*Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования*

***«Московский государственный технический университет имени Н.Э. Баумана»***

***(МГТУ им. Н.Э. Баумана)***

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии

**РА С Ч Ё Т Н О - П О Я С Н И Т Е Л Ь Н А Я З А П И С К А**

**к курсовому проекту на тему:**

«База данных охранной системы с функцией распознавания голоса»

Студент

Монахов Д.И.

(Подпись) (И.О.Фамилия)

Руководитель курсового проекта

Деон А .Ф.

(Подпись, дата) (И.О.Фамилия)

Москва, 2019

**Оглавление**

[Введение 3](#_Toc533174316)

[1. Аналитический раздел 4](#_Toc533174317)

[1.1. Описание объектов сцены 4](#_Toc533174318)

[1.2. Сравнение алгоритмов построения трехмерного изображения 5](#_Toc533174319)

[1.3. Затенения 8](#_Toc533174320)

[1.4. Удаление невидимых поверхностей 9](#_Toc533174321)

[1.5. Задание поведения объектов 10](#_Toc533174322)

[2. Конструкторский раздел 10](#_Toc533174323)

[2.1. Структура сцены и объектов 11](#_Toc533174324)

[2.2. Симуляция поведения объектов 12](#_Toc533174325)

[2.3. Структуры данных 13](#_Toc533174326)

[2.4. Интерпретация пользовательского кода 14](#_Toc533174327)

[2.5. Процесс визуализации 14](#_Toc533174328)

[3. Технологический раздел 17](#_Toc533174329)

[3.1. Выбор и обоснование ЯП 17](#_Toc533174330)

[3.2. Интерфейс пользователя 18](#_Toc533174331)

[3.3. Входные и выходные данные 20](#_Toc533174332)

[3.4. Хранение и передача данных в системе 20](#_Toc533174333)

[3.5. Требования к аппаратуре 21](#_Toc533174334)

[3.6. Требования к программному обеспечению 21](#_Toc533174335)

[3.7. Порядок работы 21](#_Toc533174336)

[3.8. Сообщения системы 21](#_Toc533174337)

[3.9. Обзор результатов 21](#_Toc533174338)

[4. Экспериментально-исследовательский раздел 23](#_Toc533174339)

[Заключение 26](#_Toc533174340)

[Список литературы 27](#_Toc533174341)

# Введение

В наши дни проблема охраны частной собственности стоит особенно остро. В условиях быстро развивающейся экономики наблюдается тенденция к увеличению степени социального расслоения общества. Это приводит к накоплению ресурсов в руках небольшой группы людей. В то же время бедные слои населения продолжают беднеть, что приводит к социальной и финансовой незащищенности. Очевидным следствием этих процессов является рост преступности, в том числе насильственных преступлений, краж и ограблений. Проблема заключается в необходимости охранять частную собственность. Отличным решением служат автоматизированные охранные системы, которые позволяют управлять охраняемой территорией, собирать данные с датчиков, идентифицировать гостей и хозяев.

**Задачи:**

* решение проблемы автоматизированного управления данными охранной системы
* создание упрощенных требования к навыкам компьютерной грамотности пользователей
* демонстрация разработанной системы управления

**Цели:**

Разработка программного продукта, позволяющего хранить и обрабатывать данные охранной системы, анализ эффективности, ресурсоёмкости и простоты для конечного пользователя.

# Аналитический раздел

В данном разделе будут рассмотрены существующие системы организации данных.

* 1. **Базы данных**

В настоящее время объемы данных возрастают экспоненциально. С каждым годом, согласно эмпирическому закону Мура, количество транзисторов в процессоре увеличивается вдвое. Очевидно, что это связано в первую очередь с потребностью увеличения производительности компьютеров, т.к. существующих характеристик слишком быстро становится недостаточно.

Для организации больших объемов данных необходимы соответствующие системы, которые позволят упорядочивать, систематизировать и управлять данными. В связи с этим были разработаны так называемые базы данных.

Изначально база данных – это представленная в объективной форме совокупность самостоятельных материалов (статей, нормативных актов, судебных решений и иных подобных материалов), систематизированных таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью электронной вычислительной машины.

Система управления базой данных (СУБД) – это программное обеспечение, которое взаимодействует с конечным пользователем, приложениями и с самой базой данных, чтобы получать и анализировать данные. Система управления базой данных дополнительно предоставляет основные возможности, необходимые для администрирования базы данных.

В компьютерной науке системы управления базами данных могут классифицироваться в зависимости от поддерживаемых моделей данных. В 1980-х доминирующее положение захватили реляционные базы данных. В них данные структурируются по строкам и столбцам в группах таблиц. В 2000-х годах набрали популярности не-реляционные базы данных.

Также встречаются иные определения понятия «база данных».

Определения из международных стандартов и национальных стандартов, разработанных на основе международных:

База данных — совокупность данных, хранимых в соответствии со схемой данных, манипулирование которыми выполняют в соответствии с правилами средств моделирования данных.

База данных — совокупность данных, организованных в соответствии с концептуальной структурой, описывающей характеристики этих данных и взаимоотношения между ними, которая поддерживает одну или более областей применения.

Определения из авторитетных монографий:

База данных — организованная в соответствии с определёнными правилами и поддерживаемая в памяти компьютера совокупность данных, характеризующая актуальное состояние некоторой предметной области и используемая для удовлетворения информационных потребностей пользователей.

База данных — некоторый набор перманентных (постоянно хранимых) данных, используемых прикладными программными системами какого-либо предприятия.

База данных — совместно используемый набор логически связанных данных (и описание этих данных), предназначенный для удовлетворения информационных потребностей организации.

В определениях наиболее часто (явно или неявно) присутствуют следующие отличительные признаки:

БД хранится и обрабатывается в вычислительной системе.

Таким образом, любые некомпьютерные хранилища информации (архивы, библиотеки, картотеки и т. п.) базами данных не являются.

Данные в БД логически структурированы (систематизированы) с целью обеспечения возможности их эффективного поиска и обработки в вычислительной системе.

Структурированность подразумевает явное выделение составных частей (элементов), связей между ними, а также типизацию элементов и связей, при которой с типом элемента (связи) соотносится определённая семантика и допустимые операции.

БД включает схему, или метаданные, описывающие логическую структуру БД в формальном виде (в соответствии с некоторой метамоделью).

В соответствии с ГОСТ Р ИСО МЭК ТО 10032-2007, «постоянные данные в среде базы данных включают в себя схему и базу данных. Схема включает в себя описания содержания, структуры и ограничений целостности, используемые для создания и поддержки базы данных. База данных включает в себя набор постоянных данных, определённых с помощью схемы. Система управления данными использует определения данных в схеме для обеспечения доступа и управления доступом к данным в базе данных».

Из перечисленных признаков только первый является строгим, а другие допускают различные трактовки и различные степени оценки. Можно лишь установить некоторую степень соответствия требованиям к БД.

