Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего профессионального образования

**“Уральский федеральный университет**

**имени первого Президента России Б.Н.Ельцина”**

Математико-механический факультет

Кафедра вычислительной математики

РАЗРАБОТКА СПЕЦИАЛИЗИРОВАННОЙ СРЕДЫ ТРЕХМЕРНОЙ ДИНАМИЧЕСКОЙ ВИЗУАЛИЗАЦИИ

|  |  |  |
| --- | --- | --- |
| "Допущен к защите"  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   "\_\_"\_\_\_\_\_\_\_\_\_\_\_\_2011 г. |  | Дипломная работа  студента 5 курса Кулакова Александра Юрьевича  Научный руководитель  Авербух Владимир Лазаревич доцент КИПУ  заведующий сектором компьютерной  визуализации ИММ УрО РАН  к.т.н. |

Екатеринбург

2011

РЕФЕРАТ

Кулаков А. Ю. РАЗРАБОТКА СПЕЦИАЛИЗИРОВАННОЙ СРЕДЫ ТРЕХМЕРНОЙ ДИНАМИЧЕСКОЙ ВИЗУАЛИЗАЦИИ, дипломная работа: стр. 80, рис. 14, библ. 14 назв.

Ключевые слова: ДИНАМИЧЕСКАЯ ВИЗУАЛИЗАЦИЯ, МОДЕЛЬ МИКРОВЗРЫВА, ПОСАДКА САМОЛЕТА.

Объект исследования – динамическая визуализация.

Цель работы – создать среду трехмерной динамической визуализации.

В процессе работы были изучены особенности решения задачи посадки самолета в условиях ветровых помех, разработана визуальная модель микровзрыва.

В результате работы создана среда трехмерной динамической визуализации.

Оглавление

[ВВЕДЕНИЕ 4](#_Toc295054412)

[1 ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ 5](#_Toc295054413)

[1.1 Постановка задачи 5](#_Toc295054414)

[1.2 Актуальность задачи 6](#_Toc295054415)

[1.3 Посадка самолета в условиях ветровых помех 8](#_Toc295054416)

[2 ОБЗОР БИБЛИОТЕКИ OPENGL 10](#_Toc295054417)

[2.1 Общие сведения 10](#_Toc295054418)

[2.2 Графические примитивы 11](#_Toc295054419)

[2.3 Векторы нормали 13](#_Toc295054420)

[2.4 Модельные преобразования 14](#_Toc295054421)

[2.5 Видовые трансформации 16](#_Toc295054422)

[2.6 Списки отображения 18](#_Toc295054423)

[2.7 Освещение 20](#_Toc295054424)

[3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ 22](#_Toc295054425)

[3.1 Основные требования 22](#_Toc295054426)

[3.2 Структура программы 23](#_Toc295054427)

[3.3 Пример файла настроек 33](#_Toc295054428)

[3.4 Пример файла данных 35](#_Toc295054429)

[3.5 Интерфейс 36](#_Toc295054430)

[3.6 Работа программы и интерпретация результата 39](#_Toc295054431)

[ЗАКЛЮЧЕНИЕ 44](#_Toc295054432)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 45](#_Toc295054433)

[ПРИЛОЖЕНИЕ А – Программный код 47](#_Toc295054434)

# ВВЕДЕНИЕ

Трехмерная графика применяется во многих современных компьютерных приложениях: от интерактивных игр до визуализации данных в науке, медицине или бизнесе. Также не стоит забывать и о том, что мощная трехмерная графика вошла в фильмы и технические и образовательные публикации.

Трехмерная графика реального времени связана с анимацией и интерактивным взаимодействием с пользователем. Одной из первых сфер применения трехмерной графики реального времени были военные авиатренажеры. Применение трехмерной графики на персональных компьютерах практически безгранично. Пожалуй, самой популярной сферой сегодня являются компьютерные игры. Трудно назвать современную игру, которая бы не требовала графической карты 3D. Кроме того, трехмерный мир всегда был популярен в области научной визуализации и инженерных приложениях, но взрывоподобное развитие дешевого аппаратного обеспечения с возможностью трехмерной визуализации стимулировало развитие этих областей как никогда ранее. Были затронуты даже современные графические пользовательские интерфейсы, которые стали развиваться с учетом преимуществ нового аппаратного обеспечения.

# 1 ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ

## 1.1 Постановка задачи

Целью моей работы является создание специализированной среды трехмерной динамической визуализации для изучения решений задач оптимального управления и дифференциальных игр, в частности задачи посадки самолета в условиях ветровых помех. Результаты решения данной задачи были получены в ходе исследований сектора 3 отдела динамических систем ИММ УрО РАН.

Для достижения этой цели работа выполнялась в несколько этапов:

1. Изучение общих сведений о библиотеке OpenGL;
2. Изучение особенностей визуального отображения результата решения задачи о посадке самолета в зоне ветровых возмущений;
3. Разработка визуальной модели микровзрыва;
4. Программная реализация приложения.

## 1.2 Актуальность задачи

Важной областью приложения математической теории управления и дифференциальных игр является создание аппаратно-программных комплексов автоматического управления самолетом (автопилотов). Однако, если автопилоты для движения воздушного судна по эшелону вдали от поверхности земли существуют уже около 50 лет (с конца 1950-х годов), то автопилоты для взлета/посадки самолетов разрабатываются до сих пор. Активно в этом направлении работали американский математик-инженер A.Miele [1-3] и немецкий R.Bulirsch [4,5]. В России математический аппарат теории управления и дифференциальных игр к управлению самолетом в 1980-е годы стал применять В.М.Кейн [6,7].

Наибольшую опасность для движения самолета представляют резкие порывы ветра, вследствие которых резко изменяется его направление и, как следствие, подъемная сила. При движении вдали от поверхности земли это приводит к проседанию воздушного судна, к «воздушным ямам» – явление неприятное, но не критичное. Если же движение происходить на взлете или посадки, подобные незапланированные маневры могут привести к катастрофе.

Наиболее частые причины резкой смена направления воздушного потока – это сдвиги ветра и микровзрывы. Сдвиг ветра (в англоязычной литературе – windshear) – атмосферное возмущение, при котором скорость ветра резко меняется по значению и/или направлению вдоль траектории полёта летательного аппарата. Возникают в нижних слоях атмосферы (на высотах до 100 м.) при прохождении грозовых и термальных атмосферных фронтов и чаще возникают в районах со сложным рельефом местности. Микровзрыв ветра (в англоязычной литературе – microburst) образуется за счет нисходящего потока воздуха, который ударяется о поверхность земли и растекается затем с образованием вихря. При прохождении самолетом на взлете или посадке зоны микровзрыва происходит резкое изменение ветра с встречного на попутный.

Как правило, временной промежуток смены воздушных потоков весьма мал (до 10 секунд), поэтому пилот зачастую не справляется с управлением. Вследствие этого разумно применять методы автоматического управления. Однако традиционные методы управления (например, пропорциональная навигация) являются недостаточно «энергичными» для парирования порывов ветра, отсюда разумным является использование методов экстремального управления, предлагаемых математической теории управления и дифференциальных игр.

В Институте математики и механики УрО РАН с середины 1980-х годов проводятся исследования в этом направлении [8,9]. Совместно со специалистами из Академии Гражданской авиации (г.Ленинград – Санкт-Петербург) разработана модель движения транспортного самолета на посадке, проведена ее линеаризация вдоль номинальной глиссады. Полученная система рассматривается как линейная дифференциальная игра, в которой игроками являются пилот самолета, осуществляющий комплекс полезных управлений, и ветер, противодействующий пилоту. К этой задаче применены численные методы. Разработанные в ИММ УрО РАН, получены оптимальные стратегии обоих игроков (методы построения наилучшего полезного управления и наихудшей ветровой помехи).

Слабым местом этой работы является то, что все вычисления ведутся в терминах некоторых «математических» объектов, а как следствие, полученные результаты тяжело могут быть оценены с точки зрения качества управления неспециалистами в области дифференциальных игр. Актуальной является разработка визуализации движения самолета под воздействием оптимального управления при тех или иных ветровых помехах.

## 1.3 Посадка самолета в условиях ветровых помех

Движение самолета на посадке рассматривается от момента прохождения высоты 600 м. на расстоянии около 8 км. от торца взлетно-посадочной полосы (ВПП) до момента прохождения им торца ВПП. Номинальное движение самолета представляет собой равномерное движение по прямолинейной глиссаде снижения, проиллюстрировано на рисунке 1. Высота глиссады над торцом ВПП составляет 15 м. Относительно ветра считаются заданными номинальные (средние) значения компонент скорости ветра и, возможно, «разумные» отклонения компонент от этих значений. Цель управления – обеспечить удовлетворительную траекторию посадки для реализации ветрового возмущения.

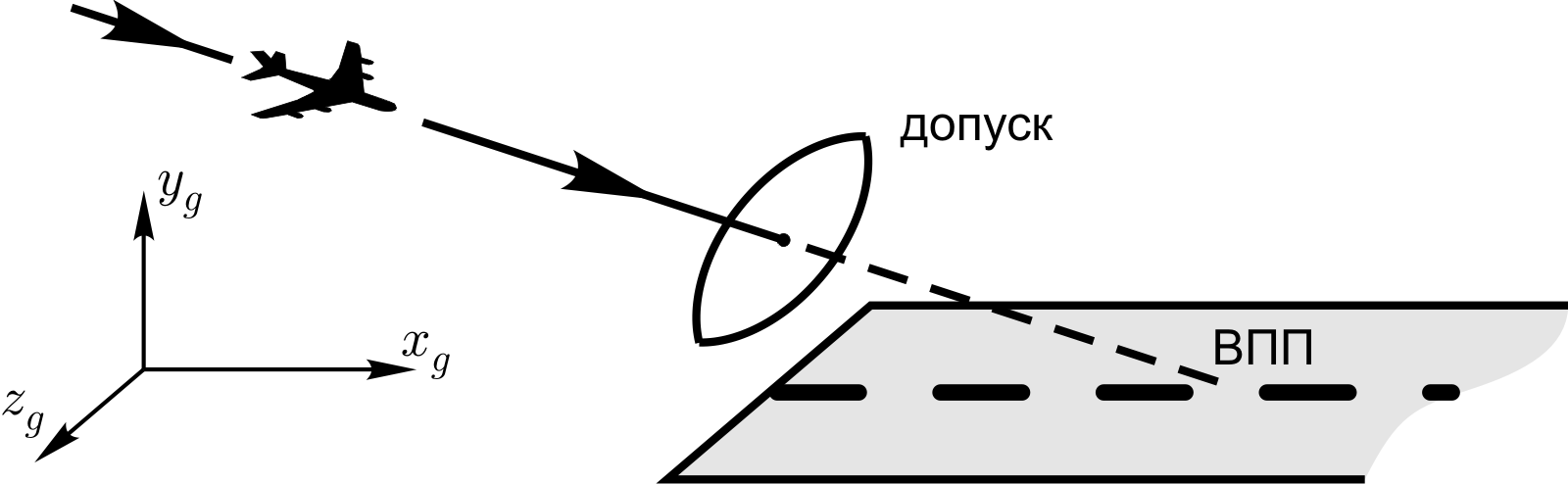


Рисунок 1 - Движение самолета при посадке.

Движение самолета описывается системой нелинейных дифференциальных уравнений 12-го порядка. Управление самолетом осуществляется за счет изменения силы тяги, рулей высоты и направления, отклонением элеронов. Ветровое возмущение, задаваемое по трем геометрическим осям, влияет на компоненты воздушной скорости самолета, которая обуславливает величину подъемной силы.

Учитываются инерционности в механизмах управления, что дает еще 4 уравнения в систему, так что полный набор уравнений динамики системы имеет 16-й порядок.

Эта система после линеаризации вдоль номинальной глиссады распадается на практически независимые подсистемы бокового и вертикального движений, которые можно рассматривать как две линейные дифференциальные игры. В рамках этих игр формируется оптимальная стратегия полезного векторного управления.

После этого проводится моделирование исходной нелинейной системы с приложением к ней полученных полезных управлений и различных ветровых возмущений. В частности, рассматривается модель микровзрыва ветра, взятая из работы [10] и разобранная в отчете [11].

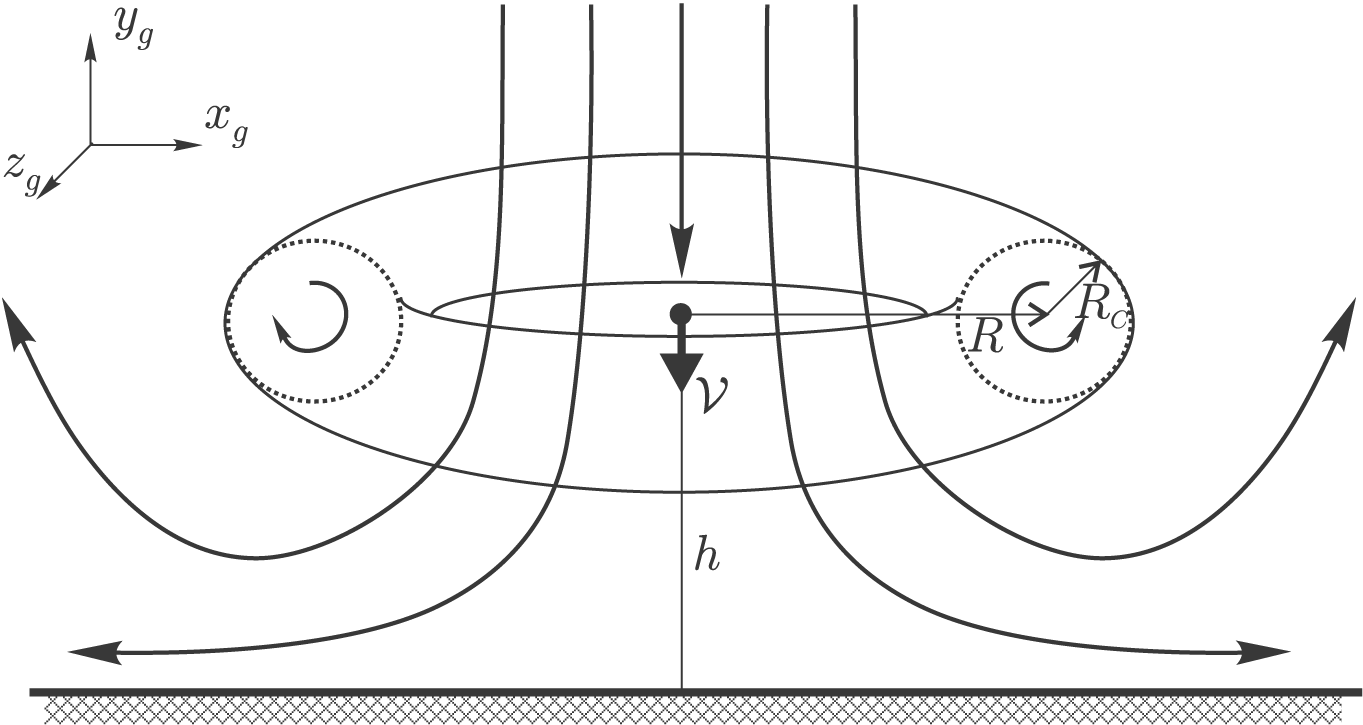


Рисунок 2 - Линии тока микровзрыва.

Микровзрыв можно представить себе как тор, в который проходит нисходящий воздушный поток и вокруг которого «закручиваются» линии тока, модель изображена на рисунке 2. Микровзрыв задается расположением своего центра (центра тора – горизонтальные координаты и высота), параметрами кольца (радиус центральной линии и радиус кольца) и средней скоростью потока (в центре микровзрыва). По этим параметрам с использованием формул из [10,11] в любой точке может быть вычислена скорость воздушного потока (величина и направление).

# 2 ОБЗОР БИБЛИОТЕКИ OPENGL

## 2.1 Общие сведения

Рассмотрим основной инструментарий, использованный при решении поставленной задачи.

