Содержание

Введение 6

1 Аналитическая часть 7

1.1 Описание и формализация объектов сцены 7

1.2 Выбор способа определения моделей на сетке сцены 9

1.3 Выбор способа задания поверхностных моделей 10

1.4 Выбор алгоритма удаления невидимых ребер и поверхностей . . 11

1.4.1 Алгоритм Робертса 12

1.4.2 Алгоритм, использующий z-буфер 14

1.4.3 Алгоритм обратной трассировки лучей 15

1.5 Выбор алгоритма построения теней 17

1.6 Вывод 18

2 Конструкторская часть 19

2.1 Требования к программному обеспечению 19

2.2 Общий алгоритм решения поставленной задачи 19

2.3 Алгоритм, использующий z-буфер 19

2.4 Модифицированный алгоритм, использующий z-буфер 21

2.5 Представление данных в ПО 23

2.6 Вывод 23

3 Технологическая часть 24

3.1 Выбор языка программирования и среды разработки 24

3.2 Структура реализуемых классов 25

3.3 Сведения о модулях программы 28

3.4 Интерфейс программного обеспечения 30

3.5 Вывод 35

4 Исследовательская часть 36

4.1 Результаты работы ПО 36

4.2 Постановка эксперимента 40

4.2.1 Цель эксперимета 40

4.2.2 Технические характеристики 40

4.2.3 Результаты эксперимента 40
4.3 Вывод 42

Заключение 43

Список литературы 44

Введение

В настоящие время компьютерное графическое моделированию различных предметов приобрело большую популярность, многие компании стали к нему прибегать, чтобы избавиться от недопонимания между заказчиком и исполнителем. Такой подход дает возможность как можно четче обозначить детали проекта, и реже допускать ситуации, в которых становится известно, что исполнитель неправильно воспринял идею заказчика. Мною была выбрана тема, связанная с визуализацией трехмерного городского пространства. Данная программа может быть использована при планировке городских дворов.

Для исполнителя важно исключить возникновение правок на поздних стадиях выполнения работы. Поэтому важно предоставить реалистичное изображение на этапе предварительного показа. Для построения такого изображения, которое позволит понять пользователю концепцию работы, потребуется учитывать невидимость ребер объектов сцены по отношению к наблюдателю и освещение отдельных участков рабочей плоскости.

Целью курсовой работы является реализация программного обеспечения для трехмерного планировщика городского пространства.

Для достижения поставленной цели необходимо выполнить следующие задачи.

- Описать список доступных к размещению на сцене моделей, формализовать эти модели.
- Выбрать существующие алгоритмы компьютерной графики для визуализации сцены и объектов на ней.
- Выбрать язык программирования и среду разработки.
- Реализовать выбранные алгоритмы визуализации.
- Реализовать программное обеспечения для визуализации и редактирования городского пространства.

1 Аналитическая часть

1.1 Описание и формализация объектов сцены

Сцена состоит из следующих объектов.

- **Площадка сцены** – правильный параллелепипед с заданной сеткой, по которой расставляются модели. Объекты располагаются только на одной из сторон площадки. Размеры границ задаются количеством ячеек (квадратов) по ширине и длине площадки. Размер ячеек – константная величина, определяемая внутри программы. Пользователь может вращать сцену с объектами. Цвет площадки – салатовый.

- **Объекты сцены** – модели, расположенные на площадке сцены, которые занимают определенное количество клеток сетки (ячеек). Каждая модель представляет собой набор граней, описываемых точками в пространстве, которые соединены ребрами. Все доступные модели определены заранее, в программе не предусмотрена возможность добавления новых или изменения старых моделей. В ПО доступна возможность изменения положения или удаления объектов на сцене.

  Список доступных объектов сцены.

  1. **Дом.** Состоит из трех частей: стен, крыши и окон.

     – Стена дома представляет собой плоскость, перпендикулярную плоскости сцены. Длиной и шириной стены является заданное пользователем число ячеек (квадратов) на сетке сцены. По середине каждого квадрата стены оставлено квадратное отверстие для окна дома, площадь которого равна $\frac{1}{9}$ площади ячейки сцены. Высота дома задается пользователем (кол-во ячеек сцены). Высота одного этажа равна длине ячейки (квадрата) сетки сцены. Цвет стены – оранжевый.

     – Крыша дома состоит из четырех плоскостей: двух треугольных и двух четырехугольных, имеющих формы равнобедрен-

ных трапеций. Нижнии вершины этих трапеций совпадают с вершинами углов стен верхнего этажа дома. Высота крыши дома соответствует длине ячейки сетки сцены. Цвет крыши – красный.