В такой ситуации не последнюю роль играет общепринятая практика. В соответствии с ней, например, не называют базами данных файловые архивы, Интернет-порталы или электронные таблицы, несмотря на то, что они в некоторой степени обладают признаками БД. Принято считать, что эта степень в большинстве случаев недостаточна (хотя могут быть исключения).

* 1. **Реляционная модель данных**

Реляционная модель - совокупность данных, состоящая из набора двумерных таблиц. В теории множеств таблице соответствует термин отношение (relation), физическим представлением которого является таблица, отсюда и название модели – реляционная. Соответственно теория построения баз данных, которая является приложением к задачам обработки данных таких разделов математики, как теория множеств и логика первого порядка. В сравнении с иерархической и сетевой моделью данных, реляционная модель отличается более высоким уровнем абстракции данных. Реляционная модель является удобной и наиболее привычной формой представления данных, так в настоящее время эта модель является фактическим стандартом, на который ориентируются практически все современные коммерческие СУБД. На реляционной модели данных строятся реляционные базы данных.

Впервые принципы реляционной модели были сформулированы в 1969—1970 годах Э. Ф. Коддом (E. F. Codd). Идеи Кодда были впервые публично изложены в статье «A Relational Model of Data for Large Shared Data Banks». Современную трактовку идей реляционной модели данных можно найти в книге К. Дж. Дейта. «C. J. Date. An Introduction to Database Systems»

**Состав частей реляционной модели данных**

Наиболее распространенная трактовка реляционной модели данных, принадлежит Дейту, который воспроизводит ее (с различными уточнениями) практически во всех своих книгах. Согласно Дейту реляционная модель состоит из трех частей, описывающих разные аспекты реляционного подхода: структурной части, манипуляционной части и целостной части.

**Структурная часть**

Структурная часть (аспект), отвечает за принцип построения структуры реляционной базы данных на нормализированном наборе n-арных отношений, в форме таблиц. Важно что реляционная база данных, структурно может представляться только в виде отношений.

**Манипуляционная часть**

В манипуляционной части модели утверждаются операторы манипулирования отношениями - реляционная алгебра и реляционное исчисление. Первый механизм базируется в основном на классической теории множеств (с некоторыми уточнениями), а второй - на классическом логическом аппарате исчисления предикатов первого порядка. Основной функцией манипуляционной части реляционной модели является обеспечение меры реляционности любого конкретного языка реляционных БД: язык называется реляционным, если он обладает не меньшей выразительностью и мощностью, чем реляционная алгебра или реляционное исчисление.

**Целостная часть**

В целостной части реляционной модели данных фиксируются два базовых требования целостности, которые должны поддерживаться в любой реляционной СУБД. Первое требование называется требованием целостности сущностей. Объекту или сущности реального мира в реляционных БД соответствуют кортежи отношений. Конкретно требование состоит в том, что любой кортеж любого отношения отличим от любого другого кортежа этого отношения, т.е. другими словами, любое отношение должно обладать первичным ключом. Как мы видели в предыдущем разделе, это требование автоматически удовлетворяется, если в системе не нарушаются базовые свойства отношений.

Второе требование называется требованием целостности по ссылкам и является несколько более сложным. Очевидно, что при соблюдении нормализованности отношений сложные сущности реального мира представляются в реляционной БД в виде нескольких кортежей нескольких отношений. Требование целостности по ссылкам, или требование внешнего ключа состоит в том, что для каждого значения внешнего ключа, появляющегося в ссылающемся отношении, в отношении, на которое ведет ссылка, должен найтись кортеж с таким же значением первичного ключа, либо значение внешнего ключа должно быть неопределенным (т.е. ни на что не указывать).

**Структура реляционной модели данных**

При табличной организации данных отсутствует иерархия элементов. Строки и столбцы могут быть просмотрены в любом порядке, поэтому высока гибкость выбора любого подмножества элементов в строках и столбцах. Любая таблица в реляционной базе состоит из строк, которые называют записями, и столбцов, которые называют полями. На пересечении строк и столбцов находятся конкретные значения данных. Для каждого поля определяется множество его значений.

В реляционной модели данных применяются разделы реляционной алгебры, откуда и была заимствованна соответствующая терминология.В реляционной алгебре поименованный столбец отношения называется атрибутом, а множество всех возможных значений конкретного атрибута – доменом. Строки таблицы со значениями разных атрибутов называют кортежами. Атрибут, значение которого однозначно идентифицирует кортежи, называется ключевым (или просто ключом). Так ключевое поле – это такое поле, значения которого в данной таблице не повторяется. В отличие от иерархической и сетевой моделей данных в реляционной отсутствует понятие группового отношения. Для отражения ассоциаций между кортежами разных отношений используется дублирование их ключей. Сложный ключ выбирается в тех случаях, когда ни одно поле таблицы однозначно не определяет запись.

Записи в таблице хранятся упорядоченными по ключу. Ключ может быть простым, состоящим из одного поля, и сложным, состоящим из нескольких полей. Сложный ключ выбирается в тех случаях, когда ни одно поле таблицы однозначно не определяет запись.

Кроме первичного ключа в таблице могут быть вторичные ключи, называемые еще внешними ключами, или индексами. Индекс – это поле или совокупность полей, чьи значения имеются в нескольких таблицах и которое является первичным ключом в одной из них. Значения индекса могут повторяться в некоторой таблице. Индекс обеспечивает логическую последовательность записей в таблице, а также прямой доступ к записи.

По первичному ключу всегда отыскивается только одна строка, а по вторичному – может отыскиваться группа строк с одинаковыми значениями первичного ключа. Ключи нужны для однозначной идентификации и упорядочения записей таблицы, а индексы для упорядочения и ускорения поиска.

Индексы можно создавать и удалять, оставляя неизменным содержание записей реляционной таблицы. Количество индексов, имена индексов, соответствие индексов полям таблицы определяется при создании схемы таблицы.

Индексы позволяют эффективно реализовать поиск и обработку данных, формирую дополнительные индексные файлы. При корректировке данных автоматически упорядочиваются индексы, изменяется местоположение каждого индекса согласно принятому условию (возрастанию или убыванию значений). Сами же записи реляционной таблицы не перемещаются при удалении или включении новых экземпляров записей, изменении значений их ключевых полей.

С помощью индексов и ключей устанавливаются связи между таблицами. Связь устанавливается путем присвоения значений внешнего ключа одной таблицы значениям первичного ключа другой. Группа связанных таблиц называется схемой данных. Информация о таблицах, их полях, ключах и т.п. называется метаданными.

**Достоинства и недостатки реляционной модели данных**

Достоинства

Изложение информации в простой и понятной для пользователя форме (таблица).

Реляционная модель данных основана на строгом математическом аппарате, что позволяет лаконично описывать необходимые операции над данными.

Независимость данных от изменения в прикладной программе при изменении.

Позволяет создавать языки манипулирования данными не процедурного типа.

Для работы с моделью данных нет необходимости полностью знать организацию БД.

Недостатки

Относительно медленный доступ к данным.

Трудность в создании БД основанной на реляционной модели.

Трудность в переводе в таблицу сложных отношений.

Требуется относительно большой объем памяти.