OpenGL - Open Graphics Library, открытая графическая библиотека, которая реализует программный интерфейс к графическому оборудованию. Этот интерфейс содержит порядка 250 отдельных команд, которые используются для задания объектов и операций, необходимых для создания интерактивных приложений трехмерной графики.

OpenGL разработан как аппаратно-независимый интерфейс для работы на различных аппаратных платформах. Поэтому в OpenGL не включены команды для управления окнами и для организации пользовательского ввода. Вся такая работа ведется через операционную систему. Аналогично, OpenGL не имеет высокоуровневых команд для описания трехмерных сложных моделей: автомобилей, частей человеческого тела, самолетов или молекул. Задача OpenGL – помочь программисту построить свою модель из небольшого набора графических примитивов: точек, линий и многоугольников.

Высокоуровневые же средства предоставляются библиотеками, являющимися надстройками над OpenGL. Для моделирования кривых и поверхностей предназначена библиотека GLU, имеющая множество инструментов, таких как вычислители и NURBS. Библиотека GLU является стандартной составляющей любой реализации OpenGL.

## 2.2 Графические примитивы

Все графические примитивы задаются с помощью вершин (vertices) – координат, определяющих точки, концы сегментов линий и углы многоугольников.

Графические примитивы: точки, линии, многоугольники.

Вершина задается с помощью функции glVertex3d (x, y, z).

Для каждого типа примитива набор вершин заключается в операторные скобки – команды glBegin() и glEnd(). Аргумент функции glBegin() определяет тип описываемого вершинами графического примитива.

В роли аргумента могут выступать следующие значения:

* GL\_POINTS – отдельные точки;
* GL\_LINES – каждая пара вершин задает отрезок;
* GL\_LINES\_STRIP – серия соединенных отрезков;
* GL\_LINE\_LOOP – серия соединенных отрезков, последняя точка соединяется с первой;
* GL\_TRIANGLES – тройка вершин задает треугольник;
* GL\_TRIANGLE\_STRIP – рисуются треугольники с общей стороной;
* GL\_TRIANGLE\_FAN – рисуются треугольники с общей стороной, другой способ соединения сторон;
* GL\_QUADS – четыре вершины задают четырехугольник;
* GL\_QUAD\_STRIP – четырехугольники с общей стороной;
* GL\_POLYGON – рисуется простой правильный многоугольник.

Для изменения размера отображаемой точки используется команда glPointSize(), аргументом которой является нужный размер в пикселах.

Функция glLineWidth() задает толщину отображаемой линии.

По умолчанию многоугольники рисуются со сплошной закраской, но их можно рисовать в виде контура или в виде точек в вершинах. Закраска многоугольника может быть сплошной или фактурной по некоторому заданному образцу. Если соседние многоугольники имеют общие границы или вершины, пикселы на границах или вершинах рисуются всего один раз – они принадлежат только одному многоугольнику – с той целью, чтобы у частично прозрачных многоугольников границы не рисовались дважды.

Многоугольники имеют две стороны – лицевую и оборотную, которые могут быть отрисованы различно, в зависимости от того, какая сторона сейчас видна. Это позволяет получать такой разрез сплошных объектов, где ясно различаются внутренние и внешние части. По умолчанию обе поверхности рисуются одинаково. Для изменения режима по умолчанию или рисования только контура или вершин используется функция glPolygonMode(face, mode). Первый аргумент определяет сторону отрисовки: лицевую, внутреннюю или обе одновременно. Второй аргумент задает режим отрисовки: точками, линиями или заполнением.

## 2.3 Векторы нормали

Вектор нормали – перпендикулярный к поверхности вектор. Для плоских поверхностей это направление одинаково для всех точек, но для кривых поверхностей оно может быть различным. OpenGL позволяет определять нормали для каждой вершины или каждого многоугольника. Вершины одного многоугольника могут иметь одинаковые или различные нормали.

Векторы нормалей для объектов определяют ориентацию их поверхностей в пространстве – в частности, относительно источников света. Эти векторы используются OpenGL для расчета, сколько света получает объект в вершинах.

Функция glNormal() используется для установки текущей нормали, определяемой значением аргумента. Последующие вызовы glVertex() связывают определяемые вершины с текущей нормалью.

В любой точке поверхности есть два перпендикулярных вектора, направленных в противоположных направлениях. Нормалью считается тот, что ориентирован вовне от моделируемой поверхности. Векторы нормали нормируются каждый раз при выполнении модельных преобразований – при повороте и параллельном переносе. Можно автоматически нормализовать векторы нормали с помощью команды glEnable(GL\_NORMALIZE).

## 2.4 Модельные преобразования

В OpenGL определены три команды модельных преобразований: glTranslate(), glRotate(), glScale(). Эти функции трансформируют объект (или систему координат, если думать об изменении системы координат, а не объекта) посредством сдвига, вращения и масштабирования.

Перенос.

void glTranslate{fd} (TYPE x, TYPE y, TYPE z);

Умножает текущую матрицу на матрицу, передвигающую (переносящую) объект на расстояния x, y, z, переданные в качестве аргументов команды, по соответствующим осям (или перемещает локальную координатную систему на те же расстояния).

Поворот.

void glRotate{fd} (TYPE angle, TYPE x, TYPE y, TYPE z);

Умножает текущую матрицу на матрицу, которая поворачивает объект (или локальную координатную систему) в направлении против часовой стрелки вокруг луча из начала координат, проходящего через точку (x, y, z). Параметр angle задает угол поворота в градусах.

Масштабирование.

void glScale{fd} (TYPE x, TYPE y, TYPE z);

Умножает текущую матрицу на матрицу, которая растягивает, сжимает или отражает объект вдоль координатный осей. Каждая x -, y- и z- координата каждой точки объекта будет умножена на соответствующий аргумент x, y или z команды glScale\*(). При рассмотрении преобразования с точки зрения локальной координатной системы, оси этой системы растягиваются, сжимаются или отражаются с учетом факторов x, y и z, и ассоциированный с этот системой объект меняется вместе с ней.

glScale\*() – это единственная из трех команд модельных преобразований, изменяющая размер объекта: масштабирование с величинами более 1.0 растягивает объект, использование величин меньше 1.0 сжимает его. Масштабирование с величиной -1.0 отражает объект относительно оси или осей. Единичными аргументами (то есть аргументами, не имеющими эффекта) являются (1.0, 1.0, 1.0). Следует ограничивать использование glScale\*() теми случаями, когда это действительно необходимо. Использование glScale\*() снижает быстродействие расчетов освещенности, так как вектора нормалей должны быть нормализованы заново после преобразования.

## 2.5 Видовые трансформации

Видовое преобразование изменяет позицию и ориентацию точки обзора. Модельное преобразование, поворачивающее объекты сцены против часовой стрелки аналогично видовому преобразованию, которое поворачивает камеру по часовой стрелке.

OpenGl позволяет производить видовые преобразования несколькими способами.

1. Использование одной или нескольких команд модельных преобразований (glTranslate\*() или glRotate\*()). Можно представить эффект этих преобразований как перемещении камеры, или как перемещение всех объектов сцены относительно стационарной камеры.
2. Использование команды библиотеки утилит gluLookAt() для определения точки и направления обзора. Эта команда инкапсулирует в себе серию поворотов и переносов.
3. Создание своей собственной функции, инкапсулирующей повороты и переносы. В некоторых случаях такая функция может понадобиться для того, чтобы можно было задавать видовую трансформацию каким-либо специфическим путем. Например, для определения преобразования в терминах полярных координат для камеры, вращающейся вокруг объекта или в терминах углов наклона самолета в полете.

Команда gluLookAt().

Часто программисты конструируют сцену в районе начала координат или в некотором другом месте, а затем хотят посмотреть на нее с определенной точки обзора для получения лучшего вида. Команда из библиотеки утилит gluLookAt() разработана как раз для подобных целей. Она принимает три набора аргументов, которые задают точку наблюдения, прицельную точку (точку, на которую направлена камера) и направление, которое следует считать верхним. Прицельная точка, как правило, находится где-то в середине сцены. Несколько сложнее задать верный вектор верхнего направления. Например, для трехмерного симулятора полетов, верхним является направление перпендикулярное крыльям самолета.  
void gluLookAt (GLdouble eyex, GLdouble eyey, GLdouble eyez, GLdouble centerx, GLdouble centery, GLdouble centerz, GLdouble upx, GLdouble upy, GLdouble upz);

Функция задает видовую матрицу и умножает на нее текущую матрицу. Выбранная точка обзора задается аргументами eyex, eyey и eyez. Аргументы centerx, centery и centerz задают любую точку на линии обзора, но обычно они задают точку где-то в середине обозреваемой сцены. Аргументы upz, upyи upz определяют, какое направление считается верхним (то есть направление от дна до вершины объема видимости).  
По умолчанию камера находится в начале координат, направлена вдоль отрицательного направления оси z, а вектор верхнего направления совпадает с положительным направлением оси y.

GluLookAt() является частью библиотеки утилит, а не базовой командой OpenGL. Это произошло не потому, что gluLookAt() бесполезна, а потому, что она инкапсулирует несколько базовых команд OpenGL, а именно glTranslate\*() и glRotate\*(). Чтобы отменить действия gluLookAt(), необходимо поместить камеру в начало координат и установить направление обзора совпадающим с отрицательным направлением оси z (то есть привести видовое преобразование к ситуации по умолчанию).

## 2.6 Списки отображения

Список отображения (display list) —список команд или список вывода — это группа команд OpenGL, сохраненных для последующего выполнения. Когда список отображения вызывается на выполнение, команды обрабатываются в порядке их появления в списке. Большинство команд OpenGL может сохраняться в списке или обрабатываться в режиме непосредственного выполнения.

Списки отображения позволяют улучшить производительность за счет использования запомненных для последующего выполнения команд OpenGL. Максимум отдачи дает кэширование команд в списке, если вы планируете перерисовывать один и тот же объект несколько раз или существует набор изменяемых состояний, которые нужно воспроизводить многократно. С помощью списков отображения можно определить геометрию и/или изменяемое состояние однажды и повторить их столько раз, сколько нужно.

Случай, когда список отображения используется для единичного запоминания, можно проиллюстрировать на примере рисования трехколесного велосипеда. Его два задних колеса имеют одинаковый размер, но требуют отдельной прорисовки, так как смещены друг относительно друга. Переднее колесо больше по диаметру и также находится в индивидуальной позиции. Эффективный способ передать вращение колес заключается в запоминании геометрии одного колеса в списке отображения и последующем трехкратном вызове этого списка. Необходимо построить соответствующую модельно-видовую матрицу перед каждым вызовом списка для вычисления корректных размеров и положения каждого колеса.

С целью поднятия производительности команды списков отображения OpenGL кэшируются, причем так, что однажды созданный список отображения не может быть модифицирован. В противном случае выигрыш в производительности был бы сведен к минимуму за счет поиска по списку и лишних операций управления памятью. Если бы части списка отображения были изменяемыми, распределение и высвобождение памяти привело бы к дефрагментации памяти. Любые модификации, внесенные реализацией OpenGL в порядок команд списка отображения с целью повышения эффективности рендеринга, все равно потребовали бы отката.

Вместе с тем имеются некоторые издержки, обусловленные процессом перехода к списку отображения. Если такой список невелик, эти издержки способны свести на нет все преимущества списочной организации. Наиболее предпочтительные места для оптимизации перечислены ниже:

* Матричные операции. Большинство матричных операций требуют вычисления обратных матриц. В зависимости от реализации OpenGL, обе матрицы — расчетная и обратная — сохраняются или не сохраняются в списке отображения.
* Растровые двоичные карты и изображения. При компиляции списка отображения OpenGL может преобразовать данные в представление, предпочтительное для оборудования. Это оказывает существенный эффект на скорость рисования растровых символов, поскольку символьные строки обычно состоят из совокупности небольших образов (двоичных карт).
* Источники света, свойства материала и модели распространения света. Когда отрисовывается сцена со сложными условиями освещения, можно изменять материалы для каждого элемента сцены. Задание материалов само по себе медленная операция, поскольку оно предполагает множество вычислений. Если возможно поместить описание свойств материалов в список отображения, эти вычисления не будут выполняться всякий раз при переключении между материалами, так как сохранению подлежат лишь результаты вычислений; в итоге рендеринг освещенной сцены может ускориться.

## 2.7 Освещение

OpenGL рассчитывает свет и освещение так, как будто свет может быть разделен на красный, зеленый и синий компоненты. Таким образом, источник света характеризуется количеством красного, зеленого и синего света, которое он излучает, а материал поверхности характеризуется долями красного, зеленого и синего компонентов, которые он отражает в различных направлениях. Уравнения освещенности в OpenGL являются всего лишь аппроксимациями, но зато они работают достаточно хорошо и могут быть вычислены относительно быстро. В модели освещения OpenGL свет исходит от нескольких источников, которые могут включаться и выключаться индивидуально. Часть света обычно исходит из какого-либо определенного направления или позиции, часть распределена по всей сцене. Например, если вы включите лампочку в комнате, большая часть света будет исходить от нее, но часть света падает на поверхности предметов в комнате после того, как он отразился от одной, двух, трех или более стен. Считается, что этот многократно отраженный свет (называемый фоновым светом) распределен настолько сильно, что не существует никакого способа определить его исходное направление, однако он исчезает при выключении определенного источника света. Наконец, в сцене может также присутствовать общий фоновый свет, у которого нет никакого конкретного источника, как будто он был отражен столько раз и распределен так сильно, что его оригинальный источник установить невозможно. В модели OpenGL эффект от источника света присутствует только тогда, когда есть поверхности поглощающие или отражающие свет. Считается, что каждая поверхность состоит из материала с несколькими свойствами. Материал может излучать свой собственный свет, он может распределять некоторое количество входящего света во всех направлениях, также он может отражать часть света в определенном направлении.

В модели освещения OpenGL предполагается, что освещение может быть разделено на 4 компонента: фоновое (ambient), диффузное (diffuse), зеркальное (specular) и исходящее (эмиссионное – emissive). Все 4 компонента рассчитываются независимо и только затем суммируются.

Фоновое излучение – это свет, который настолько распределен средой (предметами, стенами и так далее), что его направление определить невозможно. Когда фоновый свет падает на поверхность, он одинаково распределяется во всех направлениях.

Диффузный компонент – это свет, идущий из одного направления, таким образом, он выглядит ярче, если падает на поверхность под прямым углом, и выглядит тусклым, если касается ее всего лишь вскользь. Однако, когда он падает на поверхность, он распределяется одинаково во всех направлениях, то есть его яркость одинакова вне зависимости от того, с какой стороны вы смотрите на поверхность.

Зеркальный свет исходит из определенного направления и отражается от поверхности в определенном направлении.

Помимо фонового, диффузного и зеркального цветов, материалы могут также иметь исходящий цвет, имитирующий свет, исходящий от самого объекта. В модели освещения OpenGL исходящий свет поверхности добавляет объекту интенсивности, но на него не влияют никакие источники света, и он не производит дополнительного света для сцены в целом.  
Хотя источник света излучает единое распределение частот, фоновый, диффузный и зеркальный компоненты могут быть различны. OpenGL позволяет устанавливать значения красного, зеленого и синего независимо для каждого компонента света.

# 3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

## 3.1 Основные требования

Для функционирования программы необходимо наличие следующих файлов в корневой директории программы: Tao.FreeGlut.dll, Tao.OpenGl.dll, Tao.Platform.Windows.dll, freeglut.dll.

Для улучшения взаимодействия пользователя и среды был использован файл с настройками по умолчанию - applicationSettings.xml. Это позволило сократить время работы пользователя по конфигурированию программы.

Для отображения модели самолета необходим файл с трехмерной моделью. По умолчанию был выбран самолет Ту-154(файл tu154.3DS). Исходные характеристики этого самолета были использованы для решения задачи посадки на ВПП в зоне ветровых возмущений. Формат .3DS довольно популярен, его можно создать, например, в 3d Studio Max, а также можно импортировать и экспортировать многими другими программами.

Загрузка информации о посадке самолета может совершаться из произвольного файла с расширением .xml, содержащего сериализованные данные. Имя файла можно указать в файле настроек программы, и он будет использован при последующих запусках.