   – Окно дома представляют собой квадратную плоскость, перпендикулярную плоскости сцены. Площадь окна равна $\frac{1}{9}$ площади ячейки сетки сцены. Окна расставляются в оставленных при построение стен пустотах. Цвет окна – желтый.

2. **Машина.** Занимает одну клетку сетки сцены по ширине и две клетки по длине. Представляется набором плоскостей, образующими в отдельности кузов машины, ее колеса и стекла. Основой кузова является параллелепипед, расположенный на высоте $\frac{1}{6}$ длины клетки над плоскостью сцены. Длина и высота колеса машины соответствуют $\frac{1}{4}$ длины ячейки сетки сцены. Машина может быть расположена как вдоль оси X, так и вдоль оси Y. Цвет кузова машины – фиолетовый, колес – серый, окон – белый.

3. **Дорога.** Представляет собой квадратную плоскость, параллельную плоскости сцены. Занимает одну клетку по ширине и длине. Состоит из двух частей: асфальта и полосы. Ширина полосы соответствует $\frac{1}{5}$ длины ячейки сетки сцены, а длина – $\frac{3}{4}$ длины ячейки. Дорога может быть расположена как вдоль оси X, так и вдоль оси Y. Цвет асфальта – серый, полосы – белый.

4. **Дерево.** Занимает одну клетку по ширине, длине и две по высоте. Состоит из двух частей: ствола и листвы.

   – Ствол дерева представляет собой параллелепипед, высотой одна клетка, ширина и высота которого соответствуют $\frac{1}{5}$ длины ячейки сетки сцены.

   – Листва дерева состоит из набора параллелепипедов, которые вместе занимают одну клетку по ширине, длине и высоте. Цвет листвы – зеленый.

Деревья, дома, дороги разрешено расставлять только на свободных клетках сетки сцены, машины – только на дорогах. Дороги не должны прилегать к дому.

- **Источник света** – материальная точка пространства. Принимает ортогональную проекцию визуализируемой сцены из своего положения с некоторым ограниченным обзором. В зависимости от расположения источника и направления распространения лучей света, определяется тень от объектов, расположенных на сцене. Положение источника света задается относительно текущей точки наблюдения последовательными поворотами по осям X и Y.

1.2 Выбор способа определения моделей на сетке сцены

Использование моделей позволяет правильно отображать форму и размеры объектов сцены.

Модели могут задаваться в следующих формах.

- **Каркасная модель.** В данной модели задается информация о вершинах и ребрах объекта. Это одна из простейших форм задания модели. Основная проблема отображения объектов с помощью каркасной модели заключается в том, что модель не всегда однозначно передает представление о форме объекта.

- **Поверхностная модель.** Данный тип модели часто используется в компьютерной графике. Поверхность может описываться аналитически, либо задаваться другим способом. Недостатком поверхностной модели является отсутствие информации о том, с какой стороны поверхности находится материал.

- **Твердотельная модель.** Данная форма задания модели отличается от поверхностной формы тем, что в объемных моделях к информации о поверхностях добавляется информация о том, с какой стороны расположен материал. Это можно сделать путем указания направления внутренней нормали.

Для решения поставленной задачи не подойдет каркасная форма модели, так как такое представление будет приводить к неправильному восприятию

заказчиком форм объекта. Твердотельная модель также не подойдет, так как пользователю совершенно не важно, из какого материала будет выполнен тот или иной объект сцены. Поэтому выбираем поверхностную форму модели.

1.3 Выбор способа задания поверхностных моделей

Существует несколько способов задать поверхностную модель.

- **Аналитический способ.** Этот способ задания модели характеризуется описанием модели объекта, которое доступно в неявной форме, то есть для получения визуальных характеристик необходимо дополнительно вычислять некоторую функцию, которая зависит от параметра.
- **Полигональная сетка.** Данный способ характеризуется совокупностью вершин, ребер и граней, определяющих форму объекта в трехмерном пространстве.

Рассмотрим существующие способы хранения информации о сетке.

- **Список граней.** Объект – это множество граней и множество вершин. В каждую грань входят как минимум 3 вершины.
- **«Крылатое» представление.** Каждая точка ребра указывает на две вершины, две грани и четыре ребра, которые ее касаются.
- **Полуреберные сетки.** То же «крылатое» представление, но информация обхода хранится для половины грани.
- **Таблица углов.** Таблица, хранящая вершины. Обход заданной таблицы неявно задает полигоны. Такое представление более компактно и более производительно для нахождения полигонов, но, в связи с тем, что вершины присутствуют в описании нескольких углов, операции по их изменению медленны.

- **Вершинное представление.** Хранятся лишь вершины, которые указывают на другие вершины. Простота представления дает возможность проводить над сеткой множество операций.

