**ООО АвиАРеаЛ.**

**Программный продукт «ОКО»**

**Общие сведения**

Программа предназначена для выполнения в среде разработки 3D приложений Unity (текущая версия 2018.3.0f2) написана на языке С#. Для обеспечения части функциональности использует свободно распространяемые модули для Unity сторонних разработчиков:

1. Загрузка по сети Интернет и отображение карты местности Mapbox (<https://www.mapbox.com>).
2. Создание и отображение дорог EasyRoads3D от AndaSoft (<https://assetstore.unity.com/packages/3d/characters/easyroads3d-free-v3-987>)

Программа выполняется в режиме группы асинхронных потоков. В действительности, настоящим фоновым потоком C# является только часть обработки данных – «Первичная обработка», в которую вынесены все операции, не работающие напрямую с объектами сцены. Это связано с тем, что Unity выполняется в одном потоке и позволят осуществлять доступ к этим объектам только из своего основного потока. Для асинхронных операций имеются псевдопотоки, реализуемые через «сопрограммы» (coroutines), а также используется асинхронный веб запрос UnityWebRequest и вызовы Update для разных объектов сцены.

Детали реализации потока загрузки и отрисовки карты Mapbox неизвестны. Общая схема потоков приведена на рисунке 1.

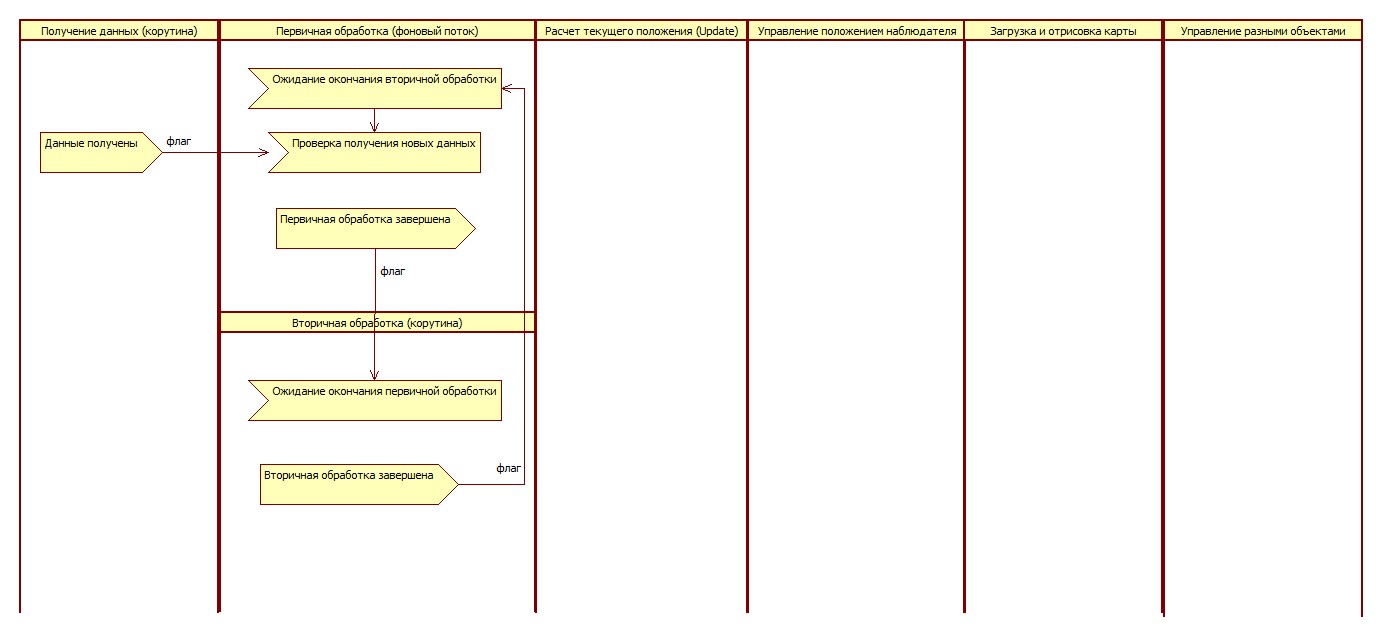


Рис. 1. Общая схема потоков выполнения.

Основная функциональность программы (запросы и получение данных состояний самолетов, обработка данных, построение траекторий, отображение моделей самолетов и их перемещение по траекториям) сосредоточены в классе sFligthRadar служебного объекта сцены Boss.