Для начала работы приложения необходимо запустить файл «Посадка самолета и микровзрыв.exe».

Требования к операционной системе: Windows XP sp3 и выше, Microsoft .NET Framework 2.0 и выше.

## 3.2 Структура программы

Программный код был написан на языке C#. Были реализованы различные классы и структуры, изображенные на рисунке 3.

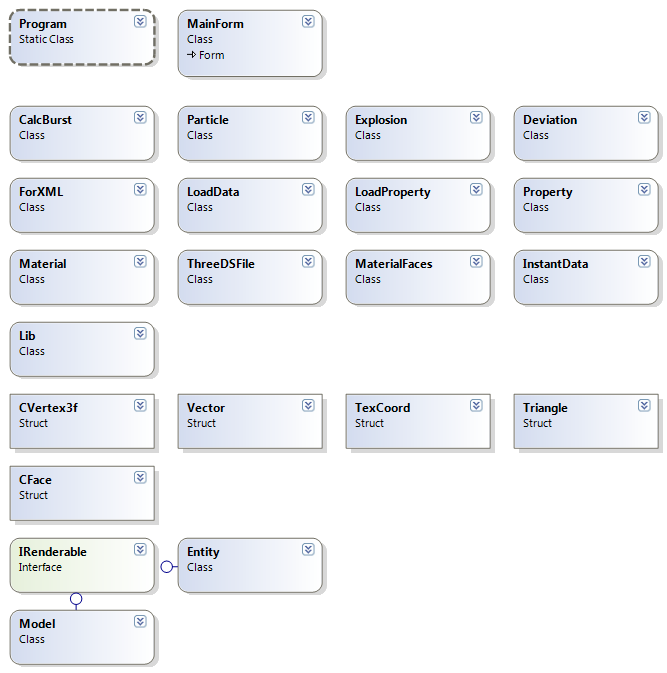


Рисунок 3 – Структура программы.

**Класс Program**

Является статическим классом, определяющим точку входа в приложение. Определяет свойства на уровне всего приложения. Содержит **static void Main()**, в которой создается экземпляр класса главного окна приложения. После создания вызывается метод

Application.Run(MF), который запускает главное окно программы, где MF – экземпляр главного окна.

**Класс MainForm**

Класс, определяющий главное окно программы, самый существенный класс из всех, поскольку служит главным «узлом», в котором обрабатываются все взаимодействия вспомогательных классов. Вот описание некоторых ключевых полей и функций этого класса:

private float[] whiteLight, sourceLight, lightPos – массивы, которые определяют настройки яркости и позицию источника освещения.

private float[] shadowM – матрица преобразования для реализации плоской тени.

private LoadData load – экземпляр класса-загрузчика данных о перемещении самолета по сцене.

static ThreeDSFile file – экземпляр класса-загрузчика трехмерной модели любого объекта из файла с расширением .3ds.

static Model model – хранит данные о модели самолета. Функционал программы позволяет пользователю самому выбирать необходимую модель, которая соответствует задаче.

private Deviation dev – экземпляр класса, который вычисляет и рисует отклонения самолета от глиссады.

private LoadProperty loadProperty – осуществляет загрузку настроек из xml-файла.

private Property property – хранит в себе основные настройки приложения.

private Explosion burst – используется для рендеринга модели микровзрыва.

public MainForm()

Конструктор главного окна задает первоначальные настройки приложения. Вычисляются размеры клиентской области, устанавливаются необходимые флаги, объявляются все необходимые переменные. Происходит инициализация области, в которой будет происходить отрисовка сцены и объектов.

private void Form1\_Load(object sender, EventArgs e)

Функция вызывается непосредственно перед отрисовкой главной формы, содержит в себе единовременное определение конеткста устройства, производит настройку всех необходимых конвейеров. Инициализируется библиотека Glut. Здесь объектно-видовая и модельная матрицы очищаются, становясь единичными. Включается настройка проверка глубины, необходимая для корректного отображения взаимного расположения объектов на сцене. Также задается матрица перспективной проекции. Данный режим позволяется более реалистично изображать модели, визуально искажая их размеры. Отдаленные объекты имеют меньшие размеры, чем те, которые расположенные рядом с камерой. В Form1\_Load происходит настройка всех параметров освещения, положения источника света. Вызывает метод loadModel(), Begin(). Задается матрица проекции на плоскость поверхности сцены. Инициализируется объект, отвечающий за модель микровзрыва и вычисление его параметров.

private void Draw()

Функция отрисовки объектов. При каждой перерисовке формы вызывается этот метод.

Последовательно отрисовываются все объекты сцены, начиная с ландшафта и заканчивая полупрозрачными плоскостями. Порядок имеет значение, потому что может происходить смешивание цвета материала объекта или некорректно вычисляться невидимые области.

private void Begin()

Функция, в которой происходит первоначальная загрузка данных о перемещении самолета. Необходима для того, чтобы минимизировать количество действий пользователя. Функция будет обращаться к файлу, путь к которому задан в настройках приложения. Происходит вычисление параметров, необходимых для отрисовки отклонений от номинальной траектории. Создается экземпляр класса Deviation. Задается частота обновления кадров. Также из исходных данных происходит вычисление размеров сцены.

private void loadModel()

Метод, в котором загружается модель самолета.

private void loadToolStripMenuItem\_Click

Метод пункта меню, который позволяет пользователю выбирать файлы с данными решения задачи оптимального управления, в данном случае задачи посадки самолета.

void DrawGround(float x, float y, float len, float width)

Функция отрисовки поверхности сцены.

void DrawRect(float x, float y, float len, float width)

Функция отрисовки ВПП.

private void ShowDisplay\_MouseMove(object sender, MouseEventArgs e)

Функция реализует возможность вращения сцены с помощью манипулятора (мышь). Камера перемещается по сфере, в центре которой либо самолет, либо торец ВПП.

Класс MainForm содержит также стандартные процедуры, отвечающие заданным событиям, таким как движения мыши, нажатие кнопок мыши. Каждое такое событие соответствующим образом обрабатывается.

**Класс CalcBurst**

Класс реализует алгоритм вычисления скорости в заданной точке микровызрыва.

float V0, H0, R0

V0 - скорость в центральной части микровзрыва, [м/с],

H0 - высота центральной части, [м],

R0 - радиус вихря, [м].

public float Wx, Wz - разложение радиальной скорости параллельно и перпендикулярно оси ВПП.

public float Wy - вертикальная скорость ветра в искомой точке.

public float WR - радиальная скорость ветра в искомой точке.

float Dx, Dy, Dz – координаты центра микровзрыва.

Float CIR - циркуляционная сила вихря.

public CalcBurst(float V0\_, float H0\_, float R0\_, float Dx\_, float Dy\_, float Dz\_)

Конструктор класса.

public float Sign(float A, float B) – функция, вычисляющая знак А относительно B.

public void VV(float Xg, float Yg, float Zg) – функция вычисления составляющих WX, WY, WZ.

public float STR(float Y\_, float R\_) - метод вычисления функции потока STR.

public void WAP(float Y\_, float R\_, out float WR\_, out float WY\_) - вычисление составляющих WR, WY на основе функции STP.

**Class Particle**

Класс, описывающий частицу. В этом классе частица создается, вычисляются ее основные параметры.

public Particle() конструктор класса.

public void Calculate() функция вычисления координат каждой частицы, составляющей микровзрыв. Создается экземпляр класса **CalcBurst.** Вычисляются скорости.

**Class Explosion**

Класс, реализующий модель микровзрыва. Создается система частиц, при первом вызове в память видеокарты передается описание, как отрисовывать частицу. Это повышает скорость выполнения операции отображения всей системы. Для каждой частицы выбирается случайная точка из области микровзрыва, из которой будет высчитываться координаты всех положений частицы.

private float[] position – координаты микровзрыва.

private int MAX\_PARTICLES = 50000 максимальное количество частиц.

private int \_particles\_now – текущее количество частиц, величина передается из файла настроек.

private bool isStart – индикатор. Показывает, произошло ли вычисление параметров микровзрыва.

private Particle[] ParticleArray – массив частиц. Хранит в себе все данные о частицах.

private bool isDisplayList – индикатор. Определяет, создан ли список отображения частицы. Необходим для улучшения быстродействия.

public Explosion(float x, float y, float z, float power, int particle\_count) – конструктор класса.

public void SetNewPosition(float x, float y, float z) – функция. Вызывается для того, чтобы задать новую позицию микровзрыва.

private void CreateDisplayList() – создание списка отображения.

public void Boooom() - метод, в котором происходит создание массива частиц, которые составляют визуальную модель микровзрыва.

Вызывается процедура CreateNewParticle(), включается индикатор начала взрыва.

public Particle CreateNewParticle() – процедура создания частицы со случайными координатами.

public void Render() – метод отрисовки микровзрыва. Происходит вызов списка отображения для каждой частицы.

**Class Deviation**

Класс описывает глиссаду, отклонения, плоскости. Также отвечает за их отрисовку. Экземпляр класса создается при загрузке программы.

public Deviation() – конструктор класса.

public void Calculate() – функция вычисляет основные параметры отклонения от номинальной траектории, глиссадной и курсовой плоскостей, координаты номинальной траектории.

public void Render(bool blGLide, bool blDev, bool blScale, float transparency) – метод, отвечающий за отрисовку элементов, описанных выше. На вход получает индикаторы включения дополнительных опций.

**Class InstantData**

Класс описывает составной элемент класса ForXML.

public double t - момент времени кадра.

public double x, public double y, public double z - координаты центра масс самолета. Начало координат расположено в центре торца взлетно-посадочной полосы (ВПП). x - продольная координата, y - высота, z - боковое отклонение от осевой линии ВПП.

public double vx, public double vy, public double vz - скорости по соответствующим осям координат.

public double theta, public double psi, public double gamma – угловые координаты (в радианах): theta - тангаж, угол поворота вокруг оси z, psi - рыскание/курс, угол поворота вокруг оси y, gamma - крен, угол поворота вокруг оси x. Положительное направление – против часовой стрелки, если смотреть с положительного направления соответствующей оси.

public double wtheta, public double wpsi, public double wgamma - соответствующие угловые скорости.

**Class ForXML**

Класс описывает параметры, которые задаются для каждого момента времени посадки самолета. Используется для десериализации данных из файла.

Сериализация - процесс перевода какой-либо структуры данных в последовательность битов. Обратной к операции сериализации является операция десериализации восстановление начального состояния структуры данных из битовой последовательности.

**Class LoadData**

Класс описывает загрузку данных из файла. Файл данных формируется во время решения задачи о посадки самолета с различными начальными данными.

public ForXML forXML – экземпляр класса ForXML, используется для хранения в памяти данных.

private void ReadPO (string filename) - создаются и инициализируются переменные, необходимые для десериализации. На входе получает имя файла, из которого будут извлекаться данные. Данный параметр может быть получен либо из файла с настройками приложения, либо имя файла пользователь укажет сам. Вызывается из MainForm.Begin().

**Class Property**

Класс описывает структуру данных, в которой хранятся параметры анимации.

**Class LoadProperty**

Класс описывает загрузку настроек из файла.

public Property Load() – результатом выполнения данного метода является экземпляр класса Property, в котором хранятся десереализованные настройки.

Вызывается в MainForm. Form1\_Load().

public void WriteProperty(Property pr) – функция создания файла с настройками, если он отсутствует в корневой директории программы.

**Class Lib**

В данном классе собраны вспомогательные структуры и методы.

struct CVertex3f – структура вершины, характеризуется тремя координатами.

struct Triangle – структура треугольного полигона, хранит в себе номера вершин из списка.

struct CFace – структура грани, хранит в себе номера вершин из списка.

struct TexCoord – структура текстурных координат.

struct Vector – структура вектора в трехмерном пространстве, реализованы перегруженные методы сложения векторов, умножения на скаляр.

public void gltGetPlaneEquation() – функции вычисления матрицы тени. На входе получает координаты трех точек плоскости, позицию источника света.

public void gltGetNormalVector() – функция вычисления вектора нормали к плоскости.

public void gltSubtractVectors() – функция нахождения разности двух векторов.

public void gltVectorCrossProduct() – используется для нахождения векторного произведения.

public void gltNormalizeVector() – функция нормализации вектора.

**Class MaterialFaces**

Класс создает соответствие грани и ее материала.

**Class Material**

Класс используется для наложения текстуры на грань модели.

public void BindTexture() – функция наложения текстуры. Если загружаемая модель содержит в своем файле описания текстуры, то вызывается этот метод.

**Class ThreeDSFile**

Класс описывает загрузку трехмерной модели из файла .3ds.

Реализует вложенный класс ThreeDSChunk(), который удобно использовать при дальнейшем чтении данных из файла.

void ProcessChunk() – чтение Chunk.

void ProcessMaterialChunk - чтение Chunk, хранящего информацию о материалах.

void ProcessTexMapChunk - чтение Chunk с данными о текстурной карте.

void ProcessColorChunk - чтение Chunk с цветом модели.

Triangle[] ReadIndices – функция загрузки треугольных полигонов. Использует структуру Triangle.

Vector[] ReadVertices() – функция загрузки вершин. Использует структуру Vector.

**Interface IRenderable**

Интерфейс, который описывает метод Render().

**Class Entity**

Класс, который отвечает за сборку трехмерной модели. Реализует интерфейс IRenderable.

public void CalculateNormals () – функция вычисления нормалей ко всем полигонам объекта. Используются структуры Vector и Triangle.

public void Render () – метод, к котором происходит отрисовка составных частей модели.

**Class Model**

Класс описывает и отображает все части модели.

public void Render () – модель может состоять из нескольких частей, например, самолет состоит из крыльев, фюзеляжа и шасси. Данная функция вызывает метод отрисовки для каждой составной части трехмерной модели.

## 3.3 Пример файла настроек

Файл настроек applicationSettings.xml.

Ниже представлен пример, как должны быть оформлены настройки.

<?xml version="1.0" encoding="utf-8"?>

<Property xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<dataFile>default.xml</dataFile>

<modelFile>tu154.3DS</modelFile>

<particleNumber>1000</particleNumber>

<particleState>true</particleState>

<glideState>false</glideState>

<deviationState>false</deviationState>

<planeState>false</planeState>

<explosion>

<x>0</x>

<y>0</y>

<z>0</z>

</explosion>

<transparency>0.3</transparency>

<V0>20</V0>

<H0>600</H0>

<R0>1000</R0>

<beginGlide>

<x>0</x>

<y>0</y>

<z>0</z>

</beginGlide>

<endGlide>

<x>-1000</x>

<y>46.44</y>

<z>0</z>

</endGlide>

</Property>

dataFile – имя или путь к файлу, содержащему данные о посадке.

modelFile – имя или путь к файлу трехмерной модели.

particleNumber – количество частиц, которые будут использованы при моделировании микровзрыва.

particleState – индикатор, показывающий состояние режима отображения микровзрыва.

Explosion.X, explosion.Y, explosion.Z – координаты центра микровзрыва.

Transparency – величина прозрачности глиссадной и курсовых плоскостей.

( полная прозрачность 0 <= transparency <= 1.0 непрозрачный объект)

V0 - скорость в центральной части микровзрыва, [м/с].

H0 - высота центральной части, [м].

R0 - радиус вихря, [м].

beginGlide, endGlide – точки начала и конца траектории глиссады.

## 3.4 Пример файла данных

Ниже представлен пример файла данных, который программа корректно распознает:

<?xml version="1.0" encoding="utf-8"?>

<Trajectory xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<data>

<InstantData>

<t>-120.00000000000001</t>

<x>-8055.6</x>

<y>430.6</y>

<z>150</z>

<vx>67.13</vx>

<vy>-3.13</vy>

<vz>0</vz>

<theta>0.0513126800086333</theta>

<psi>0</psi>

<gamma>0</gamma>

<wtheta>0</wtheta>

<wpsi>0</wpsi>

<wgamma>0</wgamma>

</InstantData>

</data>

</Trajectory>

t – время кадра.

x, y, z – координаты центра масс самолета.

vx, vy, z – скорости по соответствующим осям координат.

theta, psi, gamma – угловые координаты.

wtheta, wpsi, wgamma – угловые скорости.