При выборе способа задания модели в курсовом проекте решающим фактором стала скорость выполнения преобразований над объектами сцены.

Оптимальное для реализации представление - модель, заданная полигональной сеткой. Такая модель позволит избежать проблем при описании сложных объектов сцены.

Способом хранения полигональной сетки был выбран список граней, так как это даст явное описание граней. Этот способ позволит эффективно преобразовывать модели, так как структура будет включать в себя список вершин.

1.4 Выбор алгоритма удаления невидимых ребер и поверхностей

Задача удаления невидимых линий и поверхностей является одной из наиболее сложных в машинной графике. 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. Решать поставленную задачу удаления можно как в объектном пространстве (в мировой системе координат), так и в пространстве изображения (в экранных координатах).

Свойства, которыми должен обладать алгоритм, для оптимальной работы реализуемого программного обеспечения:

- алгоритм должен быть достаточно быстрым при работе с множеством объектов сцены, чтобы пользователь не ожидал долгой загрузки изображения;

- алгоритм может работать в любом пространстве (скорость важнее точности).

Рассмотрим алгоритмы для удаления невидимых ребер и поверхностей.

1.4.1 Алгоритм Робертса

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

Алгоритм выполняется в 3 этапа.

1. **Этап подготовки исходных данных.**

   На данном этапе должна быть задана информация о телах. Для каждого тела сцены должна быть сформирована матрица тела V. Размерность матрицы – $4 * n$, где n – количество граней тела.

   Каждый столбец матрицы представляет собой четыре коэффициента уравнения плоскости $ax + by + cz + d = 0$, проходящей через очередную грань.

   Таким образом, матрица тела будет представлена в следующем виде:

$$V = \begin{pmatrix} a_1 & a_2 & \dots & a_n \\ b_1 & b_2 & \dots & b_n \\ c_1 & c_2 & \dots & c_n \\ d_1 & d_2 & \dots & d_n \end{pmatrix} \tag{1.1}$$

   Матрица тела должна быть сформирована корректно, то есть любая точка, расположенная внутри тела, должна располагаться по положительную сторону от каждой грани тела. В случае, если для очередной грани условие не выполняется, соответствующий столбец матрицы надо умножить на -1. Для проведения проверки следует взять точку, расположенную внутри тела. Координаты такой точки можно получить путем усреднения координат всех вершин тела.

2. **Этап удаления ребер, экранируемых самим телом.**

   На данном этапе рассматривается вектор взгляда $E = \{0, 0, -1, 0\}$. Для определения невидимых граней достаточно умножить вектор E на матрицу тела V. Отрицательные компоненты полученного вектора будут соответствовать невидимым граням.

3. **Этап удаления невидимых ребер, экранируемых другими телами сцены.**

   На данном этапе для определения невидимых точек ребра требуется построить луч, соединяющий точку наблюдения с точкой на ребре. Точка будет невидимой, если луч на своем пути встречает в качестве преграды рассматриваемое тело. Если тело является преградой, то луч должен пройти через тело. Если луч проходит через тело, то он находится по положительную сторону от каждой грани тела.

Преимущества и недостатки алгоритма Робертса

Преимущества:

- алгоритм работает в объектном пространстве, точность вычислений высокая.

Недостатки:

- теоретический рост сложности алгоритма – квадрат числа объектов. Для решения данной проблемы достаточно воспользоваться модифицированными реализациями, например, с использованием габаритных тестов или сортировки по оси z;
- все тела сцены должны быть выпуклыми. Данная проблема также приводит к усложнению алгоритма, так как потребуется прибегнуть к проверке объектов на выпуклость и их разбиению на выпуклые многоугольники.

Вывод

Алгоритм Робертса не подходит для решения поставленной задачи по следующим причинам:

- от программного обеспечения не требуется той точности визуализации объектов, которую предоставляет алгоритм;

- на сцене может находиться множество объектов, что замедлит скорость работы алгоритма. Это не удовлетворяет преподнесенным требованиям к скорости выполнения алгоритма. Особенные проблемы также могут наблюдаться при наличии множества невыпуклых тел на сцене;

- реализация модификаций, позволяющих приблизить рост сложности алгоритма к линейной, очень труднозатратна.

1.4.2 Алгоритм, использующий z-буфер

Данный алгоритм работает в пространстве изображения. Используется два буфера:

- буфер кадра, в котором хранятся атрибуты каждого пикселя в пространстве изображения;

- z-буфер, куда помещается информация о координате z для каждого пикселя.

Первоначально в z-буфере находятся минимально возможные значения z, а в буфере кадра располагаются пиксели, описывающие фон. Каждый многоугольник преобразуется в растровую форму и записывается в буфер кадра.

