*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. **Сравнение алгоритмов построения трехмерного изображения**

Построение двумерного изображения на растре из трехмерной сцены – нетривиальная задача, требующая алгоритмического решения. Следовательно, необходимо рассмотреть доступные алгоритмы и выбрать наиболее подходящие для поставленных целей.

Алгоритм трассировки лучей

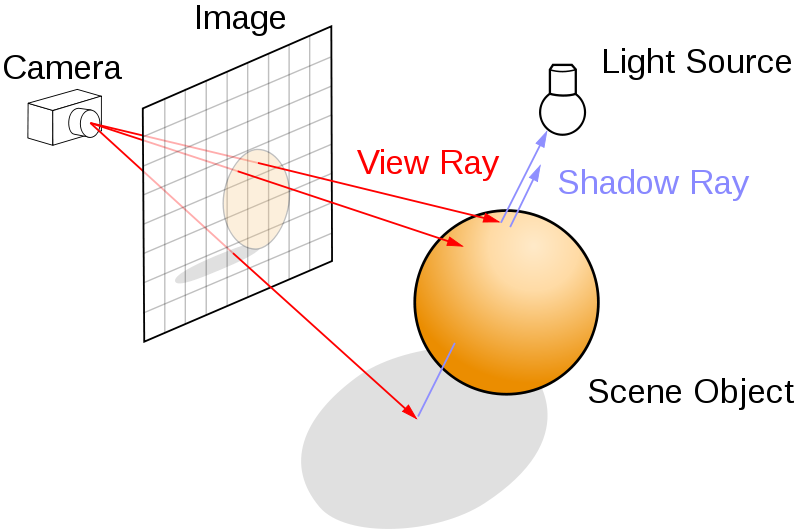


Рисунок 3. Схема алгоритма трассировки лучей

В узком смысле алгоритм трассировки лучей — это технология построения изображения трёхмерной моделей, при котором отслеживается траектория распространения луча (от экрана к источнику света или наоборот). Достоинства этого алгоритма впечатляют:

1. Возможность рендеринга гладких объектов без аппроксимации их полигональными (построенными из множества плоских поверхностей) поверхностями
2. Вычислительная сложность метода слабо зависит от сложности сцены
3. Возможность распараллеливания вычислений, т.к. каждый луч не зависит от других и обрабатывается отдельно
4. Отсечение невидимых поверхностей, перспектива и корректное изменение поля зрения являются следствиями алгоритма и не являются отдельно решаемыми задачами
5. Принцип работы, основанный на реальных физических законах

Несмотря на эти преимущества, для решения поставленных задач алгоритм непригоден. Основной его недостаток – производительность. Программное обеспечение должно формировать динамически-изменяющуюся сцену и отрисовывать его на растровый дисплей без задержек, заметных для человеческого глаза (для примера, частота кадров в современных видеофайлах около 60 кадров в секунду, что дает задержку между кадрами 0,0167 секунд). Алгоритм трассировки лучей не способен без глубоких модернизаций и хорошей вычислительной мощности ЭВМ давать такие показатели скорости, поэтому необходимо выбрать другой алгоритм.

Алгоритмы растеризации

Растеризация – это перевод математически описанных объектов в растровую форму. Данный процесс можно рассматривать как обратный трассировке: в случае с трассировкой лучей, алгоритм испускает луч из каждого пикселя растрового дисплея и анализирует его поведение в математически-описанного виртуальном пространстве. Алгоритмы растеризации – вторая основная группа алгоритмов рендеринга, которая подходит к задаче с другого конца – задается вопросом, что можно отрисовать на экране, и как выполнить этот процесс как можно быстрее. Результат работы этих алгоритмов очень близок к результатам, полученным путем трассировки лучей. Так как эти методы достаточно быстры, именно они и будут использоваться в построении изображения. Далее необходимо выбрать подходящие алгоритмы, для выполнения задач растеризации.

Первая задача – это растеризация отрезков. Отрезок сам по себе это объект, заданный четырьмя параметрами (координаты начала и конца). Математический отрезок не имеет толщины, поэтому для его изображения на растре необходимо выбирать ближайший пиксель растра. В качестве алгоритма для решения этой задачи был выбран алгоритм Брезенхема в связи с его высокой скоростью.