## 3.5 Интерфейс

Главное и единственное окно программы разделено на несколько логических зон.

1. Меню.

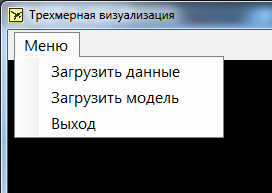


Рисунок 4 – Меню.

Выпадающее меню, представленное на рисунке 4, содержит три пункта: загрузить данные, загрузить модель, выход.

1. Область вывода анимации.



Рисунок 5 – Область вывода анимации

Здесь происходит отрисовка каждого кадра трехмерной анимации (рисунок 5)

1. Панель включения параметров отображения.

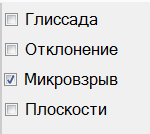


Рисунок 6 – Панель опций

Панель содержит четыре элемента check-box (рисунок 6). Пользователь может выбирать, какую функцию включить при просмотре анимации. Функции независимы друг от друга и могут быть включены в любой момент выполнения программы. Пользователь может указывать опции на включении из файла настроек.

1. Информационная панель.

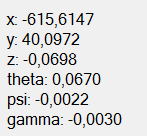


Рисунок 7 – Информационная панель

В данной области окна происходит вывод данных в каждый момент времени.

Обновление данных происходит в тот же момент, когда и обновляется кадр области отображения анимации.

1. Функциональная панель.

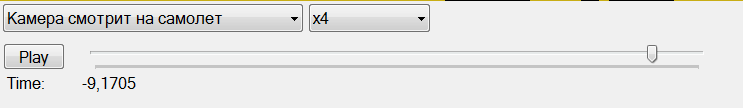


Рисунок 8 – Функциональная панель

Функциональная панель содержит несколько элементов:

* Выпадающий список выбора камеры наблюдения.

Доступные камеры: камера смотрит на самолет и камера смотрит на ВПП.

* Выпадающий список выбора скорости обновления кадра.

Доступные опции: х0.5, х1, х2, х4. При х1 время обновления кадров равняется 5 миллисекунд.

* Кнопка Play. Запускает или останавливает процесс анимации. Изначально находится в выключенном состоянии.
* Панель прокрутки.

Позволяет перейти на любой момент анимации.

* Время.

Отображает значение времени для текущего кадра.

При запуске данные элементы неактивны. Становятся доступны после загрузки данных о полете и модели самолета.

1. Общий вид и расположение панелей.

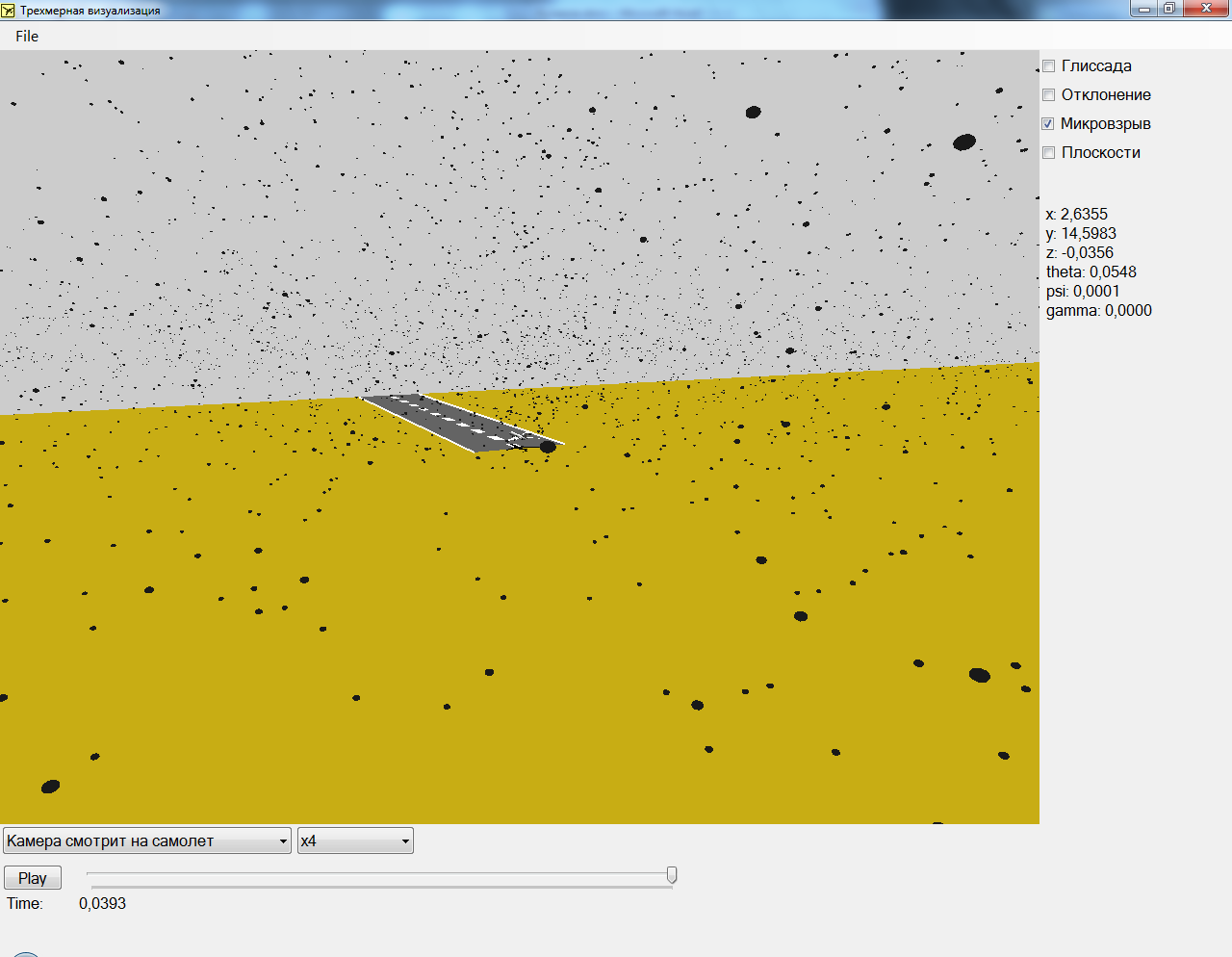


Рисунок 9 – Основное окно приложения

Большую часть рабочего окна занимает область анимации, так как там отображаются основные визуальные модели. Все остальные панели расположены максимально доступно и не отвлекают от наблюдения за анимацией.

## 3.6 Работа программы и интерпретация результата

В результате решения задачи о посадке самолета пользователь получил файл с данными. Для начала работы можно настроить приложение (не является необходимым). Рассмотрим несколько сценариев работы.

Сценарий 1

Пользователь хочет увидеть посадку самолета в условиях ветровых возмущений, в частности микровзрыв. Для этого в файле настроек необходимо указать флаг на включение отображения микровзрыва при запуске. Если этого не требуется, то данную опцию можно включить в процессе выполнения программы. Также можно указать количество частиц, которые будут формировать микровзрыв. Данную величину можно подобрать оптимально после нескольких запусков с различными значениями. Далее следует задать параметры микровзрыва. Если значения не столь важны, как сама визуальная модель, то данный шаг можно пропустить. В файле настроек можно задать имя файла с данными, если эти данные будут визуализироваться несколько раз. Тогда при последующих запусках программа будет обращаться именно к этому файлу. Теперь запускается исполняемый файл. Если пользователь еще не указал файл с данными, то это можно сделать из «Меню/Загрузить данные». Возможно, что задача была решена для самолета, отличного от Ту-154. С помощью пункта меню «Загрузить модель» следует указать путь к трехмерной модели воздушного судна. Теперь пользователь может выбрать камеру, с помощью которой будет наблюдать за анимацией. Например, в начале анимации пользователь хочет видеть перемещение самолета в динамике, а затем выбрать статичное положение камеры, направленной на торец ВПП. Выбрав нужную камеру, нужно запустить анимацию с помощью кнопки «Play». Камера следует за самолетом. Выбрав необходимую дистанцию до объекта с помощью ролика мыши, пользователь может наблюдать все изменения в положении самолета. На информационной панели в правой части окна будут выводиться значения координат и угловые скорости самолета. Если в данных решения задачи были допущены ошибки, то это отразится на положении модели в сцене. Например, наблюдатель заметил, что самолет слишком сильно накренился, хотя для этого не было никаких предпосылок. В этом случае пользователю надо анализировать ситуацию, искать возможные причины возникновения таких данных.

В некоторых момент времени необходимо наблюдать всю сцену с помощью статичной камеры. Пользователь выбирает камеру, смотрящую на торец ВПП. Моделируется ситуация, когда в зоне посадки самолета находится эпицентр микровзрыва. С помощью созданной модели этого явления наблюдатель со стороны видит, как самолет влетает в зону ветровых возмущений, также визуально сможет сравнить скорости потока воздуха и самолета в каждый момент времени посадки (рисунок 10).

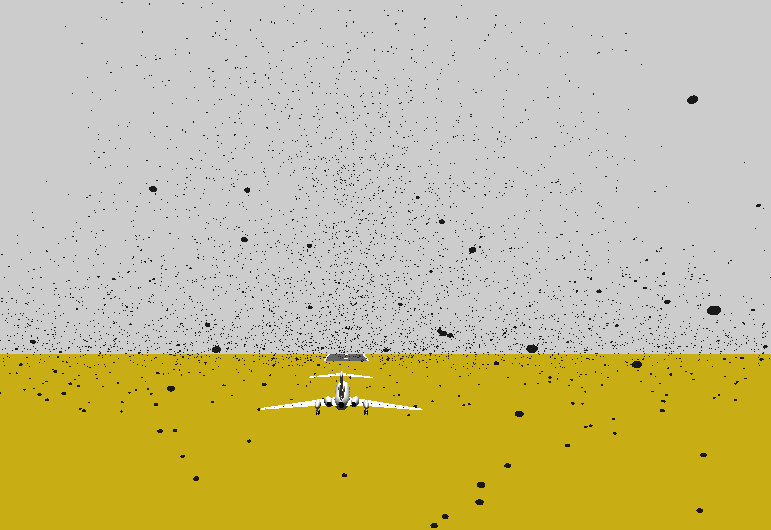


Рисунок 10 – Область микровзрыва

Статичная модель микровзрыва выглядит как бесформенное скопление частиц. Совсем другая картина получается, если смотреть на эту же сцену в динамике.

Сценарий 2

Пользователь хочет видеть отклонения воздушного судна от номинальной траектории на протяжении всего полета. Можно указать путь к файлу с данными в конфигурационном файле, либо выбрать его после запуска программы. Для вычисления отклонений нужно указать данные о глиссаде: координаты точек начала и конца.

Совершив подготовительные действия, нужно запустить исполняемый файл.

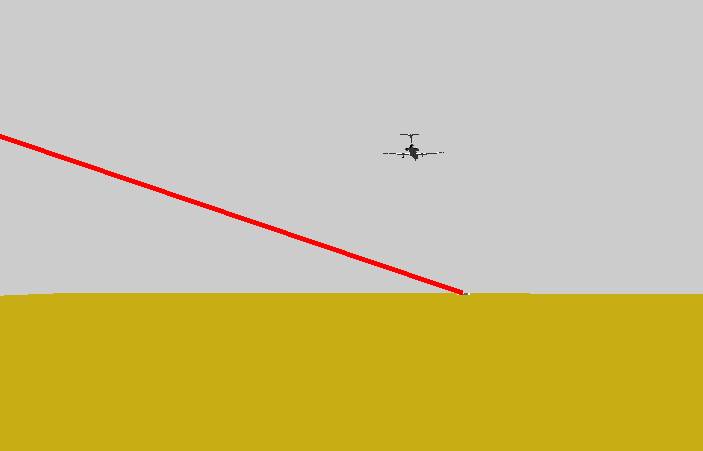


Рисунок 11 - Глиссада

Для отображения глиссады необходимо включить опцию «Глиссада». На экране будет рисоваться красная линия номинальной траектории, относительно которой совершает посадку самолет (рисунок 11).

Чтобы визуально можно было оценить отклонения от глиссады, нужно включить опцию «Отклонение».

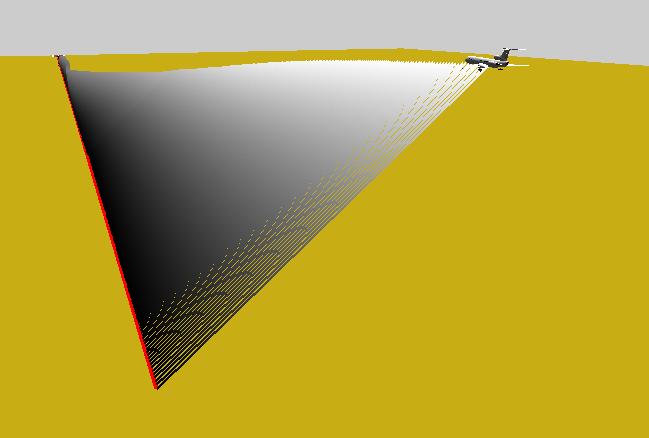


Рисунок 12 – Цветовая градация

В этом режиме на экран будут выводиться линии, соединяющие центр масс самолета и соответствующую точку на глиссаде (рисунок 12) Для большей наглядности цвет линии меняется от белого (максимальное отклонение) до черного (незначительное отклонение). Такой метод позволяет увидеть отклонения самолета, моменты полета, когда он резко меняет свое положение относительно глиссады.

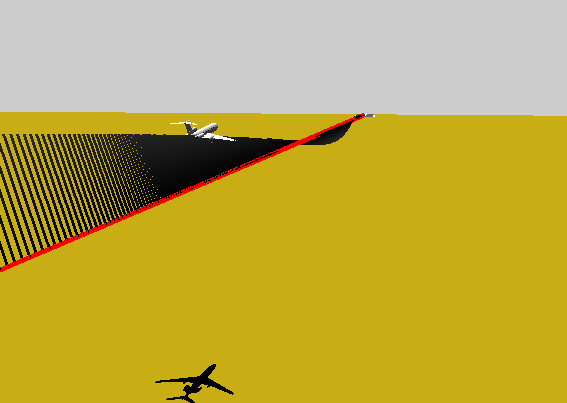


Рисунок 13 - Отклонения

На рисунке 13 видно, как самолет облетает линию глиссады с левой стороны, затем пролетает на некотором расстоянии под ней и только в конце полета выходит на оптимальную траекторию.

Для того, что отобразить глиссадную и курсовую плоскости, необходимо включить опцию «Плоскости».



Рисунок 14 - Плоскости

На рисунке 14 показано, как самолет летит левее вертикальной плоскости и практически полностью оказывается ниже плоскости глиссады.

# ЗАКЛЮЧЕНИЕ

В результате данной работы были получены следующие результаты:

1. Создана специализированная среда трехмерной динамической визуализации.
2. Разработана и реализована анимация для отображения решений задачи посадки самолета в условиях ветровых возмущений.
3. Реализована трехмерная динамическая анимация явления микровзрыва.

Развитие данного проекта возможно в нескольких направлениях.