Управление положением наблюдателя осуществляется в классе sMortarMovement объекта сцены Mortar (является родителем для главной камеры сцены).

**Данные состояния самолетов.**

Основные данные о полете, используемые в программе:

* ID самолета. 24 битный код уникальный код транспондера, присваиваемый ИКАО. Представлен 6-значной HEX строкой.
* Позывной. В большинстве случаев содержит буквенный код авиакомпании и номер рейса.
* Время создания данных (предположительно формируется транспондером).
* Географические координаты (широта, долгота).
* Высота полета.
* Скорость относительно земли (горизонтальная скорость).
* Вертикальная скорость.
* Курсовой угол (в градусах, по часовой стрелке от направления на север).

Данные о состоянии самолетов программа получает путем выполнения HTTP запросов к бесплатным онлайн сервисам (раньше был <https://www.adsbexchange.com>, в настоящее время <https://opensky-network.org>, для улучшения работы их следует использовать оба).

В ответ оба сервиса возвращают строку в формате JSON, содержащую данные.

1. Adsbexchange

Запрос: <http://public-api.adsbexchange.com/VirtualRadar/AircraftList.json?lat=Latitude&lng=Longitude&fDstL=0&fDstU=Distance>

Запрашиваются данные всех самолетов, находящихся в окрестности точки с координатами **Latitude**, **Longitude** на расстоянии <= **Distance**.

Ответ: <http://www.virtualradarserver.co.uk/Documentation/Formats/AircraftList.aspx#response>

Дополнительная информация: <https://www.adsbexchange.com/datafields/>

Данный запрос удален с сайта, но продолжает работать. Требуется замена на новый формат, предполагающий регистрацию и поставку данных в Adsbexchange с собственного фидера.

1. Opensky Network

Запрос: <https://aka2001:MyOpenSky@opensky-network.org/api/states/all?lamin=WestMargin&lomin=SouthMargin&lamax=EastMarginlomax=NorthMargin>

aka2001:MyOpenSky – логин:пароль к учетной записи. Запрашиваются данные всех самолетов в прямоугольнике с координатами WestMargin, SouthMargin, EastMargin, NorthMargin.

Ответ: <https://opensky-network.org/apidoc/rest.html#response>

Набор возвращаемых данных и их форматы у источников отличаются.

Например, Adsbexchange передает горизонтальную скорость ВС в узлах, а вертикальную – футах в минуту, а Opensky – обе скорости в метрах в секунду.

Adsbexchange поставляет также данные о модели ВС, авиакомпании, исходном и конечном пунктах выполняемого рейса, которых нет у Opensky.

Пример строки JSON от Adsbexchange.

{"src":1,"feeds":[{"id":1,"name":"From Cons","polarPlot":false}],"srcFeed":1,"showSil":true,"showFlg":true,"showPic":true,"flgH":20,"flgW":85,"acList":[{"Id":11099300,"Rcvr":1,"HasSig":true,"Sig":6,"Icao":"A95CA4","Bad":false,"Reg":"N702GT","FSeen":"\/Date(1547858841329)\/","TSecs":4994,"CMsgs":1572,"Alt":33000,"GAlt":32903,"InHg":29.822834,"AltT":0,"Call":"SOO7950","Lat":51.706269,"Long":-0.37117,"PosTime":1547863835107,"Mlat":false,"Tisb":false,"Spd":507.5,"Trak":110.3,"TrkH":false,"Type":"B77L","Mdl":"Boeing 777-F16","Man":"Boeing","CNum":"38091","From":"KCVG Cincinnati Northern Kentucky, United States","To":"VHHH Chek Lap Kok, Hong Kong, Hong Kong","Stops":["OBBI Bahrain, Manama, Bahrain"],"Op":"Southern Air","OpIcao":"SOO","Sqk":"1153","Help":false,"Vsi":0,"VsiT":0,"Dst":26.88,"Brng":11.9,"WTC":3,"Species":1,"Engines":"2","EngType":3,"EngMount":0,"Mil":false,"Cou":"United States","HasPic":false,"Interested":false,"FlightsCount":0,"Gnd":false,"SpdTyp":0,"CallSus":false,"Trt":2,"Year":"2012","Sat":true}],"totalAc":6262,"lastDv":"636834313183610944","shtTrlSec":65,"stm":1547863835761}

Пример (сокращенный) строки JSON от Opensky Network.