**Вывод**

Взяв во внимание вышеперечисленное, в качестве основы для проекта была выбрана реляционная модель данных. Изначальные предположения дают понимание, что количество обращение в единицу времени к данной базе будут небольшими, поэтому отсутствует требование к быстродействию. Однако для работы базы данных необходимо понятное для человека представление данных, что однозначно склоняет к выбору реляционной модели.

**1.3. Реляционные базы данных**

**Реляционная база данных** - база данных, построенная на основе реляционной модели. В реляционной базе каждый объект задается записью (строкой) в таблице. Реляционная база создается и затем управляется с помощью реляционной системы управления базами данных. Фактически реляционная база данных это тело связанной информации, сохраняемой в двухмерных таблицах. Связь между таблицами может находить свое отражение в структуре данных, а может только подразумеваться, то есть присутствовать на неформализованном уровне. Каждая таблица БД представляется как совокупность строк и столбцов, где строки соответствуют экземпляру объекта, конкретному событию или явлению, а столбцы - атрибутам (признакам, характеристикам, параметрам) объекта, события, явления. Реляционные базы данных предоставляют более простой доступ к оперативно составляемым отчетам (обычно через SQL) и обеспечивают повышенную надежность и целостность данных благодаря отсутствию избыточной информации.

Реляционные системы берут свое начало в математической теории множеств. Эдгар Кодд, сотрудник исследовательской лаборатории корпорации IBM в Сан-Хосе, по существу, создал и описал концепцию реляционных баз данных в своей основополагающей работе «Реляционная модель для крупных, совместно используемых банков данных» (A Relational Model of Data for Large Shared Data Banks. Communications of the ACM, июнь 1970).

Нечеткость многих терминов, используемых в сфере обработки данных, заставила Кодда отказаться от них и придумать новые или дать более точные определения существующим. Так, он не мог использовать широко распространенный термин "запись", который в различных ситуациях может означать экземпляр записи, либо тип записей, запись в стиле Кобола (которая допускает повторяющиеся группы) или плоскую запись (которая их не допускает), логическую запись или физическую запись, хранимую запись или виртуальную запись и т.д. Вместо этого он использовал термин "кортеж длины n" или просто "кортеж", которому дал точное определение.

Кодд предложил модель, которая позволяет разработчикам разделять свои базы данных на отдельные, но взаимосвязанные таблицы, что увеличивает производительность, но при этом внешнее представление остается тем же, что и у исходной базы данных. С тех пор Кодд считается отцом-основателем отрасли реляционных баз данных. Кодд сформулировал 12 правил для реляционных баз данных, большинство которых касаются целостности и обновления данных, а также доступа к ним.

**Правила Кодда**

Правило 0: Основное правило (Foundation Rule):

Система, которая рекламируется или позиционируется как реляционная система управления базами данных, должна быть способна управлять базами данных, используя исключительно свои реляционные возможности.

Правило 1: Информационное правило (The Information Rule):

Вся информация в реляционной базе данных на логическом уровне должна быть явно представлена единственным способом: значениями в таблицах.

Правило 2: Гарантированный доступ к данным (Guaranteed Access Rule):

В реляционной базе данных каждое отдельное (атомарное) значение данных должно быть логически доступно с помощью комбинации имени таблицы, значения первичного ключа и имени столбца.

Правило 3: Систематическая поддержка отсутствующих значений (Systematic Treatment of Null Values):

Неизвестные, или отсутствующие значения NULL, отличные от любого известного значения, должны поддерживаться для всех типов данных при выполнении любых операций. Например, для числовых данных неизвестные значения не должны рассматриваться как нули, а для символьных данных — как пустые строки.

Правило 4: Доступ к словарю данных в терминах реляционной модели (Active On-Line Catalog Based on the Relational Model):

Словарь данных должен сохраняться в форме реляционных таблиц, и СУБД должна поддерживать доступ к нему при помощи стандартных языковых средств, тех же самых, которые используются для работы с реляционными таблицами, содержащими пользовательские данные.

Правило 5: Полнота подмножества языка (Comprehensive Data Sublanguage Rule):

Система управления реляционными базами данных должна поддерживать хотя бы один реляционный язык, который

(а) имеет линейный синтаксис,

(б) может использоваться как интерактивно, так и в прикладных программах,

(в) поддерживает операции определения данных, определения представлений, манипулирования данными (интерактивные и программные), ограничители целостности, управления доступом и операции управления транзакциями (begin, commit и rollback).

Правило 6: Возможность изменения представлений (View Updating Rule):

Каждое представление должно поддерживать все операции манипулирования данными, которые поддерживают реляционные таблицы: операции выборки, вставки, изменения и удаления данных.

Правило 7: Наличие высокоуровневых операций управления данными (High-Level Insert, Update, and Delete):

Операции вставки, изменения и удаления данных должны поддерживаться не только по отношению к одной строке реляционной таблицы, но и по отношению к любому множеству строк.

Правило 8: Физическая независимость данных (Physical Data Independence):

Приложения не должны зависеть от используемых способов хранения данных на носителях, от аппаратного обеспечения компьютеров, на которых находится реляционная база данных.

Правило 9: Логическая независимость данных (Logical Data Independence):

Представление данных в приложении не должно зависеть от структуры реляционных таблиц. Если в процессе нормализации одна реляционная таблица разделяется на две, представление должно обеспечить объединение этих данных, чтобы изменение структуры реляционных таблиц не сказывалось на работе приложений.

Правило 10: Независимость контроля целостности (Integrity Independence):

Вся информация, необходимая для поддержания целостности, должна находиться в словаре данных. Язык для работы с данными должен выполнять проверку входных данных и автоматически поддерживать целостность данных.

Правило 11: Независимость от расположения (Distribution Independence):

База данных может быть распределённой, может находиться на нескольких компьютерах, и это не должно оказывать влияния на приложения. Перенос базы данных на другой компьютер не должен оказывать влияния на приложения.

Правило 12: Согласование языковых уровней (The Nonsubversion Rule):

Если используется низкоуровневый язык доступа к данным, он не должен игнорировать правила безопасности и правила целостности, которые поддерживаются языком более высокого уровня.

Сущность реляционной базы данных

Реляционная база данных представляет собой набор таблиц (сущностей). Таблицы состоят из колонок и строк (кортежей). Внутри таблиц могут быть определены ограничения, между таблицами существуют отношения. При помощи SQL можно выполнять запросы, которые возвращают наборы данных, получаемых из одной или нескольких таблиц. В рамках одного запроса данные получаются из нескольких таблиц путем их соединения (JOIN), чаще всего для соединения используются те же колонки, которые определяют отношения между таблицами.

Нормализация — это процесс структурирования модели данных, обеспечивающий связность и отсутствие избыточности в данных. Целью нормализации реляционной базы данных является устранение недостатков структуры базы данных, приводящих к избыточности, которая, в свою очередь, потенциально приводит к различным аномалиям и нарушениям целостности данных. Теоретики реляционных баз данных в процессе развития теории выявили и описали типичные примеры избыточности и способы их устранения. Реляционные хранилища обеспечивают наилучшую смесь простоты, устойчивости, гибкости, производительности, масштабируемости и совместимости. Касаемо масштабируемости, реляционные БД хорошо масштабируются только в том случае, если располагаются на единственном сервере.