Следующая задача — это заполнение многоугольников. Самым простым многоугольником является треугольник – он однозначно задается тремя вершинами вне зависимости от их порядка, он гарантированно выпуклый (если не вырожден в отрезок) и однозначно задает плоскость, на которой лежит. Очевидно, что работа с треугольниками быстрее чем работа с другими видами многоугольников, что важно для отрисовки динамической сцены. В связи с этим все трёхмерные модели должны быть представлены в виде набора треугольников. Однако, возникает проблема – входные данные моделей являются внешними и отсутствуют гарантии, что они будут представлены исключительно треугольниками. В таком случае к модели будет применяться алгоритм триангуляции, который преобразует не-треугольный многоугольник в несколько треугольников.

Из-за специфики работы алгоритмов трассировки, в них задача удаления невидимых поверхностей не возникает (луч сталкивается с ближайшим объектом, не проникая дальше). Но в алгоритмах растеризации необходимо выбрать, какие именно элементы видимы для наблюдателя, а какие не видимы и, соответственно, не должны быть обработаны. Для решения этой задачи был выбран алгоритм, использующий Z-буфер. Он прост в использовании, принцип его работы очевиден, и он обладает вполне пригодными показателями скорости, которые почти не зависят от сложности сцены. Среди недостатков можно отметить большой объем памяти, возрастающий с увеличением размеров дисплея (но не сцены) и трудоёмкость реализации эффектов прозрачности. Первый из перечисленных недостатков не критичен, потому что современные ЭВМ обладают избыточными (для поставленных задач) ресурсами памяти (для растрового дисплея формата Full HD, размером 1920 x 1080 пикселей потребуется около 16 Мб памяти). Второй недостаток исправляется с помощью различных модификаций алгоритма, но в этой работе они не будут рассматриваться, так как это не соответствует поставленным задачам.

В связи с тем, что выбор пал на алгоритм, использующий Z-буфер, алгоритм заполнения треугольников должен уметь вычислять глубину (Z координату) каждого пикселя треугольника. Поиск точек пересечения сканирующей строки с рёбрами можно выполнить с помощью алгоритма Брезенхема для построения отрезков, но заполнение отрезка между пересечениями на очередной сканирующей строке выполнить тем же способом не получится. Алгоритм Брезенхема рассчитан на работу с целочисленными координатами очередной точки, что мешает его использовать для поиска Z-координаты пикселей. По этой причине заполнение строк будет проводиться с помощью интерполяции граничных значений.

* 1. **Затенения**

Определившись с отрисовкой трехмерных объектов, требуется выбрать алгоритм обработки затенения.

Одной из самых простых моделей затенения является модель плоскостного затенения. Её суть заключается в том, что интенсивность света вычисляется в центре каждого многоугольник в зависимости от направления к источнику света, и каждый пиксель многоугольника закрашивается с вычисленной интенсивностью. Этот метод очень прост и обладает хорошим быстродействием, но результат выглядит нереалистично и может сбивать с толку. Поэтому этот алгоритм будет рассмотрен и заложен в программе, но не будет основным.

Более хорошим вариантом является затенение по Гуро. Принцип метода состоит в вычислении нормалей вершин путём усреднения нормалей всех примыкающих к вершине граней. На основании значений нормалей вычисляется освещённость каждой вершины, которая представляется интенсивностью цвета в вершине. Многоугольники закрашиваются путём интерполяции значений интенсивности цвета в вершинах. Данный алгоритм обладает хорошей производительностью, но все еще обладает рядом недостатков (например, при слишком сильном приближении источника к грани яркость перестает увеличиваться), но они не столь существенны.

Еще одной моделью является затенение по Фонгу, в ней используется интерполяция вектора нормали. Этот метод значительно более затратный по времени, чем метод Гуро, поэтому выбор остановился на последнем.

* 1. Удаление невидимых поверхностей

Особое внимание следует уделить алгоритмам, которые осуществляют удаление невидимых поверхностей. В данном программном продукте объект «камера» является подвижным, в связи с чем некоторые её параметры могут значительно изменяться за единицу времени.