{"time":1546777009,"states":[["4249b8","SYL474 ","United Kingdom",1546776935,1546776988,37,2387,55,5898,167,64,false,73,29,66,85,-3,9,null,160,02,"2457",false,0],["42496d","SVR728 ","United Kingdom",1546777009,1546777009,37,3563,55,5668,11277,6,false,221,73,89,73,-0,33,null,10721,34,"5165",false,0],["42491f","AFL1078 ","United Kingdom",1546777009,1546777009,37,2861,56,0518,3749,04,false,194,79,252,95,7,8,null,3596,64,"2130",false,0], ... ["157711","RSD032 ","Russian Federation",1546777009,1546777009,37,4651,55,4935,2385,06,false,144,75,165,17,15,61,null,2278,38,"1545",false,0]]}

Полученная от источника строка JSON содержит информацию, общую для всего запроса и массив данных состояний самолетов в зоне наблюдения.

Программа десериализует строку с помощью .NET фреймворка Newtonsoft Json.NET, затем все данные по каждому самолету переписываются из массива состояний в структуру C# (при этом проводится проверка и приведение типов данных и допустимых значений). Экземпляры структур данных по каждому самолету записываются в словарь (специальный массив в C# с доступом к элементам не по индексу, а по ключу). Ключом является код ИКАО транспондера самолета. В программе используется ряд словарей, отражающих разные аспекты обработки данных самолетов, все они используют в качестве ключа код ИКАО.

**Порядок работы программы**

Схема работы потоков, обеспечивающих получение данных, построение траекторий и подготовку моделей к отображению приведена на рисунке 2.

После начальной инициализации запускаются три подпрограммы – сопрограмма получения данных, фоновый поток первичной обработки и сопрограмма вторичной обработки. Все три реализованы как бесконечные циклы с фазами ожидания или разрешения от других подпрограмм (путем установки/проверки флагов – общедоступных логических переменных) или окончания периода обработки (если обработка завершилась до истечения времени очередного цикла, то подпрограмма ждет оставшееся время, а затем продолжает работу).

1. Сопрограмма получения данных.

Самая простая процедура: выполняет асинхронный веб запрос UnityWebRequest к источнику данных (называется асинхронным, так как выполняется в инструкции **yield return** и на время ожидания ответа отдает управление другим скриптам Unity), помещает полученный ответ в текстовую строку и поднимает флаг NewWebData = true. После выполнения ожидает до окончания рекомендованного цикла выполнения запросов, также с отдачей управления другим скриптам. Время цикла запросов - 5 секунд (подпрограмма ждет 5 секунд, минус время от выполнения веб запроса до получения ответа – это может быть и 2, и 1 секунду, и даже 0, если ответ на запрос пришел позже). После этого снова выполняется веб запрос.