В процессе подсчета глубины нового пикселя, он сравнивается с тем значением, которое уже лежит в z-буфере. Если новый пиксель расположен ближе к наблюдателю, чем предыдущий, то он заносится в буфер кадра и происходит корректировка z-буфера.

Для решения задачи вычисления глубины z каждый многоугольник описывается уравнением $ax + by + cz + d = 0$. При $c = 0$ многоугольник для наблюдателя вырождается в линию.

Для некоторой сканирующей строки $y = const$, поэтому имеется возможность рекуррентно высчитывать z' для каждого $x' = x + dx$:

$$z' - z = -\frac{ax' + d}{c} + \frac{ax + d}{c} = \frac{a(x - x')}{c} \tag{1.2}$$

Получим: $z' = z - \frac{a}{c'}$, так как $x - x' = dx = 1$.

При этом стоит отметить, что для невыпуклых многогранников предварительно потребуется удалить нелицевые грани.

Преимущества и недостатки алгоритма, использующего z-буфер

Преимущества:

- простота реализации;
- оценка вычислительной трудоемкости алгоритма линейна;
- экономия вычислительного времени, так как элементы сцены не сортируются.

Недостатки:

- большой объем требуемой памяти;
- реализация эффектов прозрачности сложна.

Вывод

Алгоритм отвечает главному требованию – скорости работы с множеством объектов. Также простота алгоритма позволит достаточно быстро реализовать данный алгоритм и, что важнее, в полной мере отладить его.

При этом следует отметить, что само изображение будет относительно малых размеров, что приведет к некритично большим затратам памяти для выполнения данного алгоритма.

1.4.3 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 z, 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 рассмотрении фигурирует перспективная проекция, то предполагается, что сам наблюдатель по-прежнему находится на положительной полуоси z, а сам растр при этом перпендикуляром оси z. Задача будет состоять в том, чтобы построить одноточечную центральную проекцию на картинную плоскость.

Определения пересечений происходит с помощью погружения объектов в некоторую выпуклую оболочку – например, сферическую. Поиск пересечения с такой оболочкой происходит проще: достаточно проверить превосходит ли радиус сферы-оболочки расстояние от центра этой сферы до луча.

Предположим, что некоторая прямая проходит через две точки $P_1(x_1, y_1, z_1)$ и $P_2(x_2, y_2, z_2)$:

$$P(t) = P_1 + (P_2 - P_1)t \tag{1.3}$$

Компоненты при этом:

$$x = x_1 + (x_2 - x_1)t = x_1 + at \tag{1.4}$$

$$y = y_1 + bt \tag{1.5}$$

$$z = z_1 + ct \tag{1.6}$$

Таким образом минимальное расстояние от этой прямой до некоторой точки $P_0(x_0, y_0, z_0)$:

$$d^2 = (x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 \tag{1.7}$$

Где параметр t, который определяет ближайшую точку $P(t)$:

$$t = -\frac{a(x_1 - x_0) + b(y_1 - y_0) + c(z_1 - z_0)}{a^2 + b^2 + c^2} \tag{1.8}$$

Если $d^2 > R^2$, где R – радиус сферической оболочки, то луч не может пересечься с объектом.

Преимущества и недостатки алгоритма обратной трассировки лучей

Преимущества:

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

Недостатки:

- производительность.

Вывод

Алгоритм не отвечает главному требованию – скорости работы. Также от реализуемого продукта не требуется высокой реалистичности синтезируемого изображения и возможности работы с поверхностями, заданными в математической форме. Указанные факты говорят о том, что обратная трассировка лучей не подходит для решения поставленной задачи.

Таким образом, в качестве алгоритма удаления невидимых ребер и поверхностей был выбран алгоритм с использованием z-буфера.

1.5 Выбор алгоритма построения теней

При использовании алгоритма обратной трассировки лучей, рассмотренного ранее, построение теней происходит по ходу выполнения алгоритма: пиксел будет затемнен, если испускаемый луч попадает на объект, но не попадает в источник света. Данный алгоритм не подходит для решения поставленной задачи, так как при проведении анализа алгоритмов удаления невидимых ребер он не был выбран в качестве реализуемого.

В качестве алгоритма удаления невидимых линий и поверхностей в предыдущем подпункте был выбран алгоритм с использованием z-буфера, поэтому одним из наилучших вариантов будет модификация указанного метода путем добавления вычисления теневого z-буфера из точки наблюдения, совпадающей с источником света.

Такой подход позволит не усложнять структуру программы, а также избежать проблем адаптации двух различных методов друг к другу, а, следовательно, уменьшить время отладки алгоритма.

1.6 Вывод

Были рассмотрены способы задания трехмерных моделей и выбрана поверхностная форма задания моделей. Также были рассмотрены алгоритмы удаления невидимых ребер:

- алгоритм, использующий z-буфер;
- алгоритм Робертса;
- алгоритм обратной трассировки лучей.