1. Создание более общей оболочки, в которой будут представлена трехмерная анимация для различных задач оптимального управления, объединенных в логические группы.
2. Оптимизация программного кода для улучшения быстродействия. Использование различных возможностей библиотеки OpenGL, например, уменьшение времени выполнения за счет загрузки в память видеоадаптера всех возможных структур, моделей.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Miele A., Wang T., Melvin W.W., Optimal take-off trajectories in the presence of windshear // Journal of Optimization Theory and Applications, Vol.49, No.1, 1986, pp.1–45.
2. Miele A., Wang T., Tzeng C.Y., Melvin W.W., Optimal abort landing trajectories in the presence of windshear // Journal of Optimization Theory and Applications, Vol.55, No.2, 1987, pp.165–202.
3. Miele A., Wang T., Wang H., Melvin W.W., Optimal penetration landing trajectories in the presence of windshear // Journal of Optimization Theory and Applications, Vol.57, No.1, 1988, pp.1–40.
4. Bulirsch R., Montrone F., Pesch H.J., Abort landing in the presence of windshear as a minimax optimal control problem, part 1: Necessary conditions // Journal of Optimization Theory and Applications, Vol.70, No.1, 1991, pp.1–23.
5. Bulirsch R., Montrone R., Pesch H.J., Abort landing in the presence of windshear as a minimax optimal control problem, part 2: Multiple shooting and homotopy // Journal of Optimization Theory and Applications, Vol.70, No.2, 1991, pp.223–254.
6. Кейн В.М., Париков А.Н., Смуров М.Ю., Об одном способе оптимального управления по методу экстремального прицеливания // Прикладная математика и механика, Т.44, №3, 1980, С.434–440.
7. Кейн В.М., Оптимизация систем управления по минимаксному критерию. М.: Наука, 1985.
8. Боткин Н.Д., Кейн В.М., Пацко В.С., Модельная задача об управлении боковым движением самолета на посадке // Прикладная математика и механика, Т.48, №4, 1984, С.560–567.
9. Ganebny S.A., Kumkov S.S., Patsko V.S., Pyatko S.G., Constructing Robust Control in Differential Games. Application to Aircraft Control during Landing // Annals of the International Society of Dynamic Games, Vol.9: Advances in Dynamic Games and Applications, S.Jorgensen, M.Quincampoix, and T.L.Vincent (Eds.), Birkhauser, Boston, MA, 2007, pp.69–92.
10. Ivan M., A ring-vortex downburst model for real-time flight simulation of severe windshear // AIAA Flight Simulation Technologies Conf., July 22–24, 1985, St.Louis, Miss., pp.57–61.
11. Боткин Н.Д., Пацко В.С., Турова В.Л., [Разработка численных методов построения экстремальных ветровых возмущений, действующих на самолет на этапе посадки. Разработка алгоритмов построения экстремальных ветровых возмущений.](http://home.imm.uran.ru/kumkov/rapevv1987/index.html) Отчет о научно-исследовательской работе (промежуточный). Свердловск, 1987, № ГР01880003467, Инв. № 02880054701. 58 стр.
12. Ричард С. Райт-мл., Липчак Б., OpenGL. Суперкнига, издательство Вильямс, 2006 г., 1040 стр.
13. Херн, Бейкер, Компьютерная графика и стандарт OpenGL, издательство Вильямс, 2005 г., 1168 стр.
14. М. Ву, Т. Девис, Дж. Нейдер, Д. Шрайнер, OpenGL. Руководство по программированию, 4-е издание, издательство Питер, 2006 г., 624 стр.

# ПРИЛОЖЕНИЕ А – Программный код

**class CalcBurst**

using System;

using System.Collections.Generic;

using System.Text;

namespace Самолет

{

class CalcBurst

{

float X;

float Y;

float Z;

float V0, H0, R0;

float Dx, Dy, Dz;

public float Wx, Wy, Wz;

public float WR;

public float Len;

float CIP;

public CalcBurst(float V0\_, float H0\_, float R0\_, float Dx\_, float Dy\_, float Dz\_)

{

V0 = V0\_;

H0 = H0\_;

R0 = R0\_;

Dx = Dx\_;

Dy = Dy\_;

Dz = Dz\_;

}

public float Sign(float A, float B)

{

if (B >= 0)

return (float)Math.Abs(A);

else

return (float)(-1) \* Math.Abs(A);

}

public void VV(float Xg, float Yg, float Zg)

{

X = Xg - Dx;

Y = Yg - Dy;

Z = Zg - Dz;

float WR;

WR = 0.0f;

Wy = 0.0f;

CIP = 2 \* V0 \* R0 / (1.0f - 1.0f /(float) Math.Pow(1.0f + Math.Pow(2 \* H0 / R0, 2), 1.5));

float RC, RD, R;

RC=0.8f\*H0;

R = (float) Math.Sqrt( X \* X + Z \* Z );

RD = (float)Math.Sqrt((Math.Abs(R) - R0) \* (Math.Abs(R) - R0) + (Math.Abs(Y) - H0) \* (Math.Abs(Y) - H0));

if (RD - RC >= 0)

{

if (Math.Abs(R) <= 0.5)

R = 0.5f;

this.WAP(Y, R, out WR, out Wy);

}

else

{

float Q;

float WR1, WY1;

WR1 = 0.0f;

WY1 = 0.0f;

Q = RD / RC;

if (Q != 0)

{

float R1, Y1;

R1=R-Sign(R0,R);

Y1=Y-Sign(H0,Y);

this.WAP((Sign(H0, Y) + Y1 / Q), (Sign(R0, R) + R1 / Q),out WR, out Wy);

}

WR = Q \* WR;

Wy = Q \* Wy;

}

Wx = WR \* X / R;

Wz = WR \* Z / R;

Wy = -Wy;

Len = (float)Math.Sqrt(Wx \* Wx + Wz \* Wz + Wy \* Wy);

}

public float STR(float Y\_, float R\_)

{

float R1, R2, R3, R4, RK1, RK2,AR,AM;

R1 = (float)Math.Sqrt((Y\_ - H0) \* (Y\_ - H0) + (R\_ - R0) \* (R\_ - R0));

R2 = (float)Math.Sqrt((Y\_ + H0) \* (Y\_ + H0) + (R\_ - R0) \* (R\_ - R0));

R3 = (float)Math.Sqrt((Y\_ - H0) \* (Y\_ - H0) + (R\_ + R0) \* (R\_ + R0));

R4 = (float)Math.Sqrt((Y\_ + H0) \* (Y\_ + H0) + (R\_ + R0) \* (R\_ + R0));

RK1 = ((R3 - R1) / (R3 + R1)) \* ((R3 - R1) / (R3 + R1));

RK2 = ((R4 - R2) / (R4 + R2)) \* ((R4 - R2) / (R4 + R2));

AR = 0.788f \* RK1 / (0.25f + 0.75f \* (float)Math.Sqrt(1.0 - RK1));

AM = 0.788f \* RK2 / (0.25f + 0.75f \* (float)Math.Sqrt(1.0 - RK2));

return -CIP \* ((R1 + R3) \* AR - (R2 + R4) \* AM) / 6.2831852f;

}

public void WAP(float Y\_, float R\_, out float WR\_, out float WY\_)

{

float D = 0.2f;

float STR1 = STR(Y\_,R\_);

WY\_ = (STR1 - STR(Y\_, R\_ + D)) / (R\_ \* D);

WR\_ = (STR(Y\_ - D, R\_) - STR1) / (R\_ \* D);

}

}

}

**class Deviation**

using System;

using System.Collections.Generic;

using System.Text;

using Tao.OpenGl;

using Tao.FreeGlut;

namespace Самолет

{

class Deviation

{

private List<InstantData> data;

private bool flag = false;

private float[] x;

private float[] y;

private float[] z;

private float[] deviation;

private int length;

private int DisplayListNomGlide;

private int DisplayListNomDev;

private float maxDeviation = 0.0f;

private CVertex3f beginGlide;

private CVertex3f endGlide;

public Deviation(List<InstantData> m\_data, CVertex3f begin, CVertex3f end)

{

data = m\_data;

length = data.Count;

x = new float[length];

y = new float[length];

z = new float[length];

deviation = new float[length];

beginGlide = begin;

endGlide = end;

}

public void Calculate()

{

float tempDeviation = 0.0f;

for (int i = 0; i < length; i++)

{

x[i] = (float)data[i].x;

y[i] = (x[i] - beginGlide.x) \* (endGlide.y - beginGlide.y) / (endGlide.x - beginGlide.x) + beginGlide.y;

z[i] = (x[i] - beginGlide.x) \* (endGlide.z - beginGlide.z) / (endGlide.x - beginGlide.x) + beginGlide.z;

tempDeviation = (float)Math.Sqrt(data[i].z \* data[i].z + (data[i].y - y[i]) \* (data[i].y - y[i]));

deviation[i] = tempDeviation;

if (tempDeviation > maxDeviation)

{

maxDeviation = tempDeviation;

}

}

//глиссада

DisplayListNomGlide = Gl.glGenLists(1);

Gl.glNewList(DisplayListNomGlide, Gl.GL\_COMPILE);

Gl.glColor3f(1.0f, 0.0f, 0.0f);

Gl.glLineWidth(5.0f);

Gl.glBegin(Gl.GL\_LINE\_STRIP);

for (int i = 0; i < length; i++)

{

Gl.glVertex3f(x[i] , y[i], z[i]);

}

Gl.glEnd();

Gl.glEndList();

//отклонение

DisplayListNomDev = Gl.glGenLists(1);

Gl.glNewList(DisplayListNomDev, Gl.GL\_COMPILE);

Gl.glLineWidth(3.0f);

Gl.glBegin(Gl.GL\_LINES);

for (int i = 0; i < length; i++)

{

Gl.glColor3f(deviation[i] / maxDeviation, deviation[i] / maxDeviation, deviation[i] / maxDeviation);

Gl.glVertex3f((float)data[i].x , (float)data[i].y - 1, (float)data[i].z);

Gl.glColor3f(0.0f, 0.0f, 0.0f);

Gl.glVertex3f(x[i], y[i], z[i]);

}

Gl.glEnd();

Gl.glEndList();

flag = true;

}

public void Render(bool blGLide, bool blDev, bool blScale, float transparency)

{

if (flag == false)

{

this.Calculate();

flag = true;

}

if (blGLide)

{

Gl.glCallList(DisplayListNomGlide);

}

if (blDev)

{

Gl.glCallList(DisplayListNomDev);

}

if (blScale)

{

Gl.glBlendFunc(Gl.GL\_SRC\_ALPHA, Gl.GL\_ONE\_MINUS\_SRC\_ALPHA);

Gl.glEnable(Gl.GL\_ALPHA\_TEST);

Gl.glEnable(Gl.GL\_BLEND);

Gl.glFrontFace(Gl.GL\_CW);

Gl.glBegin(Gl.GL\_TRIANGLES);

Gl.glColor4f(0.0f, 0.5f, 0, transparency);

Gl.glVertex3f(x[0], y[0], -500.0f);

Gl.glVertex3f(x[0], y[0], 500.0f);

Gl.glVertex3f(0, y[length - 1], 0.0f);

Gl.glColor4f(0.0f, 0.0f, 0.5f, transparency);

Gl.glVertex3f(x[0] , y[0] + 400.0f, 0.0f);

Gl.glVertex3f(x[0] , y[0] - 400.0f, 0.0f);

Gl.glVertex3f(0, y[length - 1], 0.0f);

Gl.glEnd();

Gl.glFrontFace(Gl.GL\_CCW);

Gl.glBegin(Gl.GL\_TRIANGLES);

Gl.glColor4f(0.0f, 0.5f, 0, transparency);

Gl.glVertex3f(x[0] , y[0], -500.0f);

Gl.glVertex3f(x[0] , y[0], 500.0f);

Gl.glVertex3f(0, y[length - 1], 0.0f);

Gl.glColor4f(0.0f, 0.0f, 0.5f, transparency);

Gl.glVertex3f(x[0], y[0] + 400.0f, 0.0f);

Gl.glVertex3f(x[0], y[0] - 400.0f, 0.0f);

Gl.glVertex3f(0, y[length - 1], 0.0f);

Gl.glEnd();

Gl.glDisable(Gl.GL\_BLEND);

Gl.glDisable(Gl.GL\_ALPHA\_TEST);

}

}

}

}

**class Entity**

using System;

using System.Collections.Generic;

using Tao.OpenGl;

namespace Самолет

{

public class MaterialFaces

{

public Material Material;

public ushort[] Faces;

}

public class Entity : IRenderable

{

int glListNo = -1;

List<MaterialFaces> materialFaces = new List<MaterialFaces>();

public Vector[] vertices;

public Vector[] normals;

public Triangle[] indices;

public TexCoord[] texcoords;

bool normalized = false;

public List<MaterialFaces> MaterialFaces {

get {

return materialFaces;

}

}

public void CalculateNormals ()

{

if ( indices == null ) return;

normals = new Vector [vertices.Length];

Vector[] temps = new Vector [ indices.Length ];

for ( int ii=0 ; ii < indices.Length ; ii++ )

{

Triangle tr = indices [ii];

Vector v1 = vertices [ tr.Vertex1 ] - vertices [ tr.Vertex2 ];

Vector v2 = vertices [ tr.Vertex2 ] - vertices [ tr.Vertex3 ];

temps [ii] = v1.CrossProduct ( v2 );

}

for ( int ii = 0; ii < vertices.Length ; ii++ )

{

Vector v = new Vector ();

for ( int jj = 0; jj < indices.Length ; jj++ )

{

Triangle tr = indices [jj];

if ( tr.Vertex1 == ii || tr.Vertex2 == ii || tr.Vertex3 == ii )

{

v += temps [jj];

}

}

normals [ii] = v.Normalize ();

}

normalized = true;

}

public void Render ()

{

if ( indices == null ) return;

if (glListNo >= 0)

{

Gl.glCallList(glListNo);

return;

}

glListNo = Gl.glGenLists(1);

System.Console.WriteLine(glListNo);

Gl.glNewList(glListNo, Gl.GL\_COMPILE);

bool noMaterials = materialFaces.Count == 0;

if (noMaterials)

{

MaterialFaces m = new MaterialFaces();

m.Material = new Material();

materialFaces.Add(m);

}

foreach (MaterialFaces m in materialFaces)

{

Material material = m.Material;

Gl.glMaterialfv (Gl.GL\_FRONT\_AND\_BACK, Gl.GL\_AMBIENT, material.Ambient);

Gl.glMaterialfv (Gl.GL\_FRONT\_AND\_BACK, Gl.GL\_DIFFUSE, material.Diffuse);

Gl.glMaterialfv (Gl.GL\_FRONT\_AND\_BACK, Gl.GL\_SPECULAR, material.Specular);

Gl.glMaterialf (Gl.GL\_FRONT\_AND\_BACK, Gl.GL\_SHININESS, material.Shininess);

if (material.TextureId >= 0 )

{

Gl.glBindTexture ( Gl.GL\_TEXTURE\_2D, material.TextureId );

Gl.glEnable( Gl.GL\_TEXTURE\_2D );

}

Gl.glBegin ( Gl.GL\_TRIANGLES);

for (int ii = 0; ii < (noMaterials ? indices.Length : m.Faces.Length); ii++)

{

Triangle tri = noMaterials ? indices[ii] : indices[m.Faces[ii]];

if (normalized) Gl.glNormal3d ( normals[tri.Vertex1].X, normals[tri.Vertex1].Y, normals[tri.Vertex1].Z );

if ( material.TextureId >= 0 ) Gl.glTexCoord2f ( texcoords [ tri.Vertex1 ].U, texcoords [ tri.Vertex1 ].V);

Gl.glVertex3d ( vertices[tri.Vertex1].X, vertices[tri.Vertex1].Y, vertices[tri.Vertex1].Z );

if (normalized) Gl.glNormal3d ( normals[tri.Vertex2].X, normals[tri.Vertex2].Y, normals[tri.Vertex2].Z );

if ( material.TextureId >= 0 ) Gl.glTexCoord2f ( texcoords [ tri.Vertex2 ].U, texcoords [ tri.Vertex2 ].V);

Gl.glVertex3d ( vertices[tri.Vertex2].X, vertices[tri.Vertex2].Y, vertices[tri.Vertex2].Z );

if (normalized) Gl.glNormal3d ( normals[tri.Vertex3].X, normals[tri.Vertex3].Y, normals[tri.Vertex3].Z );

if ( material.TextureId >= 0 ) Gl.glTexCoord2f( texcoords [ tri.Vertex3 ].U, texcoords [ tri.Vertex3 ].V);

Gl.glVertex3d ( vertices[tri.Vertex3].X, vertices[tri.Vertex3].Y, vertices[tri.Vertex3].Z );

}

Gl.glEnd();

}

Gl.glDisable( Gl.GL\_TEXTURE\_2D );

Gl.glEndList();