Особенностью реляционной базы данных является использование в ней реляционной модели данных и вытекающие из этого последствия:

Модель данных в реляционных БД определена заранее. Является строго типизированной, содержит ограничения и отношения для обеспечения целостности данных.

Модель данных основана на естественном представлении содержащихся данных, а не на функциональности приложения.

Модель данных подвергается нормализации, чтобы избежать дублирования данных. Нормализация порождает отношения между таблицами. Отношения связывают данные разных таблиц.

В реляционной базе данных данные создаются, обновляются, удаляются и запрашиваются с использованием языка структурированных запросов (SQL). SQL-запросы могут извлекать данные как из одиночной таблица, так и из нескольких таблиц. Такие запросы могут включать агрегации и сложные фильтры. Реляционная БД обычно содержит встроенную логику, такую как триггеры, хранимые процедуры и функции.

Реляционная система управления базой данных (РСУБД) - СУБД, управляющая реляционными базами данных.

Доступ к реляционным базам данных осуществляется через реляционные системы управления базами данных (РСУБД). Почти все системы баз данных, которые мы используем, являются реляционными, такие как Oracle, SQL Server, MySQL, Sybase, DB2, TeraData и так далее. Причины такого доминирования неочевидны. На протяжении всего существования реляционных БД они постоянно предлагали наилучшую смесь простоты, устойчивости, гибкости, производительности, масштабируемости и совместимости в сфере управлении данными.

Например, простой SELECT запрос может иметь сотни потенциальных путей выполнения, которые оптимизатор оценит непосредственно во время выполнения запроса. Все это скрыто от пользователей, однако внутри РСУБД создает план выполнения, основывающийся на вещах вроде алгоритмов оценки стоимости и наилучшим образом отвечающий запросу. Однако чтобы обеспечить все эти особенности, реляционные хранилища невероятно сложны внутри.

Реляционная система управления базой данных содержит:

командный язык;

язык программирования с ориентацией на обработку таблиц;

интерпретирующую и/или компилирующую систему; и

пользовательскую оболочку.

**Таблица реляционной базы данных -** это совокупность связанных данных, хранящихся в структурированном виде в [базе данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85). Она состоит из столбцов и строк.

В [реляционных базах данных](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%B1%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) и плоских [файлах баз данных](https://en.wikipedia.org/wiki/Flat_file_database), **таблица** — это набор элементов данных (значений), использующий модель вертикальных столбцов (имеющих уникальное имя) и горизонтальных строк. Ячейка — место, где строка и столбец пересекаются. Таблица содержит определенное число столбцов, но может иметь любое количество строк. Каждая строка однозначно определяется одним или несколькими уникальными значениями, которые принимают её ячейки из определенного подмножества столбцов. Подмножество столбцов, которое уникально идентифицирует строку, называется [первичным ключом](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B2%D0%B8%D1%87%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BB%D1%8E%D1%87).

«Таблица» — это ещё один термин для «[отношения](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BD%D0%BE%D1%88%D0%B5%D0%BD%D0%B8%D0%B5_(%D1%80%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C))»; разница между ними в том, что таблица обычно представляет собой [мультимножество](https://ru.wikipedia.org/wiki/%D0%9C%D1%83%D0%BB%D1%8C%D1%82%D0%B8%D0%BC%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2%D0%BE) (набор) строк, тогда как отношение представляет собой [множество](https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2%D0%BE_(%D1%82%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)) и не допускает дубликатов. Помимо обычных данных, таблицы, как правило, имеют связанные с ними [метаданные](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%B0%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D0%B5), такие как [ограничения](https://en.wikipedia.org/wiki/Check_constraint), относящиеся к таблицам в целом или к значениям в определенных столбцах.

Данные в таблицах не обязательно физически хранятся в базе данных. [Представления](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D0%B4%D1%81%D1%82%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_(%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)) также функционируют, как реляционные таблицы, но их данные вычисляются во время выполнения запроса. Внешние таблицы (например, в СУБД [Informix](https://ru.wikipedia.org/wiki/Informix) или [Oracle](https://ru.wikipedia.org/wiki/Oracle_Database),) также можно рассматривать как представления.

**Хранимая процедура -** объект [базы данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), представляющий собой набор [SQL](https://ru.wikipedia.org/wiki/SQL)-инструкций, который компилируется один раз и хранится на сервере. Хранимые процедуры очень похожи на обыкновенные [процедуры](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D0%B4%D1%83%D1%80%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) [языков](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) высокого уровня, у них могут быть входные и выходные параметры и локальные [переменные](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), в них могут производиться числовые вычисления и операции над символьными данными, результаты которых могут присваиваться переменным и параметрам. В хранимых процедурах могут выполняться стандартные операции с базами данных (как [DDL](https://ru.wikipedia.org/wiki/DDL), так и [DML](https://ru.wikipedia.org/wiki/DML)). Кроме того, в хранимых процедурах возможны циклы и ветвления, то есть в них могут использоваться инструкции управления процессом исполнения.

Хранимые процедуры похожи на определяемые пользователем функции (UDF). Основное различие заключается в том, что пользовательские функции можно использовать как и любое другое выражение в SQL запросе, в то время как хранимые процедуры должны быть вызваны с помощью функции EXEC.

Хранимые процедуры могут возвращать множества результатов, то есть результаты запроса SELECT. Такие множества результатов могут обрабатываться, используя курсоры, другими сохранёнными процедурами, возвращая указатель результирующего множества, либо же приложениями. Хранимые процедуры могут также содержать объявленные переменные для обработки данных и курсоров, которые позволяют организовать цикл по нескольким строкам в таблице. Стандарт SQL предоставляет для работы выражения IF, LOOP, REPEAT, CASE и многие другие. Хранимые процедуры могут принимать переменные, возвращать результаты или изменять переменные и возвращать их, в зависимости от того, где переменная объявлена.

Реализация хранимых процедур варьируется от одной СУБД к другой. Большинство крупных поставщиков баз данных поддерживают их в той или иной форме. В зависимости от СУБД, хранимые процедуры могут быть реализованы на различных языках программирования, таких, как SQL, Java, C или C++. Хранимые процедуры, написанные не на SQL, могут самостоятельно выполнять SQL-запросы, а могут и не выполнять. Всё более широкое использование хранимых процедур привело к появлению процедурных элементов в языке SQL стандарта SQL:1999 и SQL:2003 в части [SQL/PSM](https://ru.wikipedia.org/wiki/SQL/PSM). Это сделало SQL [императивным](https://ru.wikipedia.org/wiki/%D0%98%D0%BC%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) языком программирования. Большинство СУБД предлагает собственные проприетарные и расширения производителя, сверх SQL/PSM.

