МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ ДНР

ГОУ ВПО «ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет КНТ

Кафедра ПИ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе по дисциплине:

“Архитектура и проектирование графических систем”

|  |  |
| --- | --- |
| Руководители:  Карабчевский В. В.  Щедрин С.В.  Доценко Г.В. | Выполнил:  студент группы ПИ-16а  Семик А.О. |

Донецк – 2019

РЕФЕРАТ

Отчет по курсовой работе содержит: 56 страниц, 30 рисунков, 3 приложения, 4 источника.

Объект исследования – графические системы и алгоритмы для работы с графическими моделями.

Предмет исследования – алгоритмы работы с 3D моделями, принципы рисования модели.

Цель – выполнить обоснованный проект графической системы, удовлетворяющий ряду требований, приведенный в варианте задания, реализовать возможность создания сцены с объектами и возможность работы с ней.

Задача – приобретение профессиональных навыков практического применения знаний, полученных в процессе обучения.

Результат – готовая программа, которая предоставляет возможность работы с любым количеством 3D объектов, заданных в специальном формате.

ГРАФИЧЕСКИЕ СИСТЕМЫ, ТРИПЛАН, ОТРИСОВКА, ТАНГАЖ, РЫСКАНЬЕ, ВЕКТОР, МАТРИЦА, СИСТЕМА КООРДИНАТ, УГОЛ, АФИННЫЕ ПРЕОБРАЗОВАНИЯ, Z-БУФЕР

СОДЕРЖАНИЕ

[Введение 4](#_Toc9472745)

[1 Разработка полигональной модели объекта 5](#_Toc9472746)

[1.1 Составляющие элементы объекта 5](#_Toc9472747)

[1.2 Триангуляция поверхности объекта 6](#_Toc9472748)

[2 Описание выбранных методов и алгоритмов визуализации 8](#_Toc9472749)

[2.1 Алгоритмы аффинных преобразований 8](#_Toc9472750)

[2.2 Алгоритм удаления скрытых линий и поверхностей 9](#_Toc9472751)

[2.3 Алгоритмы проекционных преобразований 10](#_Toc9472752)

[2.3.1 Алгоритм параллельного проецирования 10](#_Toc9472753)

[2.3.2 Алгоритм центрального проецирования 10](#_Toc9472754)

[2.4 Алгоритм произвольных видовых преобразований 12](#_Toc9472755)

[2.5 Алгоритм работы конвейера визуализации 13](#_Toc9472756)

[3 Разработка структур данных для хранения и описания объекта 15](#_Toc9472757)

[3.1 Описание структур данных 15](#_Toc9472758)

[3.2 Диаграммы UML взаимодействия классов 22](#_Toc9472759)

[3.3 Реализация модификации объекта 24](#_Toc9472760)

[3.4 Реализация изменения параметров 24](#_Toc9472761)

[3.5 Реализация аксонометрической и перспективной проекций 25](#_Toc9472762)

[3.6 Реализация произвольных видовых преобразований (камеры) 25](#_Toc9472763)

[3.7 Реализация сохранения сцены 26](#_Toc9472764)

[3.9 Реализация панорамирования 27](#_Toc9472765)

[4 Тестирование программной системы 28](#_Toc9472766)

[Выводы 29](#_Toc9472767)

[Перечень ссылок 30](#_Toc9472768)

[Приложение А Техническое задание 31](#_Toc9472769)

[Приложение Б Руководство пользователя 32](#_Toc9472770)

[Приложение В Листинг программы 37](#_Toc9472771)

# Введение

В современном мире пользователю часто приходится сталкиваться с ситуацией, когда необходимо смоделировать объект для отображения его инженерных решений или для изучения его структуры. Для моделирования объекта используется большое количество различных графических систем, в том числе и графических станций с различным ПО. Они предоставляют пользователю большой спектр возможностей для работы с различным количеством объектов и камер, в том числе: вращение, перемещение, удаление и другие.

Графическая система – совокупность взаимодействующих графических станций, имеющих общее устройство управление, обеспечивающее связь с ЭВМ.

Графической станцией можно назвать очень мощный компьютер, который комплектуется, чтобы обеспечить чёткой и достаточно быстрой работой пользователя в конкретной деятельности. Также в компоненты графической станции входит ПО, используя его, моделируется 3D объект.

Задача – написать программу, реализующую работу со сценой. В сцену входят: 3D объекты (трипланы).

В мире создается различное ПО графических системах, которые имеют свои преимущества и недостатки, отличаются определёнными признаками, используются для определённых целей. Самые популярные: AutoCAD, 3dsMax, Blender, Компас.

Цель работы и область применения – целью работы является написание алгоритмов создания, удаления, перемещения, вращения, изменения параметров 3D объекта и камеры, изучение технологий и алгоритмов для реализации удаления невидимых линий.

# 1 Разработка полигональной модели объекта

Полигональная модель – это модель объекта в пространстве, состоящая из множества отдельных полигонов (чаще всего используют треугольники), исходя из этого, каждый объект объединяет в себе множество точек, рёбер, и граней.

* вершины – это точки, в которых сходится и соединяется друг с другом любое число рёбер;
* рёбра – это линии границы грани;
* грани (полигоны) – это участки плоскости треугольной или четырёхугольной формы, представляющие собой элементарные ячейки сетки.

## 1.1 Составляющие элементы объекта

В данной работе разрабатывалась полигональная модель «Триплан», которая состояла из параллелепипедов, двух цилиндров, двух полусфер и двух сфер. Этого набора примитивов было достаточно для выполнения задания. Также можно использовать для создания полигональной модели пирамиды, конусы, сферы. На рисунке 1.1 представлена разработанная полигональная модель.

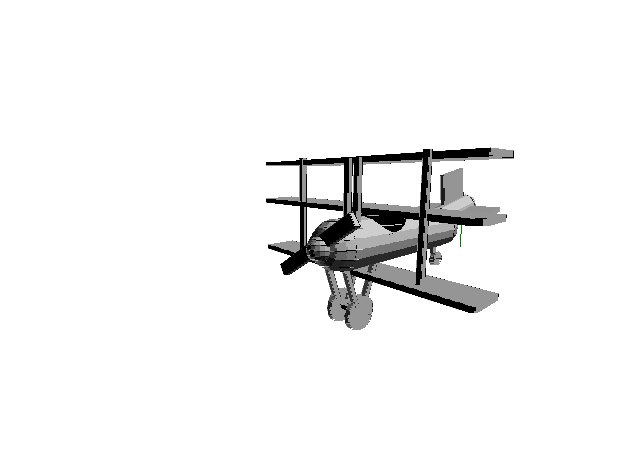


Рисунок 1.1 – Полигональная модель «Триплан»

Параллелепипед задаётся базовой точкой, которая расположена в центре нижней грани фигуры, и тремя параметрами (высотой, которая идёт вдоль оси Y, шириной (вдоль оси X), и длиной (вдоль оси Z)). Каждый параллелепипед представляется 6-ю прямоугольными гранями, которые, в свою очередь, делятся на 12 треугольников для простоты работы с гранями.

Цилиндр задаётся базовой точкой, которая расположена в центре нижнего основания, радиусом основания и высотой. Основание цилиндра делится на равные треугольники. Боковая поверхность цилиндра представляется прямоугольниками, которые тоже, в свою очередь, делятся, на треугольники. Количество треугольников зависит от количества сегментов цилиндра, которое может быть изменено программно (до 61).

Полусфера задается базовой точкой, расположенной в центре её основания, а также радиусом. Число треугольников, описывающих сферу зависит от числа сегментов, которое также может быть задано (до 61).

## 1.2 Триангуляция поверхности объекта

Триангуляция – это процесс разбиения грани, представленной многоугольником на треугольники для простоты работы с гранями и более простой визуализации. Триангуляция удобна при программировании графики потому что:

* треугольник является простейшим полигоном, вершины которого однозначно задают грань;
* любую область можно гарантированно разбить на треугольники;
* вычислительная сложность алгоритмов разбиения на треугольники существенно меньше, чем при использовании других полигонов;
* реализация процедур визуализации более проста для области, ограниченной треугольником.

В данной работе все прямоугольники разбиваются по диагонали на два треугольника.

На рисунке 1.2 представлена полигональная модель объекта в «проволочном» или «каркасном» виде.

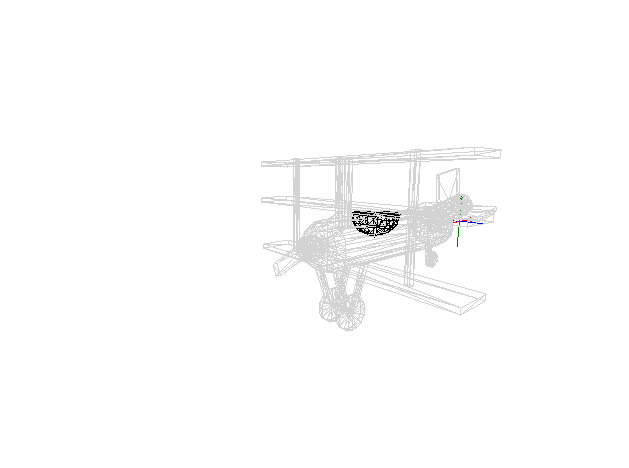


Рисунок 1.2 – Полигональная модель, представленная треугольниками

# 2 Описание выбранных методов и алгоритмов визуализации

## 2.1 Алгоритмы аффинных преобразований

Аффинное преобразование – отображение плоскости или пространства в себя, при котором параллельные прямые переходят в параллельные прямые, пересекающиеся в пересекающиеся, скрещивающиеся в скрещивающиеся.

Алгоритм любого аффинного преобразования прост: необходимо взять однородные координаты точки (x, y и z) и умножить на матрицу преобразования.

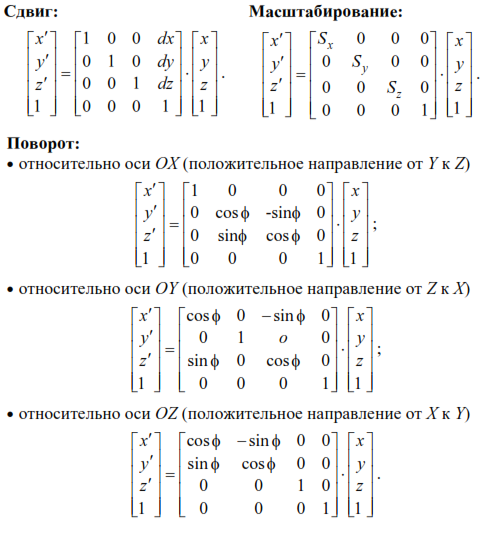


Рисунок 2.1 – Матрицы аффинных преобразований

Также важно отметить, что главный плюс аффинных преобразований в том, что их можно объединять в одну результирующую матрицу преобразований и уже её применять для модификации точек.

Например, можно отдельно высчитать матрицу преобразований из локальных в мировые координаты, из мировых координат в координаты камеры и из координат камеры в координаты экрана. Последовательно их перемножить и получим одну матрицу, которая будет переводить сразу из локальных координат в экранные.

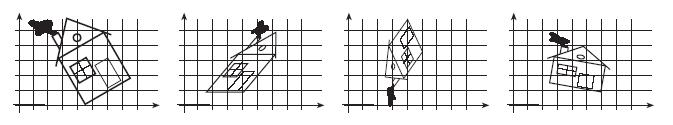


Рисунок 2.2 – Пример аффинных преобразований

## 2.2 Алгоритм удаления скрытых линий и поверхностей

Для удаления невидимых поверхностей используется алгоритм z-буфер. Это одно из решений «проблемы видимости». Он очень эффективен и, практически, не имеет недостатков, если он используется в аппаратных средствах. Программно, есть и другие методы, которые могут конкурировать с ним: z-сортировка («алгоритм художника») и двоичное разбиение пространства (BSP), но они также имеют свои преимущества и недостатки. Основным недостатком z-буферизации является потребление большого объема памяти: в работе используется «буфер глубины». Для его реализации в программе задействован буфер размером в ширину на высоту поля для отрисовки.

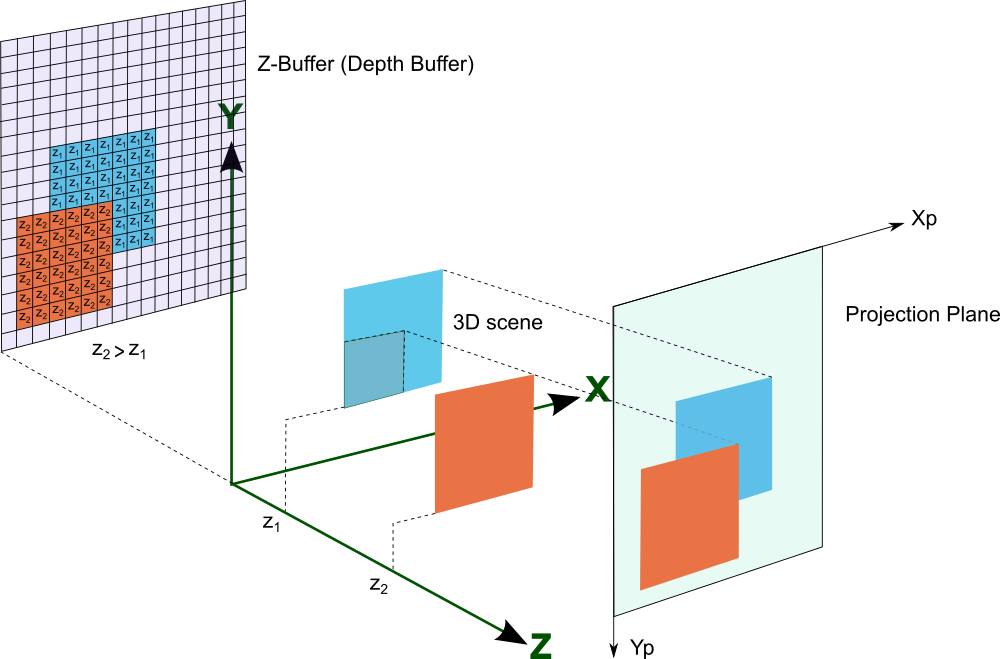


Рисунок 2.3 – Пример использования буфера глубины

## 2.3 Алгоритмы проекционных преобразований

Проекционный метод изображения предметов основан на их зрительном представлении. Если соединить все точки предмета прямыми линиями (проекционными лучами) с постоянной точкой О (центр проекции), в которой предполагается глаз наблюдателя, то на пересечении этих лучей с какой-либо плоскостью получается проекция всех точек предмета. Соединив эти точки прямыми линиями в том же порядке, как они соединены в предмете, получим на плоскости перспективное изображение предмета или центральную проекцию.

Если центр проекции бесконечно удалён от картинной плоскости, то говорят о параллельной проекции, а если при этом проекционные лучи падают перпендикулярно к плоскости — то об ортогональной проекции.

### 2.3.1 Алгоритм параллельного проецирования

В данной работе алгоритм параллельного проецирования предельно прост:

* 1. Координаты объекта переводятся в координаты камеры.
  2. Происходит отсечение по параллелепипеду видимости камеры.
  3. На экран проецируются точки, учитывая координаты х и у, а координата z отбрасывается.

### 2.3.2 Алгоритм центрального проецирования

Когда пучок проекторов исходит из заданного центра, то отрезки, бывшие изначально параллельными, на плоскости проекции уже не буду таковыми, за исключением случая, когда они лежат в плоскости, параллельной проекционной. При проецировании нескольких параллельных прямых их проекции классифицируются в зависимости от числа главных точек схода, которыми они обладают, а, следовательно, и от числа координатных осей, которые пересекает проекционная плоскость. На рисунке 2.3 приведены одно-, двух- и трёхточечная проекции параллелепипеда.

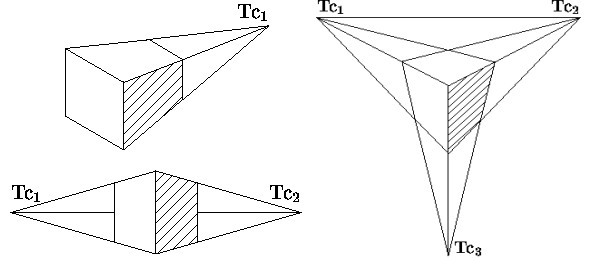


Рисунок 2.4 – Центральные (перспективные) проекции

В данной работе была использована одноточечная проекция. Алгоритм преобразования трёхмерных координат в координаты экраны при центральном проецировании очень похож на алгоритм для параллельного проецирования:

* 1. Координаты объекта переводятся в координаты камеры.
  2. Происходит отсечение по пирамиде видимости камеры.
  3. На экран проецируются точки, высчитывая новые координаты с помощью формул, использующих подобие треугольников.

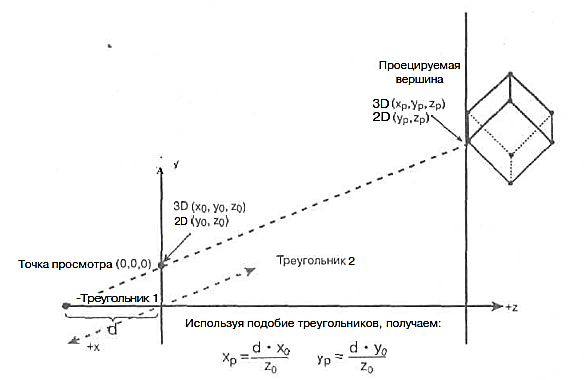


Рисунок 2.5 – Построение аксонометрического преобразования

## 2.4 Алгоритм произвольных видовых преобразований

Произвольное видовое преобразование подразумевает под собой произвольное расположение картинной плоскости, а значит расположение камеры так, как того желает пользователь.

В данном проекте существует несколько способов задания положения камеры:

* задание положения и точки цели камеры (сразу две, либо по отдельности);
* изменение углов поворота камеры (рыскание и тангаж);
* изменение дальности плоскостей отсечения и угла обзора.

Независимо от того, как задавалось положение камеры, каждый раз при перерисовке сцены камера перемещается в центр координат вместе со всеми объектами. Так как используется камера типа UVN, то все координаты объектов потом проецируются на базисные векторы камеры. И на выходе получаем объекты, находящиеся в координатах камеры, причём камера располагается по положительному направлению оси Z.

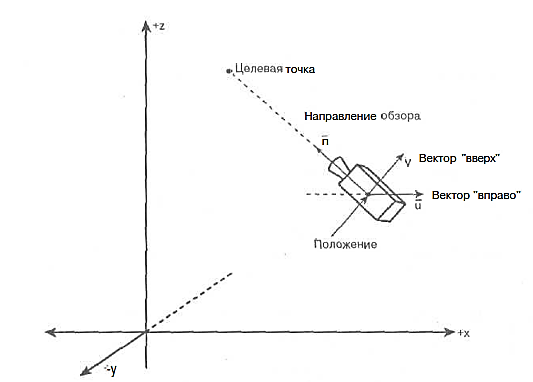


Рисунок 2.6 – Модель UVN-камеры

## 2.5 Алгоритм работы конвейера визуализации

Конвейер визуализации – это последовательная обработка данных, которые должны пройти через все этапы конвейера. Если рассмотреть конвейер визуализации данной программной системы, то можно увидеть, что на начальном этапе конвейер работает с целыми объектами, а уже под конец своей работы переходит на уровень граней и точек.

Первым этапом в данном конвейере для оптимизации стала проверка виден ли объект камере. Чтобы лишний раз не переводить все локальные координаты объекта в мировые переводится только базовая точка объекта, которая уже объявлена в мировых координатах, в координаты камеры. Проверка выполняется по отсекающей сфере, то есть входит ли окружающая объект сфера в пирамиду или параллелепипед видимости камеры. И если объект прошёл проверку, то осуществляется переход ко второму этапу, иначе осуществляется переход к первому этапу для другого объекта или выход, если других объектов нет.

Вторым этапом в данном конвейере является переход от локальных координат объекта к мировым и применение матриц преобразований. Так как все объекты созданы в локальных координатах, то переход к мировым сводится к перемещению всех точек к базовой точке в мировых координатах. На этом этапе достоверно известно, что объект виден камере.

Третий этап – это этап расчета освещения для грани. Так как в сцене может использоваться несколько источников света, на данном этапе вызывается отдельный метод.

Четвёртым этапом является переход мировых координат в координаты камеры и затем перевод координат камеры в координаты экрана.

Пятый этап заключается в отрисовке треугольников по полученным точкам с отсечением невидимых поверхностей по алгоритму z-буфера.

Эти пять этапов повторяются в цикле для каждого объекта сцены.

# 3 Разработка структур данных для хранения и описания объекта

## 3.1 Описание структур данных

Для эффективной работы программы были разработаны основные структуры данных. Так как был использован объектно-ориентированный язык программирования, то структуры данных были представлены классами данных.

Класс Point3D – это класс, содержащий три декартовы координаты точки.

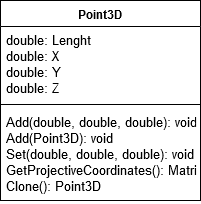


Рисунок 3.1 – Структура класса Point3D

От класса Point3D унаследован класс: Point3DSpherical. Он служит для представления точки в сферической системе координат. Point3DSpherical расширяет родителя радиусом r и углом поворота θ и добавляет угол подъёма φ.

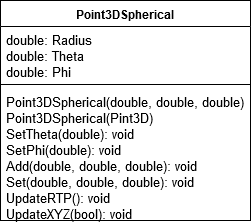


Рисунок 3.2 – Структура класса Point3DSpherical

Класс Face представляет собой грань, состоящую из трех точек. Это основной класс, с помощью которого происходит визуализация объектов.

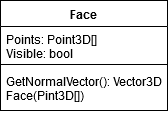


Рисунок 3.3 – Структура класса Face

Класс Primitive – это абстрактный класс, который является родительским для всех примитивов сцены, из которых можно создавать объекты сцены. У него есть базовая точка примитива, цвет фигуры и список граней.

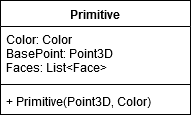


Рисунок 3.4 – Структура класса Primitive

Классы Box, Cylinder и Hemisphere являются потомками класса Primitive.

Класс Box расширяет «родителя», добавляя к нему ширину, длину и высоту.

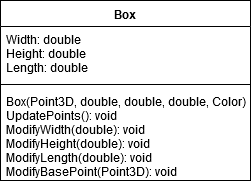


Рисунок 3.5 – Структура класса Box

Класс Cylinder расширяет «родителя» радиусом цилиндра, высотой и количеством сегментов.

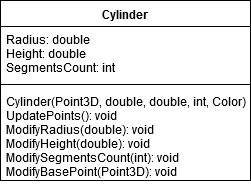


Рисунок 3.6 – Структура класса Cylinder

Класс Hemisphere расширяет «родителя» радиусом полусферы, количеством сегментов и флагом закрытия основания.

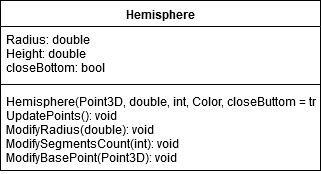


Рисунок 3.7 – Структура класса Hemisphere

Класс ModificationMatrix используется для возврата матриц модификации для различных аффинных преобразований.

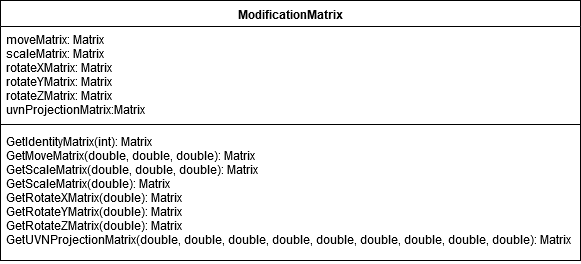


Рисунок 3.8 – Структура класса ModificationMatrix

Класс Scene представляет собой сцену, на которой могут располагаться объекты сцены. Она содержит флаг типа отрисовки (Wireframe, Solid, Z-Buffer), список объектов, список источников света, список камер, флаг отрисовки осей координат и их размеры.

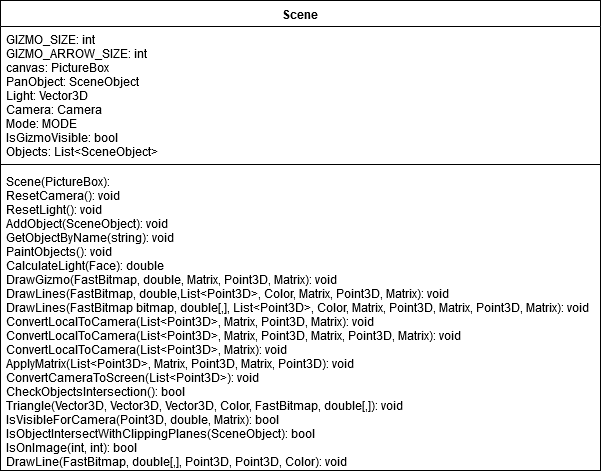


Рисунок 3.9 – Структура класса Scene

Класс Camera – это класс, который реализует UVN-камеру в программе. У него есть два экземпляра класса Point3DSpherical – это положение камеры и точка цели. Также угол обзора, ближняя и дальняя плоскости отсечения представлены типом double.

