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:

«База данных для автосервиса»

Студент ИУ7-61Б _____ Мицевич М. Д.
(Группа) (Подпись, дата) (И. О. Фамилия)

Руководитель курсовой работы _____ Тассов К. Л.
(Подпись, дата) (И. О. Фамилия)

2022 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире множество людей передвигается на автомобиле, поэтому в каждом крупном городе существует хотя бы один сервис для починки автомобилей. Большинство таких мастерских не являются дилерскими и нуждаются в приложении, способном отслеживать состояние склада с запчастями и историю закупок и продаж.

Целью данной работы является разработка приложения и создание базы данных для хранения информации о состоянии склада с автозапчастями. Для достижения поставленной цели требуется выполнить следующие задачи:

- формализовать задание, определить необходимый функционал;
- провести анализ СУБД;
- спроектировать базу данных, описать ее сущности и связи;
- спроектировать приложение для доступа к БД;
- реализовать интерфейс для доступа к базе данных;
- реализовать программное обеспечение, которое позволит получить доступ к данным по средствам REST API [1].

1 Аналитический раздел

В данном разделе будет формализована задача и проведен сравнительный анализ различных СУБД.

1.1 Постановка задачи

Необходимо разработать программу для отслеживания состояния склада с автозапчастями. В частности требуется хранить информацию о том, какой пользователь положил деталь на склад и какой ее забрал.

Некоторые запчасти можно заменить на подобные им от другого производителя, поэтому у пользователя должна быть возможность поиска на складе не только самой детали, но и ее аналогов.

1.2 Формализация данных

База данных должна содержать информацию о:

- деталях для автомобилей;
- производителях автозапчастей;
- текущем состоянии склада и истории его изменения;
- пользователях системы.

Информация о детали должна содержать:

- артикул;
- название на русском и английском языках;
- возможные варианты замены;
- идентификатор производителя.

Производитель автозапчасти характеризуется страной и названием.
Пользователь характеризуется следующими параметрами:

- Имя;
- Фамилия;

- дата рождения;
- уровень доступа к системе;
- логин;
- пароль.

1.3 Типы пользователей

В системе должно быть разделение пользователей по уровню доступа к информации и возможностям ее изменения. Каждый пользователь должен авторизоваться прежде чем начать работу с приложением. Уровни доступа различных пользователей представлены на таблице 1.1

Таблица 1.1 – Уровни доступа различных пользователей

Тип пользователя	**Функционал**
Неавторизованный	Регистрация, авторизация
Клиент	Просматривать наличие запчастей на складе
Кладовщик	Добавлять запчасти на склад
Продавец	Забирать запчасти со склада
Администратор	Изменять уровни привилегий других пользователей, модифицировать информацию о запчастях, производителях и заменах, просматривать историю изменения склада.

1.4 Use-case диаграмма

Use-case диаграмма представлена на рисунке 1.1.