В качестве реализуемого был выбран алгоритм z-буфера, отвечающего двум требованиям: быстрая работа с множеством объектов сцены и преобладание скорости над точностью. Были рассмотрены алгоритмы построения теней. В качестве реализуемого был выбран алгоритм, использующий теневые карты, в работе которого будет использоваться z-буфер.

2 Конструкторская часть

2.1 Требования к программному обеспечению

Программа должна выполнять следующие функции:

- создание сцены с заданным размером;
- поворот, перемещение и масштабирование визуализируемой сцены;
- добавление, перемещение и удаление объектов сцены;
- добавление и удаление источников света на сцене.

2.2 Общий алгоритм решения поставленной задачи

- Задать размеры сцены, на которой будут размещаться объекты.
- Разместить объекты сцены и источники света.
- Основываясь на текущем положении наблюдателя, с помощью модифицированного алгоритма, использующего z-буфер, определить падающие от объектов сцены тени и визуализировать обстановку.

2.3 Алгоритм, использующий z-буфер

- Всем элементам буфера кадра присвоить фоновое значение.
- Инициализировать z-буфер минимальным значением глубины.
- Для каждого многоугольника сцены в произвольном порядке:
  - для каждого пикселя, который принадлежит многоугольнику вычислить его глубину $z(x, y)$;

– сравнить вычисленную глубину пикселя со значением, которое находится в z-буфере. Если $z(x, y) > z_{buf}(x, y)$, то $z_{buf}(x, y) = z(x, y)$ и $colour(x, y) = specColour$.

- Вывести итоговое изображение.

На рисунке 2.1 изображена схема алгоритма z-буфера.

Рисунок 2.1 – алгоритм z-буфера

2.4 Модифицированный алгоритм, использующий z-буфер

- Для каждого направленного источника света:
  - инициализировать теневой z-буфер минимальным значением глубины;
  - определить теневой z-буфер для источника.
- Выполнить алгоритм z-буфера для точки наблюдения. При этом, если некоторая поверхность оказалась видимой относительно текущей точки наблюдения, то проверить, видима ли данная точка со стороны источников света.

  Для каждого источника света:
  - координаты рассматриваемой точки (x, y, z) линейно преобразовать из вида наблюдателя в координаты (x', y', z') на виде из рассматриваемого источника света;
  - сравнить значение $z_{shadowBuf}(x', y')$ со значением $z'(x', y')$. Если $z'(x', y') < z_{shadowBuf}(x', y')$, то пиксел высвечивается с учетом его затемнения, иначе точка высвечивается без затемнения.

На рисунке 2.2 изображена схема модифицированного алгоритма z-буфера.

Рисунок 2.2 – модифицированный алгоритма z-буфера

2.5 Представление данных в ПО

В таблице 2.1 описано представление данных в программном обеспечении.

Таблица 2.1 – Представление данных в ПО

Данные	**Представление**
Точка трехмерного пространства	Координаты по осям x, y, z
Полигон	Три точки трехмерного пространства
Объект сцены	Список полигонов
Источник света	Угол по осям X и Y относительно текущей точки наблюдения

2.6 Вывод

На основе теоретических данных, полученных из аналитического раздела, были описаны требования к програмному обеспечению, общий алгоритм решения задачи, алгоритм z-буфера и модифицированный алгоритм z-буфера, а также было обобщено представление данных в программном обеспечении.

3 Технологическая часть

3.1 Выбор языка программирования и среды разработки

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

- C++ обладает высокой вычислительной производительностью, что очень важно для выполнения поставленной задачи.
- 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 код.
- Данный язык преподавался в рамках курса Объектно-Ориентированного Программирования.
- Доступность. Для C++ существует большое количество учебной литературы.

При написании программы будет задействована среда разработки QT Creator. Данный выбор обусловлен следующими факторами.

- Основы работы с данной средой разработки изучались в рамках курса Программирования на Си.
- QT Creator позволяет работать с расширением Qt Design, которое позволит создать удобный и надежный интерфейс для программного продукта в сжатые сроки.

3.2 Структура реализуемых классов

На рисунках 3.1 - 3.2 представлена структура реализуемых классов.

Рисунок 3.1 – Структура классов Dot3D, Vertex, Facet, PolModel, Illuminant

- **Dot3D** – класс точки трехмерного пространства. Хранит координаты в пространстве, владеет методами преобразований точки.
- **Vertex** – класс вершины. Хранит координаты точки вершины и номера граней, в которых она задействована.
- **Facet** – класс грани. Хранит номера задействованных в грани вершин.
- **PolModel** – класс полигональной модели. Хранит множество вершин и граней, образующих модель, ее имя и координаты ячейки, размеры модели и ось, вдоль которой она расположена. Владеет методами перемещения и вращения по оси Z модели.
- **Illuminant** – класс источника света. Хранит теневую карту источника, матрицу преобразований и соответствующие ей углы для перемещения текущей точки обзора в точку размещения источника. Владеет методом очистки собственной теневой карты.