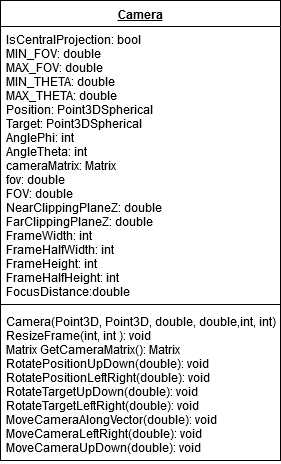


Рисунок 3.10 – Структура класса Camera

Класс SceneObject – это родительский класс для всех создаваемых фигур. Он обладает базовой точкой класса Point3D, уникальным именем, коэффициентом масштабирования, углами поворота. Также у него есть максимальный радиус, который используется для отсечения объектов и для контроля их пересечения. Данный класс содержит в себе список объектов типа ScenePrimitive, которые являются локальными составляющими объекта.

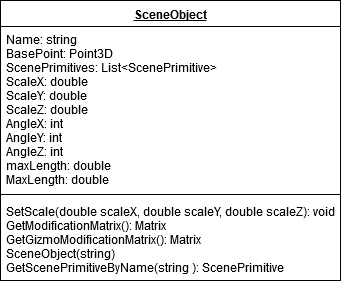


Рисунок 3.11 – Структура класса SceneObject

Объекты класса ScenePrimitive являются составляющими одного общего объекта. Данный класс содержит непосредственно список объектов класса Primitive, а также определяется базовой точкой класса Point3D, уникальным именем, коэффициентом масштабирования, углами поворота.

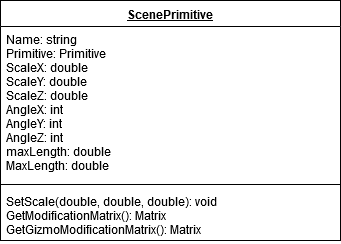


Рисунок 3.12 – Структура класса ScenePrimitive

Класс TriplaneObject наследует класс SceneObject и расширяет его параметрами объекта.

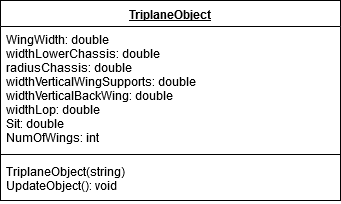


Рисунок 3.13 – Структура класса TriplaneObject

## 3.2 Диаграммы UML взаимодействия классов

Диаграммы UML помогают на этапе проектирования тщательно продумать структуру будущей программы, определить, какие функции нужны, а какие нет. Одна из таких диаграмм – это диаграмма прецедентов, она строится на начальном этапе проектирования. Она показывает, какие действия пользователь может осуществить с программой.

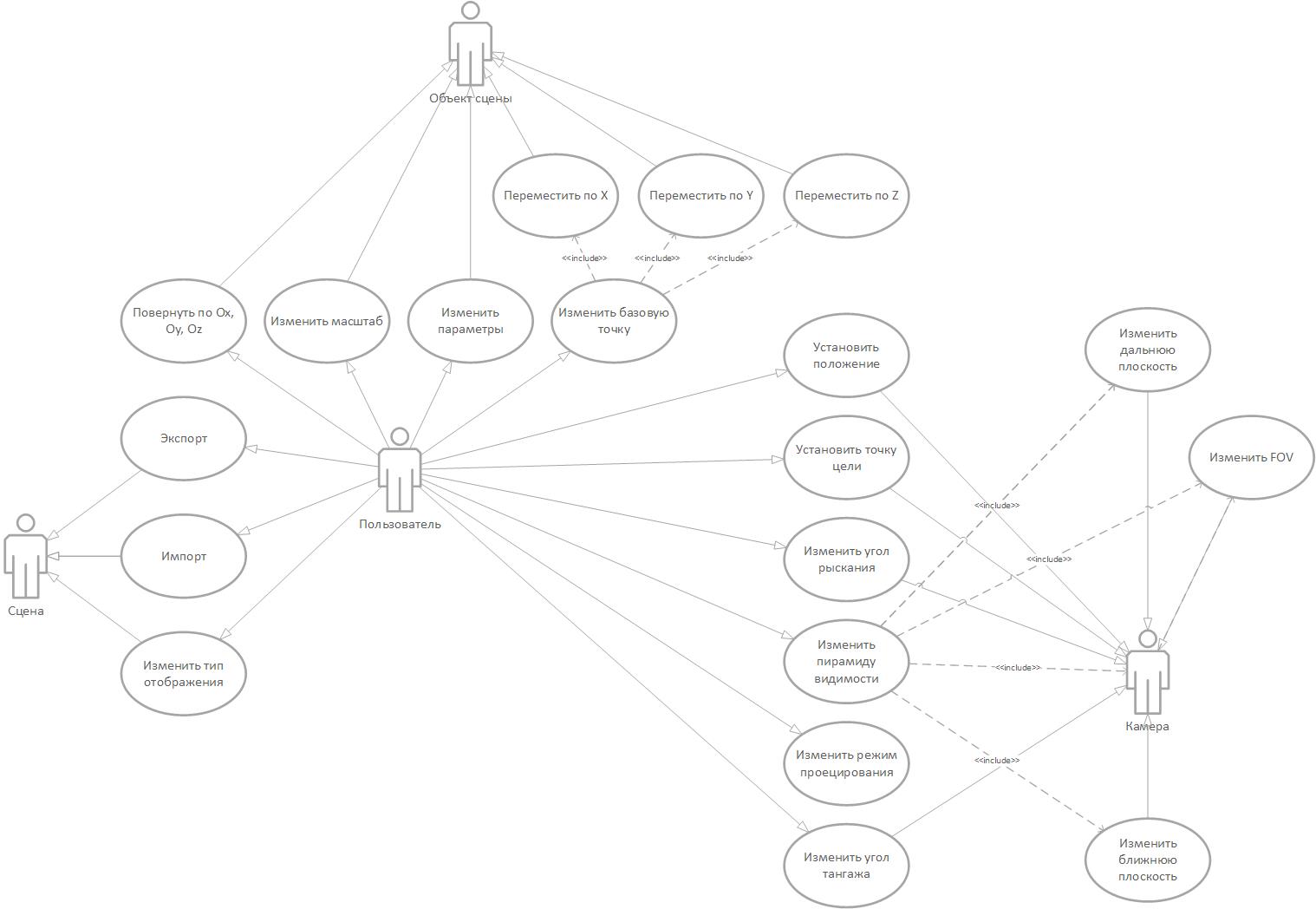


Рисунок 3.14 – Диаграмма прецедентов

Далее на рисунке 3.15 представлена диаграмма классов системы. На данной диаграмме у классов отсутствуют атрибуты и методы, так как они были представлены в подпункте 3.1.

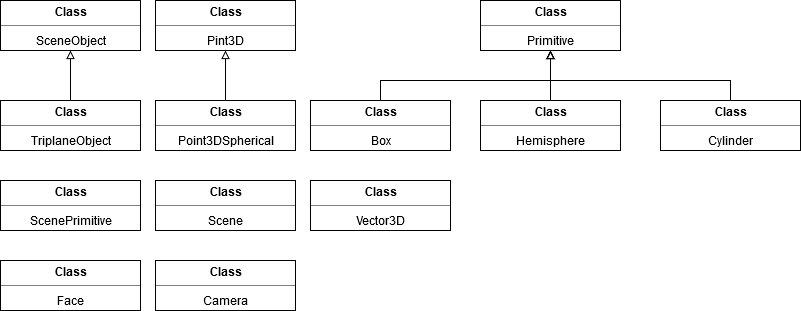


Рисунок 3.15 – Диаграмма классов

## 3.3 Реализация модификации объекта

Под модификацией объекта следует понимать масштабирование, перемещение и поворот. В данном проекте преобразование координат из локальных в экранные происходит каждый раз при перерисовке объекта, поэтому любые изменения сразу же отражаются на объекте.

У каждого объекта сцены есть свойства «коэффициент масштабирования», «угол поворота по Ох, Оу и Оz» и «базовая точка», которые может модифицировать пользователь.

Допустимые значения у коэффициента масштабирования от 0.1 до 10, у углов поворота от -180 градусов до 180. Координаты базовой точки ограничены максимальным значением типа double.

Матрица модификации из локальных координат в мировые высчитывается по такому алгоритму:

* 1. Создаётся единичная матрица размером 4х4.
  2. Она умножается на матрицу масштабирования.
  3. В случае если был поворот, то направляющие вектора объекта поворачиваются на новый угол.
  4. Умножается на матрицу проецирования на вектора направлений (аналог поворота).
  5. Умножается на матрицу сдвига с учётом базовой точки.

## 3.4 Реализация изменения параметров

Параметризация объекта возможна после добавления объекта на сцену.

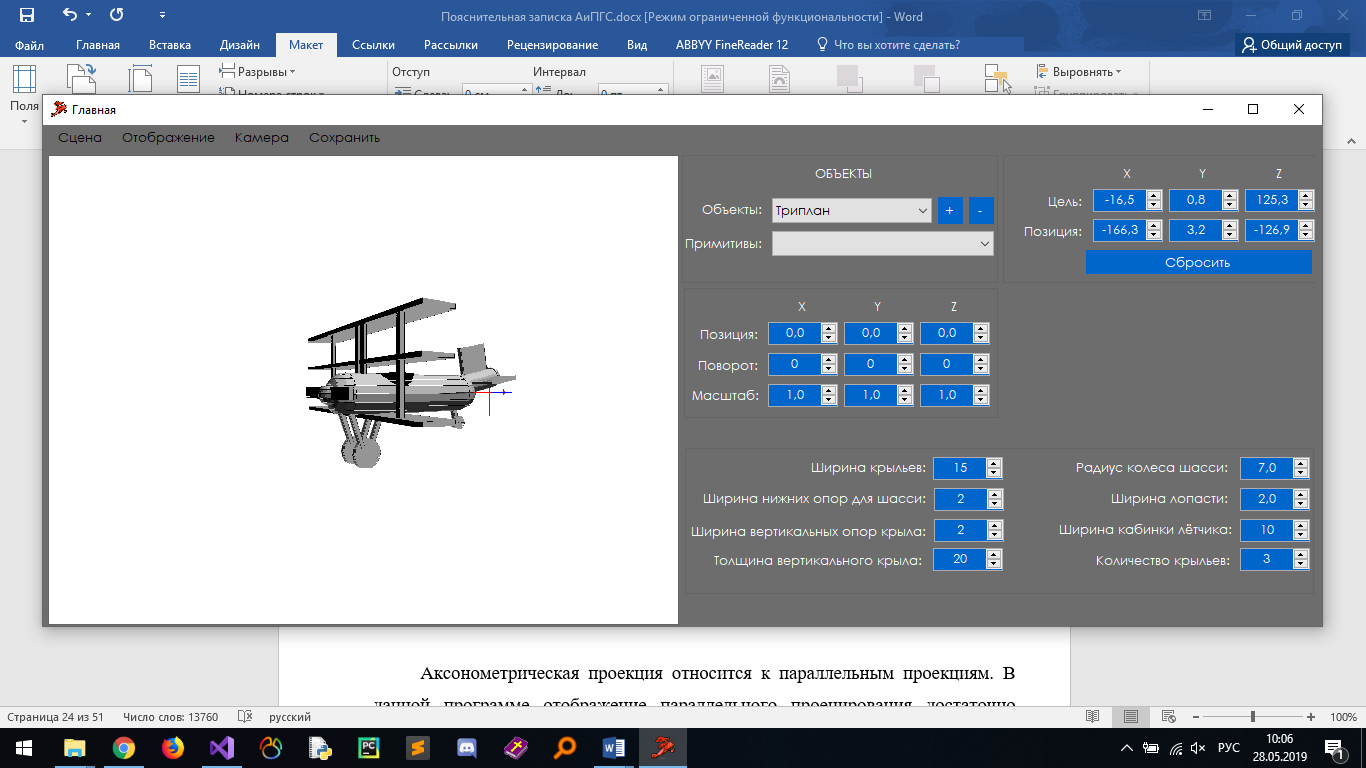


Рисунок 3.16 – Параметризация объекта

Пользователь может задавать параметры в пределе допустимых значений, которые задаются константами, исходя из здравого смысла (примитивы не должны пересекаться и объект должен выглядеть презентабельно).

## 3.5 Реализация аксонометрической и перспективной проекций

Аксонометрическая проекция относится к параллельным проекциям. В данной программе отображение параллельного проецирования достаточно тривиально: локальные координаты последовательно переводятся в мировые координаты, а потом в координаты камеры. После этого координата z просто отбрасывается и отображается на экран.

Перспективная проекция относится к центральным видам проекций, т.е. проекционные лучи направлены не параллельно друг другу, а сходятся в одной точке. Для корректного перевода в экранные координаты используется подобие треугольников, чтобы правильно спроецировать точки пространства на экран.

## 3.6 Реализация произвольных видовых преобразований (камеры)

Все виды произвольных видовых преобразований сводятся к одному и тому же действию – установка точки положения и точки цели камеры.

В ходе перерисовки изображения на экране каждый раз заново высчитывается матрица модификации камеры (матрица перевода из мировых координат в координаты камеры).

Изначально у нас есть две точки, по которым высчитывается вектор направления N. После этого вектору «вверх» V присваивается значение вертикального вектора {0, 1, 0}, так как ось Y направлена вверх. Высчитывается вектор «вправо» U с помощью векторного произведения N и V и потом высчитывается вектор V через векторное произведение векторов N и U. После этого происходит нормализация векторов и уже на их основе получаем матрицу проецирования на эти 3 вектора.

В программе реализованы повороты камеры вокруг точки цели и вокруг положения камеры. Для простоты используется сферическая система координат, так как в ней можно задавать угол поворота и угол подъёма.

Например, если будет происходить вращение вокруг точки цели, то сдвигаются обе точки камеры к началу координат со смещением точки цели так, что точка цели становится в начало координат. Потом берётся текущее положение камеры, считываются сферические углы и к ним добавляется новый угол, на который необходимо повернуть. После этого точки камеры сдвигаются с обратным смещением точки цели. Аналогичны действия и с вращением вокруг положения камеры.

## 3.7 Реализация сохранения сцены

Программная система позволяет пользователю сохранять состояние сцены и состояние объектов, находящихся на ней, в легко читаемый xml файл. В файле экспорта содержится описание камер, источников света, объектов и их модификаторов.

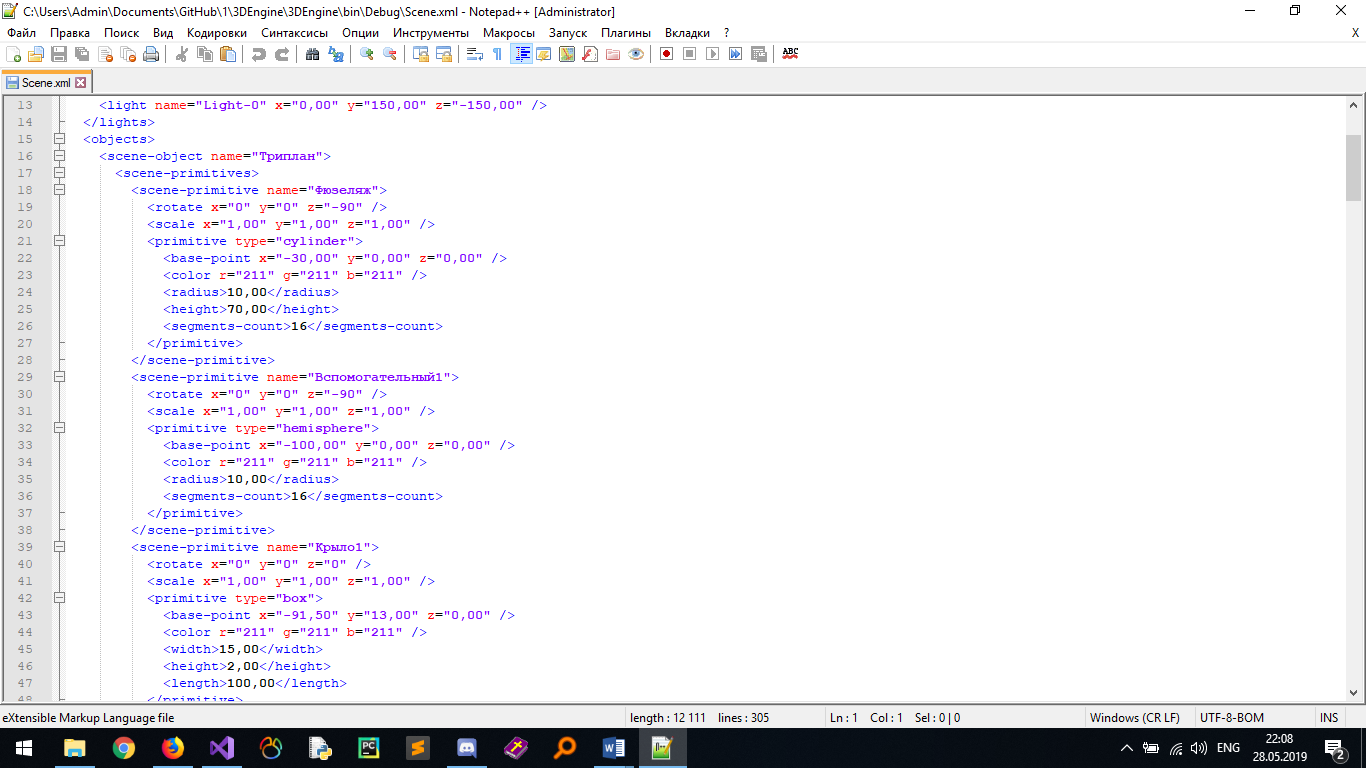


Рисунок 3.17 – Структура экспорта сцены

3.8 Реализация закраски объекта и удаления скрытых линий

Для закраски объектов и удаления скрытых линий используется алгоритм z-буфера. Теоретическая составляющая описана в подразделе 2.2.

## 3.9 Реализация панорамирования

Панорамирование – это кино- или видеосъёмка с плавным вращением камеры по горизонтали или вертикали.

В данной программной системе панорамирование сцены разработано так, что сначала вычисляется радиус описывающей сферы объекта, после этого устанавливается расстояние до дальней плоскости максимально возможным и до ближней минимальным, чтобы во время панорамирования не происходило отсечений, далее происходит установка камеры на позицию дальше радиуса описывающей сферы, устанавливается флаг панорамирования и затем каждые 30 миллисекунд вызывается метод поворота на 1 градус. Во время панорамирования остальные объекты не видны.

# 4 Тестирование программной системы

Разработанная программная система отвечает всем требованиям технического задания, предоставляет пользователю возможность работы с полигональной моделью «Триплан», возможность параметризации объекта, поворота, перемещения, масштабирования, удаления и видовых преобразований. Программа имеет удобный пользовательский оконный интерфейс на английском языке.

Программа была разработана так, чтобы максимально обезопасить пользователя от некорректного ввода данных. Например, при вводе дополнительных параметров объекта на экран выведены их минимальные и максимальные значения, что позволяет пользователю сразу ориентироваться во вводимых данных.

# Выводы

В ходе выполнения курсового проекта была разработана программа «Triplane», которая позволяет управлять параметризированными объектами сцены и камерами.

Были изучены и реализованы различные алгоритмы работы с графикой: алгоритм z-буфера, алгоритм отсечения объектов в пространстве камеры. Был изучен алгоритм работы конвейера визуализации.

Спроектированная и реализованная программная система обладает следующими достоинствами: дружелюбный интерфейс программы, возможность управления несколькими объектами, удобная параметризация объектов, понятный XML формат файла сцены, возможность сохранения и загрузки сцены, прямой манипуляцией мыши на экране.

Но в то же время программе присущи недостатки: используется отсечение по сфере, которое чаще всего даёт неправильные результаты при обнаружении пересечении объектов.

Программный продукт был разработан на языке C#, с использованием объектно-ориентированного подхода. Для корректного функционирования приложения необходим .NET Framework версии 4.6.1.

# Перечень ссылок

* 1. Трёхмерная графика: [Электронный ресурс] // wikipedia.org. URL: https://ru.wikipedia.org/wiki/Трёхмерная\_графика. (Дата обращения: 19.04.2019).
  2. Аффинное преобразование: [Электронный ресурс] // wikipedia.org. URL: https://ru.wikipedia.org/wiki/Аффинное\_преобразование. (Дата обращения: 23.04.2019).
  3. Приступа А. В. Компьютерная графика. Алгоритмические основы и базовые технологии: учебное пособие. – Томск: Изд-во НТЛ, 2012. -260с.
  4. Ламот Андре. Программирование трёхмерных игр для Windows. Советы профессионала по трёхмерной графике и растеризации: Пер. с англ. — М.: Издательский дом "Вильямc", 2004. — 1424 с.

# Приложение А Техническое задание

# Приложение Б Руководство пользователя

Для запуска программы необходимо запустить файл «Triplane.exe».

После запуска приложения у вас появится окно, которое представлено на рисунке Б.1. В главном окне отображена слева сцена, на которой будут размещаться объекты. Справа находится панель управления, с помощью которой можно манипулировать камерами, модификацией объектов и источниками света.

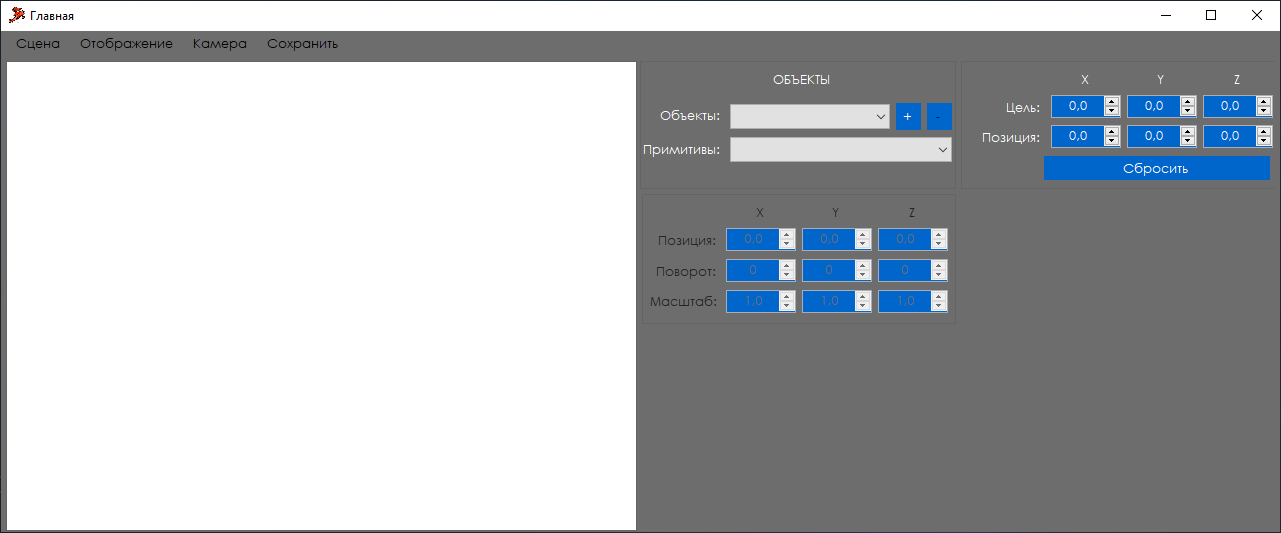


Рисунок Б.1 – Главное окно программы

Для того, чтобы добавить в сцену новый объект необходимо нажать на кнопку с плюсом возле выпадающего списка Объекты.

Далее откроется окно добавления объекта, как показано на рисунке Б.2. На нем необходимо ввести уникальное имя объекта, задать базовую позицию, углы поворота и растягивания по осям.