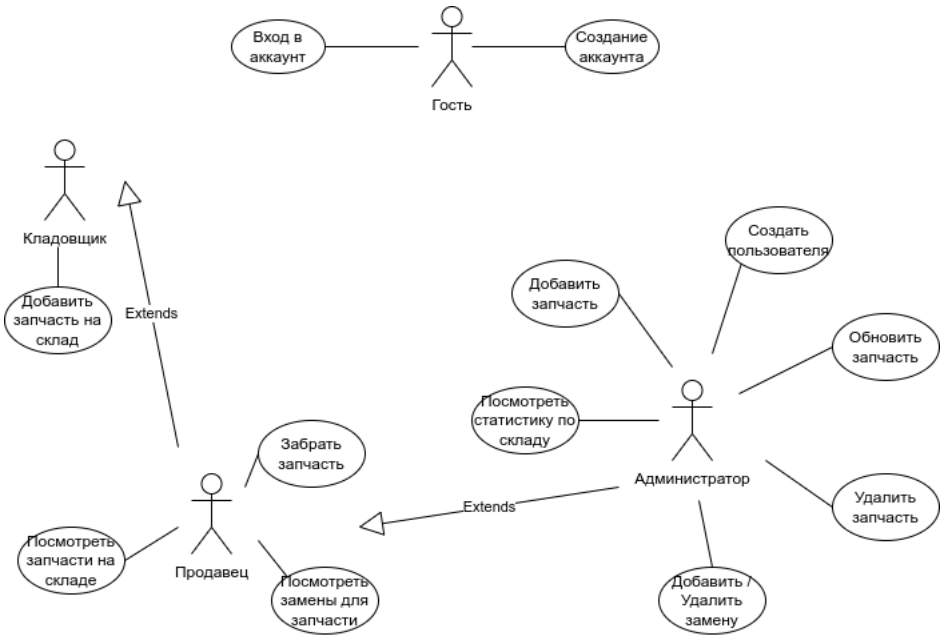

Рисунок 1.1 – Объекты находятся на одной линии

1.5 ER диаграмма

ER диаграмма представлена на рисунке 1.2

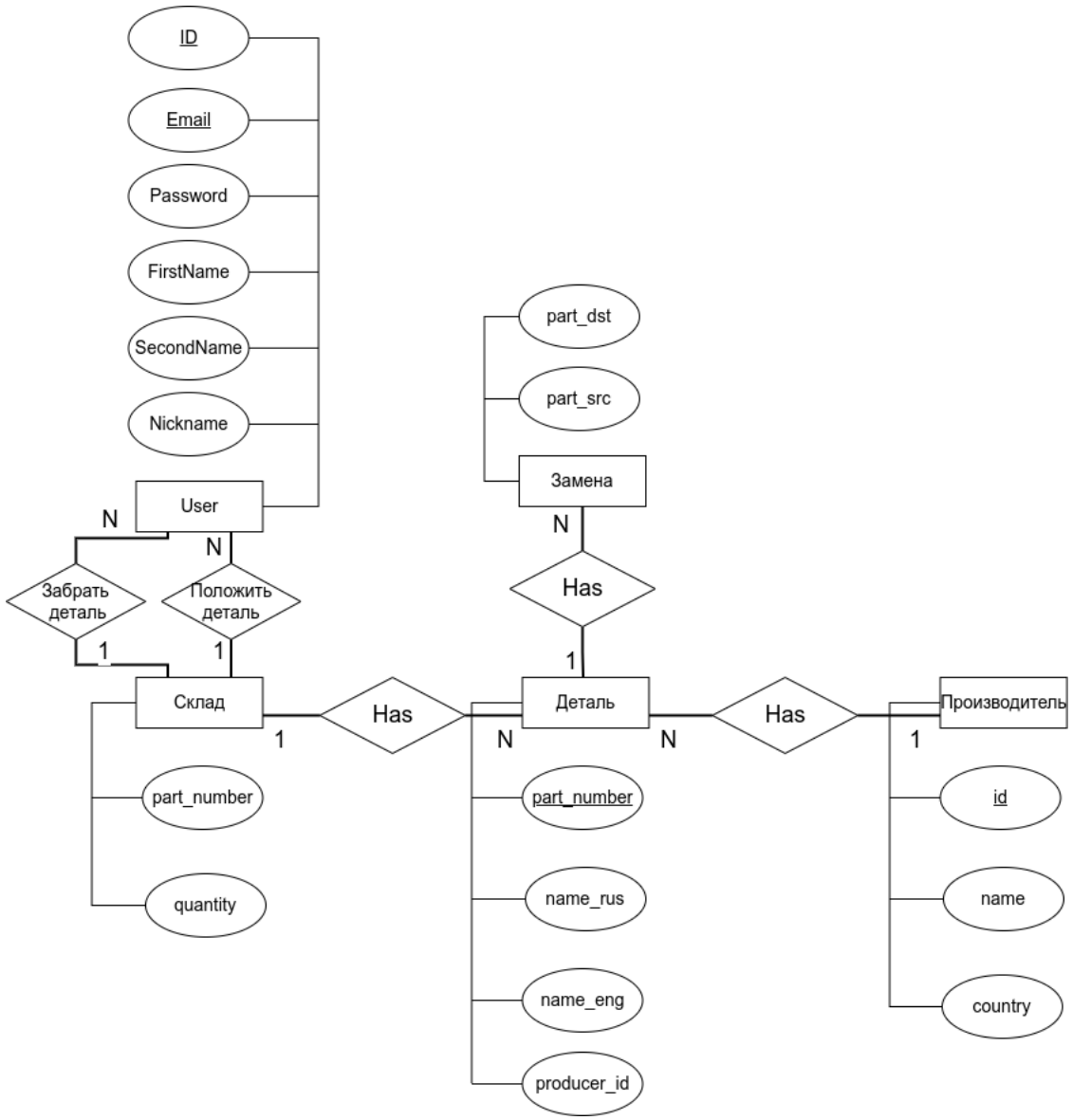

Рисунок 1.2 – Объекты находятся на одной линии

1.6 Модели хранения данных

Модели данных — вероятно, важнейшая часть разработки программного обеспе- чения в силу оказываемого ими глубочайшего воздействия не только на процесс разработки, но и на наше представление о решаемой проблеме [2]. Можно выделить три основных модели хранения данных:

- иерархическая;
- сетевая;
- реляционная.

1.6.1 Иерархическая модель

В иерархической модели данных используется представление базы данных в виде древовидной структуры, состоящей из объектов различных уровней. Между объектами существуют связи, каждый объект может включать в себя несколько объектов более низкого уровня. Такие объекты находятся в отношении предка к потомку, при этом возможна ситуация, когда объект-предок имеет несколько потомков, тогда как у объекта-потомка обязателен только один предок.