Рисунок 3.2 – Структура классов CellScene, Drawer, UsageFacade, PlaceHouseChooser, PlaceTreeChooser, PlaceRoadChooser, PlaceCarChooser, MainWindow, sizeChooser, moveChooser, illuminantPlaceChooser, ObjectChanger, ObjectHangman

- **CellScene** – класс рабочей области сцены. Хранит длину и ширину сцены, модель сцены, на которой размещаются другие объекты, матрицу преобразований для перехода в текущую точку наблюдения, множество моделей интерьера и источников света, размещенных на сцене, и их кол-во, а также точку центра выполнения преобразований. Еще в этом классе хранится матрица использованных клеток сцены. Владеет методами выполнения преобразований над сценой, построения области размещения моделей, добавления и удаления объектов и источников света, отмечания и отчищения занятых клеток в матрице использованных клеток сцены.

- **UsageFacade** – класс, отвечающий за взаимодействие пользователя с программным обеспечением. Хранит формируемую сцену и класс, отвечающий за растеризацию сцены (наложение сетки, каждая из ячеек которой обрабатывается особым образом). Владеет методами добавления моделей на сцену и преобразований сцены.

- **Drawer** – класс, отвечающий за растеризацию сцены. Хранит буфер кадра и буфер глубины. Владеет методами алгоритма теневого z-буфера и формирования объекта для отображения рисунка в главном приложении.

- **PlaceHouseChooser** – класс, связанный с интерфейсом выбора расположения дома (выбора начальной точки модели, ее длины, ширины и высоты).

- **PlaceTreeChooser** – класс, связанный с интерфейсом выбора расположения дерева (выбора начальной точки модели).

- **PlaceRoadChooser** – класс, связанный с интерфейсом выбора расположения дороги (выбора начальной точки модели, а также ее ориентации в пространстве).

- **PlaceCarChooser** – класс, связанный с интерфейсом выбора расположения машины (выбора начальной точки модели, а также ее ориентации в пространстве).

- **MainWindow** – точка входа в программу.

- **SizeChooser** – класс, связанный с интерфейсов выбора размера сцены. Хранит в себе значений длины и ширины создаваемой сцены.

- **moveChooser** – класс, связанный с интерфейсом выбора нового расположения модели (выбора новой начальной точки модели).

- **IlluminantPlaceChooser** – класс, связанный с интерфейсом выбора расположения добавляемого источника света. Хранит углы поворота относительно осей X и Y для перехода из точки наблюдения в точку расположения источника света.

- **ObjectChanger** – класс, связанный с интерфейсом перемещения объектов по сцене. Хранит фасад, с которым в данный момент работает. Владеет методом изменения модели и методом пересчёта номеров моделей на сцене.

- **ObjectHangman** – класс, связанный с интерфейсом удаления объектов со сцены. Хранит сцену, с которой в данный момент работает. Владеет методом пересчёта номеров моделей на сцене.

3.3 Сведения о модулях программы

- **main.cpp** – главная точка входа в приложение;

- **mainwindow.cpp, mainwindow.h** – описание и реализация главного окна приложения;

- **mainwindow.ui** – форма пользовательского интерфейса главного окна приложения;

- **placechooser.cpp, placechooser.hpp** – описание и реализация классов выбора расположения моделей ($PlaceHouseChooser$, $PlaceTreeChooser$, $PlaceRoadChooser$, $PlaceCarChooser$);

- **placehousechooser.ui** – форма пользовательского интерфейса класс $PlaceHouseChooser$;

- **placetreechooser.ui** – форма пользовательского интерфейса класса $PlaceTreeChooser$;
- **placeroadchooser.ui** – форма пользовательского интерфейса класса $PlaceRoadChooser$;
- **placecarchooser.ui** – форма пользовательского интерфейса класса $PlaceCarChooser$;
- **sizechooser.cpp, sizechooser.hpp** – описание и реализация класса $SizeChooser$;
- **sizechooser.ui** – форма пользовательского интерфейса класса $SizeChooser$;
- **illuminantplacechooser.cpp, illuminantplacechooser.hpp** – описание и реализация класса $IlluminantPlaceChooser$;
- **illuminantplacechooser.ui** – форма пользовательского интерфейса класса $IlluminantPlaceChoooser$;
- **movechooser.cpp, movechooser.hpp** – описание и реализация класса $moveChooser$;
- **movechooser.ui** – форма пользовательского интерфейса класса $moveChooser$;
- **objectchanger.cpp, objectchanger.hpp** – описание и реализация класса $objectChanger$;
- **objectchanger.ui** – форма пользовательского интерфейса класса $objectChanger$;
- **objecthangman.cpp, objecthangman.hpp** – описание и реализация класса $objectHangman$;
- **objecthangman.ui** – форма пользовательского интерфейса класса $objectHangman$;
- **additivemathelements.cpp, additivemathelements.hpp** – описание и реализация классов математических объектов;