Каждый объект сцены, имеющий графическое представление, должен иметь привязанный к нему модуль «контейнер трёхмерной модели». Этот модуль отвечает за предоставление данных для алгоритма растеризации. В связи с тем, что сцена может содержать значительное количество графически представимых моделей (а, следовательно, и поверхностей), необходимо реализовать алгоритм отсечения тех поверхностей, которые не попадают в поле зрения камеры. Данный алгоритм определяет вершины поверхности, которые не лежат в поле зрения и отсекает их, либо формируя новую поверхность, либо полностью удаляя ее из списка активных поверхностей.

После обработки не лежащих в поле зрения, требуется обработать многоугольники, перекрываемые другими. Для этой цели реализован алгоритм, использующий Z-буфер. В классическом варианте этого алгоритма применяется прямая глубина, однако она недостаточно точна на близких расстояниях.

Формальное описание алгоритма:

1. Заполнить буфер кадра фоновым значением цвета
2. Заполнить Z-буфер нулевыми значениями обратной глубины
3. Растеризовать каждый многоугольник (в любом порядке)
4. Для каждого пиксела в каждом полученном растровом представлении вычислить обратную глубину пикселя (1 / Z)
5. Сравнить обратную глубину, хранящуюся в z-буфере в той же позиции
6. Если новое значение меньше старого, то старое перезаписывается и пиксель заносится в буфер кадра.
   1. **Задание поведения объектов**

Проблема состоит в необходимости предоставления программному обеспечению информации о поведении объектов в виртуальном пространстве. Алгоритм необходимо задавать на каком-либо языке программирования. Разработка нового языка программирования – слишком затратный и долгосрочный процесс, поэтому было принято решение использовать существующий интерпретируемый язык программирования. В таком случае программный продукт будет выступать интерпретатором и будет передавать команды пользовательского кода объектам виртуальной схемы. В качестве такого языка программирования был выбрал Lua. Это достаточно популярный и легко встраиваемый язык с открытым исходным кодом.

Пользовательская программа на языке Lua имеет доступ к информации об объекте, с которым она ассоциирована, а также с рядом дополнительных данных, которые необходимы для анализа виртуальной обстановки.

# Конструкторский раздел

В этом разделе будет рассмотрен способ представления объектов в виртуальном пространстве, а также метод обработки их поведения.

* 1. Структура сцены и объектов

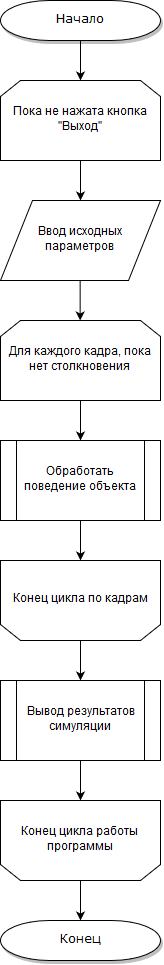
Виртуальная трёхмерная сцена представляет собой некоторый набор объектов, положение которых описываются тремя координатами (по осям X, Y и Z, из которых Y направлена вверх, X вперед, а Z вправо) и поворотом. Центр системы координат общий для всех обрабатываемых объектов сцены. Для формализации этой модели была разработана программная структура, состоящая из нескольких типов: сцена, трансформация, модуль. Объекты типа «сцена» являются контейнерами, объединяют несколько объектов симуляции в единое виртуальное пространство и обеспечивают их синхронное взаимодействие. Объекты типа «трансформация» описывают положение в сцене, в то время как «модули» наполняют объекты требуемой функциональностью.

Модули – сугубо логические субъединицы, не имеющие графического или какого-либо «материального» воплощения в сцене. Чтобы «воплотить» модуль, он должен быть привязан к какому-либо объекту типа «трансформация», что даст первому доступ ко всем параметрам второго (положение в пространстве, ориентация, другие привязанные модули и т.д.). В то же время каждый «воплощённый» модуль имеет доступ к информации обо всех объектах и их модулях в той сцене, в которой он находится и любой модуль может взаимодействовать с другим модулем в этой сцене. Такая система обеспечивает слаженное и легко синхронизируемое взаимодействие объектов, обладающих совершенно разными функциональными возможностями и моделями поведения.

Среди модулей следует особо выделить графические: камера, контейнер трехмерной модели и источник света. Для ускорения обработки графики эти модули регистрируются в соответствующей сцене особым образом так, что получение их данных значительно ускоряется. При этом в каждой сцене может быть только один объект с модулем «камера».