1.6.2 Сетевая модель

В древовидной структуре последней у каждой записи была ровно одна родительская запись. В сетевой же у каждой записи могло быть несколько родительских. Например, в базе может быть одна запись с названием города со ссылкой на нее для каждого живущего там пользователя. В сетевой модели появилась возможность моделировать связи «многие-к-одному» и «многие-ко-многим».

Главным недостатком сетевой модели данных являются жесткость и высокая сложность схемы базы данных, построенной на основе этой модели. Так как логика процедуры выбора данных зависит от физической организации этих данных, то эта модель не является полностью независимой от приложения. Иначе говоря, если будет необходимо изменить структуру данных, то нужно будет изменять и приложение.

1.6.3 Реляционная модель

Реляционная модель состоит из набора двумерных таблиц. При табличной организации отсутствует иерархия элементов. Таблицы состоят из строк – записей и столбцов – полей. На пересечении строк и столбцов находятся конкретные значения. Для каждого поля определяется множество его значений. За счет возможности просмотра строк и столбцов в любом порядке достигается гибкость выбора подмножества элементов.

В реляционной базе данных оптимизатор запросов автоматически принимает решение о том, в каком порядке выполнять части запроса и какие индексы использовать.

1.6.4 Выбор модели хранения данных

Все сущности системы имеют четкую структуру, которая не меняется от объекта к объекту, поэтому для хранения данных подойдет реляционная модель.

1.7 Аутентификация пользователей в системе

Аутентификацию пользователей в системе можно проводить с использованием спецификации OAuth1 [16]. Такой подход предполагает следующую последовательность:

- пользователь отправляет запрос, в котором указывает свои логин и пароль;
- приложение ищет пользователя, с указанными параметрами в базе данных и, в случает успеха, возвращает токен, с использованием которого клиент может осуществлять последующие запросы.

В таком случае необходимо где-то хранить связь токена с идентификаторам пользователя, которому он принадлежит. Для таких целей подойдет In Memory ключ-значение СУБД, которая обеспечивает быстрый поиск за счет того, что все данные хранятся в оперативной памяти. Главной проблемой таких СУБД является возможность утери данных, к примеру, при отключении электропитания, что не является критичным для хранения токенов, так как пользователю будет достаточно еще раз заполнить форму с логином и паролем для восстановления сессии.

В качестве СУБД для хранения сессий авторизованных пользователей может подойти Redis [17].

2 Конструкторский раздел

2.1 Вывод

В данном разделе были выбраны структуры данных для решения задачи, создана схема алгоритма z-буфера, а также были проведены расчеты для решения задачи построения теней.

3 Технологический раздел

В этом разделе будут рассмотрены средства разработки программного продукта и детали его реализации.

3.1 Средства реализации

Для реализации программного продукта был выбран язык программирования C++ [18]. Этот язык является компилируемым, что дает преимущество в скорости в сравнении с интерпретируемыми языками программирования. C++ поддерживает различные механизмы объектно-ориентированного программирования такие как инкапсуляция, наследование и полиморфизм. Эта парадигма сочетается с задачей отображения объектов реального мира, так как каждый объект можно задать с помощью отдельного класса с соответствующим набором полей и методов [19].

Для реализации интерфейса приложения был выбран фреймворк Qt [**qt**]. Данный выбор обусловлен тем, что он содержит все базовые классы, которые требуются для интерфейса приложения, а также позволяет выгружать содержимое сцены в файл.

Для создания модульных тестов был выбран фреймворк GoogleTests [20]. Данный выбор обусловлен тем, что он включает в себя определения основных классов и макросов, которые используются для создания модульных тестов.

Для автоматической сборки и тестирования работы приложения использовалась платформа gitlab.

3.2 Формат входных и выходных данных

На вход программа принимает 2 параметра:

- Имя конфигурационного файла, описывающего объекты сцены.
- Флаг, отвечающий за отключение пользовательского интерфейса. Если установлен флаг -no-gui, то сцена считывается из конфигурационного файла и сохраняется в файл, после чего программа завершает свою работу.

Результатом работы программы является сцена, которая отображается в окне пользовательского интерфейса или сохраняется в файл.