Хранимые процедуры обычно создаются с помощью языка SQL и конкретной его реализации в выбранной [СУБД](https://ru.wikipedia.org/wiki/%D0%A1%D0%A3%D0%91%D0%94). Например, для этих целей в СУБД [Microsoft SQL Server](https://ru.wikipedia.org/wiki/Microsoft_SQL_Server) существует язык [Transact-SQL](https://ru.wikipedia.org/wiki/Transact-SQL), в [Oracle](https://ru.wikipedia.org/wiki/Oracle_(%D0%A1%D0%A3%D0%91%D0%94)) — [PL/SQL](https://ru.wikipedia.org/wiki/PL/SQL), в [InterBase](https://ru.wikipedia.org/wiki/InterBase) и [Firebird](https://ru.wikipedia.org/wiki/Firebird) — [PSQL](https://ru.wikipedia.org/w/index.php?title=PSQL&action=edit&redlink=1), в [PostgreSQL](https://ru.wikipedia.org/wiki/PostgreSQL) — [PL/pgSQL](https://ru.wikipedia.org/wiki/PL/pgSQL), [PL/Tcl](https://ru.wikipedia.org/wiki/PL/Tcl), [PL/Perl](https://ru.wikipedia.org/wiki/PL/Perl), [PL/Python](https://ru.wikipedia.org/wiki/PL/Python), в [IBM DB2](https://ru.wikipedia.org/wiki/IBM_DB2) — [SQL/PL](https://ru.wikipedia.org/w/index.php?title=SQL/PL&action=edit&redlink=1) ([англ.](https://en.wikipedia.org/wiki/SQL_PL)), в [Informix](https://ru.wikipedia.org/wiki/Informix) — [SPL](https://ru.wikipedia.org/w/index.php?title=Stored_Procedure_Language&action=edit&redlink=1). [MySQL](https://ru.wikipedia.org/wiki/MySQL) достаточно близко следует стандарту [SQL:2003](https://ru.wikipedia.org/wiki/SQL:2003), её язык похож на SQL/PL.

В некоторых СУБД возможно использование хранимых процедур, написанных на любом [языке программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), способном создавать независимые исполняемые файлы, например, на [C++](https://ru.wikipedia.org/wiki/C%2B%2B) или [Delphi](https://ru.wikipedia.org/wiki/Delphi_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)). В терминологии Microsoft SQL Server такие процедуры называются *расширенными хранимыми процедурами* и являются просто функциями, содержащимися в [Win32](https://ru.wikipedia.org/wiki/Win32)-[DLL](https://ru.wikipedia.org/wiki/DLL). А, например, в Interbase и Firebird для функций, вызываемых из DLL/SO, определено другое название — [UDF](https://ru.wikipedia.org/w/index.php?title=User_Defined_Function&action=edit&redlink=1) (User Defined Function). В [MS SQL 2005](https://ru.wikipedia.org/wiki/Microsoft_SQL_Server#SQL_Server_2005) появилась возможность написания хранимых процедур на любом языке [.NET](https://ru.wikipedia.org/wiki/.NET_Framework), а от расширенных хранимых процедур в будущем планируется отказаться. СУБД Oracle, в свою очередь, допускает написание хранимых процедур на языке [Java](https://ru.wikipedia.org/wiki/Java). В IBM DB2 написание хранимых процедур и функций на обычных языках программирования является традиционным способом, поддерживаемым с самого начала, а процедурное расширение SQL было добавлено в эту СУБД только в достаточно поздних версиях, после его включения в стандарт ANSI. Также процедуры на [Java](https://ru.wikipedia.org/wiki/Java) и [С](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) поддерживает [Informix](https://ru.wikipedia.org/wiki/Informix).

В СУБД [Oracle](https://ru.wikipedia.org/wiki/Oracle_(%D0%A1%D0%A3%D0%91%D0%94)) хранимые процедуры могут объединяться в так называемые пакеты ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *packages*). Пакет состоит из двух частей — спецификации ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *package specification*), в которой указывается определение хранимой процедуры, и тела ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *package body*), где находится её реализация. Таким образом Oracle позволяет отделить интерфейс программного кода от его реализации.

В СУБД [IBM DB2](https://ru.wikipedia.org/wiki/IBM_DB2) хранимые процедуры можно объединять в [модули](http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.apdv.sqlpl.doc/doc/c0053740.html).

Хранимые процедуры позволяют повысить производительность, расширяют возможности программирования и поддерживают функции безопасности данных.

Вместо хранения часто используемого запроса, клиенты могут ссылаться на соответствующую хранимую процедуру. При вызове хранимой процедуры её содержимое сразу же обрабатывается сервером.

Кроме собственно выполнения запроса, хранимые процедуры позволяют также производить вычисления и манипуляцию данными — изменение, удаление, выполнять DDL-операторы (не во всех СУБД!) и вызывать другие хранимые процедуры, выполнять сложную транзакционную логику. Один-единственный оператор позволяет вызвать сложный сценарий, который содержится в хранимой процедуре, что позволяет избежать пересылки через сеть сотен команд и, в особенности, необходимости передачи больших объёмов данных с клиента на сервер.

В большинстве СУБД при первом запуске хранимой процедуры она компилируется (выполняется синтаксический анализ и генерируется план доступа к данным). В дальнейшем её обработка осуществляется быстрее. В СУБД Oracle выполняется интерпретация хранимого процедурного кода, сохраняемого в словаре данных. Начиная с версии Oracle 10g поддерживается так называемая естественная компиляция (native compilation) хранимого процедурного кода в Си и затем в машинный код целевой машины, после чего при вызове хранимой процедуры происходит прямое выполнение её скомпилированного объектного кода.

Возможности программирования

Созданную хранимую процедуру можно вызвать в любой момент, что обеспечивает модульность и стимулирует повторное использование кода. Последнее облегчает сопровождение базы данных, так как она становится изолированной от меняющихся бизнес-правил. Модифицировать хранимую процедуру в соответствии с новыми правилами можно в любой момент. После этого все приложения, использующие её, автоматически придут в соответствие с новыми бизнес-правилами без непосредственной модификации.

Для упрощения тестирования, независимости бизнес-логики приложений от СУБД существует подход, в котором СУБД выступает лишь в роли хранилища, с минимальным количеством хранимых процедур или полном отказе от них. При этом используется отображение программных сущностей бизнес-логики на хранилище. См. ORM (англ. Object-relational mapping, рус. Объектно-реляционное отображение)

Безопасность

Использование хранимых процедур позволяет ограничить или вообще исключить непосредственный доступ пользователей к таблицам базы данных, оставив пользователям только разрешения на выполнение хранимых процедур, обеспечивающих косвенный и строго регламентированный доступ к данным. Кроме того, некоторые СУБД поддерживают шифрование текста (wrapping) хранимой процедуры.

Эти функции безопасности позволяют изолировать от пользователя структуру базы данных, что обеспечивает целостность и надежность базы.

Снижается вероятность таких действий как «внедрение SQL-кода», поскольку хорошо написанные хранимые процедуры дополнительно проверяют входные параметры перед тем, как передать запрос СУБД.