* 1. Симуляция поведения объектов

Рисунок 3. Схема процесса симуляции.



Моделирование динамической сцены осуществляется с помощью временно́й синхронизации. Все время выполнения программы разбивается на т.н. кадры, каждый из которых является независимым дискретным состоянием системы. Для каждого кадра программный комплекс осуществляет моделирование поведения (как предписанного при разработке, так и заданного пользователем), в том числе рендеринг сцены на растровый дисплей. Время между каждым кадром ограничено ради достижения комфортной для человека частоты смены изображения на дисплее (примерно 0,02 секунды). Однако в связи с необходимостью выполнения трудоёмких вычислений допустимо незначительное превышение этой задержки.

Каждый кадр после очередной итерации обработки поведения программа производит расчет расстояния между ракетой и целью. Если это расстояние меньше, чем дистанция срабатывания взрывателя, то ракета взрывается. Взрыв изображается в замедленном режиме и во время него ракета и цель неподвижны. После окончания взрыва посылается команда на остановку симуляции. Сразу после этого выводятся результаты эксперимента.

* 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.

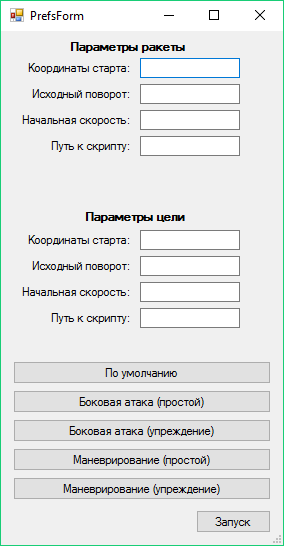


Рисунок 5. Вид окна настроек



Рисунок 6. Вид окна симуляции

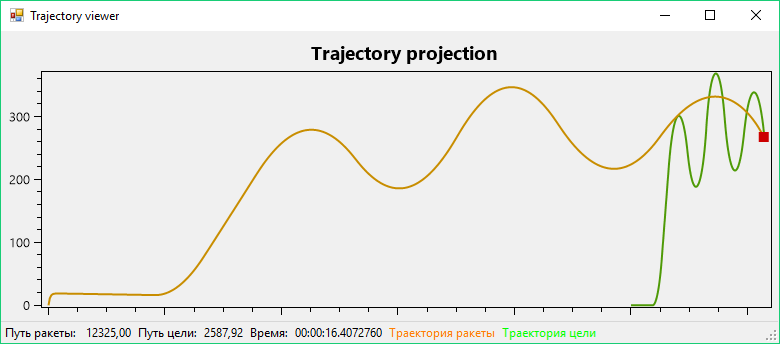


Рисунок 7. Вид окна вывода результатов

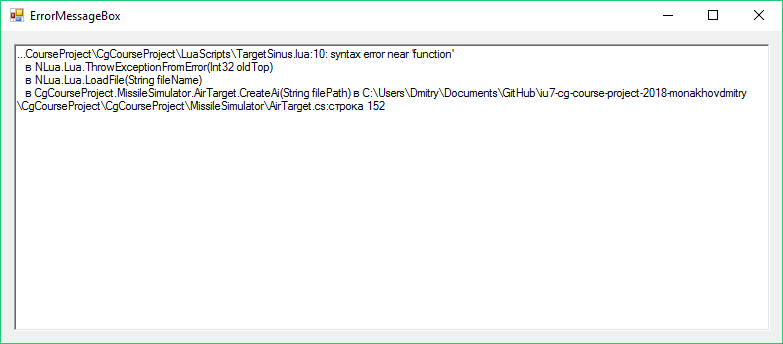


Рисунок 8. Вид окна с сообщением об ошибке в пользовательском коде

* 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.

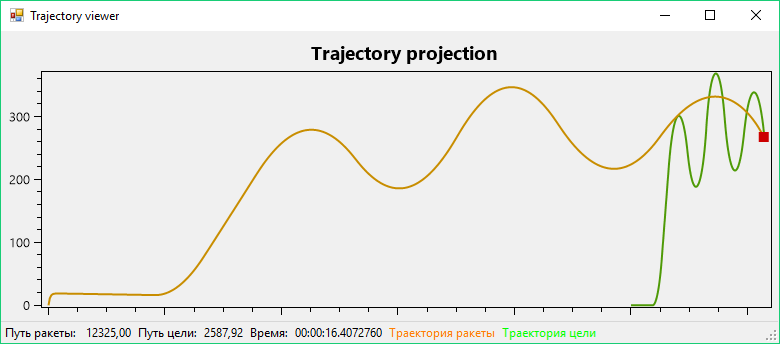


Рисунок 9. Пример результатов работы

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.