}

}

}

**class Explosion**

using System;

using System.Collections.Generic;

using System.Text;

using Tao.OpenGl;

using Tao.FreeGlut;

using Tao.Platform.Windows;

namespace Самолет

{

class Explosion

{

private float[] position = new float[3];

private int MAX\_PARTICLES = 50000;

private int \_particles\_now;

private bool isStart = false;

private Particle[] ParticleArray;

private bool isDisplayList = false;

private int DisplayListNom = 0;

private static int[] k = { -1, 1 };

private CalcBurst test;

private Random rnd = new Random();

public Explosion(float x, float y, float z, float V0, float H0, float R0, int particle\_count)

{

position[0] = x;

position[1] = y;

position[2] = z;

test = new CalcBurst(V0, H0, R0, position[0], position[1], position[2]);

\_particles\_now = particle\_count;

if (\_particles\_now > MAX\_PARTICLES)

{

\_particles\_now = MAX\_PARTICLES;

}

System.Console.WriteLine(\_particles\_now);

ParticleArray = new Particle[\_particles\_now];

}

public void SetNewPosition(float x, float y, float z)

{

position[0] = x;

position[1] = y;

position[2] = z;

}

private void CreateDisplayList()

{

DisplayListNom = Gl.glGenLists(1);

Gl.glNewList(DisplayListNom, Gl.GL\_COMPILE);

Gl.glColor4f(0.1f, 0.1f, 0.1f, 1.0f);

Glut.glutSolidSphere(1.2, 10, 10);

Gl.glEndList();

isDisplayList = true;

}

public void Boooom()

{

if (!isDisplayList)

{

CreateDisplayList();

}

for (int ax = 0; ax < \_particles\_now; ax++)

{

ParticleArray[ax] = CreateNewParticle();

ParticleArray[ax].Calculate();

}

isStart = true;

}

public Particle CreateNewParticle()

{

Particle returnParticle = new Particle(position[0], position[1], position[2], test, rnd);

return returnParticle;

}

public void Render()

{

if (isStart)

{

for (int ax = 0; ax < \_particles\_now; ax++)

{

int i ;

i=ParticleArray[ax].index;

Gl.glPushMatrix();

Gl.glTranslated(ParticleArray[ax].points[i].x, ParticleArray[ax].points[i].y, ParticleArray[ax].points[i].z);

Gl.glCallList(DisplayListNom);

Gl.glPopMatrix();

ParticleArray[ax].index++;

if (ParticleArray[ax].index==ParticleArray[ax].maxIndex)

{

ParticleArray[ax].index = 0;

}

}

}

}

}

}

**class ForXML, class InstantData**

using System;

using System.Collections.Generic;

using System.Text;

using System.Xml.Serialization;

using System.IO;

[XmlRootAttribute("InstantData")]

public class InstantData

{

// Time mark

public double t;

/\*

\* Coordinates of the aircraft mass center:

\* x - longitudinal coordinate

\* y - height

\* z - lateral deviation

\*/

public double x;

public double y;

public double z;

// Respective geometric velocities

public double vx;

public double vy;

public double vz;

// Angular coordinates:

// theta (pitch) - angle of rotation around z-axis

//psi (yaw) - angle of rotation around y-axis

// gamma (bank) - angle of rotation around x-axis

public double theta;

public double psi;

public double gamma;

// Respective angular velocities

public double wtheta;

public double wpsi;

public double wgamma;

}

[XmlRootAttribute("Trajectory")]

public class ForXML

{

[XmlArrayAttribute("data")]

public List<InstantData> data;

public ForXML()

{

data = new List<InstantData>();

}

}

**interface IRenderable**

using System;

namespace Самолет

{

public interface IRenderable

{

void Render();

}

}

**Class Lib**

using System;

using System.Collections.Generic;

using System.Text;

namespace Самолет

{

public struct CVertex3f

{

public float x;

public float y;

public float z;

public CVertex3f(float v1, float v2, float v3)

{

x = v1;

y = v2;

z = v3;

}

public override string ToString()

{

return String.Format("v1: {0} v2: {1} v3: {2}", x, y, z);

}

}

public struct Triangle

{

public int Vertex1;

public int Vertex2;

public int Vertex3;

public Triangle(int v1, int v2, int v3)

{

Vertex1 = v1;

Vertex2 = v2;

Vertex3 = v3;

}

public override string ToString()

{

return String.Format("v1: {0} v2: {1} v3: {2}", Vertex1, Vertex2, Vertex3);

}

}

public struct CFace

{

public int Vertex1;

public int Vertex2;

public int Vertex3;

public CFace(int v1, int v2, int v3)

{

Vertex1 = v1;

Vertex2 = v2;

Vertex3 = v3;

}

public override string ToString()

{

return String.Format("v1: {0} v2: {1} v3: {2}", Vertex1, Vertex2, Vertex3);

}

}

public struct TexCoord

{

public float U;

public float V;

public TexCoord(float u, float v)

{

U = u;

V = v;

}

}

public struct Vector

{

public double X;

public double Y;

public double Z;

public Vector(double x, double y, double z)

{

X = x;

Y = y;

Z = z;

}

public Vector CrossProduct(Vector v)

{

return new Vector(Y \* v.Z - Z \* v.Y,

Z \* v.X - X \* v.Z,

X \* v.Y - Y \* v.X);

}

public double DotProduct(Vector v)

{

return X \* v.X + Y \* v.Y + Z \* v.Z;

}

public Vector Normalize()

{

double d = Length();

if (d == 0) d = 1;

return this / d;

}

public double Length()

{

return Math.Sqrt(DotProduct(this));

}

public override string ToString()

{

return String.Format("X: {0} Y: {1} Z: {2}", X, Y, Z);

}

public static Vector operator +(Vector v1, Vector v2)

{

Vector vr;

vr.X = v1.X + v2.X;

vr.Y = v1.Y + v2.Y;

vr.Z = v1.Z + v2.Z;

return vr;

}

public static Vector operator /(Vector v1, double s)

{

Vector vr;

vr.X = v1.X / s;

vr.Y = v1.Y / s;

vr.Z = v1.Z / s;

return vr;

}

public static Vector operator -(Vector v1, Vector v2)

{

Vector vr;

vr.X = v1.X - v2.X;

vr.Y = v1.Y - v2.Y;

vr.Z = v1.Z - v2.Z;

return vr;

}

}

public class Lib

{

public Lib()

{

}

public void gltGetNormalVector(float[] vP1, float[] vP2, float[] vP3, float[] vNormal)

{

float[] vV1, vV2;

vV1 = new float[3];

vV2 = new float[3];

gltSubtractVectors(vP2, vP1, vV1);

gltSubtractVectors(vP3, vP1, vV2);

gltVectorCrossProduct(vV1, vV2, vNormal);

gltNormalizeVector(vNormal);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Вычесть один вектор из другого

public void gltSubtractVectors(float[] vFirst, float[] vSecond, float[] vResult)

{

vResult[0] = vFirst[0] - vSecond[0];

vResult[1] = vFirst[1] - vSecond[1];

vResult[2] = vFirst[2] - vSecond[2];

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Вычислить векторное произведение двух векторов

public void gltVectorCrossProduct(float[] vU, float[] vV, float[] vResult)

{

vResult[0] = vU[1] \* vV[2] - vV[1] \* vU[2];

vResult[1] = -vU[0] \* vV[2] + vV[0] \* vU[2];

vResult[2] = vU[0] \* vV[1] - vV[0] \* vU[1];

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Привести вектор к единичной длине (нормировать)

public void gltNormalizeVector(float[] vNormal)

{

float fLength = 1.0f / (float)Math.Sqrt(vNormal[0] \* vNormal[0] + vNormal[1] \* vNormal[1] + vNormal[2] \* vNormal[2]);

gltScaleVector(vNormal, fLength);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Умножить вектор на скаляр

public void gltScaleVector(float[] vVector, float fScale)

{

vVector[0] = fScale; vVector[1] \*= fScale; vVector[2] \*= fScale;

}

public void gltMakeShadowMatrix(float[] Point1, float[] Point2, float[] Point3, float[] vLightPos, float[] destMat)

{

float[] vPlaneEquation = new float[4];

float dot;

gltGetPlaneEquation(Point1, Point2, Point3, vPlaneEquation);

// Вычисляет скалярное произведение направляющего вектора плоскости

// и вектора положения источника света

dot = vPlaneEquation[0] \* vLightPos[0] +

vPlaneEquation[1] \* vLightPos[1] +

vPlaneEquation[2] \* vLightPos[2] +

vPlaneEquation[3] \* vLightPos[3];

// Формируем матрицу проекции

// Первый столбец

destMat[0] = dot - vLightPos[0] \* vPlaneEquation[0];

destMat[4] = 0.0f - vLightPos[0] \* vPlaneEquation[1];

destMat[8] = 0.0f - vLightPos[0] \* vPlaneEquation[2];

destMat[12] = 0.0f - vLightPos[0] \* vPlaneEquation[3];

// Второй столбец

destMat[1] = 0.0f - vLightPos[1] \* vPlaneEquation[0];

destMat[5] = dot - vLightPos[1] \* vPlaneEquation[1];

destMat[9] = 0.0f - vLightPos[1] \* vPlaneEquation[2];

destMat[13] = 0.0f - vLightPos[1] \* vPlaneEquation[3];

// Третий столбец

destMat[2] = 0.0f - vLightPos[2] \* vPlaneEquation[0];

destMat[6] = 0.0f - vLightPos[2] \* vPlaneEquation[1];

destMat[10] = dot - vLightPos[2] \* vPlaneEquation[2];

destMat[14] = 0.0f - vLightPos[2] \* vPlaneEquation[3];

// Четвертый столбец

destMat[3] = 0.0f - vLightPos[3] \* vPlaneEquation[0];

destMat[7] = 0.0f - vLightPos[3] \* vPlaneEquation[1];

destMat[11] = 0.0f - vLightPos[3] \* vPlaneEquation[2];

destMat[15] = dot - vLightPos[3] \* vPlaneEquation[3];

}

// Возвращает коэффициенты уравнения плоскости по трем точкам

public void gltGetPlaneEquation(float[] vPoint1, float[] vPoint2, float[] vPoint3, float[] vPlane)

{

// Вычислить вектор нормали

gltGetNormalVector(vPoint1, vPoint2, vPoint3, vPlane);

vPlane[3] = -(vPlane[0] \* vPoint3[0] + vPlane[1] \* vPoint3[1] + vPlane[2] \* vPoint3[2]);

}

}

}

**class LoadData**

using System;

using System.Collections.Generic;

using System.Text;

using System.Xml.Serialization;

using System.IO;

public class LoadData

{

public ForXML forXML;

public LoadData()

{

}

private void CreateData(string filename)

{

XmlSerializer serializer =

new XmlSerializer(typeof(ForXML));

TextWriter writer = new StreamWriter(filename);

ForXML Data = new ForXML();

int i = 0;

InstantData instData = new InstantData();

instData.x = i++;

instData.y = i++;

instData.z = i++;

Data.data.Add(instData);

serializer.Serialize(writer, Data);

writer.Close();

}

public void ReadPO(string filename)

{

XmlSerializer serializer = new XmlSerializer(typeof(ForXML));

serializer.UnknownNode += new

XmlNodeEventHandler(serializer\_UnknownNode);

serializer.UnknownAttribute += new

XmlAttributeEventHandler(serializer\_UnknownAttribute);

FileStream fs = new FileStream(filename, FileMode.Open);

forXML = (ForXML)serializer.Deserialize(fs);

fs.Close();

}

private void serializer\_UnknownNode(object sender, XmlNodeEventArgs e)

{

Console.WriteLine("Unknown Node:" + e.Name + "\t" + e.Text);

}

private void serializer\_UnknownAttribute(object sender, XmlAttributeEventArgs e)

{

System.Xml.XmlAttribute attr = e.Attr;

Console.WriteLine("Unknown attribute " +

attr.Name + "='" + attr.Value + "'");

}

}

**class Property**

using System;

using System.Collections.Generic;

using System.Text;

using System.Data;

using System.IO;

using System.Xml.Serialization;

using System.Runtime.Serialization;

using System.Windows.Forms;

namespace Самолет

{

public class Property

{

public string dataFile;

public string modelFile;

public int particleNumber;

public bool particleState;

public bool glideState;

public bool deviationState;

public bool planeState;

public CVertex3f explosion;

public float transparency;

public float V0;

public float H0;

public float R0;

public CVertex3f beginGlide;

public CVertex3f endGlide;

}

class LoadProperty

{

public Property property;

public Property defProperty;

public LoadProperty()

{

defProperty = new Property();

defProperty.dataFile = "default.xml";

defProperty.modelFile = "tu154.3DS";

defProperty.particleNumber = 15000;

defProperty.particleState = true;

defProperty.glideState = false;

defProperty.deviationState = false;

defProperty.planeState = false;

defProperty.explosion = new CVertex3f(0.0f, 0.0f, 0.0f);

defProperty.V0 = 20.0f;

defProperty.H0 = 600.0f;

defProperty.R0 = 1000.0f;

defProperty.transparency = 0.3f;

defProperty.beginGlide = new CVertex3f(0.0f, 0.0f, 0.0f);

defProperty.endGlide = new CVertex3f(-1000.0f, 46.44f, 0.0f);

}

public Property Load()

{

XmlSerializer xmlser = new XmlSerializer(typeof(Property));

string filename = System.Environment.CurrentDirectory + "\\applicationSettings.xml";

FileStream filestream= null;

property = defProperty;

try

{

filestream = new FileStream(filename, FileMode.Open);

property = (Property)xmlser.Deserialize(filestream);

if (!File.Exists(property.dataFile))

{

MessageBox.Show("Файла" + property.dataFile + " не существует");

property.dataFile = defProperty.dataFile;

}

if (!File.Exists(property.modelFile))

{

MessageBox.Show("Файла" + property.modelFile + " не существует");

property.modelFile = defProperty.modelFile;

}

}

catch (FileNotFoundException ae)

{

MessageBox.Show("Файл конфигурации не найден");

WriteProperty(defProperty);

}

catch (InvalidOperationException ae)

{

MessageBox.Show("Неправильный формат данных");

WriteProperty(defProperty);

}

finally

{

if (filestream != null)

filestream.Close();

}

return property;

}

public void WriteProperty(Property pr)

{

XmlSerializer serializer = new XmlSerializer(typeof(Property));

TextWriter writer = new StreamWriter("applicationSettings.xml");

serializer.Serialize(writer,pr);

writer.Close();

}

private void serializer\_UnknownNode(object sender, XmlNodeEventArgs e)

{

Console.WriteLine("Unknown Node:" + e.Name + "\t" + e.Text);

}

private void serializer\_UnknownAttribute(object sender, XmlAttributeEventArgs e)

{

System.Xml.XmlAttribute attr = e.Attr;

Console.WriteLine("Unknown attribute " +

attr.Name + "='" + attr.Value + "'");

}

}

}

**class Material**

using System;

using Tao.OpenGl;

namespace Самолет

{

public class Material

{

public float[] Ambient = new float [] { 0.5f, 0.5f, 0.5f };

public float[] Diffuse = new float [] { 0.5f, 0.5f, 0.5f };

public float[] Specular = new float [] { 0.5f, 0.5f, 0.5f };

public int Shininess = 50;

int textureid = -1;

public int TextureId {

get {

return textureid;

}

}

public void BindTexture ( int width, int height, IntPtr data )

{

Gl.glEnable( Gl.GL\_TEXTURE\_2D );

int[] textures = new int [1];

Gl.glGenTextures(1, textures);

textureid = textures[0];

Gl.glBindTexture( Gl.GL\_TEXTURE\_2D, textureid );

Gl.glTexParameteri(Gl.GL\_TEXTURE\_2D, Gl.GL\_TEXTURE\_MAG\_FILTER, Gl.GL\_LINEAR);

Gl.glTexParameteri(Gl.GL\_TEXTURE\_2D, Gl.GL\_TEXTURE\_MIN\_FILTER, Gl.GL\_LINEAR\_MIPMAP\_NEAREST);

Gl.glTexEnvf(Gl.GL\_TEXTURE\_ENV, Gl.GL\_TEXTURE\_ENV\_MODE, Gl.GL\_MODULATE);