3.2.1 Формат файла-описателя модели

Пример файла описателя планеты представлен на листинге 3.1

Листинг 3.1 – Пример файла описателя планеты

```
10 0 0
6
1 0 0
6.123233995736766e-17 1.0 0
-1.0 1.2246467991473532e-16 0
-1.8369701987210297e-16 -1.0 0
0.0 0.0 1
0.0 0.0 -1
8
0 1 4 9 1
0 1 5 2 1
1 2 4 4 1
1 2 5 0 1
2 3 4 0 1
2 3 5 7 1
3 0 4 0 1
3 0 5 4 1
6
0 10.0 0.0 0
1.0 3.0901699437494745 4.755282581475767 0
2.0 -8.090169943749473 2.9389262614623664 0
3.0 -8.090169943749476 -2.938926261462365 0
4.0 3.0901699437494723 -4.755282581475768 0
5.0 10.0 -1.2246467991473533e-15 0
```

Первые три значения в файле - координаты центра объекта, относительно которого заданы все остальные точки. После идет число точек, из которых состоит объект и их координаты. Дальше указано число поверхностей и их свойства. Первые три числа в строке с поверхностью - индексы точек, из которых она состоит, после указывается номер цвета в таблице цветов и коэффициент диффузного отражения поверхности. После указано число узлов, по которым строится анимация объекта. Каждый узел состоит из координат и момента времени, когда центр модели находится в них.

3.2.2 Формат файла-описателя источника света

Пример файла описателя источника света представлен на листинге 3.2

Здесь первое число отвечает за интенсивность источника (от 0 до 255). Дальше идет описание геометрических свойств аналогично файлу-описателю

Листинг 3.2 – Пример файла-описателя источника света

```
255
0 0 0
5
1 0 0
0 1.0 0
-1.0 0 0
0 -1.0 0
0.0 0.0 1
6
0 1 4 4 1
1 2 4 2 1
2 3 4 3 1
3 0 4 6 1
1 2 3 1 1
0 1 3 5 1
```

планеты.

3.2.3 Формат файла-описателя камеры

Пример файла описателя камеры представлен на листинге 3.3

Листинг 3.3 – Пример файла-описателя камеры

```
0 0 0
0 0 0
```

Первые три числа - координаты задающие положение наблюдателя. На следующей строке задаются углы поворота камеры относительно осей x, y и z соответственно.

3.2.4 Формат файла описателя сцены

Пример файла описателя камеры представлен на листинге 3.4

Источник света, камера и таблица цветов создаются из конфигурационных файлов, которые расположены по базовым путям *./data/light_source.txt*, *./data/camera.txt* и *./data/color_table.txt* соответственно.

В поле *planets* идет перечисление файлов-описателей планет, из которых состоит система. Группа атрибутов *time* описывает время в системе. Если атрибут *is_video* равен *true*, то создается несколько фотографий сцены со временем в системе от *time_start* до *time_finish* с шагом *time_step* не включительно. Иначе создается только одно изображение со временем из атри-

Листинг 3.4 – Пример файла-описателя сцены

```
planets:
        - ./tests/planet1.txt
        - ./tests/planet2.txt
        - ./tests/planet3.txt

time:
        is_video: true
        time_start: 0.0
        time_finish: 1000
        time_step: 1

scaling:
        kx: 15
        ky: 15

scene_params:
        width: 1800
        height: 1000

camera_rotation:
        xangle: -20
        yangle: 0
        zangle: 0
```

бута *cur_time*. Группа атрибутов *scaling* отвечает за масштаб изображения. *kx* за масштаб по оси x, *ky* — y. Группа $scene_params$ отвечает за размеры изображения сцены. Атрибут *width* содержит значение ширины, а *height* — высоты. Группа атрибутов *camera_rotation* отвечает за углы поворота камеры. Атрибуты *xangle*, *yangle* и *zangle* содержат значения относительно осей x, y и z соответственно.

3.3 Реализация основных модулей системы

Реализация алгоритма z-буфера представлена на листинге 3.5

3.4 Тестирование системы

В рамках тестирования программы были выделены следующие случаи

- Все объекты находятся на одной линии и перекрывают друг друга
- Один объект находится в тени другого
- Повороты камеры

Листинг 3.5 – z-буфер

```
void draw_manager::draw_surface(const surface &_surface) {
  surface surf_copy = _surface;
  bool shadow = is_in_shadow(surf_copy);

  sort_points(surf_copy);

  point p1 = cur_proected_points[surf_copy.get_p1()];
  point p2 = cur_proected_points[surf_copy.get_p2()];
  point p3 = cur_proected_points[surf_copy.get_p3()];

  if (p1.get_y() == p2.get_y() && p1.get_y() == p3.get_y())
    return;

  point left_move_1, right_move_1, left_move_2, right_move_2;
  color left_color_1, right_color_1, left_color_2, right_color_2;
  find_steps(left_move_1, right_move_1, left_move_2, right_move_2
     , surf_copy);

  if (!is_light_source && !shadow)
    find_color_steps(left_color_1, right_color_1, left_color_2,
      right_color_2,
                     surf_copy);

  point leftp = p1;
  point rightp = p1;

  color leftc, rightc;

  leftc = surf_copy.get_color();
  if (!is_light_source) {
    double intensity;
    if (!shadow)
      intensity =
          find_surface_intensity(surf_copy, cur_points[surf_copy.
             get_p1()]);
    else
      intensity = SHADOW_INTENSITY;
    leftc = leftc * (intensity / 255);
  }
  rightc = leftc;
  draw_surface_part(leftp, rightp, leftc, rightc, (int)p2.get_y()
     , left_move_1,
                    left_color_1, right_move_1, right_color_1);

  draw_surface_part(leftp, rightp, leftc, rightc, (int)p3.get_y()
     , left_move_2,
                    left_color_2, right_move_2, right_color_2);
}
```