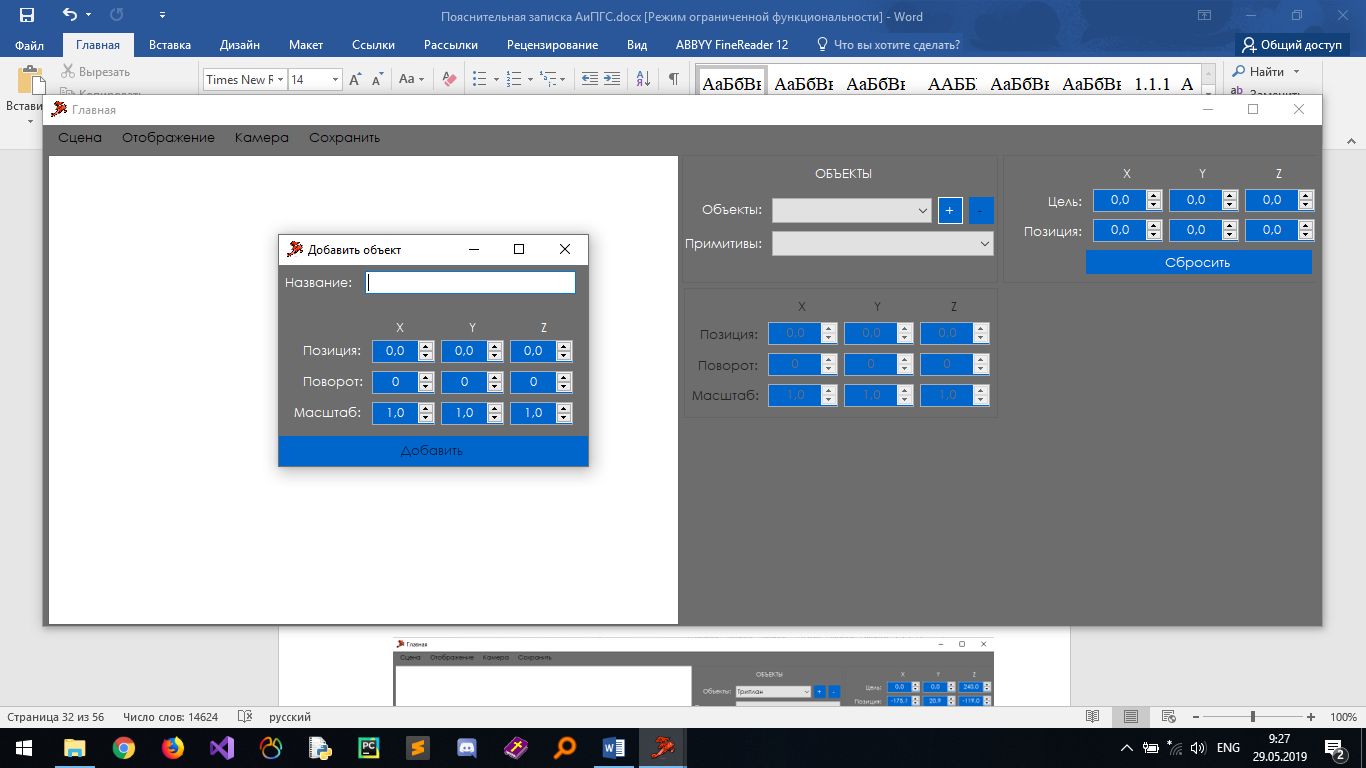


Рисунок Б.2 – Окно добавления объекта

После нажатия кнопки «Добавить» при успешном вводе данных объект появится в сцене.

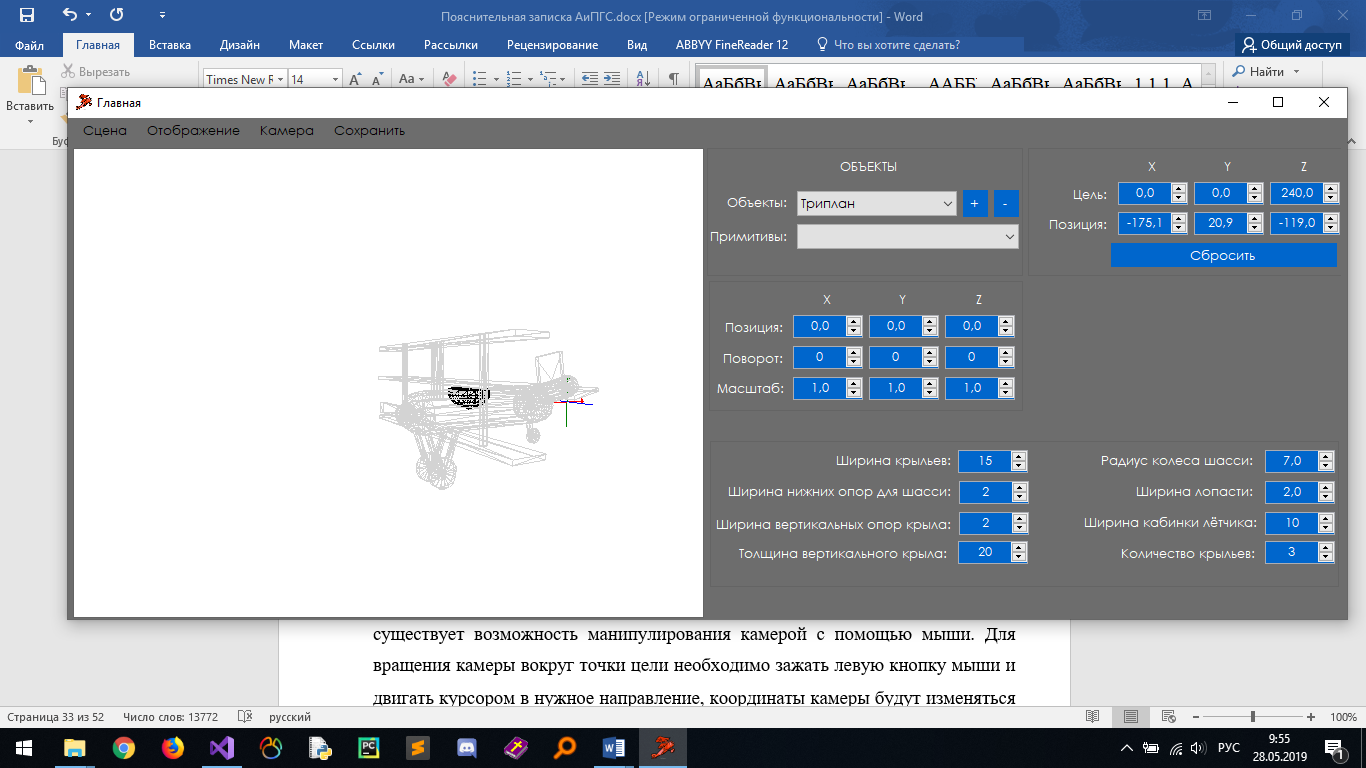


Рисунок Б.3 – Сцена с добавленным объектом

После добавления объекта типа игровой консоли в сцену и выборе его из выпадающего списка, на форме будут доступны параметры для изменения.

Для манипулирования камерой необходимо либо задать новую точку цели или позицию камеры, либо изменить угол рыскания или тангажа. Также существует возможность манипулирования камерой с помощью мыши. Для вращения камеры вокруг точки цели необходимо зажать левую кнопку мыши и двигать курсором в нужное направление, координаты камеры будут изменяться в соответствии с манипуляцией. Если предварительно нажать колесико мыши или выбрать Позиция в выпадающем списке Поворот, то вращение будет вокруг позиции камеры. Также при зажатии правой кнопки мыши камеру можно перетаскивать вдоль осей, что является удобным при просмотре сцены.

Чтобы изменить режим заливки необходимо в верхнем левом выпадающем списке выбрать режим Объёмное или же обратно Каркасное.

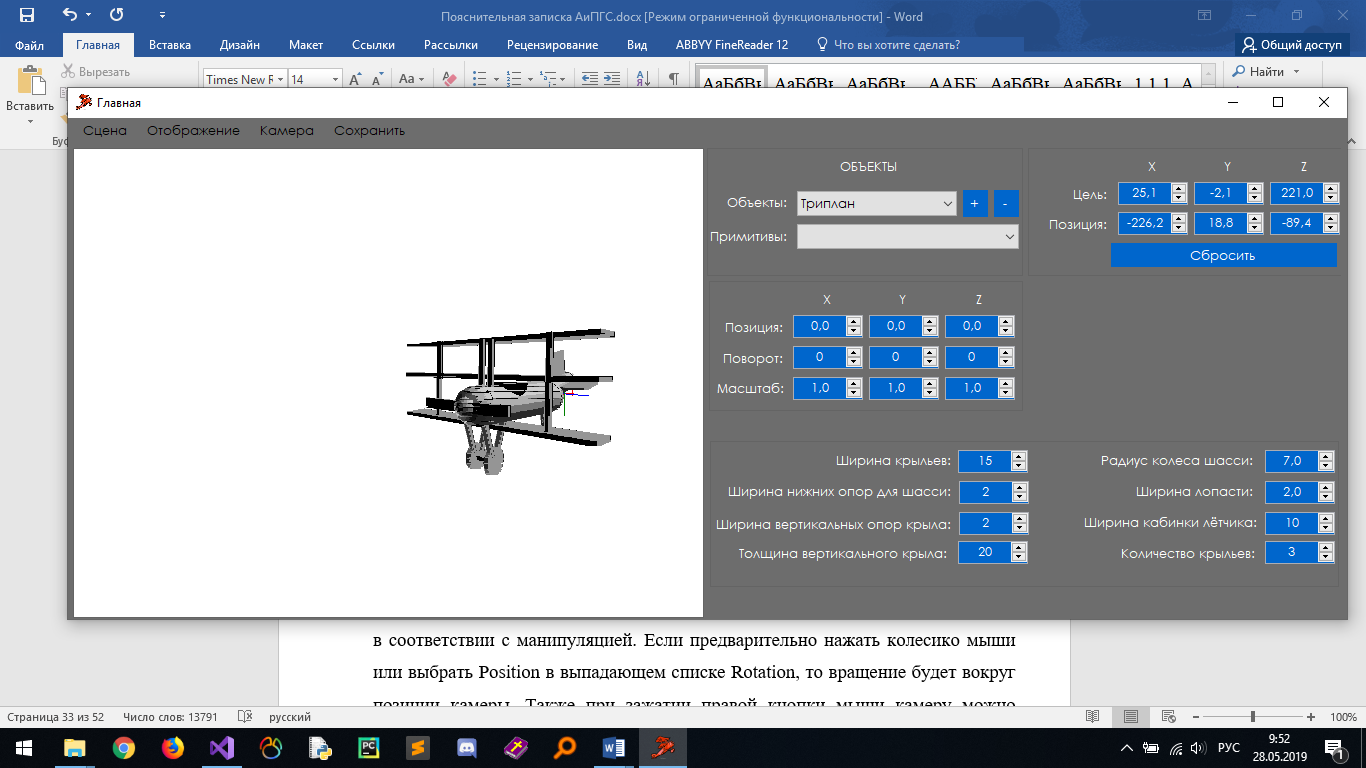


Рисунок Б.4 – Отображение объекта с заполнением треугольников

Также, для тестирования z-буфера в программе был создан дополнительный режим просмотра, отображающий карту глубин сцены.

После того, как нужный объект выбран из списка, можно задавать коэффициенты масштабирования, углы поворота вокруг осей и положение объекта на сцене, задавая его базовую точку.

Для сохранения сцены необходимо нажать кнопку «Экспорт», далее будет запрошен путь сохранения и имя файла. Для загрузки будет использована кнопка «Импорт».

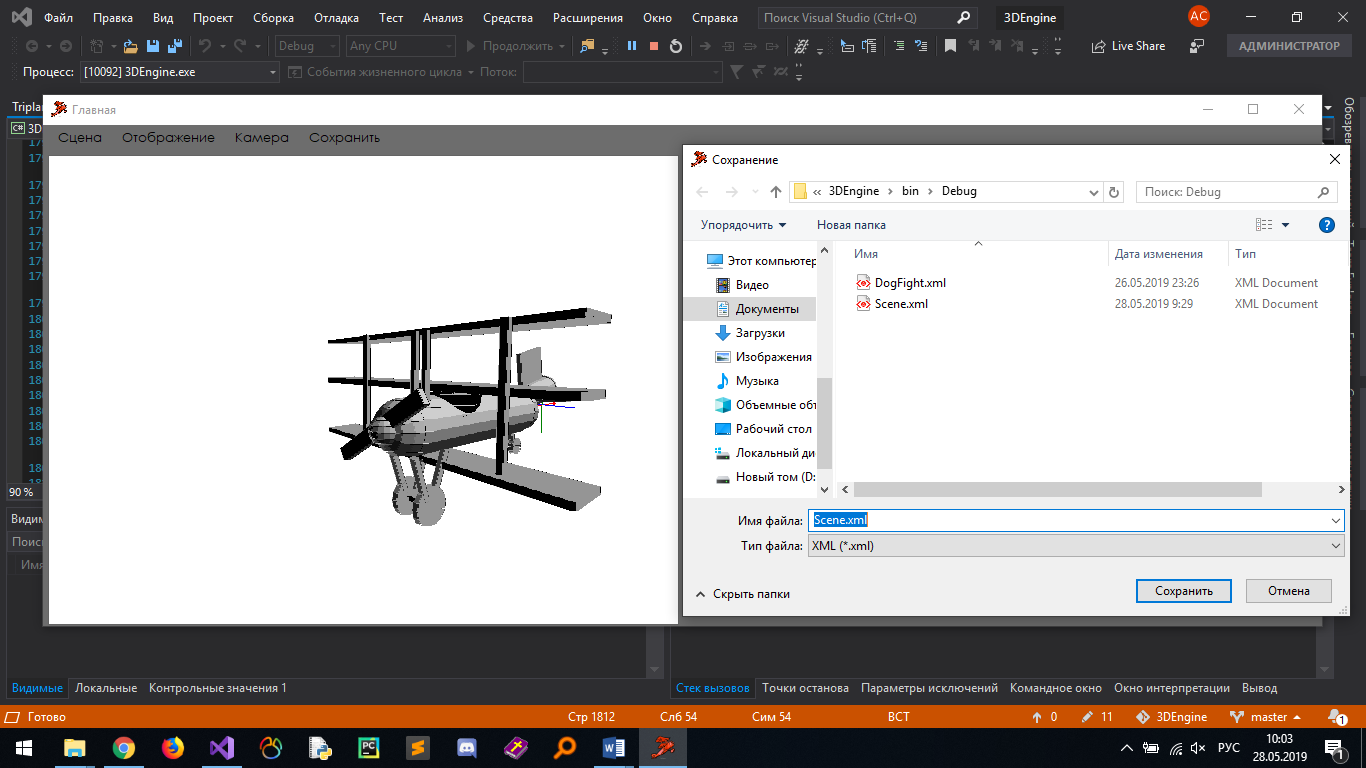


Рисунок Б.5 – Сохранение сцены в файл

Лишние объекты могут быть удалены со сцены, либо нажатием кнопки с минусом после выбора удаляемого объекта или нажатием кнопки «Reset Scene» для удаления всех объектов и сброса всех камер с источниками света.

Также на сцене могут быть созданы произвольные объекты, отдельные части которых могут изменяться непосредственно на форме при выборе частей из списка. При выборе пустой части объекта, будет модифицироваться сам объект.

# Приложение В Листинг программы

Camera.cs

using System;

namespace Game\_Consoles{

public class Camera{

public bool IsCentralProjection { get; set; } = true;

public const double MIN\_FOV = 60.0;

public const double MAX\_FOV = 135.0;

public const double MIN\_THETA = 1;

public const double MAX\_THETA = 179;

public Point3DSpherical Position;

public Point3DSpherical Target;

public int AnglePhi { get; private set; } = 270;

public int AngleTheta { get; private set; } = 90;

private Matrix cameraMatrix = ModificationMatrix.GetIdentityMatrix(4);

private double fov = 90;

public double FOV{

get{

return fov;}

set{

fov = Math.Min(Math.Max(value, MIN\_FOV), MAX\_FOV);}}

public double NearClippingPlaneZ { get; set; }

public double FarClippingPlaneZ { get; set; }

public int FrameWidth { get; private set; }

public int FrameHalfWidth { get; private set; }

public int FrameHeight { get; private set; }

public int FrameHalfHeight { get; private set; }

public double FocusDistance { get { return FrameWidth / 2.0 \* Math.Tan(MathHelp.DegreesToRadians(fov / 2)); } }

public Camera(){}

public Camera(

Point3D position, Point3D target,

double nearClippingPaneZ, double farClippingPaneZ,

int frameWidth, int frameHeight){

Position = new Point3DSpherical(position);

Target = new Point3DSpherical(target);

NearClippingPlaneZ = nearClippingPaneZ;

FarClippingPlaneZ = farClippingPaneZ;

ResizeFrame(frameWidth, frameHeight);}

public void ResizeFrame(int width, int height){

FrameWidth = width;

FrameHalfWidth = FrameWidth / 2;

FrameHeight = height;

FrameHalfHeight = FrameHeight / 2;}

public Matrix GetCameraMatrix(){

Matrix matrixInverted = ModificationMatrix.GetMoveMatrix(-Position.X, -Position.Y, -Position.Z);

Vector3D N = new Vector3D(Target.X - Position.X, Target.Y - Position.Y, Target.Z - Position.Z);

Vector3D V = new Vector3D(0, 1, 0);

Vector3D U = V ^ N;

V = U ^ N;

U.Normalize();

V.Normalize();

N.Normalize();

cameraMatrix = ModificationMatrix.GetUVNProjectionMatrix(

U.X, U.Y, U.Z,

V.X, V.Y, V.Z,

N.X, N.Y, N.Z);

cameraMatrix = cameraMatrix.Multiply(matrixInverted);

return cameraMatrix;}

public void RotatePositionUpDown(double angle){

Matrix rotateMatrix = ModificationMatrix.GetMoveMatrix(-Target.X, -Target.Y, -Target.Z);

rotateMatrix = rotateMatrix.Multiply(Position.GetProjectiveCoordinates());

Position.Set(rotateMatrix[0, 0], rotateMatrix[1, 0], rotateMatrix[2, 0]);

angle += MathHelp.RadiansToDegrees(Position.Theta);

angle = Math.Min(Math.Max(angle, MIN\_THETA), MAX\_THETA);

Position.SetTheta(MathHelp.DegreesToRadians(angle));

AngleTheta = 180 - (int)angle;

rotateMatrix = ModificationMatrix.GetMoveMatrix(Target.X, Target.Y, Target.Z);

rotateMatrix = rotateMatrix.Multiply(Position.GetProjectiveCoordinates());

Position.Set(rotateMatrix[0, 0], rotateMatrix[1, 0], rotateMatrix[2, 0]);}

public void RotatePositionLeftRight(double angle){

Matrix rotateMatrix = ModificationMatrix.GetMoveMatrix(-Target.X, -Target.Y, -Target.Z);

rotateMatrix = rotateMatrix.Multiply(Position.GetProjectiveCoordinates());

Position.Set(rotateMatrix[0, 0], rotateMatrix[1, 0], rotateMatrix[2, 0]);

angle += MathHelp.RadiansToDegrees(Position.Phi);

angle %= 360;

angle = angle < 0 ? 360 + angle : angle;

Position.SetPhi(MathHelp.DegreesToRadians(angle));

AnglePhi = (int)angle;

rotateMatrix = ModificationMatrix.GetMoveMatrix(Target.X, Target.Y, Target.Z);

rotateMatrix = rotateMatrix.Multiply(Position.GetProjectiveCoordinates());

Position.Set(rotateMatrix[0, 0], rotateMatrix[1, 0], rotateMatrix[2, 0]);}

public void RotateTargetUpDown(double angle){

angle = MathHelp.DegreesToRadians(angle);

Matrix rotateMatrix = ModificationMatrix.GetMoveMatrix(-Position.X, -Position.Y, -Position.Z);

rotateMatrix = rotateMatrix.Multiply(Target.GetProjectiveCoordinates());

Target.Set(rotateMatrix[0, 0], rotateMatrix[1, 0], rotateMatrix[2, 0]);

angle += MathHelp.RadiansToDegrees(Target.Theta);

angle = Math.Min(Math.Max(angle, MIN\_THETA), MAX\_THETA);

Target.SetTheta(MathHelp.DegreesToRadians(angle));

AngleTheta = 180 - (int)angle;

rotateMatrix = ModificationMatrix.GetMoveMatrix(Position.X, Position.Y, Position.Z);

rotateMatrix = rotateMatrix.Multiply(Target.GetProjectiveCoordinates());

Target.Set(rotateMatrix[0, 0], rotateMatrix[1, 0], rotateMatrix[2, 0]);}

public void RotateTargetLeftRight(double angle){

Matrix rotateMatrix = ModificationMatrix.GetMoveMatrix(-Position.X, -Position.Y, -Position.Z);

rotateMatrix = rotateMatrix.Multiply(Target.GetProjectiveCoordinates());

Target.Set(rotateMatrix[0, 0], rotateMatrix[1, 0], rotateMatrix[2, 0]);

angle += MathHelp.RadiansToDegrees(Target.Phi);

angle %= 360;

angle = angle < 0 ? 360 + angle : angle;

Target.SetPhi(MathHelp.DegreesToRadians(angle));

AnglePhi = (int)angle;

rotateMatrix = ModificationMatrix.GetMoveMatrix(Position.X, Position.Y, Position.Z);

rotateMatrix = rotateMatrix.Multiply(Target.GetProjectiveCoordinates());

Target.Set(rotateMatrix[0, 0], rotateMatrix[1, 0], rotateMatrix[2, 0]);}

public void MoveCameraAlongVector(double distance){

Vector3D vector = new Vector3D(Target) - new Vector3D(Position);

vector.Normalize();

vector = vector \* distance;

Target.Add(vector.X, vector.Y, vector.Z);

Position.Add(vector.X, vector.Y, vector.Z);}

public void MoveCameraLeftRight(double distance){

Vector3D vector = new Vector3D(Target) - new Vector3D(Position);

vector.Normalize();

vector = vector ^ new Vector3D(0, 1, 0);

vector = vector \* distance;

Target.Add(vector.X, vector.Y, vector.Z);

Position.Add(vector.X, vector.Y, vector.Z);}

public void MoveCameraUpDown(double distance){

Vector3D vector = new Vector3D(Target) - new Vector3D(Position);

vector.Normalize();

vector = vector ^ (vector ^ new Vector3D(0, 1, 0));

vector = vector \* distance;

Target.Add(vector.X, vector.Y, vector.Z);

Position.Add(vector.X, vector.Y, vector.Z);}}}

Box.cs

using System.Drawing;

using System.Collections.Generic;

namespace Game\_Consoles{

public class Box : Primitive{

public double Width { get; private set; }

public double Height { get; private set; }

public double Length { get; private set; }

public Box(Point3D basePoint, double width, double height, double length, Color color) : base(basePoint, color){

Width = width;

Height = height;

Length = length;

UpdatePoints();}

private void UpdatePoints(){

double x = Width / 2;

double z = Length / 2;

List<Point3D> points = new List<Point3D>();

points.Add(new Point3D(-x, 0, -z));

points.Add(new Point3D(-x, 0, z));

points.Add(new Point3D(x, 0, z));

points.Add(new Point3D(x, 0, -z));

points.Add(new Point3D(-x, Height, -z));

points.Add(new Point3D(-x, Height, z));

points.Add(new Point3D(x, Height, z));

points.Add(new Point3D(x, Height, -z));

Faces.Clear();

Faces.Add(new Face(new Point3D[] { points[7], points[4], points[0] }));

Faces.Add(new Face(new Point3D[] { points[0], points[3], points[7] }));

Faces.Add(new Face(new Point3D[] { points[6], points[7], points[3] }));

Faces.Add(new Face(new Point3D[] { points[2], points[6], points[3] }));

Faces.Add(new Face(new Point3D[] { points[5], points[6], points[2] }));

Faces.Add(new Face(new Point3D[] { points[1], points[5], points[2] }));

Faces.Add(new Face(new Point3D[] { points[4], points[5], points[1] }));

Faces.Add(new Face(new Point3D[] { points[0], points[4], points[1] }));

Faces.Add(new Face(new Point3D[] { points[2], points[3], points[0] }));

Faces.Add(new Face(new Point3D[] { points[1], points[2], points[0] }));

Faces.Add(new Face(new Point3D[] { points[6], points[4], points[7] }));

Faces.Add(new Face(new Point3D[] { points[6], points[5], points[4] }));}

public void ModifyWidth(double width){

Width = width;

UpdatePoints();}

public void ModifyHeight(double height){

Height = height;

UpdatePoints();}

public void ModifyLength(double length){

Length = length;

UpdatePoints();}

public void ModifyBasePoint(Point3D basePoint){

BasePoint = basePoint;

UpdatePoints();}}}

Vector3D.cs

using System;

namespace Game\_Consoles{

public class Vector3D{

public double X;

public double Y;

public double Z;

public double Length { get { return (float)Math.Sqrt(X \* X + Y \* Y + Z \* Z); } }

public Vector3D(){

X = 0;

Y = 0;

Z = 0;}

public Vector3D(double x, double y, double z){

X = x;

Y = y;