# Конструкторский раздел

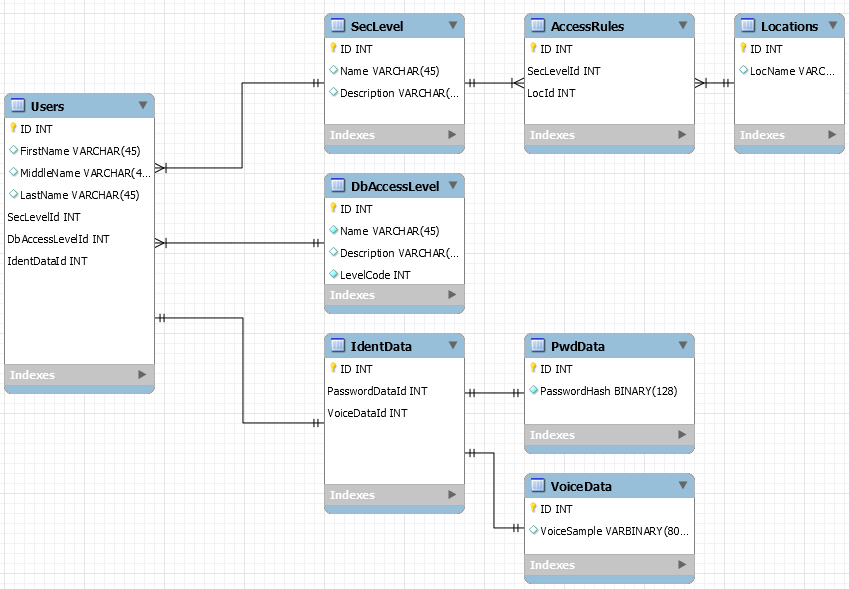
В этом разделе будет рассмотрено построение базы данных и способы доступа к данным из внешнего приложения.

* 1. Структура данных

База данных состоит из нескольких таблиц, которые в совокупности представляют описание всех данных охранной системы.

Охранная система включает в себя несколько категорий компонентов: исполнительные механизмы, база данных, приложения для осуществления доступа к данным и ряд хранимых процедур, предоставляющих базовые возможности для обработки данных.

ER-диаграмма базы данных:



* 1. Алгоритм проверки наличия доступа



Рис. 1. Алгоритм проверки наличия доступа

На вход алгоритму подается идентификатор пользователя и идентификатор помещения. Алгоритм обращается к базе данных и получает идентификатор уровня доступа (SecLevelId) данного пользователя. Далее с помощью этого идентификатора алгоритм получает список территорий, для которых данного уровня доступа достаточно. В случае, если список содержит идентификатор указанной территории, алгоритм решает, что доступ разрешен и возвращает истину. В противном случае (если идентификатор пользователя не найден в базе или не указан, если его уровня доступа недостаточно) возвращается ложь.

Этот алгоритм выполняется на стороне сервера, который обслуживает базу данных. Таким образом вызывающее приложение не получает никакой дополнительной информации, что снижает вероятность неправомерного использования. Вызывающее приложение может быть установлено непосредственно у входа на охраняемую территорию и может управлять исполнительным механизмом, например, электронным замком.

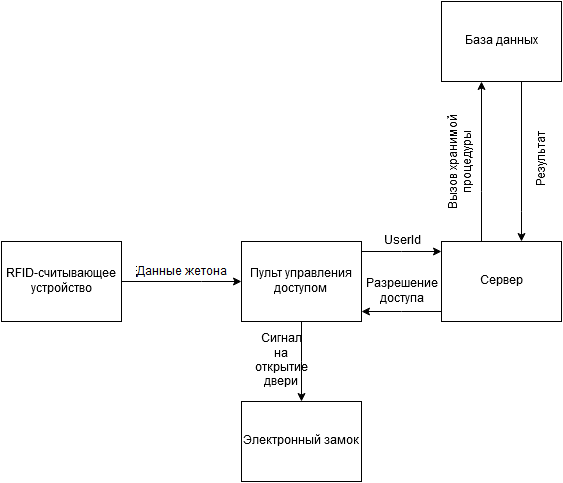


Рис. 1. Пример исполнения контрольно-пропускного пункта

* 1. Структуры данных

В качестве фундаментальной структуры данных для описания виртуального вектора в пространстве был принят трёхмерный математический вектор – структура, хранящая три поля, которые отвечают за трёхмерные координаты вектора. Эта структура была выбрана, т.к. даёт наглядное представление как любого вектора в трёхмерном пространстве, так и любой точки.

Для хранения поворота выбраны кватернионы – структура из четырех полей, описывающих ось вращения и угол поворота. Для этой роли также рассматривались углы Эйлера, но их применение ограничено возникновениями т.н. эффекта «Шарнирного замка».

Трансформации координат осуществляются структурами, представляющими квадратные матрицы с размером четыре на четыре. Эта структура позволяет удобно выполнять работы с трёхмерными векторами (в однородных координатах) и кватернионами. За счет заранее известного размера матрицы и некоторых свойств матриц преобразования многие операции (например, умножение матриц) можно значительно оптимизировать.

* 1. Интерпретация пользовательского кода

Пользовательский код должен быть ассоциирован с одним определенным объектом сцены (с ракетой или с целью). Поэтому было принято решение реализовать модуль управления для каждого из объектов. Модуль в своей области памяти имеет объект, представляющий экземпляр интерпретатора языка Lua. Каждый экземпляр имеет свою выделенную и изолированную область памяти, поэтому могу работать совершенно независимо. Пользовательская программа должна иметь определенный формат. Во-первых программа обязательно должна содержать функцию Update(), которая вызывается периодически с малым интервалом. В этой функции предполагается наличие всей логики поведения объекта, анализ окружающей обстановки и обновление локальных данных. Также допускается наличие программного кода вне какой-либо функции, но следует учитывать, что этот код будет выполнен только один раз во время инициализации соответствующего экземпляра интерпретатора. По желанию пользователя может быть добавлена функция OnStart(), которая вызывается один раз во время следующего кадра после инициализации интерпретатора. Ее наличие не обязательно. Запрещено размещение бесконечных циклов и любых неделимых трудоемких вычислений где-либо в программе. Вместо этого предполагается растягивание операции по нескольким вызовам функции Update().

* 1. Процесс визуализации

Все видимые объекты сцены представляются набором треугольных многоугольников. Перед началом визуализации очередного кадра все объекты разбираются на треугольники, каждый из которых будет обработан в совокупности с треугольниками других объектов.

Для преобразований перемещения, поворота, масштабирования и перспективы используются представленные структуры данных:

где (1) – матрица перемещения, (2) – матрица масштабирования, (3), (4) и (5) – матрицы поворота вокруг ортогональных осей X, Y и Z соответственно. Для перспективного преобразования используется матрица (6).

Ниже представлена схема процесса обработки графики в каждом кадре.



Рисунок 4. Алгоритм рендеринга кадра

# Технологический раздел

В этом разделе будет обоснован выбор языка программирования, рассмотрен интерфейс пользователя, входные данные, вывод результатов и структура программного продукта.