- **objects.cpp, objects.hpp** – описание и реализация класса полигональных моделей, источников света и сцены.
- **usagefacade.cpp, usagefacade.hpp** – описание и реализация классов $UsageFacade$ и $Drawer$;

3.4 Интерфейс программного обеспечения

На рисунке 3.3 изображен интерфейс главного окна программного обеспечения.

Рисунок 3.3 – Интерфейс главного окна ПО

Интерфейс главного окна ПО позволяет:

- создавать сцену с объектами, масштабировать ее и перемещать в центр преобразования;
- расставлять объекты на клетки сетки сцены, перемещать и удалять их, а также вращать объекты вместе с сценой.

На рисунке 3.4 изображен интерфейс окна выбора размеров новой сцены.

Рисунок 3.4 – Интерфейс окна выбора размеров новой сцены

На рисунках 3.5 - 3.8 изображены интерфейсы окон выбора размеров новых объектов.

Рисунок 3.5 – Интерфейс окна выбора распоожения дома

Рисунок 3.6 – Интерфейс окна выбора распоожения дерева

Рисунок 3.7 – Интерфейс окна выбора распоожения дороги

Рисунок 3.8 – Интерфейс окна выбора распоожения машины

Рисунок 3.9 – Интерфейс окна выбора распоожения источника света

На рисунках 3.10 - 3.11 изображены интерфейсы окон для перемещения объетов по сетке сцены.

Рисунок 3.10 – Интерфейс окна выбора перемещаемого объекта

Рисунок 3.11 – Интерфейс окна выбора нового расположения объекта

На рисунке 3.12 изображен интерфейс окна для удаления объетов со сцены.

Рисунок 3.12 – Интерфейс окна выбора удаляемого объекта

3.5 Вывод

В данном разделе были выбраны язык программирования и среда разработки, описаны поля реализуемых классов, сведения о модулях программы, а также интерфейс програмного обеспечения.

4 Исследовательская часть

В данном разделе будут приведены результаты работы программного обеспечения и проведен экмперимент с использованием библиотеки $OpenMP$.

4.1 Результаты работы ПО

На рисунке 4.1 приведен результат работы программы при наличии единственного источника света с заданными углами поворота в 30° по осям X и Y.

Рисунок 4.1 – Результат работы ПО с одним источником света

На рисунке 4.2 приведен результат работы программы при наличии двух рядом расположенных источников света с заданными углами поворота в 20°, 30° и 30°, 30° по осям X и Y. В сравнении с рисунком 4.1 тени уменьшились.

Рисунок 4.2 – Результат работы ПО с двумя источниками света

На рисунке 4.3 приведен результат работы программы при наличии единственного источника света с заданными углами поворота в 45° по осям X и Y.

Рисунок 4.3 – Результат работы ПО с измененным положением источника света

На рисунках 4.4 - 4.5 приведены результаты работы этой же программы, но с другого ракурса на сцену.

Рисунок 4.4 – Результат работы ПО с другого ракурса

Рисунок 4.5 – Результат работы ПО с другого ракурса

4.2 Постановка эксперимента

Для ускорения отрисовки сцены и объектов на ней было использовано распараллеливание циклов с помощью директив библиотеки $OpenMP$.

4.2.1 Цель эксперимета

Целью эксперимента является проведение сравнения скорости работы алгоритма z-буфера без использования и с использованием распараллеливания циклов при различных размерах сцены.

4.2.2 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование представлены далее.

- Операционная система: macOS 11.5.2.
- Память: 8 GiB.
- Процессор: 2,3 GHz 4-ядерный процессор Intel Core i5.

При тестировании ноутбук был включен в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также системой тестирования.

4.2.3 Результаты эксперимента

Результаты замеров времени работы z-буфера без использования и с использованием распараллеливания циклов при различных размерах сцены приведены в таблице 4.1. На рисунке 4.2 приведён график зависимости времени выполнения алгоритма z-буфера от размеров задаваемой сцены.