Z = z;}

public Vector3D(Vector3D Vector3D){

X = Vector3D.X;

Y = Vector3D.Y;

Z = Vector3D.Z;}

public Vector3D(Point3D point3D){

X = point3D.X;

Y = point3D.Y;

Z = point3D.Z;}

public void Normalize(){

double length = Length;

double scale = 1 / length;

if (length != 0){

X \*= scale;

Y \*= scale;

Z \*= scale;}}

public Vector3D GetNormalized(){

Vector3D result = new Vector3D(X, Y, Z);

double length = Length;

double scale = 1 / length;

if (length != 0){

result.X \*= scale;

result.Y \*= scale;

result.Z \*= scale;}

return result;}

public Vector3D CrossProduct(Vector3D vector){

return new Vector3D(

Y \* vector.Z - Z \* vector.Y,

Z \* vector.X - X \* vector.Z,

X \* vector.Y - Y \* vector.X

);}

public static Vector3D CrossProduct(Vector3D first, Vector3D second){

return new Vector3D(

first.Y \* second.Z - first.Z \* second.Y,

first.Z \* second.X - first.X \* second.Z,

first.X \* second.Y - first.Y \* second.X

);}

public static Vector3D operator ^(Vector3D first, Vector3D second){

Vector3D result = new Vector3D(

first.Y \* second.Z - first.Z \* second.Y,

first.Z \* second.X - first.X \* second.Z,

first.X \* second.Y - first.Y \* second.X

);

return result;}

public double DotProduct(Vector3D vector){

return X \* vector.X + Y \* vector.Y + Z \* vector.Z;}

public static double DotProduct(Vector3D first, Vector3D second){

return first.X \* second.X + first.Y \* second.Y + first.Z \* second.Z;}

public static double operator \*(Vector3D first, Vector3D second){

return first.X \* second.X + first.Y \* second.Y + first.Z \* second.Z;}

public static double AngleCos(Vector3D first, Vector3D second){

return (first \* second) / Math.Max((first.Length \* second.Length), 1);}

public static double Angle(Vector3D first, Vector3D second){

return Math.Acos(AngleCos(first, second));}

public static Vector3D operator +(Vector3D first, Vector3D second){

Vector3D result = new Vector3D(

first.X + second.X,

first.Y + second.Y,

first.Z + second.Z

);

return result;}

public static Vector3D operator -(Vector3D first, Vector3D second){

Vector3D result = new Vector3D(

first.X - second.X,

first.Y - second.Y,

first.Z - second.Z

);

return result;}

public static Vector3D operator \*(Vector3D Vector3D, double scale){

return new Vector3D(Vector3D.X \* scale, Vector3D.Y \* scale, Vector3D.Z \* scale);}

public static Vector3D operator /(Vector3D Vector3D, double scale){

return new Vector3D(Vector3D.X / scale, Vector3D.Y / scale, Vector3D.Z / scale);}}}

Sphere.cs

using System;

using System.Drawing;

namespace Game\_Consoles{

public class Sphere : Primitive{

public double Radius { get; private set; }

public int SegmentsCount { get; private set; }

public Sphere(Point3D basePoint, double radius, int segmentsCount, Color color) : base(basePoint, color){

Radius = radius;

SegmentsCount = Math.Max(segmentsCount, 4);

UpdatePoints();}

private void UpdatePoints(){

int quater = Math.Max(SegmentsCount / 4 - 1, 1);

double angle = 360.0 / SegmentsCount;

double sin = Math.Sin(MathHelp.DegreesToRadians(-angle));

double cos = Math.Cos(MathHelp.DegreesToRadians(-angle));

Point3D[,] points = new Point3D[quater, SegmentsCount + 1];

points[0, 0] = new Point3D(-Radius, 0, 0);

for (int i = 1; i < quater; ++i){

double x = points[i - 1, 0].X \* cos - points[i - 1, 0].Y \* sin;

double y = points[i - 1, 0].X \* sin + points[i - 1, 0].Y \* cos;

points[i, 0] = new Point3D(x, y, 0);}

Faces.Clear();

for (int i = 1; i <= SegmentsCount; ++i){

for (int j = 0; j < quater; ++j){

double x = points[j, i - 1].X \* cos - points[j, i - 1].Z \* sin;

double z = points[j, i - 1].X \* sin + points[j, i - 1].Z \* cos;

points[j, i] = new Point3D(x, points[j, i - 1].Y, z);

if (i > 0){

if (j > 0){

Faces.Add(new Face(new Point3D[] {

new Point3D(points[j - 1, i - 1].X, points[j - 1, i - 1].Y, points[j - 1, i - 1].Z),

new Point3D(points[j - 1, i].X, points[j - 1, i].Y, points[j - 1, i].Z),

new Point3D(points[j, i - 1].X, points[j, i - 1].Y, points[j, i - 1].Z)}));

Faces.Add(new Face(new Point3D[] {

new Point3D(points[j, i - 1].X, points[j, i - 1].Y, points[j, i - 1].Z),

new Point3D(points[j, i].X, points[j, i].Y, points[j, i].Z),

new Point3D(points[j - 1, i].X, points[j - 1, i].Y, points[j - 1, i].Z)}));

Faces.Add(new Face(new Point3D[] {

new Point3D(points[j - 1, i - 1].X, -points[j - 1, i - 1].Y, points[j - 1, i - 1].Z),

new Point3D(points[j - 1, i].X, -points[j - 1, i].Y, points[j - 1, i].Z),

new Point3D(points[j, i - 1].X, -points[j, i - 1].Y, points[j, i - 1].Z)}));

Faces.Add(new Face(new Point3D[] {

new Point3D(points[j, i - 1].X, -points[j, i - 1].Y, points[j, i - 1].Z),

new Point3D(points[j, i].X, -points[j, i].Y, points[j, i].Z),

new Point3D(points[j - 1, i].X, -points[j - 1, i].Y, points[j - 1, i].Z)}));}

if (j == quater - 1){

Faces.Add(new Face(new Point3D[] {

new Point3D(points[j, i - 1].X, points[j, i - 1].Y, points[j, i - 1].Z),

new Point3D(points[j, i].X, points[j, i].Y, points[j, i].Z),

new Point3D(0, Radius, 0)}));

Faces.Add(new Face(new Point3D[] {

new Point3D(points[j, i - 1].X, -points[j, i - 1].Y, points[j, i - 1].Z),

new Point3D(points[j, i].X, -points[j, i].Y, points[j, i].Z),

new Point3D(0, -Radius, 0)}));}}}}}

public void ModifyRadius(double radius){

Radius = radius;

UpdatePoints();}

public void ModifySegmentsCount(int segmentsCount){

SegmentsCount = segmentsCount;

UpdatePoints();}

public void ModifyBasePoint(Point3D basePoint){

BasePoint = basePoint;

UpdatePoints();}}}

ScenePrimitive.cs

using System;

namespace Game\_Consoles{

public class ScenePrimitive{

public string Name { get; private set; }

public Primitive Primitive { get; set; }

public ScenePrimitive(Primitive primitive, string name){

Primitive = primitive;

Name = name;}

public double ScaleX { get; private set; } = 1.0;

public double ScaleY { get; private set; } = 1.0;

public double ScaleZ { get; private set; } = 1.0;

public int AngleX { get; set; } = 0;

public int AngleY { get; set; } = 0;

public int AngleZ { get; set; } = 0;

private double maxLength = 0;

public double MaxLength{

get{

if (maxLength == 0){

double result = 0;

foreach (Face face in Primitive.Faces){

foreach (Point3D point in face.Points){

double length = point.Length;

result = Math.Max(length, result);}}

maxLength = result \* Math.Max(Math.Max(ScaleX, ScaleY), ScaleZ);}

return maxLength;}}

public void SetScale(double scaleX, double scaleY, double scaleZ){

ScaleX = scaleX;

ScaleY = scaleY;

ScaleZ = scaleZ;

maxLength = 0;}

public Matrix GetModificationMatrix(){

Matrix matrix = ModificationMatrix.GetIdentityMatrix(4);

matrix = matrix.Multiply(ModificationMatrix.GetScaleMatrix(ScaleX, ScaleY, ScaleZ));

matrix = matrix.Multiply(ModificationMatrix.GetRotateXMatrix(MathHelp.DegreesToRadians(AngleX)));

matrix = matrix.Multiply(ModificationMatrix.GetRotateYMatrix(MathHelp.DegreesToRadians(AngleY)));

matrix = matrix.Multiply(ModificationMatrix.GetRotateZMatrix(MathHelp.DegreesToRadians(AngleZ)));

return matrix;}

public Matrix GetGizmoModificationMatrix(){

Matrix matrix = ModificationMatrix.GetIdentityMatrix(4);

matrix = matrix.Multiply(ModificationMatrix.GetRotateXMatrix(MathHelp.DegreesToRadians(AngleX)));

matrix = matrix.Multiply(ModificationMatrix.GetRotateYMatrix(MathHelp.DegreesToRadians(AngleY)));

matrix = matrix.Multiply(ModificationMatrix.GetRotateZMatrix(MathHelp.DegreesToRadians(AngleZ)));

return matrix;}}}

SceneObject.cs

using System;

using System.Collections.Generic;

namespace Game\_Consoles{

public class SceneObject{

public string Name { get; private set; }

public Point3D BasePoint { get; set; }

public List<ScenePrimitive> ScenePrimitives { get; private set; } = new List<ScenePrimitive>();

public SceneObject(string name){

Name = name;}

public void AddScenePrimitive(ScenePrimitive scenePrimitive){

ScenePrimitives.Add(scenePrimitive);}

public ScenePrimitive GetScenePrimitiveByName(string name){

foreach (ScenePrimitive scenePrimitive in ScenePrimitives){

if (name.Equals(scenePrimitive.Name)){

return scenePrimitive;}}

return null;}

public double ScaleX { get; private set; } = 1.0;

public double ScaleY { get; private set; } = 1.0;

public double ScaleZ { get; private set; } = 1.0;

public int AngleX { get; set; } = 0;

public int AngleY { get; set; } = 0;

public int AngleZ { get; set; } = 0;

private double maxLength = 0;

public double MaxLength{

get{

if (maxLength == 0){

double result = 0;

foreach (ScenePrimitive primitive in ScenePrimitives){

result = Math.Max(primitive.MaxLength, result);}

maxLength = result \* Math.Max(Math.Max(ScaleX, ScaleY), ScaleZ);}

return maxLength;}}

public void SetScale(double scaleX, double scaleY, double scaleZ){

ScaleX = scaleX;

ScaleY = scaleY;

ScaleZ = scaleZ;

maxLength = 0;}

public Matrix GetModificationMatrix(){

Matrix matrix = ModificationMatrix.GetIdentityMatrix(4);

matrix = matrix.Multiply(ModificationMatrix.GetScaleMatrix(ScaleX, ScaleY, ScaleZ));

matrix = matrix.Multiply(ModificationMatrix.GetRotateXMatrix(MathHelp.DegreesToRadians(AngleX)));

matrix = matrix.Multiply(ModificationMatrix.GetRotateYMatrix(MathHelp.DegreesToRadians(AngleY)));

matrix = matrix.Multiply(ModificationMatrix.GetRotateZMatrix(MathHelp.DegreesToRadians(AngleZ)));

return matrix;}

public Matrix GetGizmoModificationMatrix(){

Matrix matrix = ModificationMatrix.GetIdentityMatrix(4);

matrix = matrix.Multiply(ModificationMatrix.GetRotateXMatrix(MathHelp.DegreesToRadians(AngleX)));

matrix = matrix.Multiply(ModificationMatrix.GetRotateYMatrix(MathHelp.DegreesToRadians(AngleY)));

matrix = matrix.Multiply(ModificationMatrix.GetRotateZMatrix(MathHelp.DegreesToRadians(AngleZ)));

return matrix;}}}

Scene.cs

using System;

using System.Drawing;

using System.Windows.Forms;

using System.Collections.Generic;

namespace Game\_Consoles{

public class Scene{

private const int GIZMO\_SIZE = 15;

private const int GIZMO\_ARROW\_SIZE = 2;

private PictureBox canvas;

public enum MODE{

WIREFRAME = 0,

SOLID = 1,

Z\_BUFFER = 2}

public SceneObject PanObject = null;

public Vector3D Light { get; set; }

public List<Vector3D> Lights { get; set; } = new List<Vector3D>();

public Camera Camera { get; set; }

public List<Camera> Cameras { get; set; } = new List<Camera>();

public MODE Mode { get; set; } = MODE.WIREFRAME;

public bool IsGizmoVisible { get; set; } = true;

public List<SceneObject> Objects { get; private set; } = new List<SceneObject>();

public Scene(PictureBox canvas){

this.canvas = canvas;

Camera = new Camera();

ResetCamera();

Cameras.Add(Camera);

Lights.Add(new Vector3D(0, 150, -150));

Light = Lights[0];}

public void ResetCamera(){

Camera = new Camera(

new Point3D(0, 0, -200),

new Point3D(0, 0, 200),

10,

1000.0,

canvas.ClientSize.Width,

canvas.ClientSize.Height

);}

public void ResetLight(){

Light = new Vector3D(0, 150, -150);}

public void AddObject(SceneObject sceneObject){

Objects.Add(sceneObject);}

public SceneObject GetObjectByName(string name){

foreach (SceneObject sceneObject in Objects){

if (name.Equals(sceneObject.Name)){

return sceneObject;}}

return null;}

public void PaintObjects(){

FastBitmap bitmap = new FastBitmap(new Bitmap(Camera.FrameWidth, Camera.FrameHeight));

bitmap.Fill(Color.White);

double[,] buffer = new double[Camera.FrameWidth, Camera.FrameHeight];

for (int i = 0; i < Camera.FrameWidth; ++i)

for (int j = 0; j < Camera.FrameHeight; ++j)

buffer[i, j] = double.MaxValue;

Matrix cameraMatrix = Camera.GetCameraMatrix();

if (PanObject == null){

if (IsGizmoVisible && IsVisibleForCamera(new Point3D(0, 0, 0), GIZMO\_SIZE, cameraMatrix)){

DrawGizmo(bitmap, buffer, ModificationMatrix.GetIdentityMatrix(4), new Point3D(0, 0, 0), cameraMatrix);}

Matrix lightMatrix = ModificationMatrix.GetIdentityMatrix(4);

foreach (Vector3D light in Lights){

Point3D lightPosition = new Point3D(light);

if (IsVisibleForCamera(lightPosition, lightPosition.Length, cameraMatrix)){

Sphere lightSphere = new Sphere(lightPosition, 3, 4, Color.Yellow);

foreach (Face face in lightSphere.Faces){

List<Point3D> points = new List<Point3D>();

foreach (Point3D point in face.Points){

points.Add(point.Clone());}

DrawLines(bitmap, buffer, points, lightSphere.Color, lightMatrix, lightPosition, lightMatrix, lightPosition, cameraMatrix);}}}}

foreach (SceneObject sceneObject in Objects){

if (IsVisibleForCamera(sceneObject.BasePoint, sceneObject.MaxLength, cameraMatrix) && (PanObject == null || PanObject.Equals(sceneObject))){

Matrix objectMatrix = sceneObject.GetModificationMatrix();

Matrix gizmoModificationMatrix = sceneObject.GetGizmoModificationMatrix();

if (Mode == MODE.WIREFRAME && IsGizmoVisible){

DrawGizmo(bitmap, buffer, gizmoModificationMatrix, sceneObject.BasePoint, cameraMatrix);}

foreach (ScenePrimitive scenePrimitive in sceneObject.ScenePrimitives){

Matrix primitiveMatrix = scenePrimitive.GetModificationMatrix();

foreach (Face face in scenePrimitive.Primitive.Faces){

List<Point3D> points = new List<Point3D>();

foreach (Point3D point in face.Points){

points.Add(point.Clone());}

if (Mode == MODE.WIREFRAME){

DrawLines(bitmap, buffer, points, scenePrimitive.Primitive.Color, primitiveMatrix, scenePrimitive.Primitive.BasePoint, objectMatrix, sceneObject.BasePoint, cameraMatrix);}

else{

ApplyMatrix(points, primitiveMatrix, scenePrimitive.Primitive.BasePoint, objectMatrix, sceneObject.BasePoint);

double brightness = CalculateLight(new Face(new Point3D[] { points[0], points[1], points[2] }));

Color color = Color.FromArgb(

(int)(scenePrimitive.Primitive.Color.R \* brightness),

(int)(scenePrimitive.Primitive.Color.G \* brightness),

(int)(scenePrimitive.Primitive.Color.B \* brightness)

);

ConvertLocalToCamera(points, cameraMatrix);

ConvertCameraToScreen(points);

Triangle(new Vector3D(points[0]), new Vector3D(points[1]), new Vector3D(points[2]), color, bitmap, buffer);}}}}}

if (Mode == MODE.Z\_BUFFER){

FastBitmap bufferImage = new FastBitmap(new Bitmap(canvas.ClientSize.Width, canvas.ClientSize.Height));

double minimum = double.MaxValue;

double maximum = double.MinValue;

for (int i = 0; i < Camera.FrameWidth; ++i)

for (int j = 0; j < Camera.FrameHeight; ++j){

if (buffer[i, j] != double.MinValue){

minimum = Math.Min(minimum, buffer[i, j]);}

if (buffer[i, j] != double.MaxValue){

maximum = Math.Max(maximum, buffer[i, j]);}}

double step = (maximum - minimum) / 255;

for (int i = 0; i < Camera.FrameWidth; ++i)

for (int j = 0; j < Camera.FrameHeight; ++j){

int gray = buffer[i, j] == double.MaxValue ?

0 :

buffer[i, j] == double.MinValue ?

255 :

255 - (int)((buffer[i, j] - minimum) / step);

bufferImage.SetPixel(i, j, Color.FromArgb(gray, gray, gray));}

canvas.Image = bufferImage.GetBitmap();}

else{

canvas.Image = bitmap.GetBitmap();}}

private double CalculateLight(Face face){

double result = 0;

Vector3D normal = face.GetNormalVector();

foreach (Vector3D light in Lights){

result += Math.Abs(Vector3D.AngleCos(normal, light));}

return Math.Min(result, 1);}

private void DrawGizmo(FastBitmap bitmap, double[,] buffer, Matrix objectMatrix, Point3D basePoint, Matrix cameraMatrix){

List<Point3D> axisX = new List<Point3D>(){

new Point3D(-GIZMO\_SIZE, 0, 0),

new Point3D(GIZMO\_SIZE, 0, 0)};

DrawLines(bitmap, buffer, axisX, Color.Red, objectMatrix, basePoint, cameraMatrix);

List<Point3D> axisXArrow = new List<Point3D>(){

new Point3D(GIZMO\_SIZE, 0, 0),

new Point3D(GIZMO\_SIZE - GIZMO\_ARROW\_SIZE, GIZMO\_ARROW\_SIZE, 0),

new Point3D(GIZMO\_SIZE - GIZMO\_ARROW\_SIZE, -GIZMO\_ARROW\_SIZE, 0)};

DrawLines(bitmap, buffer, axisXArrow, Color.Red, objectMatrix, basePoint, cameraMatrix);

List<Point3D> axisY = new List<Point3D>(){

new Point3D(0, -GIZMO\_SIZE, 0),

new Point3D(0, GIZMO\_SIZE, 0)};

DrawLines(bitmap, buffer, axisY, Color.Green, objectMatrix, basePoint, cameraMatrix);

List<Point3D> axisYArrow = new List<Point3D>(){

new Point3D(0, GIZMO\_SIZE, 0),

new Point3D(GIZMO\_ARROW\_SIZE, GIZMO\_SIZE - GIZMO\_ARROW\_SIZE, 0),

new Point3D(-GIZMO\_ARROW\_SIZE, GIZMO\_SIZE - GIZMO\_ARROW\_SIZE, 0)};

DrawLines(bitmap, buffer, axisYArrow, Color.Green, objectMatrix, basePoint, cameraMatrix);

List<Point3D> axisZ = new List<Point3D>(){

new Point3D(0, 0, -GIZMO\_SIZE),

new Point3D(0, 0, GIZMO\_SIZE)};

DrawLines(bitmap, buffer, axisZ, Color.Blue, objectMatrix, basePoint, cameraMatrix);

List<Point3D> axisZArrow = new List<Point3D>(){

new Point3D(0, 0, GIZMO\_SIZE),

new Point3D(GIZMO\_ARROW\_SIZE, 0, GIZMO\_SIZE - GIZMO\_ARROW\_SIZE),

new Point3D(-GIZMO\_ARROW\_SIZE, 0, GIZMO\_SIZE - GIZMO\_ARROW\_SIZE)};

DrawLines(bitmap, buffer, axisZArrow, Color.Blue, objectMatrix, basePoint, cameraMatrix);}

private void DrawLines(FastBitmap bitmap, double[,] buffer, List<Point3D> points, Color color, Matrix objectMatrix, Point3D basePoint, Matrix cameraMatrix){

ConvertLocalToCamera(points, objectMatrix, basePoint, cameraMatrix);

ConvertCameraToScreen(points);

for (int pi = 0; pi < points.Count - 1; ++pi){

for (int pj = pi + 1; pj < points.Count; ++pj){

DrawLine(bitmap, buffer, points[pi], points[pj], color);}}}

private void DrawLines(FastBitmap bitmap, double[,] buffer, List<Point3D> points, Color color, Matrix primitiveMatrix, Point3D primitiveBasePoint, Matrix objectMatrix, Point3D objectBasePoint, Matrix cameraMatrix){

ConvertLocalToCamera(points, primitiveMatrix, primitiveBasePoint, objectMatrix, objectBasePoint, cameraMatrix);

ConvertCameraToScreen(points);

for (int pi = 0; pi < points.Count - 1; ++pi){

for (int pj = pi + 1; pj < points.Count; ++pj){

DrawLine(bitmap, buffer, points[pi], points[pj], color);}}}

private void ConvertLocalToCamera(List<Point3D> points, Matrix objectMatrix, Point3D basePoint, Matrix cameraMatrix){

foreach (Point3D point in points){

Matrix result = objectMatrix.Multiply(point.GetProjectiveCoordinates());

point.Set(result[0, 0] + basePoint.X, result[1, 0] + basePoint.Y, result[2, 0] + basePoint.Z);

result = cameraMatrix.Multiply(point.GetProjectiveCoordinates());

point.Set(result[0, 0], result[1, 0], result[2, 0]);}}

private void ConvertLocalToCamera(List<Point3D> points, Matrix primitiveMatrix, Point3D primitiveBasePoint, Matrix objectMatrix, Point3D objectBasePoint, Matrix cameraMatrix){

foreach (Point3D point in points){

Matrix result = primitiveMatrix.Multiply(point.GetProjectiveCoordinates());

point.Set(result[0, 0] + primitiveBasePoint.X, result[1, 0] + primitiveBasePoint.Y, result[2, 0] + primitiveBasePoint.Z);

result = objectMatrix.Multiply(point.GetProjectiveCoordinates());

point.Set(result[0, 0] + objectBasePoint.X, result[1, 0] + objectBasePoint.Y, result[2, 0] + objectBasePoint.Z);

result = cameraMatrix.Multiply(point.GetProjectiveCoordinates());

point.Set(result[0, 0], result[1, 0], result[2, 0]);}}

private void ConvertLocalToCamera(List<Point3D> points, Matrix cameraMatrix){

foreach (Point3D point in points){

Matrix result = cameraMatrix.Multiply(point.GetProjectiveCoordinates());

point.Set(result[0, 0], result[1, 0], result[2, 0]);}}

private void ApplyMatrix(List<Point3D> points, Matrix primitiveMatrix, Point3D primitiveBasePoint, Matrix objectMatrix, Point3D objectBasePoint){

foreach (Point3D point in points){

Matrix result = primitiveMatrix.Multiply(point.GetProjectiveCoordinates());

point.Set(result[0, 0] + primitiveBasePoint.X, result[1, 0] + primitiveBasePoint.Y, result[2, 0] + primitiveBasePoint.Z);

result = objectMatrix.Multiply(point.GetProjectiveCoordinates());

point.Set(result[0, 0] + objectBasePoint.X, result[1, 0] + objectBasePoint.Y, result[2, 0] + objectBasePoint.Z);}}

private void ConvertCameraToScreen(List<Point3D> points){

foreach (Point3D point in points){

double x = point.X;

double y = point.Y;

double z = point.Z;

if (Camera.IsCentralProjection && z != 0){

x = Camera.FocusDistance \* x / z;

y = Camera.FocusDistance \* y / z;}

x += Camera.FrameHalfWidth;

y += Camera.FrameHalfHeight;

point.Set((int)x, (int)y, z);}}

public bool CheckObjectsIntersection(){

bool intersecting = false;

for (int i = 0; i < Objects.Count - 1 && !intersecting; ++i){

SceneObject first = Objects[i];