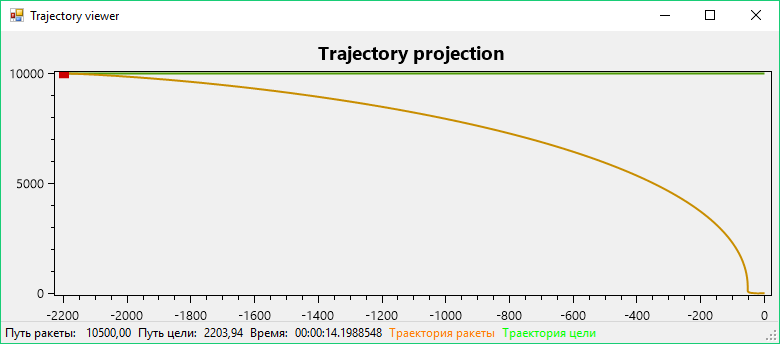


Рисунок 10. Результаты эксперимента 1

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.

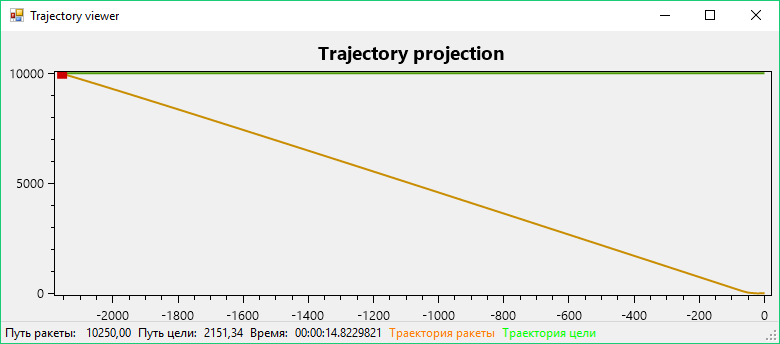


Рисунок 11. Результаты эксперимента 2

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.

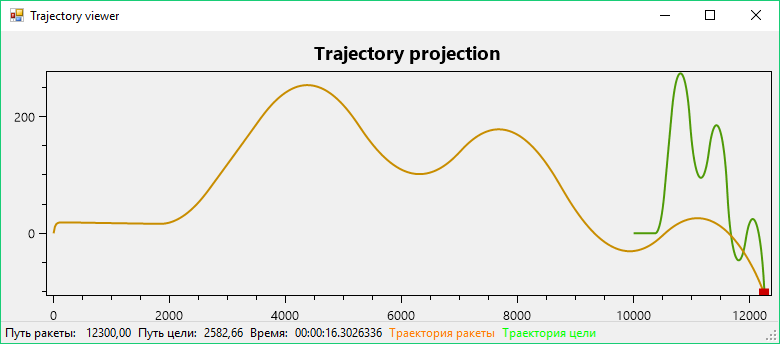


Рисунок 12. Результат эксперимента 3

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

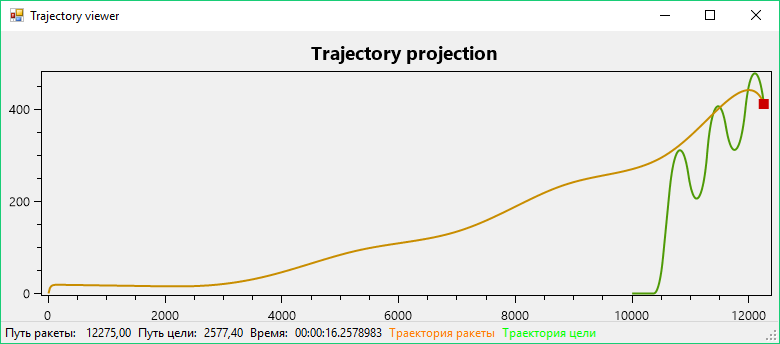


Рисунок 13. Результат эксперимента 4

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.