- Изменение масштаба

  Результаты тестов приведены на рисунках 3.1 - 3.5

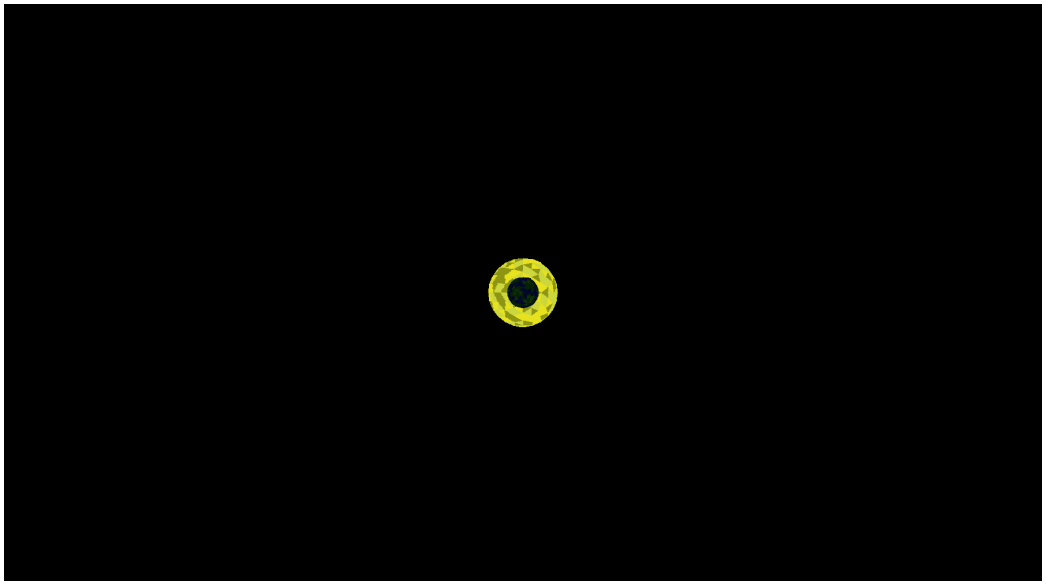

Рисунок 3.1 – Объекты находятся на одной линии

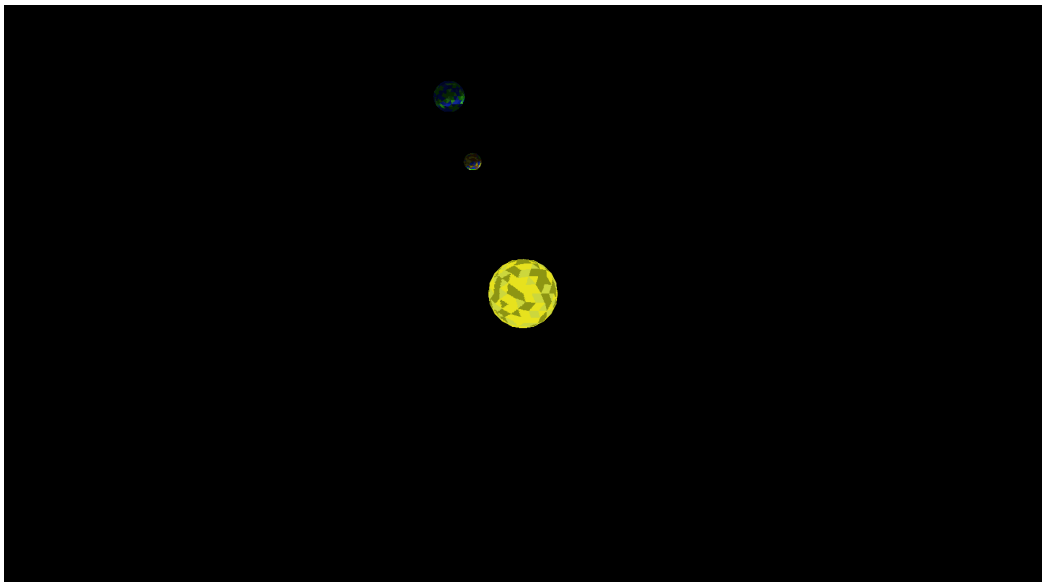

Рисунок 3.2 – Один объект находится в тени другого

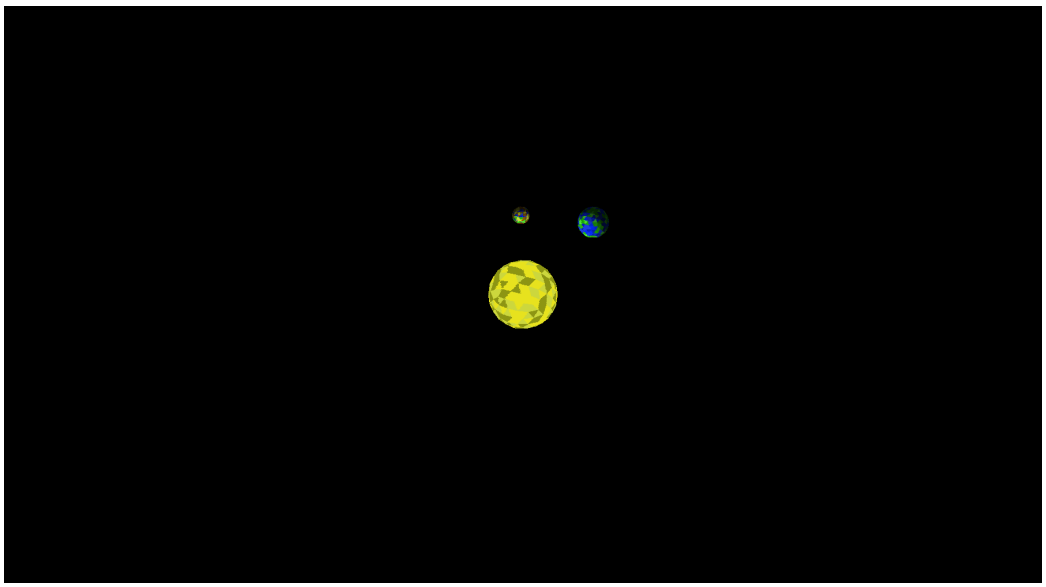

Рисунок 3.3 – Поворот вокруг оси x

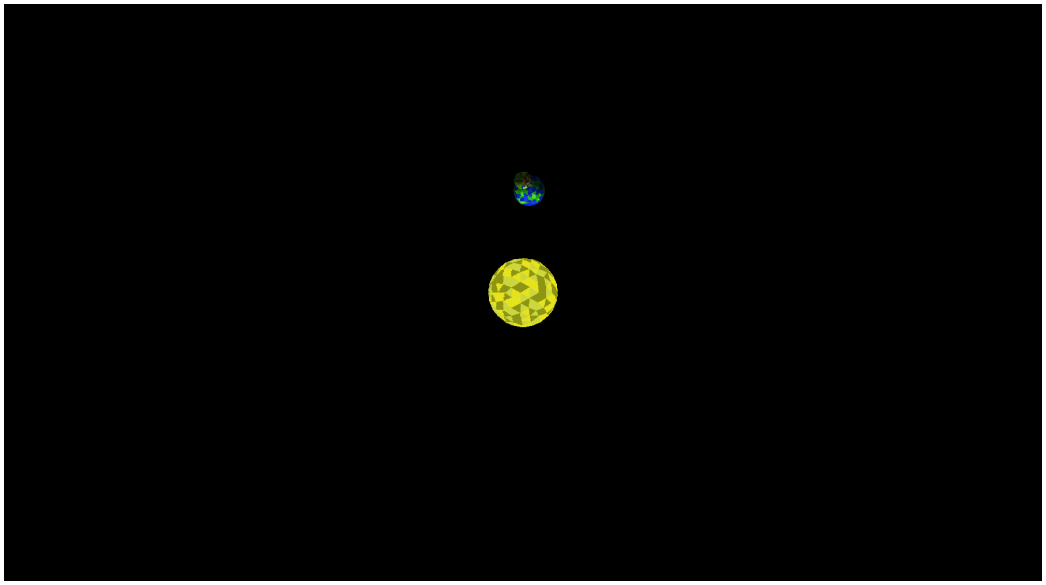

Рисунок 3.4 – Поворот вокруг оси y

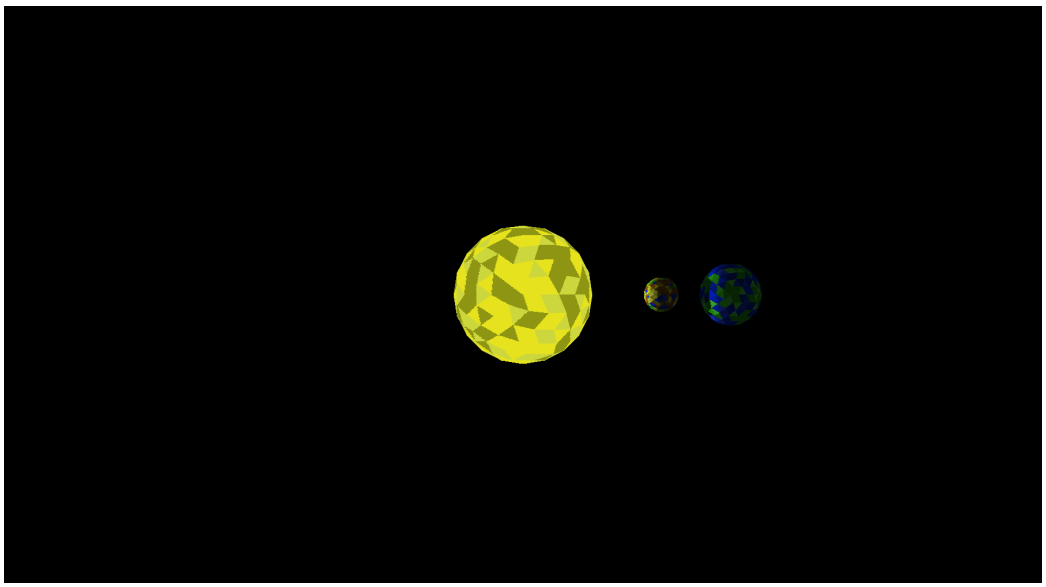

Рисунок 3.5 – Изменение масштаба

Все тесты успешно пройдены.

3.5 Вывод

В данном разделе были выбраны средства реализации программного продукта, определены форматы входных и выходных данных, а также был реализован и протестирован алгоритм z-буфера.

4 Исследовательский раздел

В данном разделе будет проведено сравнение временных характеристик обычного алгоритма z-буфера и его распаралелленной версии.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование и исследование, приведены ниже.

- Операционная система: Ubuntu Linux 64-bit.
- Оперативная память: 16 GB.
- Количество логических ядер - 8.