Point3D a = first.BasePoint;

for (int j = i + 1; j < Objects.Count && !intersecting; ++j){

SceneObject second = Objects[j];

Point3D b = second.BasePoint;

double distance = Math.Sqrt((a.X - b.X) \* (a.X - b.X) + (a.Y - b.Y) \* (a.Y - b.Y) + (a.Z - b.Z) \* (a.Z - b.Z));

double size = first.MaxLength + second.MaxLength;

intersecting = distance < size;}}

return intersecting;}

private void Triangle(Vector3D p0, Vector3D p1, Vector3D p2, Color color, FastBitmap bitmap, double[,] buffer){

if (p0.Y != p1.Y || p0.Y != p2.Y){

if (p0.Y > p1.Y){

MathHelp.Swap(ref p0, ref p1);}

if (p0.Y > p2.Y){

MathHelp.Swap(ref p0, ref p2);}

if (p1.Y > p2.Y){

MathHelp.Swap(ref p1, ref p2);}

int totalHeight = (int)Math.Round(p2.Y - p0.Y);

for (int i = 0; i < totalHeight; ++i){

bool secondHalf = i > p1.Y - p0.Y || p1.Y == p0.Y;

int segmentHeight = (int)Math.Round(secondHalf ? p2.Y - p1.Y : p1.Y - p0.Y);

double alpha = (double)i / totalHeight;

double beta = (i - (secondHalf ? p1.Y - p0.Y : 0.0)) / segmentHeight;

Vector3D a = p0 + (p2 - p0) \* alpha;

Vector3D b = (secondHalf ? p1 + (p2 - p1) \* beta : p0 + (p1 - p0) \* beta);

if (a.X > b.X){

MathHelp.Swap(ref a, ref b);}

for (int j = (int)Math.Round(a.X); j <= (int)Math.Round(b.X); ++j){

double scale = (a.X == b.X) ? 1 : (j - a.X) / (b.X - a.X);

Vector3D p = a + (b - a) \* scale;

int x = (int)Math.Round(p.X);

int y = (int)Math.Round(p.Y);

if (IsOnImage(x, y) && buffer[x, y] >= p.Z){

buffer[x, y] = p.Z;

bitmap.SetPixel(x, y, color);}}}}}

private bool IsVisibleForCamera(Point3D basePoint, double maxLength, Matrix cameraMatrix){

bool result = false;

Point3D point = basePoint.Clone();

Matrix cameraToBase = cameraMatrix.Multiply(point.GetProjectiveCoordinates());

point.Set(cameraToBase[0, 0], cameraToBase[1, 0], cameraToBase[2, 0]);

if (point.Z + maxLength < Camera.FarClippingPlaneZ && point.Z - maxLength > Camera.NearClippingPlaneZ){

if (Camera.IsCentralProjection){

double testZ = Camera.FrameHalfWidth \* point.Z / Camera.FocusDistance;

if (point.X - maxLength < testZ && point.X + maxLength > -testZ){

testZ = Camera.FrameHalfHeight \* point.Z / Camera.FocusDistance;

if (point.Y - maxLength < testZ && point.Y + maxLength > -testZ){

result = true;}}}

else{

if (Math.Abs(point.X) + maxLength < Camera.FrameHalfWidth && Math.Abs(point.Y) + maxLength < Camera.FrameHalfHeight){

result = true;}}}

return result;}

public bool IsObjectIntersectWithClippingPlanes(SceneObject sceneObject){

bool result = true;

Point3D point = sceneObject.BasePoint.Clone();

Matrix cameraToBase = Camera.GetCameraMatrix().Multiply(point.GetProjectiveCoordinates());

point.Set(cameraToBase[0, 0], cameraToBase[1, 0], cameraToBase[2, 0]);

if (point.Z + sceneObject.MaxLength < Camera.FarClippingPlaneZ && point.Z - sceneObject.MaxLength > Camera.NearClippingPlaneZ){

result = false;}

return result;}

private bool IsOnImage(int x, int y){

return x > 0 && x < canvas.ClientSize.Width && y > 0 && y < canvas.ClientSize.Height;}

private void DrawLine(FastBitmap bitmap, double[,] buffer, Point3D start, Point3D end, Color color){

double dx = end.X - start.X;

double dy = end.Y - start.Y;

if (Math.Abs(dx) > Math.Abs(dy)){

if (dx < 0){

MathHelp.Swap(ref start, ref end);}

double[] ys = MathHelp.Interpolate(start.X, start.Y, end.X, end.Y);

double[] zs = MathHelp.Interpolate(start.X, start.Z, end.X, end.Z);

for (int x = (int)start.X; x <= (int)end.X; ++x){

int y = (int)ys[(int)(x - start.X)];

double z = zs[(int)(x - start.X)];

if (IsOnImage(x, y) && buffer[x, y] >= z){

buffer[x, y] = z;

bitmap.SetPixel(x, y, color);}}}

else{

if (dy < 0){

MathHelp.Swap(ref start, ref end);}

double[] xs = MathHelp.Interpolate(start.Y, start.X, end.Y, end.X);

double[] zs = MathHelp.Interpolate(start.Y, start.Z, end.Y, end.Z);

for (int y = (int)start.Y; y <= (int)end.Y; ++y){

int x = (int)xs[(int)(y - start.Y)];

double z = zs[(int)(y - start.Y)];

if (IsOnImage(x, y) && buffer[x, y] >= z){

buffer[x, y] = z;

bitmap.SetPixel(x, y, color);}}}}}}

Primitive.cs

using System.Drawing;

using System.Collections.Generic;

namespace Game\_Consoles{

public class Primitive{

public Color Color;

public Point3D BasePoint;

public List<Face> Faces { get; protected set; } = new List<Face>();

public Primitive(Point3D basePoint, Color color){

BasePoint = basePoint;

Color = color;}}}

Point3DSpherical.cs

using System;

namespace Game\_Consoles{

public class Point3DSpherical : Point3D{

public double Radius { get; set; }

public double Theta { get; private set; }

public void SetTheta(double value){

Theta = Math.Max(value, 0);

UpdateXYZ();}

public double Phi { get; private set; }

public void SetPhi(double value){

Phi = value;

UpdateXYZ(false);}

public Point3DSpherical(double radius, double theta, double phi) : base(0, 0, 0){

Radius = radius;

Theta = Math.Max(Theta, 0);

Phi = phi;

UpdateXYZ();}

private void UpdateXYZ(bool updateY = true){

X = Radius \* Math.Sin(Theta) \* Math.Cos(Phi);

if (updateY){

Y = Radius \* Math.Cos(Theta);}

Z = Radius \* Math.Sin(Theta) \* Math.Sin(Phi);}

private void UpdateRTP(){

Radius = Math.Sqrt(X \* X + Y \* Y + Z \* Z);

Theta = Math.Max(Math.Acos(Y / Radius), 0);

Phi = Math.Atan(Z / X);}

public Point3DSpherical(Point3D point) : base(point){

Set(point.X, point.Y, point.Z);}

public override void Add(double dx, double dy, double dz){

base.Add(dx, dy, dz);

UpdateRTP();}

public override void Set(double x, double y, double z){

base.Set(x, y, z);

UpdateRTP();

if (X < 0){

Phi = Math.PI + Phi;}

else if (Z < 0){

Phi = 2 \* Math.PI + Phi;}}}}

Point3D.cs

using System;

namespace Game\_Consoles{

public class Point3D{

public double X { get; set; }

public double Y { get; set; }

public double Z { get; set; }

public double Length { get { return (float)Math.Sqrt(X \* X + Y \* Y + Z \* Z); } }

public Point3D(double x, double y, double z){

X = x;

Y = y;

Z = z;}

public Point3D(Point3D point){

X = point.X;

Y = point.Y;

Z = point.Z;}

public Point3D(Vector3D vector){

X = vector.X;

Y = vector.Y;

Z = vector.Z;}

public virtual void Add(double dx, double dy, double dz){

X += dx;

Y += dy;

Z += dz;}

public virtual void Add(Point3D point){

X += point.X;

Y += point.Y;

Z += point.Z;}

public virtual void Set(double x, double y, double z){

X = x;

Y = y;

Z = z;}

public Matrix GetProjectiveCoordinates(){

double[] result = new double[4];

result[0] = X;

result[1] = Y;

result[2] = Z;

result[3] = 1.0;

return new Matrix(result);}

public Point3D Clone(){

return new Point3D(this);}}}

ModificationMatrix.cs

using System;

namespace Game\_Consoles{

public class ModificationMatrix{

private static Matrix moveMatrix = GetIdentityMatrix(4);

private static Matrix scaleMatrix = GetIdentityMatrix(4);

private static Matrix rotateXMatrix = GetIdentityMatrix(4);

private static Matrix rotateYMatrix = GetIdentityMatrix(4);

private static Matrix rotateZMatrix = GetIdentityMatrix(4);

private static Matrix uvnProjectionMatrix = GetIdentityMatrix(4);

public static Matrix GetIdentityMatrix(int n){

double[,] result = new double[n, n];

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

for (int j = 0; j < n; ++j) {

if (i == j) {

result[i, j] = 1.0;} else {

result[i, j] = 0.0;}}}

return new Matrix(result);}

public static Matrix GetMoveMatrix(double dx, double dy, double dz){

moveMatrix[0, 3] = dx;

moveMatrix[1, 3] = dy;

moveMatrix[2, 3] = dz;

return moveMatrix;}

public static Matrix GetScaleMatrix(double sx, double sy, double sz){

scaleMatrix[0, 0] = sx;

scaleMatrix[1, 1] = sy;

scaleMatrix[2, 2] = sz;

return scaleMatrix;}

public static Matrix GetScaleMatrix(double scale){

return GetScaleMatrix(scale, scale, scale);}

public static Matrix GetRotateXMatrix(double angle){

double cos = Math.Cos(angle);

double sin = Math.Sin(angle);

rotateXMatrix[1, 1] = cos;

rotateXMatrix[2, 1] = -sin;

rotateXMatrix[1, 2] = sin;

rotateXMatrix[2, 2] = cos;

return rotateXMatrix;}

public static Matrix GetRotateYMatrix(double angle){

double cos = Math.Cos(angle);

double sin = Math.Sin(angle);

rotateYMatrix[0, 0] = cos;

rotateYMatrix[0, 2] = -sin;

rotateYMatrix[2, 0] = sin;

rotateYMatrix[2, 2] = cos;

return rotateYMatrix;}

public static Matrix GetRotateZMatrix(double angle){

double cos = Math.Cos(angle);

double sin = Math.Sin(angle);

rotateZMatrix[0, 0] = cos;

rotateZMatrix[1, 0] = -sin;

rotateZMatrix[0, 1] = sin;

rotateZMatrix[1, 1] = cos;

return rotateZMatrix;}

public static Matrix GetUVNProjectionMatrix(double ux, double uy, double uz, double vx, double vy, double vz, double nx, double ny, double nz){

uvnProjectionMatrix[0, 0] = ux;

uvnProjectionMatrix[1, 0] = vx;

uvnProjectionMatrix[2, 0] = nx;

uvnProjectionMatrix[0, 1] = uy;

uvnProjectionMatrix[1, 1] = vy;

uvnProjectionMatrix[2, 1] = ny;

uvnProjectionMatrix[0, 2] = uz;

uvnProjectionMatrix[1, 2] = vz;

uvnProjectionMatrix[2, 2] = nz;

return uvnProjectionMatrix;}}}

Matrix.cs

namespace Game\_Consoles{

public class Matrix{

public int Width { get; private set; }

public int Height { get; private set; }

private double[,] values;

public double this[int j, int i]{

get{

return values[j, i];}

set{

values[j, i] = value;}}

public Matrix(int height, int width){

Height = height;

Width = width;

values = new double[Height, Width];}

public Matrix(double[] values){

Height = values.GetLength(0);

Width = 1;

this.values = new double[Height, Width];

for (int j = 0; j < Height; ++j){

this.values[j, 0] = values[j];}}

public Matrix(double[,] values){

Height = values.GetLength(0);

Width = values.GetLength(1);

this.values = new double[Height, Width];

for (int j = 0; j < Height; ++j){

for (int i = 0; i < Width ; ++i){

this.values[j, i] = values[j, i];}}}

public Matrix Multiply(Matrix matrix){

if (Width == matrix.Height){

Matrix result = new Matrix(Height, matrix.Width);

for (int i = 0; i < result.Width; ++i){

for (int j = 0; j < result.Height; ++j){

for (int k = 0; k < matrix.Height; ++k){

result[j, i] += this[j, k] \* matrix[k, i];}}}

return result;}

else{

return this;}}}}

MathHelper.cs

using System;

namespace Game\_Consoles{

public class MathHelp{

public static double DegreesToRadians(double angle){

return Math.PI \* angle / 180.0;}

public static double RadiansToDegrees(double angle){

return angle \* 180.0 / Math.PI;}

public static void Swap<T>(ref T a, ref T b){

T temp = a;

a = b;

b = temp;}

public static double[] Interpolate(double sx, double sy, double ex, double ey){

double[] values;

if (sx == ex){

values = new double[] { sy };}

else{

values = new double[(int)Math.Abs(ex - sx + 1)];

double step = (ey - sy) / (ex - sx);

double value = sy;

for (var i = sx; i <= ex; ++i){

values[(int)(i - sx)] = value;

value += step;}}

return values;}}}

MainForm.cs

using System;

using System.Drawing;

using System.Xml.Linq;

using System.Drawing.Imaging;

using System.Windows.Forms;

namespace \_3DEngine

{

public partial class Main : Form

{

public long MinCoordinate = long.MinValue;

public long MaxCoordinate = long.MaxValue;

public Scene scene = null;

private int Rotation = 0;

private bool IsPan = false;

private Camera tempCamera = null;

private SceneObject currentObject = null;

private ScenePrimitive currentPrimitive = null;

public Main()

{

InitializeComponent();

scene = new Scene(RenderPicture);

IsUpdating = true;

CameraTargetX.Minimum = CameraTargetY.Minimum = CameraTargetZ.Minimum = MinCoordinate;

CameraPositionX.Minimum = CameraPositionY.Minimum = CameraPositionZ.Minimum = MinCoordinate;

CameraTargetX.Maximum = CameraTargetY.Maximum = CameraTargetZ.Maximum = MaxCoordinate;

CameraPositionX.Maximum = CameraPositionY.Maximum = CameraPositionZ.Maximum = MaxCoordinate;

PositionX.Minimum = PositionY.Minimum = PositionZ.Minimum = MinCoordinate;

PositionX.Maximum = PositionY.Maximum = PositionZ.Maximum = MaxCoordinate;

IsUpdating = false;

RenderPicture.MouseWheel += new MouseEventHandler(Canvas\_MouseWheel);

}

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

{

scene.ResetCamera();

UpdateCameraValues();

scene.PaintObjects();

}

bool IsUpdating = false;

private void UpdateCameraValues()

{

IsUpdating = true;

CameraTargetX.Value = (decimal)scene.Camera.Target.X;

CameraTargetY.Value = (decimal)scene.Camera.Target.Y;

CameraTargetZ.Value = (decimal)scene.Camera.Target.Z;

CameraPositionX.Value = (decimal)scene.Camera.Position.X;

CameraPositionY.Value = (decimal)scene.Camera.Position.Y;

CameraPositionZ.Value = (decimal)scene.Camera.Position.Z;

IsUpdating = false;

}

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

{

}

private bool IsCanvasAction = false;

private bool LeftMouse = false;

private bool RightMouse = false;

private Point start;

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

{

start = e.Location;

IsCanvasAction = true;

if (e.Button == MouseButtons.Left && !RightMouse)

{

LeftMouse = true;

}

else if (e.Button == MouseButtons.Right && !LeftMouse)

{

RightMouse = true;

}

}

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

{

if (IsCanvasAction)

{

int dx = e.Location.X - start.X;

int dy = e.Location.Y - start.Y;

if (LeftMouse)

{

if (Rotation == 0)

{

scene.Camera.RotatePositionLeftRight(-dx);

scene.Camera.RotatePositionUpDown(-dy);

}

else

{

scene.Camera.RotateTargetLeftRight(-dx);

scene.Camera.RotateTargetUpDown(dy \* 25);

}

}

else if (RightMouse)

{

scene.Camera.MoveCameraLeftRight(dx);

scene.Camera.MoveCameraUpDown(-dy);

}

start = e.Location;

UpdateCameraValues();

scene.PaintObjects();

}

}

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

{

IsCanvasAction = false;

if (e.Button == MouseButtons.Left)

{

LeftMouse = false;

}

else if (e.Button == MouseButtons.Right)

{

RightMouse = false;

}

}

private void Canvas\_MouseClick(object sender, MouseEventArgs e)

{

if (e.Button == MouseButtons.Middle)

{

Rotation = Rotation == 0 ? 1 : 0;

}

}

private void Canvas\_MouseWheel(object sender, MouseEventArgs e)

{

if (scene.Camera.IsCentralProjection)

{

scene.Camera.MoveCameraAlongVector(e.Delta / 12);

UpdateCameraValues();

scene.PaintObjects();

}

}

private void CoordinateBox\_KeyPress(object sender, KeyPressEventArgs e)

{

TextBox box = (TextBox)sender;

if (!char.IsDigit(e.KeyChar) && e.KeyChar != '-' && e.KeyChar != 8)

{

e.Handled = true;

}

else

{

if (e.KeyChar == '-')

{

if (box.TextLength > 0 && (box.SelectionStart != 0 || box.SelectionLength == 0 && box.Text[0] == '-'))

{

e.Handled = true;

}

}

else if (char.IsDigit(e.KeyChar))

{

if (box.TextLength > 0 && box.SelectionStart > 0 && box.Text[0] == '0')

{

int start = box.SelectionStart;

box.Text = box.Text.Substring(1);

box.SelectionStart = start - 1;

}

else if (box.TextLength > 1 && box.SelectionStart > 1 && box.Text[0] == '-' && box.Text[1] == '0')

{

int start = box.SelectionStart;

box.Text = "-" + box.Text.Substring(2);

box.SelectionStart = start - 1;

}

}

}

}

private void CoordinateBox\_Leave(object sender, EventArgs e)

{

TextBox box = (TextBox)sender;

int value = 0;

if (!int.TryParse(box.Text, out value))

{

box.Text = "0";

}

}

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

{

currentPrimitive = null;

currentObject = scene.GetObjectByName(ObjectsList.SelectedItem.ToString());

UpdateCameraValues();

DeleteObject.Enabled = true;

SpacePanel.Enabled = true;

PrimitivesList.Items.Clear();

PrimitivesList.Items.Add("");

PrimitivesList.SelectedIndex = 0;

if (!currentObject.Name.Equals("Lorenz"))

{

if (!(currentObject is TriplaneObject))

{

foreach (ScenePrimitive scenePrimitive in currentObject.ScenePrimitives)

{

PrimitivesList.Items.Add(scenePrimitive.Name);

}

ConsoleParameters.Visible = ConsoleParameters.Enabled = false;

}

else

{

WingWidth.Value = (decimal)((TriplaneObject)currentObject).WingWidth;

widthLowerChassis.Value = (decimal)((TriplaneObject)currentObject).widthLowerChassis;

widthVerticalWingSupports.Value = (decimal)((TriplaneObject)currentObject).widthVerticalWingSupports;

radiusChassis.Value = (decimal)((TriplaneObject)currentObject).radiusChassis;

widthLop.Value = (decimal)((TriplaneObject)currentObject).widthLop;

Sit.Value = (decimal)((TriplaneObject)currentObject).Sit;

widthVerticalBackWing.Value = (decimal)((TriplaneObject)currentObject).widthVerticalBackWing;

ConsoleParameters.Visible = ConsoleParameters.Enabled = true;

ParametersPanel.Visible = ParametersPanel.Enabled = false;

}

}

}

private void PositionX\_ValueChanged(object sender, EventArgs e)

{

if (!IsUpdating)

{

if (currentPrimitive != null)

{

currentPrimitive.Primitive.BasePoint.X = (double)PositionX.Value;

}

else

{

currentObject.BasePoint.X = (double)PositionX.Value;

}

scene.PaintObjects();

}

}

private void PositionY\_ValueChanged(object sender, EventArgs e)

{

if (!IsUpdating)

{

if (currentPrimitive != null)

{

currentPrimitive.Primitive.BasePoint.Y = (double)PositionY.Value;

}

else

{

currentObject.BasePoint.Y = (double)PositionY.Value;

}

scene.PaintObjects();

}

}

private void PositionZ\_ValueChanged(object sender, EventArgs e)

{

if (!IsUpdating)

{

if (currentPrimitive != null)

{

currentPrimitive.Primitive.BasePoint.Z = (double)PositionZ.Value;

}

else

{

currentObject.BasePoint.Z = (double)PositionZ.Value;

}

scene.PaintObjects();

}

}

private void RotateX\_ValueChanged(object sender, EventArgs e)

{

if (!IsUpdating)

{

if (currentPrimitive != null)

{

currentPrimitive.AngleX = (int)RotateX.Value;

}

else

{

currentObject.AngleX = (int)RotateX.Value;

}

scene.PaintObjects();

}

}

private void RotateY\_ValueChanged(object sender, EventArgs e)

{

if (!IsUpdating)

{

if (currentPrimitive != null)

{

currentPrimitive.AngleY = (int)RotateY.Value;

}

else

{

currentObject.AngleY = (int)RotateY.Value;

}

scene.PaintObjects();

}

}

private void RotateZ\_ValueChanged(object sender, EventArgs e)

{

if (!IsUpdating)

{

if (currentPrimitive != null)

{

currentPrimitive.AngleZ = (int)RotateZ.Value;

}

else

{

currentObject.AngleZ = (int)RotateZ.Value;

}

scene.PaintObjects();

}

}

private void MainForm\_Resize(object sender, EventArgs e)

{

if (WindowState == FormWindowState.Minimized)

{

MainTimer.Enabled = false;

}

else

{

MainTimer.Enabled = true;

if (scene != null)

{

scene.Camera.ResizeFrame(RenderPicture.ClientSize.Width, RenderPicture.ClientSize.Height);

scene.PaintObjects();

}

}

}

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

{

AddObject form = new AddObject(this, null);

form.ShowDialog();

}

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

{

//CreateObject form = new CreateObject(this, currentObject);

//form.ShowDialog();

}

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

{

IsUpdating = true;

RadiusBox.Enabled = false;

SegmentsBox.Enabled = false;

WidthBox.Enabled = false;

HeightBox.Enabled = false;

LengthBox.Enabled = false;

RadiusBox.Value = WidthBox.Value = HeightBox.Value = LengthBox.Value = 25;

SegmentsBox.Value = 16;

ColorBox.BackColor = Color.LightGray;

ParametersPanel.Visible = ParametersPanel.Enabled = ColorBox.Enabled = PrimitivesList.SelectedIndex != 0;

if (PrimitivesList.SelectedIndex == 0)

{

currentPrimitive = null;

}

else

{

string name = PrimitivesList.SelectedItem.ToString();

currentPrimitive = currentObject.GetScenePrimitiveByName(name);

if (currentPrimitive.Primitive is Box)

{

Box primitive = (Box)currentPrimitive.Primitive;

WidthBox.Enabled = true;

WidthBox.Value = (decimal)primitive.Width;

HeightBox.Enabled = true;

HeightBox.Value = (decimal)primitive.Height;

LengthBox.Enabled = true;

LengthBox.Value = (decimal)primitive.Length;

}

else if (currentPrimitive.Primitive is Cylinder)

{

Cylinder primitive = (Cylinder)currentPrimitive.Primitive;

RadiusBox.Enabled = true;

RadiusBox.Value = (decimal)primitive.Radius;

HeightBox.Enabled = true;

HeightBox.Value = (decimal)primitive.Height;

SegmentsBox.Enabled = true;

SegmentsBox.Value = (decimal)primitive.SegmentsCount;

}

else if (currentPrimitive.Primitive is Sphere)

{

Sphere primitive = (Sphere)currentPrimitive.Primitive;

RadiusBox.Enabled = true;

RadiusBox.Value = (decimal)primitive.Radius;

SegmentsBox.Enabled = true;

SegmentsBox.Value = (decimal)primitive.SegmentsCount;

}

else if (currentPrimitive.Primitive is Hemisphere)

{

Hemisphere primitive = (Hemisphere)currentPrimitive.Primitive;

RadiusBox.Enabled = true;

RadiusBox.Value = (decimal)primitive.Radius;

SegmentsBox.Enabled = true;

SegmentsBox.Value = (decimal)primitive.SegmentsCount;

}

ColorBox.BackColor = currentPrimitive.Primitive.Color;

}

RotateX.Value = currentPrimitive != null ? currentPrimitive.AngleX : currentObject.AngleX;