Gl.glTexImage2D(Gl.GL\_TEXTURE\_2D, 0, Gl.GL\_RGBA8, width, height, 0, Gl.GL\_BGRA\_EXT, Gl.GL\_UNSIGNED\_BYTE, data );

Glu.gluBuild2DMipmaps( Gl.GL\_TEXTURE\_2D, 4, width, height, Gl.GL\_BGRA\_EXT, Gl.GL\_UNSIGNED\_BYTE, data );

Gl.glDisable( Gl.GL\_TEXTURE\_2D );

}

}

}

**class Model**

using System;

using System.Collections.Generic;

namespace Самолет

{

public class Model : IRenderable

{

public List<Entity> Entities = new List<Entity> ();

public void Render ()

{

foreach ( Entity e in Entities )

e.Render ();

}

}

}

**class ThreeDSFile**

using System;

using System.IO;

using System.Text;

using System.Collections.Generic;

using System.Drawing;

namespace Самолет

{

public class ThreeDSFile

{

#region classes

public class ThreeDSChunk

{

public ushort ID;

public uint Length;

public int BytesRead;

public ThreeDSChunk ( BinaryReader reader )

{

// 2 byte ID

ID = reader.ReadUInt16();

// 4 byte length

Length = reader.ReadUInt32 ();

// = 6

BytesRead = 6;

}

}

#endregion

#region Enums

enum Groups

{

C\_PRIMARY = 0x4D4D,

C\_OBJECTINFO = 0x3D3D,

C\_VERSION = 0x0002,

C\_EDITKEYFRAME = 0xB000,

C\_MATERIAL = 0xAFFF,

C\_MATNAME = 0xA000,

C\_MATAMBIENT = 0xA010,

C\_MATDIFFUSE = 0xA020,

C\_MATSPECULAR = 0xA030,

C\_MATSHININESS = 0xA040,

C\_MATMAP = 0xA200,

C\_MATMAPFILE = 0xA300,

C\_OBJECT = 0x4000,

C\_OBJECT\_MESH = 0x4100,

C\_OBJECT\_VERTICES = 0x4110,

C\_OBJECT\_FACES = 0x4120,

C\_OBJECT\_MATERIAL = 0x4130,

C\_OBJECT\_UV = 0x4140

}

#endregion

#region Vars

Dictionary < string, Material > materials = new Dictionary < string, Material > ();

string base\_dir;

BinaryReader reader;

double maxX, maxY, maxZ, minX, minY, minZ;

int version = -1;

#endregion

#region public properties

Model model = new Model ();

public Model Model {

get {

return model;

}

}

public int Version {

get {

return version;

}

}

public double MaxX {

get {

return maxX;

}

}

public double MaxY {

get {

return maxY;

}

}

public double MaxZ {

get {

return maxZ;

}

}

public double MinX {

get {

return minX;

}

}

public double MinY {

get {

return minY;

}

}

public double MinZ {

get {

return minZ;

}

}

#endregion

#region Constructors

public ThreeDSFile ( string file\_name )

{

if (string.IsNullOrEmpty(file\_name))

{

throw new ArgumentNullException("file\_name");

}

if (!File.Exists(file\_name))

{

throw new ArgumentException("3ds file could not be found", "file\_name");

}

base\_dir = new FileInfo ( file\_name ).DirectoryName + "/";

maxX = maxY = maxZ = double.MinValue;

minX = minY = minZ = double.MaxValue;

FileStream file = null;

try

{

file = new FileStream(file\_name, FileMode.Open, FileAccess.Read);

reader = new BinaryReader ( file );

reader.BaseStream.Seek (0, SeekOrigin.Begin);

ThreeDSChunk chunk = new ThreeDSChunk ( reader );

if ( chunk.ID != (short) Groups.C\_PRIMARY )

{

throw new FormatException ( "Not a proper 3DS file." );

}

ProcessChunk ( chunk );

}

finally

{

if (reader != null) reader.Close ();

if (file != null) file.Close ();

}

}

#endregion

#region Helper methods

void ProcessChunk ( ThreeDSChunk chunk )

{

while ( chunk.BytesRead < chunk.Length )

{

ThreeDSChunk child = new ThreeDSChunk ( reader );

switch ((Groups) child.ID)

{

case Groups.C\_VERSION:

version = reader.ReadInt32 ();

child.BytesRead += 4;

break;

case Groups.C\_OBJECTINFO:

break;

case Groups.C\_MATERIAL:

ProcessMaterialChunk ( child );

break;

case Groups.C\_OBJECT:

ProcessString ( child );

Entity e = ProcessObjectChunk ( child );

e.CalculateNormals ();

model.Entities.Add ( e );

break;

default:

SkipChunk ( child );

break;

}

chunk.BytesRead += child.BytesRead;

}

}

void ProcessMaterialChunk ( ThreeDSChunk chunk )

{

string name = string.Empty;

Material m = new Material ();

while ( chunk.BytesRead < chunk.Length )

{

ThreeDSChunk child = new ThreeDSChunk ( reader );

switch ((Groups) child.ID)

{

case Groups.C\_MATNAME:

name = ProcessString ( child );

break;

case Groups.C\_MATAMBIENT:

m.Ambient = ProcessColorChunk ( child );

break;

case Groups.C\_MATDIFFUSE:

m.Diffuse = ProcessColorChunk ( child );

break;

case Groups.C\_MATSPECULAR:

m.Specular = ProcessColorChunk ( child );

break;

case Groups.C\_MATSHININESS:

m.Shininess = ProcessPercentageChunk ( child );

break;

case Groups.C\_MATMAP:

ProcessPercentageChunk ( child );

ProcessTexMapChunk ( child , m );

break;

default:

SkipChunk ( child );

break;

}

chunk.BytesRead += child.BytesRead;

}

materials.Add ( name, m );

}

void ProcessTexMapChunk ( ThreeDSChunk chunk, Material m )

{

while ( chunk.BytesRead < chunk.Length )

{

ThreeDSChunk child = new ThreeDSChunk ( reader );

switch ((Groups) child.ID)

{

case Groups.C\_MATMAPFILE:

string name = ProcessString ( child );

Bitmap bmp;

try

{

bmp = new Bitmap ( base\_dir + name );

}

catch ( Exception ex )

{

Console.WriteLine ( " ERROR: could not load file '{0}': {1}", base\_dir + name, ex.Message );

break;

}

bmp.RotateFlip(RotateFlipType.RotateNoneFlipY);

System.Drawing.Imaging.BitmapData imgData = bmp.LockBits ( new Rectangle(new Point(0, 0), bmp.Size),

System.Drawing.Imaging.ImageLockMode.ReadOnly,

System.Drawing.Imaging.PixelFormat.Format32bppArgb);

m.BindTexture ( imgData.Width, imgData.Height, imgData.Scan0 );

bmp.UnlockBits(imgData);

bmp.Dispose();

break;

default:

SkipChunk ( child );

break;

}

chunk.BytesRead += child.BytesRead;

}

}

float[] ProcessColorChunk ( ThreeDSChunk chunk )

{

ThreeDSChunk child = new ThreeDSChunk ( reader );

float[] c = new float[] { (float) reader.ReadByte() / 256 , (float) reader.ReadByte() / 256 , (float) reader.ReadByte() / 256 };

chunk.BytesRead += (int) child.Length;

return c;

}

int ProcessPercentageChunk ( ThreeDSChunk chunk )

{

ThreeDSChunk child = new ThreeDSChunk ( reader );

int per = reader.ReadUInt16 ();

child.BytesRead += 2;

chunk.BytesRead += child.BytesRead;

return per;

}

Entity ProcessObjectChunk ( ThreeDSChunk chunk )

{

return ProcessObjectChunk ( chunk, new Entity() );

}

Entity ProcessObjectChunk ( ThreeDSChunk chunk, Entity e )

{

while ( chunk.BytesRead < chunk.Length )

{

ThreeDSChunk child = new ThreeDSChunk ( reader );

switch ((Groups) child.ID)

{

case Groups.C\_OBJECT\_MESH:

ProcessObjectChunk ( child , e );

break;

case Groups.C\_OBJECT\_VERTICES:

e.vertices = ReadVertices ( child );

break;

case Groups.C\_OBJECT\_FACES:

e.indices = ReadIndices ( child );

if ( child.BytesRead < child.Length )

ProcessObjectChunk ( child, e );

break;

case Groups.C\_OBJECT\_MATERIAL:

string name2 = ProcessString ( child );

Material mat;

if ( materials.TryGetValue ( name2, out mat ) )

{

MaterialFaces m = new MaterialFaces();

m.Material = mat;

int nfaces = reader.ReadUInt16 ();

child.BytesRead += 2;

m.Faces = new UInt16[nfaces];

for ( int ii=0; ii<nfaces; ii++)

{

m.Faces[ii] = reader.ReadUInt16 ();

child.BytesRead += 2;

}

e.MaterialFaces.Add(m);

}

else

{

SkipChunk ( child );

}

break;

case Groups.C\_OBJECT\_UV:

int cnt = reader.ReadUInt16 ();

child.BytesRead += 2;

e.texcoords = new TexCoord [ cnt ];

for ( int ii=0; ii<cnt; ii++ )

e.texcoords [ii] = new TexCoord ( reader.ReadSingle (), reader.ReadSingle () );

child.BytesRead += ( cnt \* ( 4 \* 2 ) );

break;

default:

SkipChunk ( child );

break;

}

chunk.BytesRead += child.BytesRead;

}

return e;

}

void SkipChunk ( ThreeDSChunk chunk )

{

int length = (int) chunk.Length - chunk.BytesRead;

reader.ReadBytes ( length );

chunk.BytesRead += length;

}

string ProcessString ( ThreeDSChunk chunk )

{

StringBuilder sb = new StringBuilder ();

byte b = reader.ReadByte ();

int idx = 0;

while ( b != 0 )

{

sb.Append ( (char) b);

b = reader.ReadByte ();

idx++;

}

chunk.BytesRead += idx+1;

return sb.ToString();

}

Vector[] ReadVertices ( ThreeDSChunk chunk )

{

ushort numVerts = reader.ReadUInt16 ();

chunk.BytesRead += 2;

Vector[] verts = new Vector[numVerts];

for ( int ii=0; ii < verts.Length ; ii++ )

{

float f1 = reader.ReadSingle();

float f2 = reader.ReadSingle();

float f3 = reader.ReadSingle();

Vector v = new Vector ( f1, f3, -f2 );

if (v.X > maxX) maxX = v.X;

if (v.Y > maxY) maxY = v.Y;

if (v.Z > maxZ) maxZ = v.Z;

if (v.X < minX) minX = v.X;

if (v.Y < minY) minY = v.Y;

if (v.Z < minZ) minZ = v.Z;

verts[ii] = v;

}

chunk.BytesRead += verts.Length \* ( 3 \* 4 ) ;

return verts;

}

Triangle[] ReadIndices ( ThreeDSChunk chunk )

{

ushort numIdcs = reader.ReadUInt16 ();

chunk.BytesRead += 2;

Triangle[] idcs = new Triangle[numIdcs];

for ( int ii=0; ii < idcs.Length ; ii++ )

{

idcs [ii] = new Triangle ( reader.ReadUInt16(), reader.ReadUInt16(), reader.ReadUInt16() );

reader.ReadUInt16 ();

}

chunk.BytesRead += ( 2 \* 4 ) \* idcs.Length;

return idcs;

}

#endregion

}

}

**class Particle**

using System;

using System.Collections.Generic;

using System.Collections;

using System.Text;

namespace Самолет

{

class Particle

{

private float angle;

private float radius;

private float [] position\_emmit = new float[3];

public CVertex3f[] points;

public int index;

public int maxIndex;

private Random rnd;

private static float RADIUS\_BOOM = 600.0f;

float x, y,z;

private CalcBurst burst;

public Particle(float x\_, float y\_, float z\_, CalcBurst burst\_, Random rnd\_)

{

position\_emmit[0] = x\_;

position\_emmit[1] = y\_;

position\_emmit[2] = z\_;

burst = burst\_;

rnd = rnd\_;

index = 0;

angle = 6.2831852f \* (float)rnd.NextDouble();

radius = 5.0f + (RADIUS\_BOOM - 5.0f) \* (float)rnd.NextDouble();

x = radius \* (float)Math.Cos(angle) + position\_emmit[0];

z = radius \* (float)Math.Sin(angle) + position\_emmit[2];

y = 600 + 400 \* (float)rnd.NextDouble() + position\_emmit[1];

}

public void Calculate()

{

List<CVertex3f> list = new List<CVertex3f>();

float deltaTime = 0.50f;

Boolean flag = true;

while (flag)

{

CVertex3f temp = new CVertex3f();

burst.VV(x, y, z);

if (burst.Len < 1 || Math.Sqrt(Math.Pow(position\_emmit[0] - x, 2) + Math.Pow(position\_emmit[2] - z, 2)) > 2000|| list.Count > 700) // Math.Abs(y) + Math.Abs(z) + Math.Abs(x) < 2 ||

{

flag = false;

}

else

{

x = x + burst.Wx \* deltaTime;

y = y + burst.Wy \* deltaTime;

z = z + burst.Wz \* deltaTime;

temp.x = x;

temp.y = y;

temp.z = z;

list.Add(temp);

}

}

points = list.ToArray();

maxIndex = points.GetLength(0);

index = rnd.Next(maxIndex);

}

}

}

**class MainForm**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

//using System.Linq;

using System.Text;

using System.Windows.Forms;

using Tao.OpenGl;

using Tao.FreeGlut;

using Tao.Platform.Windows;

namespace Самолет

{

public partial class MainForm : Form

{

private float[] whiteLight = { 0.2f, 0.2f, 0.2f, 1.0f };

private float[] sourceLight = { 0.8f, 1.0f, 0.8f, 1.0f };

private float[] lightPos = { 1.0f, 100.0f, 0.0f, 0.0f };

private float[] grey = { 0.1f, 0.1f, 0.1f, 1.0f};

private Boolean btnchk= true; //true, if movie stop

private string fileName ;

private string fileModel ;

private float theta =45.0f, phi = 45.0f, r = 1000.0f, pogr\_v = 0.0f;

private float x = 0.0f, y = 90.0f;

private int count = 0;

private float dAngle = 2.0f;

private float t = -40.0f;

private float size = 6000.0f;

private float[] shadowM = new float[16];

private Boolean canMove = false;

private Point p;

private LoadData load;

private ForXML xml;

static ThreeDSFile file;

static Model model;

private AirCorridor cor;

private Deviation dev;

private LoadProperty loadProperty;

private Property property;

private double def\_time = -120;

private int def\_count = 1;

private float[] speed = { 0.5f, 1.0f, 2.0f, 4.0f};

private static float WheelSpeed = 1.0f / 15;

private Explosion burst ;

private float transparency;

bool dontCount = true;

public MainForm()

{

InitializeComponent();

ShowDisplay.InitializeContexts();

this.SetDesktopLocation(0, 0);

this.ShowDisplay.MouseLeave += new EventHandler(ShowDisplay\_MouseLeave);

this.ShowDisplay.MouseDown += new MouseEventHandler(ShowDisplay\_MouseDown);

this.ShowDisplay.MouseUp += new MouseEventHandler(ShowDisplay\_MouseUp);

this.ShowDisplay.MouseMove += new MouseEventHandler(ShowDisplay\_MouseMove);

this.ShowDisplay.MouseWheel += new MouseEventHandler(ShowDisplay\_MouseWheel);

Box1.SelectedIndex=0;

comboBox1.SelectedIndex = 3;

load = new LoadData();

loadProperty = new LoadProperty();

property = loadProperty.Load();

fileModel = property.modelFile;

fileName = property.dataFile;

checkBoxParticle.Checked = property.particleState;

checkBoxGlide.Checked = property.glideState;

checkBoxDev.Checked = property.deviationState;

checkBoxPlane.Checked = property.planeState;

transparency = property.transparency;

Size size = SystemInformation.PrimaryMonitorSize;

if ( size.Width != 1366)