- Процессор: Intel(R) Core(TM) i7-8850H CPU @ 2.60GHz [**intel**].

Тестирование проводилось на компьютере, включенном в сеть электропитания. Во время тестирования компьютер был нагружен только встроенными приложениями окружения рабочего стола, окружением рабочего стола, а также непосредственно системой тестирования. Во время тестирования оптимизации компилятора были отключены.

4.2 Замеры времени

В связи с работой алгоритма в нескольких потоках, измерялось реальное время работы, а не процессорное. Время замерялось с помощью функции now [21].

Результаты замеров времени (в мс) приведены в таблицах 4.1 - 4.2. На рисунке 4.1 приведены зависимости времени работы алгоритмов от количества поверхностей модели.

Таблица 4.1 – Замер времени работы стандартного алгоритма для разного числа поверхностей

Число поверхностей	**Время, мс**
	Стандартный алгоритм
250	105
500	121
750	143
1000	166
1250	213

Таблица 4.2 – Замер времени работы распараллеленного алгоритма для разного числа поверхностей

Число рёбер	**Время в мс на n потоков**					
	1	**2**	**4**	**8**	**16**	**32**
250	100	34	19	17	21	21
500	115	43	25	23	25	25
750	140	54	30	29	29	29
1000	170	60	34	34	34	34
1250	210	72	39	37	36	36

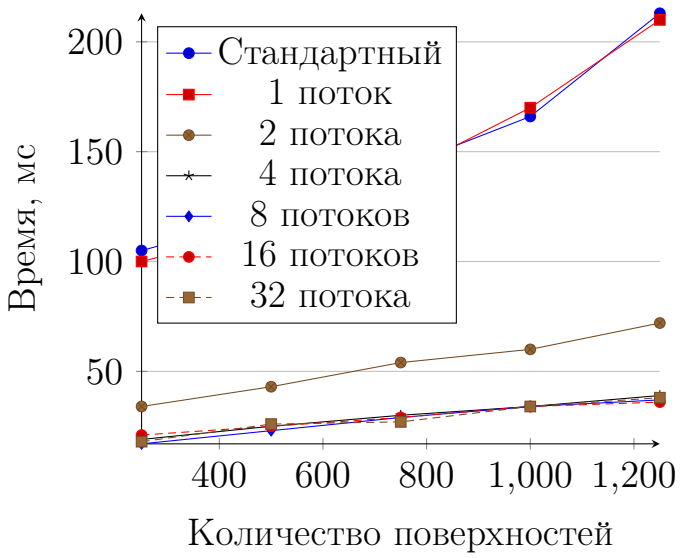

Рисунок 4.1 – Зависимость времени работы алгоритма z буфера от количества поверхностей.

Вывод

Из графиков отношения количества граней ко времени работы видно, что расчёт на одном потоке работает примерно столько же времени, как и стандартный алгоритм (разница около 1%). При использовании двух и более потоков, время работы алгоритма уменьшается в разы. В случае, когда количество потоков равно числу логических ядер процессора, алгоритм работает в 2 (при размерности 250) и в 3 (при 1250 поверхностях) раза быстрее. При использовании количества потоков, большего, чем число логических ядер процессора, время работы изменяется в пределах 2-3%, относительно предыдущего рассмотренного случая. Максимальной эффективности от параллельных вычислений добиться не удалось, так как между потоками возникает конфликт по данным, для решения которого используется мьютес.