* 1. Выбор и обоснование ЯП

В качестве платформы для разработки программного продукта была выбрана “.NET Framework” и язык C#. Выбор обоснован несколькими факторами:

1. Платформа .NET Framework снабжена мощными ресурсами документации, которые позволяют найти информацию и рассмотреть тонкие моменты практически каждой детали системы.
2. Язык C# значительно безопаснее языка C++ с точки зрения управления памятью и уровней доступа. В C# умышленно удалены многие опасные с архитектурной точки зрения возможности, такие как множественное наследование.
3. Благодаря технологии JIT-компиляции программы, написанные на этом языке, не имеют значительных потерь в производительности по отношению к компилируемым языкам, таким как C++, при этом позволяя системе контролировать безопасность кода и не допускать выполнения критических инструкций.
4. Платформа (и язык C#) позволяют комбинировать управляемый код с библиотечными вызовами кода компилируемых языков, таких как C и C++.
5. Язык C# позволяет разработчику сконцентрироваться на основных задачах проекта, не отвлекаясь на мелочи (благодаря ресурсам платформы .NET Framework)
   1. Интерфейс пользователя

Пользовательский интерфейс состоит из четырех «окон». Перед запуском очередной симуляции пользователю предлагается ввести исходные данные в окно Preferences. Во время симуляции видимым является окно Simulation, которое содержит элемент, отображающих трехмерную сцену. После окончания симуляции (например в результате попадания ракеты в цель) будет показано окно Trajectory Projection, с помощью которого можно получить данные о траекториях объектов. В случае аварийных ситуаций, спровоцированных пользовательским кодом, будет отображено окно LuaException, где можно получить информацию об ошибке.

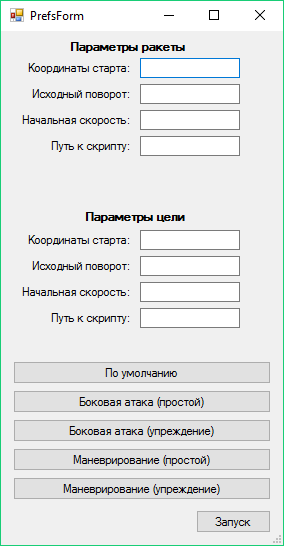


Рисунок 5. Вид окна настроек



Рисунок 6. Вид окна симуляции

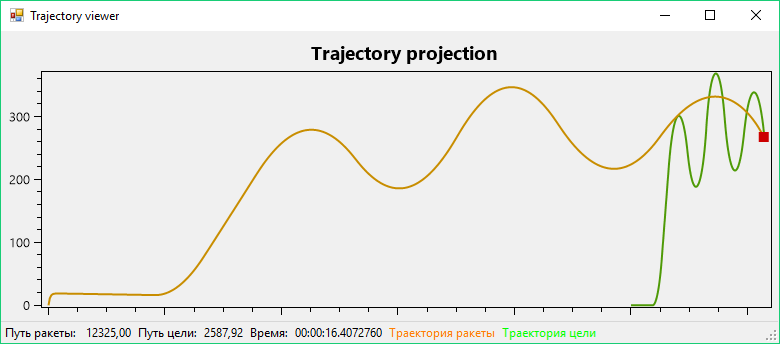


Рисунок 7. Вид окна вывода результатов

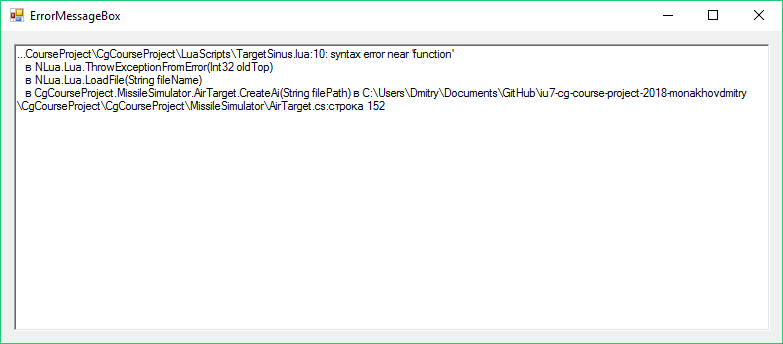


Рисунок 8. Вид окна с сообщением об ошибке в пользовательском коде

* 1. Входные и выходные данные

На вход программе подаются исходные параметры симуляции (положение в пространстве, начальная скорость) и сценарии их поведения. Во время работы симуляции, пользователь может изменять положение камеры. Выходными данными является информация об итогах симуляции: траектории объектов, время от старта до конца симуляции, расстояния, пройденные объектами.

* 1. Хранение и передача данных в системе

В связи с объектно-ориентированной парадигмой разработки, каждый программный объект является изолированным от других и предоставляет другим объектам свой интерфейс взаимодействия. С точки зрения пользовательского кода существует только ограниченное количество объектов. Передача данных возможно только к ассоциированному объекту, которым код управляем. Таким образом решена проблема несанкционированного доступа пользователя к внутренним структурам системы.

* 1. Требования к аппаратуре

Для стабильной работы программного продукта необходим многоядерный процессор. Рекомендуется количество ядер не менее четырех. Требуется как минимум 512 Мб оперативной памяти.

* 1. Требования к программному обеспечению

Необходимо запускать программный продукт на компьютере под управлением ОС Windows 7 или младше с установленным .NET Framework 4.6.5 или младше.

* 1. Порядок работы

Порядок работы состоит из нескольких этапов.

1. Разработка алгоритмов поведения
2. Ввод исходных параметров
3. Симуляция
4. Оценка результатов

После четвертого этапа пользователь имеет возможно незамедлительно вернуться к первому или второму этапу без необходимости перезапускать программу.

* 1. Сообщения системы

В случае аварийной ситуации, возникшей в пользовательском коде, программа выдаст данные об этой ошибки с подробными пояснениями.

* 1. Обзор результатов

В конце цикла симуляции пользователю будут предоставлены данные о результатах. Рассмотрим выводимые результаты на примере рисунка 8. На рисунке видны следующие данные:

1. Путь ракеты в метрах
2. Путь цели в метрах
3. Время от старта ракеты до столкновения с целью
4. Проекции траекторий ракеты и цели на горизонтальную плоскость.

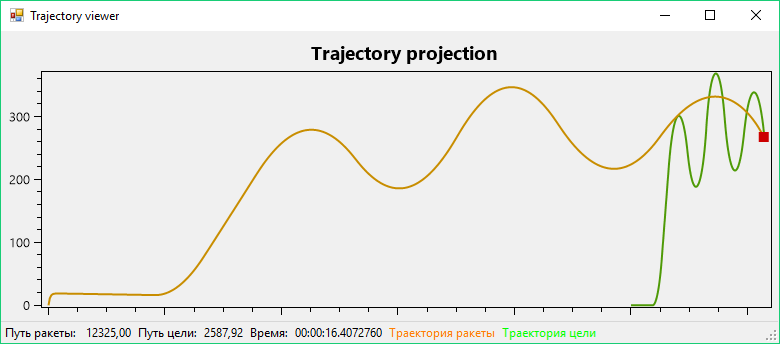


Рисунок 9. Пример результатов работы

Вышеперечисленные данные помогут сделать вывод об эффективности исследуемого алгоритма поведения реактивного снаряда.