RotateY.Value = currentPrimitive != null ? currentPrimitive.AngleY : currentObject.AngleY;

RotateZ.Value = currentPrimitive != null ? currentPrimitive.AngleZ : currentObject.AngleZ;

PositionX.Value = currentPrimitive != null ? (int)currentPrimitive.Primitive.BasePoint.X : (int)currentObject.BasePoint.X;

PositionY.Value = currentPrimitive != null ? (int)currentPrimitive.Primitive.BasePoint.Y : (int)currentObject.BasePoint.Y;

PositionZ.Value = currentPrimitive != null ? (int)currentPrimitive.Primitive.BasePoint.Z : (int)currentObject.BasePoint.Z;

ScaleX.Value = currentPrimitive != null ? (decimal)currentPrimitive.ScaleX : (decimal)currentObject.ScaleX;

ScaleY.Value = currentPrimitive != null ? (decimal)currentPrimitive.ScaleY : (decimal)currentObject.ScaleY;

ScaleZ.Value = currentPrimitive != null ? (decimal)currentPrimitive.ScaleZ : (decimal)currentObject.ScaleZ;

IsUpdating = false;

}

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

{

currentObject.ScenePrimitives.Remove(currentPrimitive);

int index = PrimitivesList.SelectedIndex;

PrimitivesList.Items.RemoveAt(index);

PrimitivesList.SelectedIndex = index - 1;

}

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

{

scene.Objects.Remove(currentObject);

int index = ObjectsList.SelectedIndex;

ObjectsList.Items.RemoveAt(index);

ObjectsList.SelectedIndex = index - 1;

if (ObjectsList.SelectedIndex == -1 && ObjectsList.Items.Count > 0)

{

ObjectsList.SelectedIndex = 0;

}

if (ObjectsList.Items.Count == 0)

{

DeleteObject.Enabled = false;

PrimitivesList.Items.Clear();

SpacePanel.Enabled = false;

PositionX.Value = PositionY.Value = PositionZ.Value = 0;

RotateX.Value = RotateY.Value = RotateZ.Value = 0;

ScaleX.Value = 1;

ScaleY.Value = 1;

ScaleZ.Value = 1;

ParametersPanel.Visible = ParametersPanel.Enabled = false;

ConsoleParameters.Visible = ConsoleParameters.Enabled = false;

}

RenderPicture.Image = null;

}

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

{

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

{

ColorBox.BackColor = ColorPicker.Color;

currentPrimitive.Primitive.Color = ColorPicker.Color;

scene.PaintObjects();

}

}

private void RadiusBox\_ValueChanged(object sender, EventArgs e)

{

if (!IsUpdating)

{

if (currentPrimitive.Primitive is Cylinder)

{

Cylinder primitive = (Cylinder)currentPrimitive.Primitive;

primitive.ModifyRadius((int)RadiusBox.Value);

}

else if (currentPrimitive.Primitive is Sphere)

{

Sphere primitive = (Sphere)currentPrimitive.Primitive;

primitive.ModifyRadius((int)RadiusBox.Value);

}

else if (currentPrimitive.Primitive is Hemisphere)

{

Hemisphere primitive = (Hemisphere)currentPrimitive.Primitive;

primitive.ModifyRadius((int)RadiusBox.Value);

}

currentPrimitive.SetScale(currentPrimitive.ScaleX, currentPrimitive.ScaleY, currentPrimitive.ScaleZ);

scene.PaintObjects();

}

}

private void SegmentsBox\_ValueChanged(object sender, EventArgs e)

{

if (!IsUpdating)

{

if (currentPrimitive.Primitive is Cylinder)

{

Cylinder primitive = (Cylinder)currentPrimitive.Primitive;

primitive.ModifySegmentsCount((int)SegmentsBox.Value);

}

else if (currentPrimitive.Primitive is Sphere)

{

Sphere primitive = (Sphere)currentPrimitive.Primitive;

primitive.ModifySegmentsCount((int)SegmentsBox.Value);

}

else if (currentPrimitive.Primitive is Hemisphere)

{

Hemisphere primitive = (Hemisphere)currentPrimitive.Primitive;

primitive.ModifySegmentsCount((int)SegmentsBox.Value);

}

currentPrimitive.SetScale(currentPrimitive.ScaleX, currentPrimitive.ScaleY, currentPrimitive.ScaleZ);

scene.PaintObjects();

}

}

private void HeightBox\_ValueChanged(object sender, EventArgs e)

{

if (!IsUpdating)

{

if (currentPrimitive.Primitive is Box)

{

Box primitive = (Box)currentPrimitive.Primitive;

primitive.ModifyHeight((int)HeightBox.Value);

}

else if (currentPrimitive.Primitive is Cylinder)

{

Cylinder primitive = (Cylinder)currentPrimitive.Primitive;

primitive.ModifyHeight((int)HeightBox.Value);

}

currentPrimitive.SetScale(currentPrimitive.ScaleX, currentPrimitive.ScaleY, currentPrimitive.ScaleZ);

scene.PaintObjects();

}

}

private void WidthBox\_ValueChanged(object sender, EventArgs e)

{

if (!IsUpdating)

{

if (currentPrimitive.Primitive is Box)

{

Box primitive = (Box)currentPrimitive.Primitive;

primitive.ModifyWidth((int)WidthBox.Value);

}

currentPrimitive.SetScale(currentPrimitive.ScaleX, currentPrimitive.ScaleY, currentPrimitive.ScaleZ);

scene.PaintObjects();

}

}

private void LengthBox\_ValueChanged(object sender, EventArgs e)

{

if (!IsUpdating)

{

if (currentPrimitive.Primitive is Box)

{

Box primitive = (Box)currentPrimitive.Primitive;

primitive.ModifyLength((int)LengthBox.Value);

}

currentPrimitive.SetScale(currentPrimitive.ScaleX, currentPrimitive.ScaleY, currentPrimitive.ScaleZ);

scene.PaintObjects();

}

}

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

{

if (!IsPan)

{

tempCamera = scene.Camera;

scene.ResetCamera();

scene.Camera.Target.Set(

currentObject.BasePoint.X,

currentObject.BasePoint.Y,

currentObject.BasePoint.Z

);

scene.Camera.Position.Set(

currentObject.BasePoint.X,

currentObject.BasePoint.Y + currentObject.MaxLength \* 2,

currentObject.BasePoint.Z - Math.Max(currentObject.MaxLength + scene.Camera.NearClippingPlaneZ, 100)

);

scene.Camera.FarClippingPlaneZ = Math.Max(scene.Camera.FarClippingPlaneZ, currentObject.MaxLength \* 1.5);

RenderPicture.Enabled = false;

scene.PanObject = currentObject;

IsPan = true;

}

else

{

IsPan = false;

scene.Camera = tempCamera;

SpacePanel.Enabled = ObjectsList.SelectedIndex >= 0;

if (!(currentObject is TriplaneObject))

{

ParametersPanel.Enabled = PrimitivesList.SelectedIndex >= 0;

}

else

{

ConsoleParameters.Enabled = true;

}

RenderPicture.Enabled = true;

scene.PanObject = null;

}

}

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

{

if (e.Button == MouseButtons.Right)

{

PositionX.Value = 0;

scene.PaintObjects();

}

}

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

{

if (e.Button == MouseButtons.Right)

{

PositionY.Value = 0;

scene.PaintObjects();

}

}

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

{

if (e.Button == MouseButtons.Right)

{

PositionZ.Value = 0;

scene.PaintObjects();

}

}

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

{

if (e.Button == MouseButtons.Right)

{

RotateX.Value = 0;

scene.PaintObjects();

}

}

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

{

if (e.Button == MouseButtons.Right)

{

RotateY.Value = 0;

scene.PaintObjects();

}

}

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

{

if (e.Button == MouseButtons.Right)

{

RotateZ.Value = 0;

scene.PaintObjects();

}

}

private void CameraTargetX\_ValueChanged(object sender, EventArgs e)

{

if (!IsUpdating)

{

scene.Camera.Target.Set(

(double)CameraTargetX.Value,

(double)CameraTargetY.Value,

(double)CameraTargetZ.Value

);

scene.PaintObjects();

}

}

private void CameraTargetY\_ValueChanged(object sender, EventArgs e)

{

if (!IsUpdating)

{

scene.Camera.Target.Set(

(double)CameraTargetX.Value,

(double)CameraTargetY.Value,

(double)CameraTargetZ.Value

);

scene.PaintObjects();

}

}

private void CameraTargetZ\_ValueChanged(object sender, EventArgs e)

{

if (!IsUpdating)

{

scene.Camera.Target.Set(

(double)CameraTargetX.Value,

(double)CameraTargetY.Value,

(double)CameraTargetZ.Value

);

scene.PaintObjects();

}

}

private void CameraPositionX\_ValueChanged(object sender, EventArgs e)

{

if (!IsUpdating)

{

scene.Camera.Position.Set(

(double)CameraPositionX.Value,

(double)CameraPositionY.Value,

(double)CameraPositionZ.Value

);

scene.PaintObjects();

}

}

private void CameraPositionY\_ValueChanged(object sender, EventArgs e)

{

if (!IsUpdating)

{

scene.Camera.Position.Set(

(double)CameraPositionX.Value,

(double)CameraPositionY.Value,

(double)CameraPositionZ.Value

);

scene.PaintObjects();

}

}

private void CameraPositionZ\_ValueChanged(object sender, EventArgs e)

{

if (!IsUpdating)

{

scene.Camera.Position.Set(

(double)CameraPositionX.Value,

(double)CameraPositionY.Value,

(double)CameraPositionZ.Value

);

scene.PaintObjects();

}

}

private void ScaleX\_ValueChanged(object sender, EventArgs e)

{

if (!IsUpdating)

{

if (currentPrimitive != null)

{

currentPrimitive.SetScale((double)ScaleX.Value, (double)ScaleY.Value, (double)ScaleZ.Value);

}

else

{

currentObject.SetScale((double)ScaleX.Value, (double)ScaleY.Value, (double)ScaleZ.Value);

}

scene.PaintObjects();

}

}

private void ScaleY\_ValueChanged(object sender, EventArgs e)

{

if (!IsUpdating)

{

if (currentPrimitive != null)

{

currentPrimitive.SetScale((double)ScaleX.Value, (double)ScaleY.Value, (double)ScaleZ.Value);

}

else

{

currentObject.SetScale((double)ScaleX.Value, (double)ScaleY.Value, (double)ScaleZ.Value);

}

scene.PaintObjects();

}

}

private void ScaleZ\_ValueChanged(object sender, EventArgs e)

{

if (!IsUpdating)

{

if (currentPrimitive != null)

{

currentPrimitive.SetScale((double)ScaleX.Value, (double)ScaleY.Value, (double)ScaleZ.Value);

}

else

{

currentObject.SetScale((double)ScaleX.Value, (double)ScaleY.Value, (double)ScaleZ.Value);

}

scene.PaintObjects();

}

}

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

{

if (e.Button == MouseButtons.Right)

{

ScaleX.Value = 1;

scene.PaintObjects();

}

}

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

{

if (e.Button == MouseButtons.Right)

{

ScaleY.Value = 1;

scene.PaintObjects();

}

}

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

{

if (e.Button == MouseButtons.Right)

{

ScaleZ.Value = 1;

scene.PaintObjects();

}

}

private void SecondDisplayDiagonal\_ValueChanged(object sender, EventArgs e)

{

((TriplaneObject)currentObject).WingWidth = (double)WingWidth.Value;

((TriplaneObject)currentObject).UpdateObject();

scene.PaintObjects();

}

private void ManipulatorRadius\_ValueChanged(object sender, EventArgs e)

{

((TriplaneObject)currentObject).widthLowerChassis = (double)widthLowerChassis.Value;

((TriplaneObject)currentObject).UpdateObject();

scene.PaintObjects();

}

private void ManipulatorBaseRadius\_ValueChanged(object sender, EventArgs e)

{

((TriplaneObject)currentObject).widthVerticalWingSupports = (double)widthVerticalWingSupports.Value;

((TriplaneObject)currentObject).UpdateObject();

scene.PaintObjects();

}

private void CylindersHeight\_ValueChanged(object sender, EventArgs e)

{

((TriplaneObject)currentObject).radiusChassis = (double)radiusChassis.Value;

((TriplaneObject)currentObject).UpdateObject();

scene.PaintObjects();

}

private void CylindersRadius\_ValueChanged(object sender, EventArgs e)

{

((TriplaneObject)currentObject).widthLop = (double)widthLop.Value;

((TriplaneObject)currentObject).UpdateObject();

scene.PaintObjects();

}

private void CrossButtonSize\_ValueChanged(object sender, EventArgs e)

{

((TriplaneObject)currentObject).Sit = (double)Sit.Value;

((TriplaneObject)currentObject).UpdateObject();

scene.PaintObjects();

}

private void VolumeSpace\_ValueChanged(object sender, EventArgs e)

{

((TriplaneObject)currentObject).widthVerticalBackWing = (double)widthVerticalBackWing.Value;

((TriplaneObject)currentObject).UpdateObject();

scene.PaintObjects();

}

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

{

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

{

try

{

IFormatProvider format = new System.Globalization.CultureInfo("es-ES");

XDocument document = new XDocument();

XElement scene1 = new XElement("scene");

XElement cameras = new XElement("cameras");

for (int i = 0; i < scene.Cameras.Count; ++i)

{

Camera camera = scene.Cameras[i];

XElement cameraElement = new XElement("camera");

cameraElement.SetElementValue("central-projection", camera.IsCentralProjection);

XElement position = new XElement("position");

position.SetAttributeValue("x", string.Format(format, "{0:0.00}", camera.Position.X));

position.SetAttributeValue("y", string.Format(format, "{0:0.00}", camera.Position.Y));

position.SetAttributeValue("z", string.Format(format, "{0:0.00}", camera.Position.Z));

cameraElement.Add(position);

XElement target = new XElement("target");

target.SetAttributeValue("x", string.Format(format, "{0:0.00}", camera.Target.X));

target.SetAttributeValue("y", string.Format(format, "{0:0.00}", camera.Target.Y));

target.SetAttributeValue("z", string.Format(format, "{0:0.00}", camera.Target.Z));

cameraElement.Add(target);

XElement clippingPlanes = new XElement("clipping-planes");

clippingPlanes.SetAttributeValue("near", camera.NearClippingPlaneZ);

clippingPlanes.SetAttributeValue("far", camera.FarClippingPlaneZ);

cameraElement.Add(clippingPlanes);

cameraElement.SetElementValue("fov", camera.FOV);

cameras.Add(cameraElement);

}

scene1.Add(cameras);

XElement lights = new XElement("lights");

for (int i = 0; i < scene.Lights.Count; ++i)

{

Vector3D light = scene.Lights[i];

XElement lightElement = new XElement("light");

lightElement.SetAttributeValue("x", string.Format(format, "{0:0.00}", light.X));

lightElement.SetAttributeValue("y", string.Format(format, "{0:0.00}", light.Y));

lightElement.SetAttributeValue("z", string.Format(format, "{0:0.00}", light.Z));

lights.Add(lightElement);

}

scene1.Add(lights);

XElement objects = new XElement("objects");

foreach (SceneObject sceneObject in scene.Objects)

{

XElement objectElement;

if (sceneObject is TriplaneObject)

{

TriplaneObject triplaneObject = (TriplaneObject)sceneObject;

objectElement = new XElement("TriplaneObject-object");

}

else

{

objectElement = new XElement("scene-object");

XElement scenePrimitivesElement = new XElement("scene-primitives");

foreach (ScenePrimitive scenePrimitive in sceneObject.ScenePrimitives)

{

XElement scenePrimitiveElement = new XElement("scene-primitive");

XElement primitiveRotate = new XElement("rotate");

primitiveRotate.SetAttributeValue("x", scenePrimitive.AngleX);

primitiveRotate.SetAttributeValue("y", scenePrimitive.AngleY);

primitiveRotate.SetAttributeValue("z", scenePrimitive.AngleZ);

scenePrimitiveElement.Add(primitiveRotate);

XElement primitiveScale = new XElement("scale");

primitiveScale.SetAttributeValue("x", string.Format(format, "{0:0.00}", scenePrimitive.ScaleX));

primitiveScale.SetAttributeValue("y", string.Format(format, "{0:0.00}", scenePrimitive.ScaleY));

primitiveScale.SetAttributeValue("z", string.Format(format, "{0:0.00}", scenePrimitive.ScaleZ));

scenePrimitiveElement.Add(primitiveScale);

XElement primitiveElement = new XElement("primitive");

XElement primitiveBasePoint = new XElement("base-point");

primitiveBasePoint.SetAttributeValue("x", string.Format(format, "{0:0.00}", scenePrimitive.Primitive.BasePoint.X));

primitiveBasePoint.SetAttributeValue("y", string.Format(format, "{0:0.00}", scenePrimitive.Primitive.BasePoint.Y));

primitiveBasePoint.SetAttributeValue("z", string.Format(format, "{0:0.00}", scenePrimitive.Primitive.BasePoint.Z));

primitiveElement.Add(primitiveBasePoint);

XElement primitiveColor = new XElement("color");

primitiveColor.SetAttributeValue("r", scenePrimitive.Primitive.Color.R);

primitiveColor.SetAttributeValue("g", scenePrimitive.Primitive.Color.G);

primitiveColor.SetAttributeValue("b", scenePrimitive.Primitive.Color.B);

primitiveElement.Add(primitiveColor);

if (scenePrimitive.Primitive is Box)

{

Box box = (Box)scenePrimitive.Primitive;

primitiveElement.SetAttributeValue("type", "box");

primitiveElement.SetElementValue("width", string.Format(format, "{0:0.00}", box.Width));

primitiveElement.SetElementValue("height", string.Format(format, "{0:0.00}", box.Height));

primitiveElement.SetElementValue("length", string.Format(format, "{0:0.00}", box.Length));

}

else if (scenePrimitive.Primitive is Cylinder)

{

Cylinder cylinder = (Cylinder)scenePrimitive.Primitive;

primitiveElement.SetAttributeValue("type", "cylinder");

primitiveElement.SetElementValue("radius", string.Format(format, "{0:0.00}", cylinder.Radius));

primitiveElement.SetElementValue("height", string.Format(format, "{0:0.00}", cylinder.Height));

primitiveElement.SetElementValue("segments-count", cylinder.SegmentsCount);

}

else if (scenePrimitive.Primitive is Sphere)

{

Sphere sphere = (Sphere)scenePrimitive.Primitive;

primitiveElement.SetAttributeValue("type", "sphere");

primitiveElement.SetElementValue("radius", string.Format(format, "{0:0.00}", sphere.Radius));

primitiveElement.SetElementValue("segments-count", sphere.SegmentsCount);

}

else if (scenePrimitive.Primitive is Hemisphere)

{

Hemisphere hemisphere = (Hemisphere)scenePrimitive.Primitive;

primitiveElement.SetAttributeValue("type", "hemisphere");

primitiveElement.SetElementValue("radius", string.Format(format, "{0:0.00}", hemisphere.Radius));

primitiveElement.SetElementValue("segments-count", hemisphere.SegmentsCount);

}

scenePrimitiveElement.Add(primitiveElement);

scenePrimitiveElement.SetAttributeValue("name", scenePrimitive.Name);

scenePrimitivesElement.Add(scenePrimitiveElement);

}

objectElement.Add(scenePrimitivesElement);

}

objectElement.SetAttributeValue("name", sceneObject.Name);

XElement objectPosition = new XElement("position");

objectPosition.SetAttributeValue("x", string.Format(format, "{0:0.00}", sceneObject.BasePoint.X));

objectPosition.SetAttributeValue("y", string.Format(format, "{0:0.00}", sceneObject.BasePoint.Y));

objectPosition.SetAttributeValue("z", string.Format(format, "{0:0.00}", sceneObject.BasePoint.Z));

objectElement.Add(objectPosition);

XElement objectRotate = new XElement("rotate");

objectRotate.SetAttributeValue("x", sceneObject.AngleX);

objectRotate.SetAttributeValue("y", sceneObject.AngleY);

objectRotate.SetAttributeValue("z", sceneObject.AngleZ);

objectElement.Add(objectRotate);

XElement objectScale = new XElement("scale");

objectScale.SetAttributeValue("x", string.Format(format, "{0:0.00}", sceneObject.ScaleX));

objectScale.SetAttributeValue("y", string.Format(format, "{0:0.00}", sceneObject.ScaleY));

objectScale.SetAttributeValue("z", string.Format(format, "{0:0.00}", sceneObject.ScaleZ));

objectElement.Add(objectScale);

objects.Add(objectElement);

}

scene1.Add(objects);

document.Add(scene);

//document.Save(ExportDialog.FileName);

}

catch (Exception exception)

{

//Log.Append(exception.StackTrace);

MessageBox.Show("Unable to export scene!", Text, MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

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

{

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

{

try

{

IFormatProvider format = new System.Globalization.CultureInfo("es-ES");

XDocument document = XDocument.Load(ImportDialog.FileName);

XElement scene1 = document.Element("scene");

scene.Cameras.Clear();

foreach (XElement cameraElement in scene1.Element("cameras").Elements("camera"))

{

scene.Camera = new Camera();

scene.ResetCamera();

scene.Camera.IsCentralProjection = bool.Parse(cameraElement.Element("central-projection").Value);

XElement position = cameraElement.Element("position");

scene.Camera.Position = new Point3DSpherical(new Point3D(

double.Parse(position.Attribute("x").Value, format),

double.Parse(position.Attribute("y").Value, format),

double.Parse(position.Attribute("z").Value, format)

));

XElement target = cameraElement.Element("target");

scene.Camera.Target = new Point3DSpherical(new Point3D(

double.Parse(target.Attribute("x").Value, format),

double.Parse(target.Attribute("y").Value, format),

double.Parse(target.Attribute("z").Value, format)

));

scene.Camera.RotatePositionLeftRight(0);

scene.Camera.RotatePositionUpDown(0);

XElement clippingPlanes = cameraElement.Element("clipping-planes");

scene.Camera.NearClippingPlaneZ = double.Parse(clippingPlanes.Attribute("near").Value, format);

scene.Camera.FarClippingPlaneZ = double.Parse(clippingPlanes.Attribute("far").Value, format);

scene.Camera.FOV = double.Parse(cameraElement.Element("fov").Value, format);

scene.Cameras.Add(scene.Camera);

}

scene.Lights.Clear();

foreach (XElement lightElement in scene1.Element("lights").Elements("light"))

{

scene.Light = new Vector3D(

double.Parse(lightElement.Attribute("x").Value, format),

double.Parse(lightElement.Attribute("y").Value, format),

double.Parse(lightElement.Attribute("z").Value, format)

);

scene.Lights.Add(scene.Light);

}

scene.Objects.Clear();

ObjectsList.Items.Clear();

PrimitivesList.Items.Clear();

ParametersPanel.Visible = ParametersPanel.Enabled = ConsoleParameters.Visible = ConsoleParameters.Enabled = false;

foreach (XElement sceneObjectElement in scene1.Element("objects").Elements("scene-object"))

{

SceneObject sceneObject = new SceneObject(sceneObjectElement.Attribute("name").Value);

XElement position = sceneObjectElement.Element("position");

sceneObject.BasePoint = new Point3D(

double.Parse(position.Attribute("x").Value, format),

double.Parse(position.Attribute("y").Value, format),

double.Parse(position.Attribute("z").Value, format)

);

XElement rotate = sceneObjectElement.Element("rotate");

sceneObject.AngleX = int.Parse(rotate.Attribute("x").Value);

sceneObject.AngleY = int.Parse(rotate.Attribute("y").Value);

sceneObject.AngleZ = int.Parse(rotate.Attribute("z").Value);

XElement scale = sceneObjectElement.Element("scale");

sceneObject.SetScale(

double.Parse(scale.Attribute("x").Value, format),

double.Parse(scale.Attribute("y").Value, format),

double.Parse(scale.Attribute("z").Value, format)

);

foreach (XElement scenePrimitiveElement in sceneObjectElement.Element("scene-primitives").Elements("scene-primitive"))

{

Primitive primitive = null;

XElement primitiveElement = scenePrimitiveElement.Element("primitive");

string type = primitiveElement.Attribute("type").Value;

XElement basePointElement = primitiveElement.Element("base-point");

Point3D basePoint = new Point3D(

double.Parse(basePointElement.Attribute("x").Value, format),

double.Parse(basePointElement.Attribute("y").Value, format),

double.Parse(basePointElement.Attribute("z").Value, format)

);

XElement colorElement = primitiveElement.Element("color");

Color color = Color.FromArgb(

int.Parse(colorElement.Attribute("r").Value),

int.Parse(colorElement.Attribute("g").Value),

int.Parse(colorElement.Attribute("b").Value)

);

switch (type)

