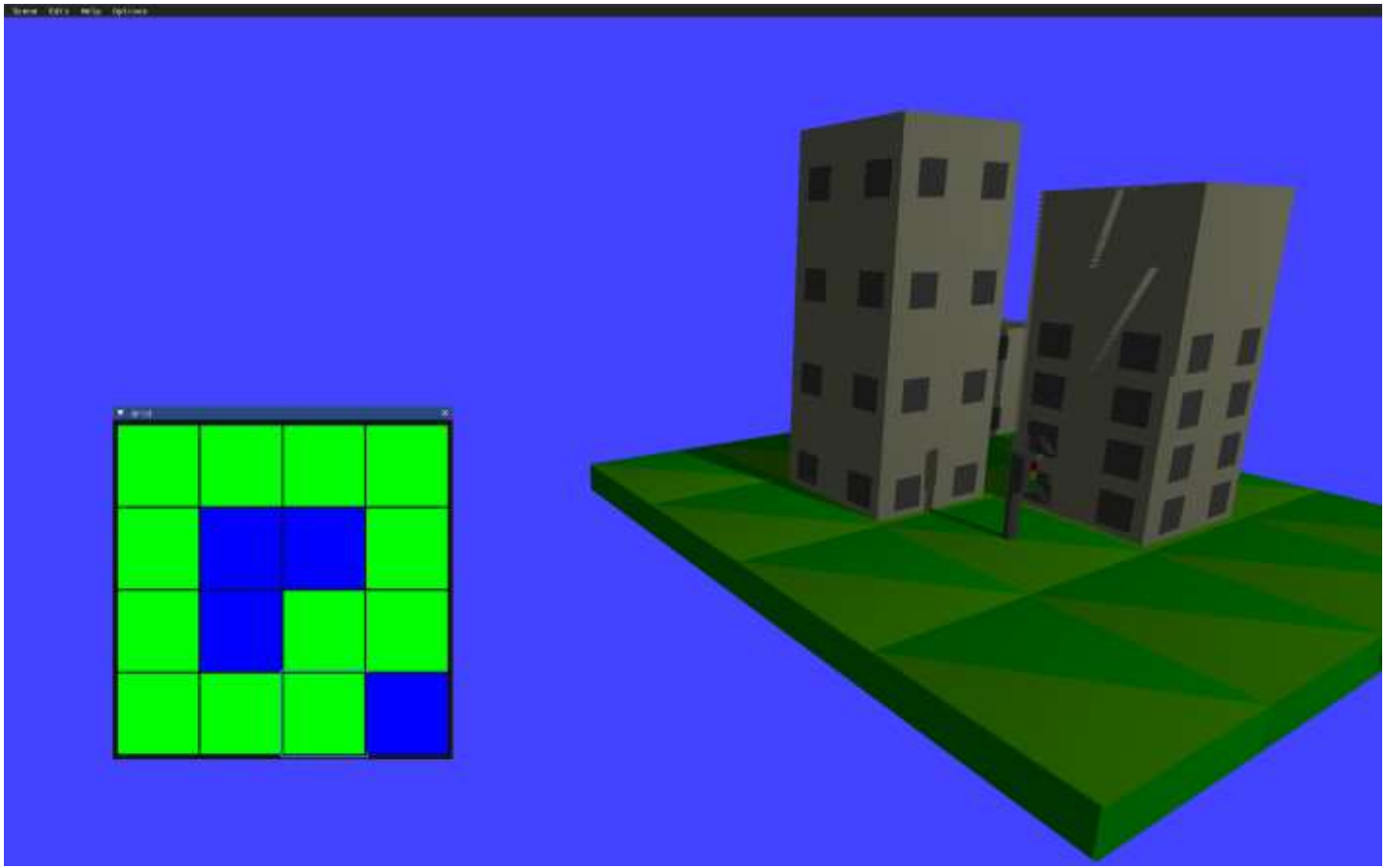


Пример работы программного обеспечения

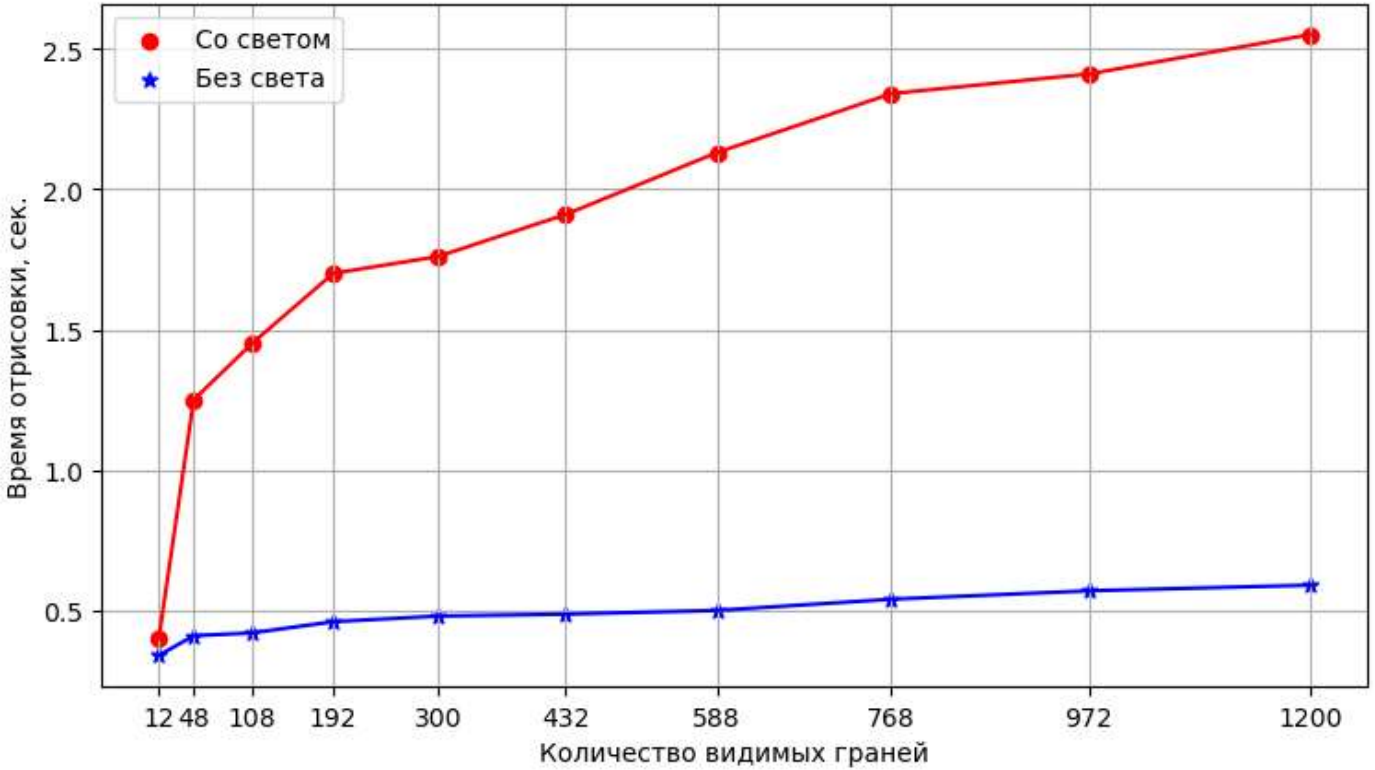


Пример работы программного обеспечения

Пример работы программного обеспечения



Зависимость времени отрисовки кадра от количества граней на сцене



Зависимость времени отрисовки кадра от количества граней на сцене

Кол-во Граней	С ист. Света, сек	Без света, сек
12	0.4	0.34
48	1.25	0.41
108	1.45	0.42
192	1.7	0.46
300	1.76	0.48
432	1.91	0.49
588	2.13	0.5
768	2.34	0.54
972	2.41	0.57
1200	2.55	0.49

ВЫВОД

Без учета освещенности алгоритм Z-буфер работает быстрее, чем с учетом освещенности. Это связано с тем, что для каждого источника света необходимо проинициализировать и заполнить теневой Z-буфер, а потом для каждого пикселя при растеризации грани выполнять преобразование координат в координаты в теневом Z-буфере для определения теней.

Заключение

В ходе выполнения курсовой работы цель курсовой работы была достигнута: было разработано программное обеспечение для моделирования сцены
улицы из разработанной библиотеки объектов с возможностью изменять положение камеры и источника света.

Для достижения цели курсовой работы были решены следующие задачи:
- выбраны алгоритмы компьютерной графики для визуализации трехмерной сцены;
- выбран язык программирования и среда разработки;
- разработано программное обеспечение и реализованы выбранные алгоритмы визуализации;
- проведены замеры временных характеристик разработанного программного обеспечения.