Таблица 4.1 – Результаты замеров времени работы z-буфера при различных размерах сцены

Размеры сцены	**Время выполнения без использования распараллеливания, мс**	**Время выполнения с использованием распараллеливания, мс**
1x1	25	33
10x10	152	92
20x20	203	123
30x30	227	163
40x40	251	195
50x50	296	242

Рисунок 4.6 – График зависимости времени выполнения алгоритма z-буфера от размеров задаваемой сцены

4.3 Вывод

В этом разделе были приведены результаты работы программного обеспечения и проведен экмперимент с использованием библиотеки $OpenMP$. Также были указаны характеристики машины, на которой происходило сравнение времени работы алгоритма z-буффера с использованием и без использования распараллеливания.

В результате эксперимента было установлено, что использование распараллеливания с помощью директив библиотеки $OpenMP$ помогло ускорить работу алгоритма z-буффера для всех рассмотренных размеров сцены, кроме сцены размером 1×1. На графике видно, что максимальная эффективность ускорения алгоритма z-буффера наблюдалась при построении сцен размерами 10×10 и 20×20 (в обоих случаях распараллеливание помогло ускорить работу алгоритма в 1.65 раз). Наименьшая эффективность (если не учитывать сцену размером 1×1) наблюдалась при построении сцены размером 50×50 (распараллеливание помогло ускорить работу алгоритма всего в 1.22 раза).

Заключение

Во время выполнения курсовой работы было реализовано программное обеспечение для решения задачи визуализации городского пространства.

Были рассмотрены способы задания трехмерных моделей и выбрана поверхностная форма задания моделей. Также были рассмотрены алгоритмы удаления невидимых ребер (алгоритм, использующий z-буфер, алгоритм Робертса, алгоритм обратной трассировки лучей).

В качестве реализуемого был выбран алгоритм z-буфера, отвечающего двум требованиям: быстрая работа с множеством объектов сцены и преобладание скорости над точностью. Были рассмотрены алгоритмы построения теней. В качестве реализуемого был выбран алгоритм, использующий теневые карты, в работе которого будет использоваться z-буфер.

В ходе выполнения эксперимента в исследовательской части было установлено, что реализация алгоритма z-буфера с использованием директив, предоставляемых библиотекой $OpenMP$, позволяет ускорить реализованный алгоритм в лучшем случае в 1.65 раза, что позволяет более быстро синтезировать изображение сцены.

Проделанная работа позволила лучше изучить как язык программирования C++, так и среду разработки Qt Creator.

Также в ходе выполнеия курсового проекта были выполнены следующие задачи:

- описан список доступных к размещению на сцене моделей, а сами модели были формализованы;
- выбраны существующие алгоритмы компьютерной графики для визуализации сцены и объектов на ней;
- выбран язык программирования и среда разработки;
- реализованы выбранные алгоритмы визуализации;
- реализовано программное обеспечения для визуализации и редактирования городского пространства.

Поставленная цель была достигнута.

Литература

[1] Демин А. Ю., Основы компьютерной графики [Электронный ресурс] // Томс, Томский политехнический университет. 2011. URL: https://portal.tpu.ru/SHARED/j/JBOLOTOVA/academic/ComputerGraphics/CGStudyBook.pdf.

[2] Набережнов Г. М., Максимов, Н. Н. Трехмерное моделирование полигональными сетками [Электронный ресурс] // Казань, Казанский государственный технический университет им. А.Н.Туполева. 2008. URL: https://studfile.net/preview/7335496/.

[3] Польский С. В., Компьютерная графика [Электронный ресурс] // Москва, Московский государственный университет леса. 2008. URL: https://mf.bmstu.ru/info/faculty/kf/caf/k3/subjects/Computer_graphics/materials/CG_RGR.pdf.

[4] Селянкин В. В., Алгоритмы трехмерной графики [Электронный ресурс] // Таганрог, Южный федеральный университет. 2007. URL: http://ntb.tgn.sfedu.ru/UML/UML_4112.pdf.

[5] Rogers D., Adams J., Matematical Elements for Computer Graphics [Электронный ресурс] // Аннаполис, United States Naval Academy. 1989. URL: https://itslearningakarmazyan.files.wordpress.com/2015/08/rodzhers_adams.pdf.

[6] Eck D., Introduction to Computer Graphics [Электронный ресурс] // Нью-Йорк, Hobart and William Smith Colleges. 2021. URL: https://math.hws.edu/eck/cs424/downloads/graphicsbook-linked.pdf.

[7] Programming Languages – C++ [Электронный ресурс] // 2020. URL: https://isocpp.org/files/papers/N4860.pdf.

[8] Qt Documentation [Электронный ресурс] // URL: https://doc.qt.io/qt-6/.

[9] OpenMP Specifications [Электронный ресурс] // URL: https://www.openmp.org/specifications/.