{

case "box":

primitive = new Box(

basePoint,

double.Parse(primitiveElement.Element("width").Value, format),

double.Parse(primitiveElement.Element("height").Value, format),

double.Parse(primitiveElement.Element("length").Value, format),

color

);

break;

case "cylinder":

primitive = new Cylinder(

basePoint,

double.Parse(primitiveElement.Element("radius").Value, format),

double.Parse(primitiveElement.Element("height").Value, format),

int.Parse(primitiveElement.Element("segments-count").Value),

color

);

break;

case "sphere":

primitive = new Sphere(

basePoint,

double.Parse(primitiveElement.Element("radius").Value, format),

int.Parse(primitiveElement.Element("segments-count").Value),

color

);

break;

case "hemisphere":

primitive = new Hemisphere(

basePoint,

double.Parse(primitiveElement.Element("radius").Value, format),

int.Parse(primitiveElement.Element("segments-count").Value),

color

);

break;

}

ScenePrimitive scenePrimitive = new ScenePrimitive(primitive, scenePrimitiveElement.Attribute("name").Value);

XElement primitiveRotate = scenePrimitiveElement.Element("rotate");

scenePrimitive.AngleX = int.Parse(primitiveRotate.Attribute("x").Value);

scenePrimitive.AngleY = int.Parse(primitiveRotate.Attribute("y").Value);

scenePrimitive.AngleZ = int.Parse(primitiveRotate.Attribute("z").Value);

XElement primitiveScale = scenePrimitiveElement.Element("scale");

scenePrimitive.SetScale(

double.Parse(primitiveScale.Attribute("x").Value, format),

double.Parse(primitiveScale.Attribute("y").Value, format),

double.Parse(primitiveScale.Attribute("z").Value, format)

);

sceneObject.AddScenePrimitive(scenePrimitive);

}

scene.AddObject(sceneObject);

ObjectsList.Items.Add(sceneObject.Name);

}

foreach (XElement triplaneObjectElement in scene1.Element("objects").Elements("console-object"))

{

TriplaneObject triplaneObject = new TriplaneObject(triplaneObjectElement.Attribute("name").Value);

XElement position = triplaneObjectElement.Element("position");

triplaneObject.BasePoint = new Point3D(

double.Parse(position.Attribute("x").Value, format),

double.Parse(position.Attribute("y").Value, format),

double.Parse(position.Attribute("z").Value, format)

);

XElement rotate = triplaneObjectElement.Element("rotate");

triplaneObject.AngleX = int.Parse(rotate.Attribute("x").Value);

triplaneObject.AngleY = int.Parse(rotate.Attribute("y").Value);

triplaneObject.AngleZ = int.Parse(rotate.Attribute("z").Value);

XElement scale = triplaneObjectElement.Element("scale");

triplaneObject.SetScale(

double.Parse(scale.Attribute("x").Value, format),

double.Parse(scale.Attribute("y").Value, format),

double.Parse(scale.Attribute("z").Value, format)

);

triplaneObject.UpdateObject();

scene.AddObject(triplaneObject);

ObjectsList.Items.Add(triplaneObject.Name);

}

ObjectsList.SelectedIndex = -1;

if (scene.Objects.Count > 0)

{

ObjectsList.SelectedIndex = 0;

}

scene.PaintObjects();

}

catch (Exception exception)

{

MessageBox.Show("Unable to import scene! File is corrupted.", Text, MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

private void MainForm\_KeyDown(object sender, KeyEventArgs e)

{

if (e.Control && e.KeyCode == Keys.L)

{

double a = 10;

double r = 24.7;

double b = 2.7;

double x = 1;

double y = 1;

double z = 1;

double time = 1000;

double dt = 0.01;

SceneObject sceneObject = new SceneObject("Lorenz");

sceneObject.BasePoint = new Point3D(0, 0, 0);

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

{

double nx = x + (-a \* x + a \* y) \* dt;

double ny = y + (-x \* z + r \* x - y) \* dt;

double nz = z + (x \* y - b \* z) \* dt;

x = nx;

y = ny;

z = nz;

int red = (int)((i / (double)time) \* 255);

int green = 0;

int blue = (int)(((time - i) / (double)time) \* 255);

Sphere box = new Sphere(new Point3D(x \* 10, y \* 10, z \* 10), 5, 4, Color.FromArgb(red, green, blue));

ScenePrimitive scenePrimitive = new ScenePrimitive(box, $"Point-{i}");

sceneObject.AddScenePrimitive(scenePrimitive);

}

scene.AddObject(sceneObject);

int index = ObjectsList.Items.Add(sceneObject.Name);

ObjectsList.SelectedIndex = index;

}

else if (e.Control && e.KeyCode == Keys.S)

{

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

{

RenderPicture.Image.Save(SaveImageDialog.FileName);

}

}

}

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

{

scene.Cameras.Clear();

scene.Camera = new Camera();

scene.ResetCamera();

scene.Cameras.Add(scene.Camera);

scene.Lights.Clear();

scene.Light = new Vector3D();

scene.ResetLight();

scene.Lights.Add(scene.Light);

scene.Objects.Clear();

currentObject = null;

ObjectsList.Items.Clear();

PrimitivesList.Items.Clear();

ParametersPanel.Visible = ConsoleParameters.Visible = false;

scene.PaintObjects();

}

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

{

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

{

try

{

IFormatProvider format = new System.Globalization.CultureInfo("es-ES");

XDocument document = XDocument.Load(ImportDialog.FileName);

XElement scene1 = document.Element("scene");

scene.Cameras.Clear();

foreach (XElement cameraElement in scene1.Element("cameras").Elements("camera"))

{

scene.Camera = new Camera();

scene.ResetCamera();

scene.Camera.IsCentralProjection = bool.Parse(cameraElement.Element("central-projection").Value);

XElement position = cameraElement.Element("position");

scene.Camera.Position = new Point3DSpherical(new Point3D(

double.Parse(position.Attribute("x").Value, format),

double.Parse(position.Attribute("y").Value, format),

double.Parse(position.Attribute("z").Value, format)

));

XElement target = cameraElement.Element("target");

scene.Camera.Target = new Point3DSpherical(new Point3D(

double.Parse(target.Attribute("x").Value, format),

double.Parse(target.Attribute("y").Value, format),

double.Parse(target.Attribute("z").Value, format)

));

scene.Camera.RotatePositionLeftRight(0);

scene.Camera.RotatePositionUpDown(0);

XElement clippingPlanes = cameraElement.Element("clipping-planes");

scene.Camera.NearClippingPlaneZ = double.Parse(clippingPlanes.Attribute("near").Value, format);

scene.Camera.FarClippingPlaneZ = double.Parse(clippingPlanes.Attribute("far").Value, format);

scene.Camera.FOV = double.Parse(cameraElement.Element("fov").Value, format);

scene.Cameras.Add(scene.Camera);

}

scene.Lights.Clear();

foreach (XElement lightElement in scene1.Element("lights").Elements("light"))

{

scene.Light = new Vector3D(

double.Parse(lightElement.Attribute("x").Value, format),

double.Parse(lightElement.Attribute("y").Value, format),

double.Parse(lightElement.Attribute("z").Value, format)

);

scene.Lights.Add(scene.Light);

}

scene.Objects.Clear();

ObjectsList.Items.Clear();

PrimitivesList.Items.Clear();

ParametersPanel.Visible = ParametersPanel.Enabled = ConsoleParameters.Visible = ConsoleParameters.Enabled = false;

foreach (XElement sceneObjectElement in scene1.Element("objects").Elements("scene-object"))

{

SceneObject sceneObject = new SceneObject(sceneObjectElement.Attribute("name").Value);

XElement position = sceneObjectElement.Element("position");

sceneObject.BasePoint = new Point3D(

double.Parse(position.Attribute("x").Value, format),

double.Parse(position.Attribute("y").Value, format),

double.Parse(position.Attribute("z").Value, format)

);

XElement rotate = sceneObjectElement.Element("rotate");

sceneObject.AngleX = int.Parse(rotate.Attribute("x").Value);

sceneObject.AngleY = int.Parse(rotate.Attribute("y").Value);

sceneObject.AngleZ = int.Parse(rotate.Attribute("z").Value);

XElement scale = sceneObjectElement.Element("scale");

sceneObject.SetScale(

double.Parse(scale.Attribute("x").Value, format),

double.Parse(scale.Attribute("y").Value, format),

double.Parse(scale.Attribute("z").Value, format)

);

foreach (XElement scenePrimitiveElement in sceneObjectElement.Element("scene-primitives").Elements("scene-primitive"))

{

Primitive primitive = null;

XElement primitiveElement = scenePrimitiveElement.Element("primitive");

string type = primitiveElement.Attribute("type").Value;

XElement basePointElement = primitiveElement.Element("base-point");

Point3D basePoint = new Point3D(

double.Parse(basePointElement.Attribute("x").Value, format),

double.Parse(basePointElement.Attribute("y").Value, format),

double.Parse(basePointElement.Attribute("z").Value, format)

);

XElement colorElement = primitiveElement.Element("color");

Color color = Color.FromArgb(

int.Parse(colorElement.Attribute("r").Value),

int.Parse(colorElement.Attribute("g").Value),

int.Parse(colorElement.Attribute("b").Value)

);

switch (type)

{

case "box":

primitive = new Box(

basePoint,

double.Parse(primitiveElement.Element("width").Value, format),

double.Parse(primitiveElement.Element("height").Value, format),

double.Parse(primitiveElement.Element("length").Value, format),

color

);

break;

case "cylinder":

primitive = new Cylinder(

basePoint,

double.Parse(primitiveElement.Element("radius").Value, format),

double.Parse(primitiveElement.Element("height").Value, format),

int.Parse(primitiveElement.Element("segments-count").Value),

color

);

break;

case "sphere":

primitive = new Sphere(

basePoint,

double.Parse(primitiveElement.Element("radius").Value, format),

int.Parse(primitiveElement.Element("segments-count").Value),

color

);

break;

case "hemisphere":

primitive = new Hemisphere(

basePoint,

double.Parse(primitiveElement.Element("radius").Value, format),

int.Parse(primitiveElement.Element("segments-count").Value),

color

);

break;

}

ScenePrimitive scenePrimitive = new ScenePrimitive(primitive, scenePrimitiveElement.Attribute("name").Value);

XElement primitiveRotate = scenePrimitiveElement.Element("rotate");

scenePrimitive.AngleX = int.Parse(primitiveRotate.Attribute("x").Value);

scenePrimitive.AngleY = int.Parse(primitiveRotate.Attribute("y").Value);

scenePrimitive.AngleZ = int.Parse(primitiveRotate.Attribute("z").Value);

XElement primitiveScale = scenePrimitiveElement.Element("scale");

scenePrimitive.SetScale(

double.Parse(primitiveScale.Attribute("x").Value, format),

double.Parse(primitiveScale.Attribute("y").Value, format),

double.Parse(primitiveScale.Attribute("z").Value, format)

);

sceneObject.AddScenePrimitive(scenePrimitive);

}

scene.AddObject(sceneObject);

ObjectsList.Items.Add(sceneObject.Name);

}

foreach (XElement triplaneObjectElement in scene1.Element("objects").Elements("console-object"))

{

TriplaneObject triplaneObject = new TriplaneObject(triplaneObjectElement.Attribute("name").Value);

XElement position = triplaneObjectElement.Element("position");

triplaneObject.BasePoint = new Point3D(

double.Parse(position.Attribute("x").Value, format),

double.Parse(position.Attribute("y").Value, format),

double.Parse(position.Attribute("z").Value, format)

);

XElement rotate = triplaneObjectElement.Element("rotate");

triplaneObject.AngleX = int.Parse(rotate.Attribute("x").Value);

triplaneObject.AngleY = int.Parse(rotate.Attribute("y").Value);

triplaneObject.AngleZ = int.Parse(rotate.Attribute("z").Value);

XElement scale = triplaneObjectElement.Element("scale");

triplaneObject.SetScale(

double.Parse(scale.Attribute("x").Value, format),

double.Parse(scale.Attribute("y").Value, format),

double.Parse(scale.Attribute("z").Value, format)

);

triplaneObject.UpdateObject();

scene.AddObject(triplaneObject);

ObjectsList.Items.Add(triplaneObject.Name);

}

ObjectsList.SelectedIndex = -1;

if (scene.Objects.Count > 0)

{

ObjectsList.SelectedIndex = 0;

}

scene.PaintObjects();

}

catch (Exception exception)

{

MessageBox.Show("Unable to import scene! File is corrupted.", Text, MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

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

{

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

{

try

{

IFormatProvider format = new System.Globalization.CultureInfo("es-ES");

XDocument document = new XDocument();

XElement scene1 = new XElement("scene");

XElement cameras = new XElement("cameras");

for (int i = 0; i < scene.Cameras.Count; ++i)

{

Camera camera = scene.Cameras[i];

XElement cameraElement = new XElement("camera");

cameraElement.SetElementValue("central-projection", camera.IsCentralProjection);

XElement position = new XElement("position");

position.SetAttributeValue("x", string.Format(format, "{0:0.00}", camera.Position.X));

position.SetAttributeValue("y", string.Format(format, "{0:0.00}", camera.Position.Y));

position.SetAttributeValue("z", string.Format(format, "{0:0.00}", camera.Position.Z));

cameraElement.Add(position);

XElement target = new XElement("target");

target.SetAttributeValue("x", string.Format(format, "{0:0.00}", camera.Target.X));

target.SetAttributeValue("y", string.Format(format, "{0:0.00}", camera.Target.Y));

target.SetAttributeValue("z", string.Format(format, "{0:0.00}", camera.Target.Z));

cameraElement.Add(target);

XElement clippingPlanes = new XElement("clipping-planes");

clippingPlanes.SetAttributeValue("near", camera.NearClippingPlaneZ);

clippingPlanes.SetAttributeValue("far", camera.FarClippingPlaneZ);

cameraElement.Add(clippingPlanes);

cameraElement.SetElementValue("fov", camera.FOV);

cameras.Add(cameraElement);

}

scene1.Add(cameras);

XElement lights = new XElement("lights");

for (int i = 0; i < scene.Lights.Count; ++i)

{

Vector3D light = scene.Lights[i];

XElement lightElement = new XElement("light");

lightElement.SetAttributeValue("name", "Light-0");

lightElement.SetAttributeValue("x", "0,00");

lightElement.SetAttributeValue("y", "150,00");

lightElement.SetAttributeValue("z", "-150,00");

lights.Add(lightElement);

}

scene1.Add(lights);

XElement objects = new XElement("objects");

foreach (SceneObject sceneObject in scene.Objects)

{

XElement objectElement;

if (sceneObject is TriplaneObject)

{

TriplaneObject triplaneObject = (TriplaneObject)sceneObject;

objectElement = new XElement("console-object");

}

else

{

objectElement = new XElement("scene-object");

XElement scenePrimitivesElement = new XElement("scene-primitives");

foreach (ScenePrimitive scenePrimitive in sceneObject.ScenePrimitives)

{

XElement scenePrimitiveElement = new XElement("scene-primitive");

XElement primitiveRotate = new XElement("rotate");

primitiveRotate.SetAttributeValue("x", scenePrimitive.AngleX);

primitiveRotate.SetAttributeValue("y", scenePrimitive.AngleY);

primitiveRotate.SetAttributeValue("z", scenePrimitive.AngleZ);

scenePrimitiveElement.Add(primitiveRotate);

XElement primitiveScale = new XElement("scale");

primitiveScale.SetAttributeValue("x", string.Format(format, "{0:0.00}", scenePrimitive.ScaleX));

primitiveScale.SetAttributeValue("y", string.Format(format, "{0:0.00}", scenePrimitive.ScaleY));

primitiveScale.SetAttributeValue("z", string.Format(format, "{0:0.00}", scenePrimitive.ScaleZ));

scenePrimitiveElement.Add(primitiveScale);

XElement primitiveElement = new XElement("primitive");

XElement primitiveBasePoint = new XElement("base-point");

primitiveBasePoint.SetAttributeValue("x", string.Format(format, "{0:0.00}", scenePrimitive.Primitive.BasePoint.X));

primitiveBasePoint.SetAttributeValue("y", string.Format(format, "{0:0.00}", scenePrimitive.Primitive.BasePoint.Y));

primitiveBasePoint.SetAttributeValue("z", string.Format(format, "{0:0.00}", scenePrimitive.Primitive.BasePoint.Z));

primitiveElement.Add(primitiveBasePoint);

XElement primitiveColor = new XElement("color");

primitiveColor.SetAttributeValue("r", scenePrimitive.Primitive.Color.R);

primitiveColor.SetAttributeValue("g", scenePrimitive.Primitive.Color.G);

primitiveColor.SetAttributeValue("b", scenePrimitive.Primitive.Color.B);

primitiveElement.Add(primitiveColor);

if (scenePrimitive.Primitive is Box)

{

Box box = (Box)scenePrimitive.Primitive;

primitiveElement.SetAttributeValue("type", "box");

primitiveElement.SetElementValue("width", string.Format(format, "{0:0.00}", box.Width));

primitiveElement.SetElementValue("height", string.Format(format, "{0:0.00}", box.Height));

primitiveElement.SetElementValue("length", string.Format(format, "{0:0.00}", box.Length));

}

else if (scenePrimitive.Primitive is Cylinder)

{

Cylinder cylinder = (Cylinder)scenePrimitive.Primitive;

primitiveElement.SetAttributeValue("type", "cylinder");

primitiveElement.SetElementValue("radius", string.Format(format, "{0:0.00}", cylinder.Radius));

primitiveElement.SetElementValue("height", string.Format(format, "{0:0.00}", cylinder.Height));

primitiveElement.SetElementValue("segments-count", cylinder.SegmentsCount);

}

else if (scenePrimitive.Primitive is Sphere)

{

Sphere sphere = (Sphere)scenePrimitive.Primitive;

primitiveElement.SetAttributeValue("type", "sphere");

primitiveElement.SetElementValue("radius", string.Format(format, "{0:0.00}", sphere.Radius));

primitiveElement.SetElementValue("segments-count", sphere.SegmentsCount);

}

else if (scenePrimitive.Primitive is Hemisphere)

{

Hemisphere hemisphere = (Hemisphere)scenePrimitive.Primitive;

primitiveElement.SetAttributeValue("type", "hemisphere");

primitiveElement.SetElementValue("radius", string.Format(format, "{0:0.00}", hemisphere.Radius));

primitiveElement.SetElementValue("segments-count", hemisphere.SegmentsCount);

}

scenePrimitiveElement.Add(primitiveElement);

scenePrimitiveElement.SetAttributeValue("name", scenePrimitive.Name);

scenePrimitivesElement.Add(scenePrimitiveElement);

}

objectElement.Add(scenePrimitivesElement);

}

objectElement.SetAttributeValue("name", sceneObject.Name);

XElement objectPosition = new XElement("position");

objectPosition.SetAttributeValue("x", string.Format(format, "{0:0.00}", sceneObject.BasePoint.X));

objectPosition.SetAttributeValue("y", string.Format(format, "{0:0.00}", sceneObject.BasePoint.Y));

objectPosition.SetAttributeValue("z", string.Format(format, "{0:0.00}", sceneObject.BasePoint.Z));

objectElement.Add(objectPosition);

XElement objectRotate = new XElement("rotate");

objectRotate.SetAttributeValue("x", sceneObject.AngleX);

objectRotate.SetAttributeValue("y", sceneObject.AngleY);

objectRotate.SetAttributeValue("z", sceneObject.AngleZ);

objectElement.Add(objectRotate);

XElement objectScale = new XElement("scale");

objectScale.SetAttributeValue("x", string.Format(format, "{0:0.00}", sceneObject.ScaleX));

objectScale.SetAttributeValue("y", string.Format(format, "{0:0.00}", sceneObject.ScaleY));

objectScale.SetAttributeValue("z", string.Format(format, "{0:0.00}", sceneObject.ScaleZ));

objectElement.Add(objectScale);

objects.Add(objectElement);

}

scene1.Add(objects);

document.Add(scene1);

document.Save(ExportDialog.FileName);

}

catch (Exception exception)

{

MessageBox.Show("Unable to export scene!", Text, MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

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

{

объёмноеToolStripMenuItem.Image = null;

каркасToolStripMenuItem.Image = Properties.Resources.Check;

scene.Mode = Scene.MODE.WIREFRAME;

scene.PaintObjects();

}

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

{

каркасToolStripMenuItem.Image = null;

объёмноеToolStripMenuItem.Image = Properties.Resources.Check;

scene.Mode = Scene.MODE.SOLID;

scene.PaintObjects();

}

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

{

scene.Camera.RotatePositionLeftRight(0);

scene.Camera.RotatePositionUpDown(0);

UpdateCameraValues();

Rotation = 0;

объекToolStripMenuItem.Image = null;

позиционныйToolStripMenuItem.Image = Properties.Resources.Check;

}

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

{

scene.Camera.RotateTargetLeftRight(0);

scene.Camera.RotateTargetUpDown(0);

UpdateCameraValues();

Rotation = 1;

позиционныйToolStripMenuItem.Image = null;

объекToolStripMenuItem.Image = Properties.Resources.Check;

}

private void NumOfWings\_ValueChanged(object sender, EventArgs e)

{

((TriplaneObject)currentObject).NumOfWings = (double)NumOfWings.Value;

((TriplaneObject)currentObject).UpdateObject();

scene.PaintObjects();

}

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

{

SaveFileDialog saveFile = new SaveFileDialog();

saveFile.Filter = "PNG Image(\*.png)|\*.png";

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

{

RenderPicture.Image.Save(saveFile.FileName, ImageFormat.Png);

}

}

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

{

SaveFileDialog saveFile = new SaveFileDialog();

saveFile.Filter = "JPG Image(\*.jpg)|\*.jpg";

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

{

RenderPicture.Image.Save(saveFile.FileName, ImageFormat.Jpeg);

}

}

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

{

scene.Camera.IsCentralProjection = true;

}

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

{

scene.Camera.IsCentralProjection = false;

}

}

}

Hemisphere.cs

using System;

using System.Drawing;

namespace Game\_Consoles{

class Hemisphere : Primitive{

public bool CloseBottom { get; set; } = false;

public double Radius { get; private set; }

public int SegmentsCount { get; private set; }

public Hemisphere(Point3D basePoint, double radius, int segmentsCount, Color color, bool closeBottom = true) : base(basePoint, color){

Radius = radius;

SegmentsCount = Math.Max(segmentsCount, 4);

CloseBottom = closeBottom;

UpdatePoints();}

private void UpdatePoints(){

int quater = Math.Max(SegmentsCount / 4 - 1, 1);

double angle = 360.0 / SegmentsCount;

double sin = Math.Sin(MathHelp.DegreesToRadians(-angle));

double cos = Math.Cos(MathHelp.DegreesToRadians(-angle));

Point3D[,] points = new Point3D[quater, SegmentsCount + 1];

points[0, 0] = new Point3D(-Radius, 0, 0);

for (int i = 1; i < quater; ++i){

double x = points[i - 1, 0].X \* cos - points[i - 1, 0].Y \* sin;

double y = points[i - 1, 0].X \* sin + points[i - 1, 0].Y \* cos;

points[i, 0] = new Point3D(x, y, 0);}

Faces.Clear();

for (int i = 1; i <= SegmentsCount; ++i){

for (int j = 0; j < quater; ++j){

double x = points[j, i - 1].X \* cos - points[j, i - 1].Z \* sin;

double z = points[j, i - 1].X \* sin + points[j, i - 1].Z \* cos;

points[j, i] = new Point3D(x, points[j, i - 1].Y, z);

if (i > 0){

if (j > 0){

Faces.Add(new Face(new Point3D[] {

new Point3D(points[j - 1, i - 1].X, points[j - 1, i - 1].Y, points[j - 1, i - 1].Z),

new Point3D(points[j - 1, i].X, points[j - 1, i].Y, points[j - 1, i].Z),

new Point3D(points[j, i - 1].X, points[j, i - 1].Y, points[j, i - 1].Z)}));

Faces.Add(new Face(new Point3D[] {

new Point3D(points[j, i - 1].X, points[j, i - 1].Y, points[j, i - 1].Z),

new Point3D(points[j, i].X, points[j, i].Y, points[j, i].Z),

new Point3D(points[j - 1, i].X, points[j - 1, i].Y, points[j - 1, i].Z)}));}

else if (closeBottom){

Faces.Add(new Face(new Point3D[] {

new Point3D(points[j, i - 1].X, points[j, i - 1].Y, points[j, i - 1].Z),

new Point3D(points[j, i].X, points[j, i].Y, points[j, i].Z),

new Point3D(0, 0, 0)}));}

if (j == quater - 1){

Faces.Add(new Face(new Point3D[] {

new Point3D(points[j, i - 1].X, points[j, i - 1].Y, points[j, i - 1].Z),

new Point3D(points[j, i].X, points[j, i].Y, points[j, i].Z),

new Point3D(0, Radius, 0)}));}}}}}

public void ModifyRadius(double radius){

Radius = radius;