# Экспериментально-исследовательский раздел

В этом разделе будет поставлен эксперимент и продемонстрирована эффективность данного программного продукта в исследовании проблемы.

Пусть воздушная цель летит равномерно и прямолинейно. Это может быть тяжелый неповоротливый грузовой самолет противника. Пусть также ракета запускается на удалении в 10 километров от цели. Примем скорость цели за 263 м/c (максимальная скорость стратегического бомбардировщика Ту-95), а скорость ракеты за 1250 м/с (максимальная скорость ракеты AIM-120 AMRAAM). Установим для ракеты модель поведения такую, что ракета в каждый момент времени стремится повернуть свой нос в сторону цели. Результаты эксперимента показаны на рисунке 9. Как видно, ракета летела по нерациональной траектории.

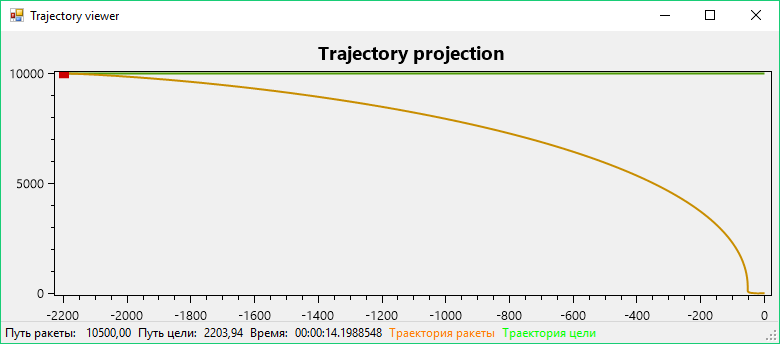


Рисунок 10. Результаты эксперимента 1

Пусть теперь ракета будет вычислять упреждение и поворачивать нос в сторону прогнозируемой точки встречи. Результат на рисунке 10. Теперь траектория цели значительно более рациональная. Ракета потратит меньше топлива, а значит увеличится максимальная дальность пуска.

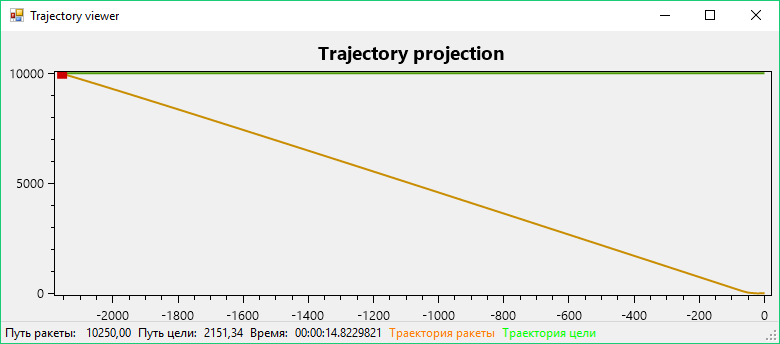


Рисунок 11. Результаты эксперимента 2

Но данная модель работает только с неповоротливой целью. Предположим теперь, что самолет противника может маневрировать, значительно изменяя свои траекторию во время движения. Используем ракету, которая движется на упреждение. Ракета начинает лететь «змейко». На каждое изменение направления ракета тратит энергию, у нее падает скорость, а значит вероятность попадания снижается. Проекция траекторий на рисунке 11.

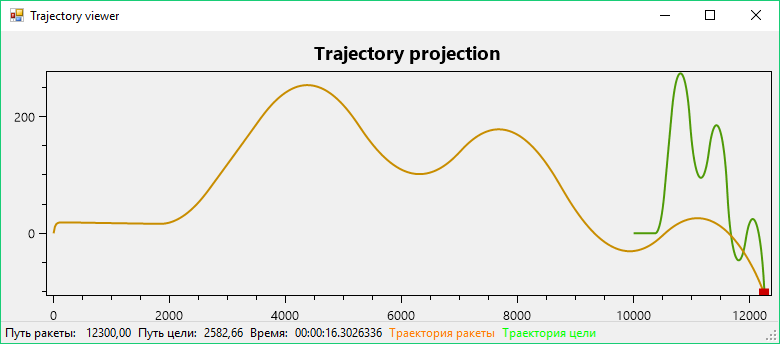


Рисунок 12. Результат эксперимента 3

Попытаемся решить проблему, снова установив ракете правильно преследования без упреждения. Теперь цель тратит энергию на маневрирование, а ракета нет. Рисунок 12.

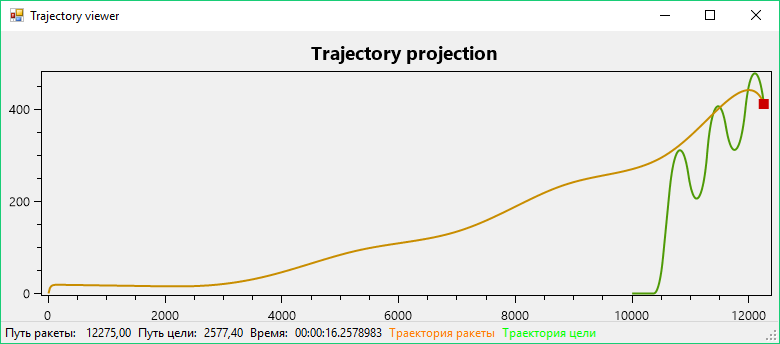


Рисунок 13. Результат эксперимента 4

Из поставленных экспериментов можно сделать вывод, что против слабо маневрирующей цели выгодно использовать алгоритм вычисления упреждения, но против подвижной цели это неэффективно и может привести к промаху из-за потери кинетической энергии. Следовательно, бортовой компьютер ракеты должен принимать во внимание в том числе поведение цели и делать вывод о том, какой алгоритм применить.

# Заключение

В рамках курсового проекта был разработан программный продукт, позволяющий моделировать поведение реактивных снарядов при преследовании воздушных целей, а также разрабатывать и анализировать алгоритмы управления для таких снарядов. В ходе эксперимента была продемонстрированы ситуации, в которых простые алгоритмы оказались неприменимы. Показана необходимость детального исследования проблемы с помощью привлечения программных средств симуляции.

# Список литературы

1. <http://www.gabrielgambetta.com/computer-graphics-from-scratch/introduction.html>
2. *Лурье А. И.* Аналитическая механика. — М.:[Физматлит](https://ru.wikipedia.org/wiki/%D0%A4%D0%B8%D0%B7%D0%BC%D0%B0%D1%82%D0%BB%D0%B8%D1%82). — 1961. — 824 с.
3. <http://vii.sfu-kras.ru/images/pdf/m8_metody-navedeniya.pdf> Разделы 3 и 6.