ЗАКЛЮЧЕНИЕ

В рамках курсового проекта были решены следующие задачи:

- изучены различные подходы к построению реалистичных сцен
- выбран наиболее подходящий под условие задачи алгоритм
- создана схема выбранного алгоритма
- определены структуры данных
- реализован алгоритм
- проведено тестирование
- создана версию алгоритма, которая может выполняться с использованием нескольких потоков одноверменно
- сравнены временные показатели обычной и распараллеленной версий

Была создана программа, способная создавать трехмерную модель планетарной системы, а также моделировать ее движение. Были реализованы автосборка и автотестирование на платформе gitlab.

В результате исследование было выяснено, что версия алгоритма с использованием параллельных вычислений может дать выигрыш во времени в 2 или 3 раза в зависимости от числа поверхностей.

Поставленная цель достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. What is a REST API? - Red Hat [Электронный ресурс]. — Режим доступа: https://www.redhat.com/en/topics/api/what-is-a-rest-api (дата обращения: 07.05.2022).
2. *Kleppmann M.* Designing Data-Intensive Applications //. — Edition. Sebastopol: O'Reilly Media, 2017. — с. 640.
3. PostgreSQL: Документация. [Электронный ресурс]. — Режим доступа: https://postgrespro.ru/docs/postgresql/ (дата обращения: 07.05.2022).
4. PostgreSQL: вчера, сегодня, завтра [Электронный ресурс]. — Режим доступа: https://postgrespro.ru/blog/media/17768 (дата обращения: 07.05.2022).
5. Транзакции, ACID, CAP | GeekBrains [Электронный ресурс]. — Режим доступа: https://gb.ru/posts/acid_cap_transactions (дата обращения: 07.05.2022).
6. Documentation: 12: 13.1. Introduction - PostgreSQL [Электронный ресурс]. — Режим доступа: https://www.postgresql.org/docs/12/mvcc-intro.html (дата обращения: 07.05.2022).
7. Применение блокировок чтения/записи | IBM [Электронный ресурс]. — Режим доступа: https://www.ibm.com/docs/ru/aix/7.2?topic=programming-using-readwrite-locks (дата обращения: 07.05.2022).
8. SQL Language | Oracle[Электронный ресурс]. — Режим доступа: https://www.oracle.com/database/technologies/appdev/sql.html (дата обращения: 07.05.2022).
9. Oracle | Integrated Cloud Applications and Platform Services [Электронный ресурс]. — Режим доступа: https://www.oracle.com/index.html (дата обращения: 07.05.2022).
10. DB-Engines Ranking [Электронный ресурс]. — Режим доступа: https://db-engines.com/en/ranking (дата обращения: 07.05.2022).

11. MySQL Database Service is a fully managed database service to deploy cloud-native applications. [Электронный ресурс]. — Режим доступа: https://www.mysql.com/ (дата обращения: 07.05.2022).

12. MySQL Reference Manual 8.0: The InnoDB Storage Enginee [Электронный ресурс]. — Режим доступа: https://dev.mysql.com/doc/refman/8.0/en/innodb-storage-engine.html (дата обращения: 07.05.2022).

13. MySQL Reference Manual 16.2: The MyISAM Storage Engine [Электронный ресурс]. — Режим доступа: https://dev.mysql.com/doc/refman/8.0/en/myisam-storage-engine.html (дата обращения: 07.05.2022).

14. PHP: Hypertext Preprocessor [Электронный ресурс]. — Режим доступа: https://www.php.net/ (дата обращения: 07.05.2022).

15. The Perl Programming Language [Электронный ресурс]. — Режим доступа: https://www.perl.org/ (дата обращения: 07.05.2022).

16. OAuth Core 1.0 [Электронный ресурс]. — Режим доступа: https://oauth.net/core/1.0/ (дата обращения: 07.05.2022).

17. Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache, and message broker [Электронный ресурс]. — Режим доступа: https://redis.io/ (дата обращения: 07.05.2022).

18. Документация по Microsoft C++, C и ассемблеру [Электронный ресурс]. — Режим доступа: https://docs.microsoft.com/ru-ru/cpp/ (дата обращения: 07.05.2022).

19. *Гради Буч Роберт А. Максимчук М. У. Э.* Объектно-ориентированный анализ и проектирование с примерами приложений //. — Издательский дом Вильямс, 2008.

20. Документация по фреймворку GoogleTests [Электронный ресурс]. — Режим доступа: https://google.github.io/googletest/ (дата обращения: 07.05.2022).

21. Документация по chrono [Электронный ресурс]. — Режим доступа: https://www.cplusplus.com/reference/chrono/system_clock/now/ (дата обращения: 07.05.2022).