UpdatePoints();}

public void ModifySegmentsCount(int segmentsCount){

SegmentsCount = segmentsCount;

UpdatePoints();}

public void ModifyBasePoint(Point3D basePoint){

BasePoint = basePoint;

UpdatePoints();}}}

FastBitmap.cs

using System;

using System.Drawing;

using System.Drawing.Imaging;

using System.Runtime.InteropServices;

public class FastBitmap{

private int stride;

private byte[] data;

public int Width { get; }

public int Height { get; }

public FastBitmap(int width, int height, Color[,] colors){

data = ConvertToByte(colors);

Width = width;

Height = height;

stride = data.Length / height;}

private byte[] ConvertToByte(Color[,] colors){

byte[] result = new byte[colors.Length \* 4];

int position = 0;

for (int y = 0; y < colors.GetLength(1); ++y)

for (int x = 0; x < colors.GetLength(0); ++x){

result[position++] = colors[x, y].B;

result[position++] = colors[x, y].G;

result[position++] = colors[x, y].R;

result[position++] = colors[x, y].A;}

return result;}

public FastBitmap(int width, int height, byte[] data){

this.data = data;

Width = width;

Height = height;

stride = data.Length / height;}

public byte[] GetData(){

return data;}

public FastBitmap(Bitmap image){

BitmapData bits = image.LockBits(new Rectangle(0, 0, image.Width, image.Height), ImageLockMode.ReadWrite, PixelFormat.Format32bppArgb);

stride = bits.Stride;

int bytes = stride \* bits.Height;

data = new byte[bytes];

Marshal.Copy(bits.Scan0, data, 0, bytes);

image.UnlockBits(bits);

Width = image.Width;

Height = image.Height;}

public Bitmap GetBitmap(){

Bitmap result = new Bitmap(Width, Height);

BitmapData bits = result.LockBits(new Rectangle(0, 0, Width, Height), ImageLockMode.WriteOnly, PixelFormat.Format32bppArgb);

Marshal.Copy(data, 0, bits.Scan0, data.Length);

result.UnlockBits(bits);

return result;}

public Color GetPixel(int x, int y){

int position = y \* stride + x \* 4;

int b = data[position++];

int g = data[position++];

int r = data[position++];

int a = data[position++];

return Color.FromArgb(a, r, g, b);}

private void SetPixel(int x, int y, byte r, byte g, byte b, byte a = 255){

if (x > 0 && x < Width && y > 0 && y < Height){

int position = y \* stride + x \* 4;

data[position] = (byte)(b \* ((float)a / 255) + data[position] \* ((float)(255 - a) / 255));

++position;

data[position] = (byte)(g \* ((float)a / 255) + data[position] \* ((float)(255 - a) / 255));

++position;

data[position] = (byte)(r \* ((float)a / 255) + data[position] \* ((float)(255 - a) / 255));

++position;

data[position] = (byte)(a \* ((float)a / 255) + data[position] \* ((float)(255 - a) / 255));}}

public void SetPixel(int x, int y, Color color){

SetPixel(x, y, color.R, color.G, color.B, color.A);}

public void SwapPixels(int x, int y, int newX, int newY){

Color color = GetPixel(x, y);

SetPixel(x, y, GetPixel(newX, newY));

SetPixel(newX, newY, color);}

public void DrawRectangle(Rectangle rectangle, Color color){

for (int y = rectangle.Y; y < rectangle.Y + rectangle.Height; ++y)

for (int x = rectangle.X; x < rectangle.X + rectangle.Width; ++x)

SetPixel(x, y, color.R, color.G, color.B, color.A);}

public void Fill(Color color){

DrawRectangle(new Rectangle(0, 0, Width, Height), color);}

public void DrawLine(int startX, int startY, int endX, int endY, Color color){

bool swap = false;

if (Math.Abs(endY - startY) > Math.Abs(endX - startX)){

Swap(ref startX, ref startY);

Swap(ref endX, ref endY);

swap = true;}

if (endX < startX){

Swap(ref startX, ref endX);

Swap(ref startY, ref endY);}

int lengthX = endX - startX;

int doubleLengthY = Math.Abs(endY - startY) \* 2;

int doubleLengthX = lengthX \* 2;

int step = startY < endY ? 1 : -1;

int y = startY;

int error = 0;

for (int x = startX; x <= endX; ++x){

if (swap) SetPixel(y, x, color);

else SetPixel(x, y, color);

error += doubleLengthY;

if (error > lengthX){

y += step;

error -= doubleLengthX;}}}

private void Swap(ref int a, ref int b){

int temp = a;

a = b;

b = temp;}}

Face.cs

using System.Drawing;

namespace Game\_Consoles{

public class Face{

public Point3D[] Points { get; private set; }

public Face(Point3D[] points){

Points = points;}

public Vector3D GetNormalVector(){

double ux = Points[1].X - Points[0].X;

double uy = Points[1].Y - Points[0].Y;

double uz = Points[1].Z - Points[0].Z;

double vx = Points[2].X - Points[0].X;

double vy = Points[2].Y - Points[0].Y;

double vz = Points[2].Z - Points[0].Z;

Vector3D u = new Vector3D(ux, uy, uz);

Vector3D v = new Vector3D(vx, vy, vz);

return (u ^ v).GetNormalized();}}}

Cylinder.cs

using System;

using System.Collections.Generic;

using System.Drawing;

namespace Game\_Consoles{

public class Cylinder : Primitive{

public double Radius { get; private set; }

public double Height { get; private set; }

public int SegmentsCount { get; private set; }

public Cylinder(Point3D basePoint, double radius, double height, int segmentsCount, Color color) : base(basePoint, color){

Radius = radius;

Height = height;

SegmentsCount = Math.Max(segmentsCount, 3);

UpdatePoints();}

private void UpdatePoints(){

double angle = 360.0 / SegmentsCount;

double sin = Math.Sin(MathHelp.DegreesToRadians(angle));

double cos = Math.Cos(MathHelp.DegreesToRadians(angle));

Point3D[] points = new Point3D[SegmentsCount + 1];

points[0] = new Point3D(-Radius, 0, 0);

Faces.Clear();

for (int i = 1; i <= SegmentsCount; ++i){

double x = points[i - 1].X \* cos - points[i - 1].Z \* sin;

double z = points[i - 1].X \* sin + points[i - 1].Z \* cos;

points[i] = new Point3D(x, 0, z);

if (i > 0){

Faces.Add(new Face(new Point3D[] {

new Point3D(points[i - 1].X, 0, points[i - 1].Z),

new Point3D(points[i].X, 0, points[i].Z),

new Point3D(0, 0, 0)}));

Faces.Add(new Face(new Point3D[] {

new Point3D(points[i - 1].X, Height, points[i - 1].Z),

new Point3D(points[i].X, Height, points[i].Z),

new Point3D(0, Height, 0)}));

Faces.Add(new Face(new Point3D[] {

new Point3D(points[i - 1].X, Height, points[i - 1].Z),

new Point3D(points[i].X, Height, points[i].Z),

new Point3D(points[i - 1].X, 0, points[i - 1].Z)}));

Faces.Add(new Face(new Point3D[] {

new Point3D(points[i - 1].X, 0, points[i - 1].Z),

new Point3D(points[i].X, 0, points[i].Z),

new Point3D(points[i].X, Height, points[i].Z)}));}}}

public void ModifyRadius(double radius){

Radius = radius;

UpdatePoints();}

public void ModifyHeight(double height){

Height = height;

UpdatePoints();}

public void ModifySegmentsCount(int segmentsCount){

SegmentsCount = segmentsCount;

UpdatePoints();}

public void ModifyBasePoint(Point3D basePoint){

BasePoint = basePoint;

UpdatePoints();}}}

CreateObject.cs

using System.Drawing;

using System.Windows.Forms;

namespace Game\_Consoles{

public partial class CreateObject : Form{

private MainForm mainForm = null;

private SceneObject parentObject = null;

public CreateObject(MainForm mainForm, SceneObject parentObject){

this.mainForm = mainForm;

this.parentObject = parentObject;

InitializeComponent();

PositionX.Minimum = PositionY.Minimum = PositionZ.Minimum = mainForm.MinCoordinate;

PositionX.Maximum = PositionY.Maximum = PositionZ.Maximum = mainForm.MaxCoordinate;

if (parentObject != null){

ObjectType.Items.Add("Box");

ObjectType.Items.Add("Cylinder");

ObjectType.Items.Add("Sphere");

ObjectType.Items.Add("Hemisphere");

ColorBox.Enabled = true;

ParametersPanel.Visible = !(parentObject is TriplaneObject);}

else{

ObjectType.Items.Add("Empty");

ObjectType.Items.Add("Console");

ParametersPanel.Visible = false;

ColorLabel.Visible = ColorBox.Visible = false;}

ObjectType.SelectedIndex = 0;}

private void ObjectType\_SelectedIndexChanged(object sender, System.EventArgs e){

RadiusBox.Enabled = false;

SegmentsBox.Enabled = false;

WidthBox.Enabled = false;

HeightBox.Enabled = false;

LengthBox.Enabled = false;

if (parentObject != null){

switch (ObjectType.SelectedItem.ToString()){

case "Box":

WidthBox.Enabled = true;

HeightBox.Enabled = true;

LengthBox.Enabled = true;

break;

case "Cylinder":

RadiusBox.Enabled = true;

HeightBox.Enabled = true;

SegmentsBox.Enabled = true;

break;

case "Sphere":

RadiusBox.Enabled = true;

SegmentsBox.Enabled = true;

break;

case "Hemisphere":

RadiusBox.Enabled = true;

SegmentsBox.Enabled = true;

break;}}}

private void ObjectName\_TextChanged(object sender, System.EventArgs e){

AddObject.Enabled = ObjectName.TextLength >= 3;}

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

Point3D basePoint = new Point3D(

(double)PositionX.Value,

(double)PositionY.Value,

(double)PositionZ.Value

);

if (parentObject != null){

if (parentObject.GetScenePrimitiveByName(ObjectName.Text) == null){

Primitive primitive = null;

switch (ObjectType.SelectedItem.ToString()){

case "Box":

primitive = new Box(basePoint, (int)WidthBox.Value, (int)HeightBox.Value, (int)LengthBox.Value, ColorPicker.Color);

break;

case "Cylinder":

primitive = new Cylinder(basePoint, (int)RadiusBox.Value, (int)HeightBox.Value, (int)SegmentsBox.Value, ColorPicker.Color);

break;

case "Sphere":

primitive = new Sphere(basePoint, (int)RadiusBox.Value, (int)SegmentsBox.Value, ColorPicker.Color);

break;

case "Hemisphere":

primitive = new Hemisphere(basePoint, (int)RadiusBox.Value, (int)SegmentsBox.Value, ColorPicker.Color);

break;}

ScenePrimitive scenePrimitive = new ScenePrimitive(primitive, ObjectName.Text);

scenePrimitive.AngleX = (int)RotateX.Value;

scenePrimitive.AngleY = (int)RotateY.Value;

scenePrimitive.AngleZ = (int)RotateZ.Value;

scenePrimitive.SetScale((double)ScaleX.Value, (double)ScaleY.Value, (double)ScaleZ.Value);

parentObject.AddScenePrimitive(scenePrimitive);

int index = mainForm.PrimitivesList.Items.Add(ObjectName.Text);

mainForm.PrimitivesList.SelectedIndex = index;

Close();}

else{

MessageBox.Show("Object with this name already exists!", Text, MessageBoxButtons.OK, MessageBoxIcon.Warning);

ObjectName.Clear();

ObjectName.Focus();}}

else{

if (mainForm.Scene.GetObjectByName(ObjectName.Text) == null){

SceneObject sceneObject;

if (ObjectType.SelectedItem.ToString().Equals("Empty")){

sceneObject = new SceneObject(ObjectName.Text);}

else{

sceneObject = new TriplaneObject(ObjectName.Text);}

sceneObject.BasePoint = basePoint;

sceneObject.AngleX = (int)RotateX.Value;

sceneObject.AngleY = (int)RotateY.Value;

sceneObject.AngleZ = (int)RotateZ.Value;

sceneObject.SetScale((double)ScaleX.Value, (double)ScaleY.Value, (double)ScaleZ.Value);

mainForm.Scene.AddObject(sceneObject);

int index = mainForm.ObjectsList.Items.Add(ObjectName.Text);

mainForm.ObjectsList.SelectedIndex = index;

mainForm.Pan.Enabled = true;

Close();}

else{

MessageBox.Show("Object with this name already exists!", Text, MessageBoxButtons.OK, MessageBoxIcon.Warning);

ObjectName.Clear();

ObjectName.Focus();}}}

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

if (ColorPicker.ShowDialog() == DialogResult.OK){

ColorBox.BackColor = ColorPicker.Color;}}}}

TriplaneObject.cs

using System.Drawing;

namespace \_3DEngine

{

public class TriplaneObject : SceneObject

{

public double WingWidth { get; set; } = 15;

public double widthLowerChassis { get; set; } = 2;

public double radiusChassis { get; set; } = 7;

public double widthVerticalWingSupports { get; set; } = 2;

public double widthVerticalBackWing { get; set; } = 20;

public double widthLop { get; set; } = 2;

public double Sit { get; set; } = 10;

public double NumOfWings { get; set; } = 3;

public TriplaneObject(string name) : base(name)

{

UpdateObject();

}

public void UpdateObject()

{

ScenePrimitives.Clear();

ScenePrimitive fuselage = new ScenePrimitive(

new Cylinder(new Point3D(-30, 0, 0), 10, 70, 16, Color.FromArgb(211, 211, 211)),

"Фюзеляж"

);

fuselage.AngleZ = -90;

ScenePrimitives.Add(fuselage);

ScenePrimitive auxiliary1 = new ScenePrimitive(

new Hemisphere(new Point3D(-100, 0, 0), 10, 16, Color.FromArgb(211, 211, 211)),

"Вспомогательный1"

);

auxiliary1.AngleZ = -90;

ScenePrimitives.Add(auxiliary1);

if(NumOfWings==2 | NumOfWings == 1)

{

ScenePrimitive wing1 = new ScenePrimitive(

new Box(new Point3D(-91.5, 13, 0), 0, 0, 0, Color.FromArgb(255, 255, 255)),

"Крыло1"

);

ScenePrimitives.Add(wing1);

}

else

{

ScenePrimitive wing1 = new ScenePrimitive(

new Box(new Point3D(-91.5, 13, 0), WingWidth, 2, 100, Color.FromArgb(211, 211, 211)),

"Крыло1"

);

ScenePrimitives.Add(wing1);

}

ScenePrimitive wing2 = new ScenePrimitive(

new Box(new Point3D(-91, -10, 0), WingWidth, 2, 100, Color.FromArgb(211, 211, 211)),

"Крыло2"

);

ScenePrimitives.Add(wing2);

if (NumOfWings == 1)

{

ScenePrimitive wing3 = new ScenePrimitive(

new Box(new Point3D(-91, 30, 0), 0, 0, 0, Color.FromArgb(255, 255, 255)),

"Крыло3"

);

ScenePrimitives.Add(wing3);

}

else

{

ScenePrimitive wing3 = new ScenePrimitive(

new Box(new Point3D(-91, 30, 0), WingWidth, 2, 100, Color.FromArgb(211, 211, 211)),

"Крыло3"

);

ScenePrimitives.Add(wing3);

}

ScenePrimitive auxiliary2 = new ScenePrimitive(

new Sphere(new Point3D(-108, 0, 0), 5, 16, Color.FromArgb(211, 211, 211)),

"Вспомогательный2"

);

ScenePrimitives.Add(auxiliary2);

ScenePrimitive screw = new ScenePrimitive(

new Box(new Point3D(-109.5, -2, 0), widthLop, 5, 45, Color.FromArgb(211, 211, 211)),

"Винт"

);

ScenePrimitives.Add(screw);

if (NumOfWings == 1)

{

ScenePrimitive support1 = new ScenePrimitive(

new Box(new Point3D(-92, -9, -30), 0, 0, 0, Color.FromArgb(211, 211, 211)),

"Поддержка1"

);

ScenePrimitives.Add(support1);

ScenePrimitive support2 = new ScenePrimitive(

new Box(new Point3D(-92, -9, 30), 0, 0, 0, Color.FromArgb(211, 211, 211)),

"Поддержка2"

);

ScenePrimitives.Add(support2);

ScenePrimitive support3 = new ScenePrimitive(

new Box(new Point3D(-92, 6, -2.5), 0, 0, 0, Color.FromArgb(211, 211, 211)),

"Поддержка3"

);

ScenePrimitives.Add(support3);

ScenePrimitive support4 = new ScenePrimitive(

new Box(new Point3D(-92, 6, 2.5), 0, 0, 0, Color.FromArgb(211, 211, 211)),

"Поддержка4"

);

ScenePrimitives.Add(support4);

}

else

{

ScenePrimitive support1 = new ScenePrimitive(

new Box(new Point3D(-92, -9, -30), widthVerticalWingSupports, 42, 2, Color.FromArgb(211, 211, 211)),

"Поддержка1"

);

ScenePrimitives.Add(support1);

ScenePrimitive support2 = new ScenePrimitive(

new Box(new Point3D(-92, -9, 30), widthVerticalWingSupports, 42, 2, Color.FromArgb(211, 211, 211)),

"Поддержка2"

);

ScenePrimitives.Add(support2);

ScenePrimitive support3 = new ScenePrimitive(

new Box(new Point3D(-92, 6, -2.5), widthVerticalWingSupports, 26, 2, Color.FromArgb(211, 211, 211)),

"Поддержка3"

);

ScenePrimitives.Add(support3);

ScenePrimitive support4 = new ScenePrimitive(

new Box(new Point3D(-92, 6, 2.5), widthVerticalWingSupports, 26, 2, Color.FromArgb(211, 211, 211)),

"Поддержка4"

);

ScenePrimitives.Add(support4);

}

ScenePrimitive bearingChassis1 = new ScenePrimitive(

new Box(new Point3D(-94, -25, 5), widthLowerChassis, 17, 2, Color.FromArgb(211, 211, 211)),

"ОпораШасси1"

);

bearingChassis1.AngleZ = -20;

ScenePrimitives.Add(bearingChassis1);

ScenePrimitive bearingChassis2 = new ScenePrimitive(

new Box(new Point3D(-94, -25, -5), widthLowerChassis, 17, 2, Color.FromArgb(211, 211, 211)),

"ОпораШасси2"

);

bearingChassis2.AngleZ = -20;

ScenePrimitives.Add(bearingChassis2);

ScenePrimitive bearingChassis3 = new ScenePrimitive(

new Box(new Point3D(-90, -25, -4.5), widthLowerChassis, 17, 2, Color.FromArgb(211, 211, 211)),

"ОпораШасси3"

);

bearingChassis3.AngleZ = 20;

ScenePrimitives.Add(bearingChassis3);

ScenePrimitive bearingChassis4 = new ScenePrimitive(

new Box(new Point3D(-90, -25, 4), widthLowerChassis, 17, 2, Color.FromArgb(211, 211, 211)),

"ОпораШасси4"

);

bearingChassis4.AngleZ = 20;

ScenePrimitives.Add(bearingChassis4);

ScenePrimitive chassis1 = new ScenePrimitive(

new Cylinder(new Point3D(-92, -27, -4), radiusChassis, 2, 16, Color.FromArgb(211, 211, 211)),

"Шасси1"

);

chassis1.AngleX = 90;

ScenePrimitives.Add(chassis1);

ScenePrimitive chassis2 = new ScenePrimitive(

new Cylinder(new Point3D(-92, -27, 6), radiusChassis, 2, 16, Color.FromArgb(211, 211, 211)),

"Шасси2"

);

chassis2.AngleX = 90;

ScenePrimitives.Add(chassis2);

ScenePrimitive bearingChassis5 = new ScenePrimitive(

new Box(new Point3D(-29, -18, 0), 2, 9, 2, Color.FromArgb(211, 211, 211)),

"ОпораШасси5"

);

bearingChassis5.AngleZ = -15;

ScenePrimitives.Add(bearingChassis5);

ScenePrimitive chassis3 = new ScenePrimitive(

new Cylinder(new Point3D(-29, -17, 0), 4, 2, 16, Color.FromArgb(211, 211, 211)),

"Шасси3"

);

chassis3.AngleX = 90;

ScenePrimitives.Add(chassis3);

ScenePrimitive bearingChassis6 = new ScenePrimitive(

new Box(new Point3D(-92, -27, 5), 2, 9, 2, Color.FromArgb(211, 211, 211)),

"ОпораШасси6"

);

bearingChassis6.AngleX = 90;

ScenePrimitives.Add(bearingChassis6);

ScenePrimitive tail1 = new ScenePrimitive(

new Cylinder(new Point3D(-30, 3.5, 0), 6, 30, 16, Color.FromArgb(211, 211, 211)),

"Хвост1"

);

tail1.AngleZ = 79;

ScenePrimitives.Add(tail1);

ScenePrimitive tail2 = new ScenePrimitive(

new Hemisphere(new Point3D(-0.5, 9.5, 0), 6, 16, Color.FromArgb(211, 211, 211)),

"Хвост2"

);

tail2.AngleZ = 75;

ScenePrimitives.Add(tail2);

ScenePrimitive auxiliary3 = new ScenePrimitive(

new Sphere(new Point3D(-29, 0, 0), 10, 16, Color.FromArgb(211, 211, 211)),

"Вспомогательный3"

);

ScenePrimitives.Add(auxiliary3);

ScenePrimitive tailWing1 = new ScenePrimitive(

new Box(new Point3D(-14, 7, 20), widthVerticalBackWing, 40, 2, Color.FromArgb(211, 211, 211)),

"КрылоХвоста1"

);

tailWing1.AngleX = 90;

tailWing1.AngleY = 10;

ScenePrimitives.Add(tailWing1);

ScenePrimitive tailWing2 = new ScenePrimitive(

new Box(new Point3D(-12, 2, 0), 20, 26, 2, Color.FromArgb(211, 211, 211)),

"КрылоХвоста2"

);

tailWing2.AngleZ = -11;

ScenePrimitives.Add(tailWing2);

ScenePrimitive sitPlace = new ScenePrimitive(

new Hemisphere(new Point3D(-73.5, 10, 0), Sit, 16, Color.FromArgb(0, 0, 0)),

"Сиденье"

);

sitPlace.AngleX = 180;

ScenePrimitives.Add(sitPlace);

}

}

}

AddObject.cs

using System;

using System.Drawing;

using System.Windows.Forms;

namespace \_3DEngine

{

public partial class AddObject : Form

{

private Main mainForm = null;

public AddObject(Main mainForm, SceneObject triplaneObject)

{

this.mainForm = mainForm;

InitializeComponent();

}

private void Button1\_Click(object sender, EventArgs e)

{

Point3D basePoint = new Point3D(

(double)PositionX.Value,

(double)PositionY.Value,

(double)PositionZ.Value

);

TriplaneObject sceneObject;

sceneObject = new TriplaneObject(ObjectName.Text);

sceneObject.BasePoint = basePoint;

sceneObject.AngleX = (int)RotateX.Value;

sceneObject.AngleY = (int)RotateY.Value;

sceneObject.AngleZ = (int)RotateZ.Value;

sceneObject.SetScale((double)ScaleX.Value, (double)ScaleY.Value, (double)ScaleZ.Value);

mainForm.scene.AddObject(sceneObject);

int index = mainForm.ObjectsList.Items.Add(ObjectName.Text);

mainForm.ObjectsList.SelectedIndex = index;

mainForm.scene.PaintObjects();

Close();

}

private void ObjectName\_TextChanged(object sender, EventArgs e)

{

AddObj.Enabled = ObjectName.TextLength >= 2;

if (mainForm.scene.GetObjectByName(ObjectName.Text) != null)

{

ObjectName.ForeColor = Color.Red;

ErrorLabel.Text = "Объект с таким именем уже существует!";

RotateX.Enabled = RotateY.Enabled = RotateZ.Enabled = ScaleX.Enabled = ScaleY.Enabled = ScaleZ.Enabled = PositionX.Enabled = PositionY.Enabled = PositionZ.Enabled = AddObj.Enabled = false;

}

else if(ObjectName.TextLength >= 2 & mainForm.scene.GetObjectByName(ObjectName.Text) == null)

{

ErrorLabel.Text = null;

ObjectName.ForeColor = Color.Green;

RotateX.Enabled = RotateY.Enabled = RotateZ.Enabled = ScaleX.Enabled = ScaleY.Enabled = ScaleZ.Enabled = PositionX.Enabled = PositionY.Enabled = PositionZ.Enabled = AddObj.Enabled = true;

}

}

}

}