{

this.Width = size.Width;

this.Height = size.Height- 30;

panel1.Location = new Point(this.Width - 200, panel1.Location.Y);//153; 121

panel3.Location = new Point(this.Width - 200, panel3.Location.Y);

this.ShowDisplay.Height = this.Height - 120 - 40-30;

this.ShowDisplay.Width = panel1.Location.X;

panel2.Location = new Point(0, this.Height - 120-40); //784; 116

}

}

private void ShowDisplay\_MouseLeave(object sender, EventArgs e)

{

canMove = false;

}

private void ShowDisplay\_MouseDown(object sender, MouseEventArgs e)

{

canMove = true;

p = e.Location;

}

private void ShowDisplay\_MouseUp(object sender, MouseEventArgs e)

{

canMove = false;

}

private void ShowDisplay\_MouseMove(object sender, MouseEventArgs e)

{

if (canMove&(xml!=null) )

{

float a, b, s;

a = (e.X - p.X);

b = (e.Y - p.Y);

s = (float)Math.Sqrt(a \* a + b \* b);

if (s >= 20)

{

theta -= dAngle \* b / s;

if (theta < -180)

{

theta += 360;

}

else

{

if (theta > 180)

{

theta -= 360;

}

}

if (theta < 0)

{

phi += dAngle \* a / s;

}

else

{

phi -= dAngle \* a / s;

}

if (phi > 360)

{

phi -= 360;

}

else

{

if (phi < 0)

{

phi += 360;

}

}

p = e.Location;

Draw();

}

}

}

void ShowDisplay\_MouseWheel(object sender, MouseEventArgs e)

{

if (r + e.Delta \* WheelSpeed > 10.0f)

{

r += e.Delta \* WheelSpeed;

}

if (xml != null)

{

Draw();

}

}

private void Form1\_Load(object sender, EventArgs e)

{

float fAspect;

Glut.glutInit();

Glut.glutInitDisplayMode(Glut.GLUT\_RGB | Glut.GLUT\_DOUBLE | Glut.GLUT\_DEPTH);

Gl.glViewport(0, 0, ShowDisplay.Width, ShowDisplay.Height);

fAspect = (float)ShowDisplay.Width / (float)ShowDisplay.Height;

Gl.glMatrixMode(Gl.GL\_PROJECTION);

Gl.glLoadIdentity();

Glu.gluPerspective(75, Width / Height, 2, 10000);

Gl.glMatrixMode(Gl.GL\_MODELVIEW);

Gl.glLoadIdentity();

Gl.glEnable(Gl.GL\_DEPTH\_TEST);

Gl.glFrontFace(Gl.GL\_CCW);

Gl.glEnable(Gl.GL\_CULL\_FACE);

Gl.glCullFace(Gl.GL\_BACK);

Gl.glEnable(Gl.GL\_LIGHTING);

Gl.glLightModelfv(Gl.GL\_LIGHT\_MODEL\_AMBIENT, whiteLight);

Gl.glLightfv(Gl.GL\_LIGHT0, Gl.GL\_POSITION, sourceLight);

Gl.glLightfv(Gl.GL\_LIGHT0, Gl.GL\_POSITION, lightPos);

Gl.glEnable(Gl.GL\_LIGHT0);

Gl.glEnable(Gl.GL\_COLOR\_MATERIAL);

Gl.glColorMaterial(Gl.GL\_FRONT, Gl.GL\_AMBIENT\_AND\_DIFFUSE);

Gl.glClearColor(0.8f, 0.8f, 0.8f, 1.0f);

loadModel();

Gl.glEnable(Gl.GL\_NORMALIZE);

float[] dot1 = { -30.0f, 0.0f, -20.0f };

float[] dot2 = { -30.0f, 0.0f, 20.0f };

float[] dot3 = { 40.0f, 0.0f, 20.0f };

Lib lib = new Lib();

lib.gltMakeShadowMatrix(dot1, dot2, dot3, lightPos, shadowM);

Begin();

burst = new Explosion(property.explosion.x, property.explosion.y, property.explosion.z, property.V0, property.H0, property.R0, property.particleNumber);

burst.Boooom();

}

private void barPlay\_Scroll(object sender, EventArgs e)

{

count = barPlay.Value;

lblShowTime.Text = xml.data[count].t.ToString("0.0000");

lblx.Text = "x: " + xml.data[count].x.ToString("0.0000");

lbly.Text = "y: " + xml.data[count].y.ToString("0.0000");

lblz.Text = "z: " + xml.data[count].z.ToString("0.0000");

lblTheta.Text = "theta: " + xml.data[count].theta.ToString("0.0000");

lblPsi.Text = "psi: " + xml.data[count].psi.ToString("0.0000");

lblGamma.Text = "gamma: " + xml.data[count].gamma.ToString("0.0000");

Draw();

}

private void btnLoadData\_Click(object sender, EventArgs e)

{

}

private void btnPause\_Click(object sender, EventArgs e)

{

if (btnPause.Text == "Pause")

{

dontCount = true;

// btnchk = true;

btnPause.Text = "Play";

}

else

{

timer1.Start();

dontCount = false;

// btnchk = false;

btnPause.Text = "Pause";

}

}

private void timer1\_Tick(object sender, EventArgs e)

{

lblShowTime.Text = xml.data[count].t.ToString("0.0000");

lblx.Text = "x: " + xml.data[count].x.ToString("0.0000");

lbly.Text = "y: " + xml.data[count].y.ToString("0.0000");

lblz.Text = "z: " + xml.data[count].z.ToString("0.0000");

lblTheta.Text = "theta: " + xml.data[count].theta.ToString("0.0000");

lblPsi.Text = "psi: " + xml.data[count].psi.ToString("0.0000");

lblGamma.Text = "gamma: " + xml.data[count].gamma.ToString("0.0000");

barPlay.Value = count;

Draw();

if (dontCount == false)

{

count += 1;

}

if (count >= barPlay.Maximum)

{

//btnchk = true;

btnPause.Text = "Play";

dontCount = true;

count = barPlay.Maximum;

}

}

private void exitToolStripMenuItem\_Click(object sender, EventArgs e)

{

Application.Exit();

}

private void Draw()

{

Gl.glClear(Gl.GL\_COLOR\_BUFFER\_BIT | Gl.GL\_DEPTH\_BUFFER\_BIT);

Gl.glMatrixMode(Gl.GL\_MODELVIEW);

Gl.glLoadIdentity();

switch (Box1.SelectedIndex)

{

case 0:

{

Glu.gluLookAt(xml.data[count].x + r \* Math.Sin(-theta \* 3.14 / 180) \* Math.Sin(phi \* 3.14 / 180),

xml.data[count].y + r \* Math.Cos(-theta \* 3.14 / 180),

xml.data[count].z + r \* Math.Sin(-theta \* 3.14 / 180) \* Math.Cos(phi \* 3.14 / 180),

xml.data[count].x, xml.data[count].y, xml.data[count].z,

0, Math.Sign(theta), 0);

break;

}

case 1:

{

Glu.gluLookAt(r \* Math.Sin(theta \* 3.14 / 180) \* Math.Sin(phi \* 3.14 / 180),

r \* Math.Cos(theta \* 3.14 / 180),

r \* Math.Sin(theta \* 3.14 / 180) \* Math.Cos(phi \* 3.14 / 180),

0, 0, 0,

0, Math.Sign(theta), 0);

break;

}

}

Gl.glPushMatrix();

Gl.glPushMatrix();

Gl.glEnable(Gl.GL\_LIGHTING);

Gl.glLightfv(Gl.GL\_LIGHT0, Gl.GL\_POSITION, lightPos);

Gl.glDisable(Gl.GL\_LIGHTING);

DrawGround(-size, -size / 2, size+500, size);

Gl.glDisable(Gl.GL\_DEPTH\_TEST);

DrawRect(0, -50, 500,100);

Gl.glEnd();

Gl.glPushMatrix();

Gl.glMultMatrixf(shadowM);

Gl.glTranslated(xml.data[count].x , xml.data[count].y, xml.data[count].z);

Gl.glRotated(xml.data[count].gamma \* 180.0f / 3.14159265f, 1, 0, 0);

Gl.glRotated(xml.data[count].psi \* 180.0f / 3.14159265f, 0, 1, 0);

Gl.glRotated(xml.data[count].theta \* 180.0f / 3.14159265f, 0, 0, 1);

Gl.glColor3ub(0, 0, 0);

//Gl.glRotated(-90, 0, 1, 0);

//Gl.glTranslated(0.0f, 4.0f, -4.0f);

//Gl.glScalef(0.7f, 1.0f, 0.65f);

model.Render();

Gl.glPopMatrix();

Gl.glPopMatrix();

Gl.glPushMatrix();

Gl.glColor3ub(0, 0, 0);

Gl.glPopMatrix();

Gl.glEnable(Gl.GL\_DEPTH\_TEST);

Gl.glEnable(Gl.GL\_LIGHTING);

Gl.glPushMatrix();

Gl.glDisable(Gl.GL\_COLOR\_MATERIAL);

Gl.glTranslated(xml.data[count].x, xml.data[count].y, xml.data[count].z);

Gl.glRotated(xml.data[count].gamma \* 180.0f / 3.14159265f, 1, 0, 0);

Gl.glRotated(xml.data[count].psi \* 180.0f / 3.14159265f, 0, 1, 0);

Gl.glRotated(xml.data[count].theta \* 180.0f / 3.14159265f, 0, 0, 1);

// Gl.glRotated(-90, 0, 1, 0);

//Gl.glTranslated(0.0f, -1.0f, -4.0f);

//Gl.glScalef(0.7f, 1.0f, 0.65f);

model.Render();

Gl.glEnable(Gl.GL\_COLOR\_MATERIAL);

Gl.glPopMatrix();

Gl.glDisable(Gl.GL\_LIGHTING);

if (checkBoxParticle.Checked)

{

burst.Render();

}

dev.Render(checkBoxGlide.Checked, checkBoxDev.Checked, checkBoxPlane.Checked, transparency);

Gl.glEnable(Gl.GL\_LIGHTING);

Gl.glPopMatrix();

Gl.glFlush();

ShowDisplay.Invalidate();

}

void DrawGround(float x, float y, float len, float width)

{

int n = 10;

float lStep = len / n;

float wStep = width / n;

Gl.glBegin(Gl.GL\_LINES);

Gl.glEnd();

Gl.glColor3ub(200, 173, 20);

for (int i = 0; i < n; i++)

{

Gl.glBegin(Gl.GL\_TRIANGLE\_STRIP);

Gl.glNormal3f(0.0f, 1.0f, 0.0f);

for (int j = n; j >= 0; j--)

{

Gl.glVertex3f(x + i \* lStep, 0, y + j \* wStep);

Gl.glVertex3f(x + (i + 1) \* lStep, 0, y + j \* wStep);

}

Gl.glEnd();

}

}

void DrawRect(float x, float y, float len, float width)

{

int n=10;

float lStep = len/n;

float wStep = width / n;

Gl.glColor3ub(100, 100, 100);

for (int i = 0; i < n; i++)

{

Gl.glBegin(Gl.GL\_TRIANGLE\_STRIP);

Gl.glNormal3f(0.0f, 1.0f, 0.0f);

for (int j = n; j >=0; j--)

{

Gl.glVertex3f(x+i\*lStep, 0, y + j\*wStep);

Gl.glVertex3f(x + (i+1) \* lStep, 0, y + j \* wStep);

}

Gl.glEnd();

}

Gl.glColor3ub(255, 255, 255);

n = 20;

lStep = len / n;

wStep = width / n;

for (int i = 0; i < n ; i += 2)

{

Gl.glBegin(Gl.GL\_QUADS);

Gl.glNormal3f(0.0f, 1.0f, 0.0f);

Gl.glVertex3f(x + i \* lStep, 0, y + width / 2 - 5);

Gl.glVertex3f(x + i \* lStep, 0, y + width / 2 + 5);

Gl.glVertex3f(x + (i + 1) \* lStep, 0, y + width / 2 + 5);

Gl.glVertex3f(x + (i + 1) \* lStep, 0, y + width / 2 - 5);

Gl.glEnd();

}

Gl.glLineWidth(2.0f);

Gl.glBegin(Gl.GL\_LINES);

Gl.glVertex3f(x , 0, y );

Gl.glVertex3f(x + len, 0, y);

Gl.glVertex3f(x, 0, y + width);

Gl.glVertex3f(x+len, 0, y + width);

Gl.glEnd();

}

private void label1\_Click(object sender, EventArgs e)

{

}

private void AxesBox\_CheckedChanged(object sender, EventArgs e)

{

Draw();

}

private void loadToolStripMenuItem\_Click(object sender, EventArgs e)

{

timer1.Stop();

OpenFileDialog openFileDialog1 = new OpenFileDialog();

openFileDialog1.InitialDirectory = AppDomain.CurrentDomain.BaseDirectory;

openFileDialog1.Filter = "xml file (\*.xml)|\*.xml|All files (\*.\*)|\*.\*";

openFileDialog1.RestoreDirectory = true;

if (openFileDialog1.ShowDialog() == DialogResult.OK)

{

try

{

fileName = openFileDialog1.FileName;

Begin();

}

catch (Exception ex)

{

MessageBox.Show("Error: Could not read file from disk. Original error: " + ex.Message);

}

}

}

private void Begin()

{

try

{

load.ReadPO(fileName);

xml = null;

xml = load.forXML;

dev = new Deviation(xml.data, property.beginGlide, property.endGlide);

dev.Calculate();

def\_time = xml.data[0].t;

def\_count = xml.data.Count;

timer1.Interval = (int)(-def\_time / def\_count \* 1000 / speed[comboBox1.SelectedIndex]);

size = (float)-xml.data[0].x;

barPlay.Maximum = xml.data.Count - 1;

btnPause.Show();

lblShowTime.Text = xml.data[0].t.ToString("0.0000");

barPlay.Value = 0;

count = 0;

barPlay.Enabled = true;

comboBox1.Enabled = true;

}

catch (System.IO.FileNotFoundException)

{

}

btnchk = true;

btnPause.Text = "Play";

}

private void comboBox1\_SelectedIndexChanged(object sender, EventArgs e)

{

try

{

timer1.Interval = (int)(-def\_time / def\_count \* 1000 / speed[comboBox1.SelectedIndex]);

}

catch (System.Exception a)

{

System.Console.WriteLine(a.TargetSite);

}

}

private void loadModelToolStripMenuItem\_Click(object sender, EventArgs e)

{

OpenFileDialog openFileDialog1 = new OpenFileDialog();

openFileDialog1.InitialDirectory = AppDomain.CurrentDomain.BaseDirectory;

openFileDialog1.Filter = "3DS file (\*.3ds)|\*.3DS|All files (\*.\*)|\*.\*";

openFileDialog1.RestoreDirectory = true;

if (openFileDialog1.ShowDialog() == DialogResult.OK)

{

try

{

fileModel = openFileDialog1.FileName;

loadModel();

}

catch (Exception ex)

{

MessageBox.Show("Error: Could not read file from disk. Original error: " + ex.Message);

}

}

}

private void loadModel()

{

file = null;

file = new ThreeDSFile(fileModel);

model = null;

model = file.Model;

}

private void checkBoxGlide\_CheckedChanged(object sender, EventArgs e)

{

if (xml != null)

{

Draw();

}

}

private void checkBoxDev\_CheckedChanged(object sender, EventArgs e)

{

if (xml != null)

{

Draw();

}

}

private void checkBoxParticle\_CheckedChanged(object sender, EventArgs e)

{

if (xml != null)

{

Draw();

}

}

private void checkBoxScale\_CheckedChanged(object sender, EventArgs e)

{

if (xml != null)

{

Draw();

}

}

private void AxesToolStripMenuItem\_Click(object sender, System.EventArgs e)

{

throw new System.NotImplementedException();

}

private void DrawUnitAxes()

{

throw new System.NotImplementedException();

}

}

}

**class Program**

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Windows.Forms;

namespace Самолет

{

static class Program

{

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Form MF = new MainForm();

MF.Text = "Трехмерная визуализация";

Application.Run(MF);

}

}

}