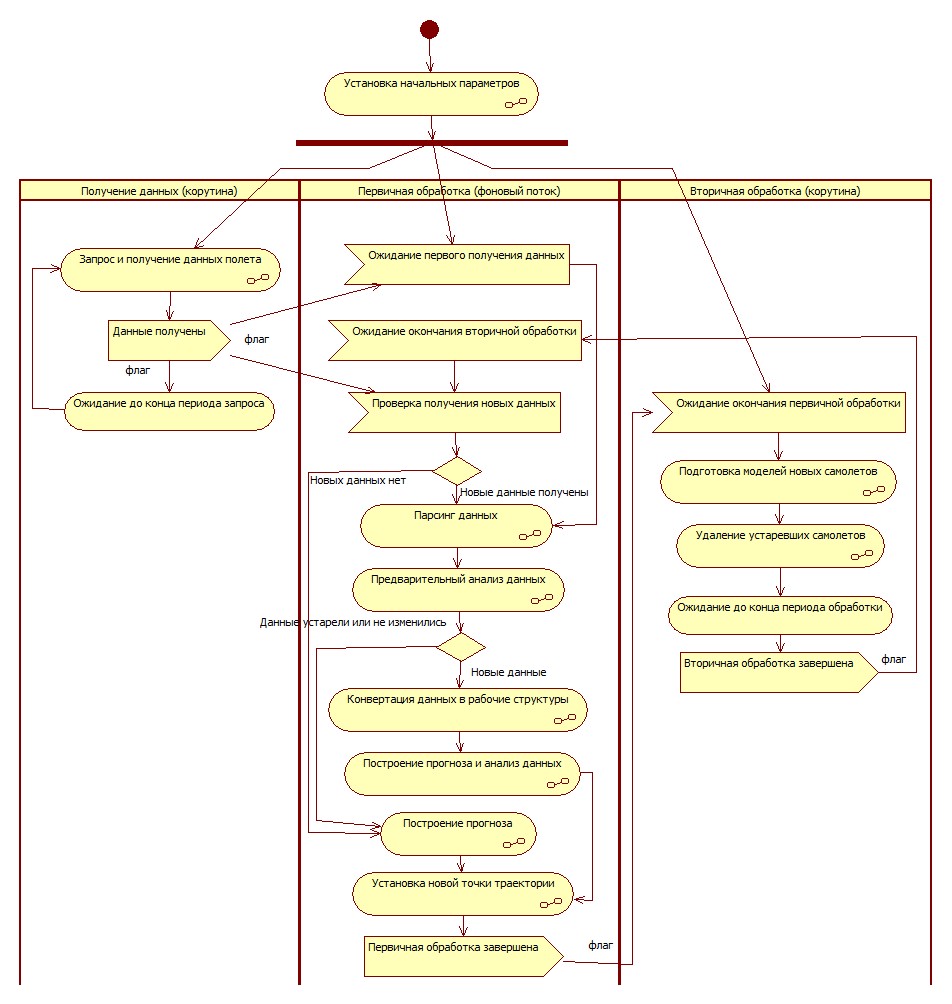


Рис. 2. Получение данных, построение траекторий, создание и удаление моделей самолетов.

1. Фоновый поток первичной обработки.

Сначала в цикле проверяет флаг получения новых данных. Если данных нет – поток «засыпает» на 200 миллисекунд, затем проверка повторяется. Так как поток фоновый, его «засыпание» не приводит к задержке работы других скриптов Unity.

Как только обнаруживается, что новые данные получены, подпрограмма парсит новую строку JSON, выбирает из нее необходимый минимум данных (код ИКАО, время формирования данных и географические координаты) и начинает их первичную проверку (см. рисунок 3).

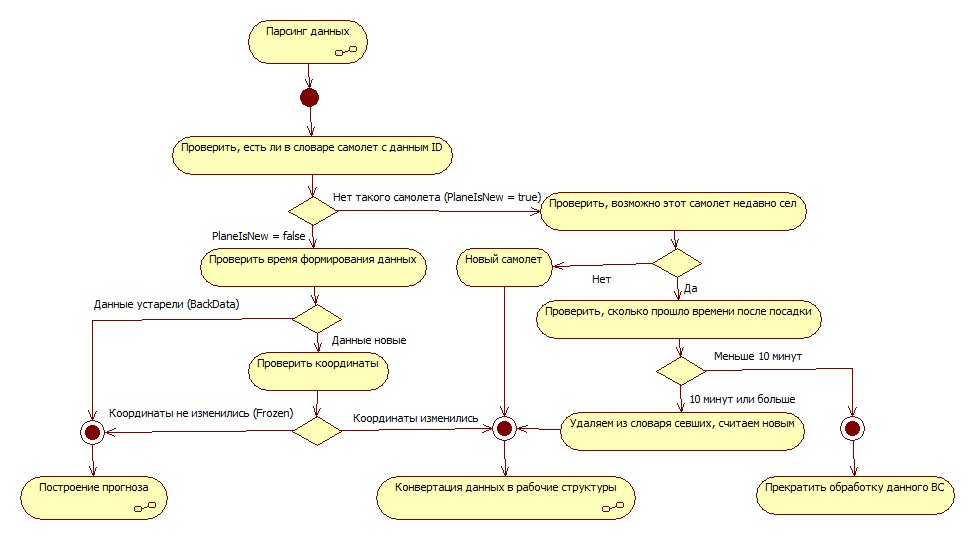


Рис. 3. Предварительный анализ данных.

Если вновь полученные данные имеют такое же (или более раннее) время формирования, что и полученные в предыдущий раз, то они считаются устаревшими (BackData). Если координаты самолета с прошлого раза не изменились, то данные считаются замороженными (Frozen). В обоих случаях новая точка траектории будет определяться экстраполяцией имеющихся данных (прогнозом).

Если данные прошли предварительный анализ, то есть они не устаревшие и не замороженные, то продолжается конвертация в рабочие структуры и дальнейшая обработка.

Если у данного самолета есть история (достаточно 2-х точек), то стоится прогноз, который затем сравнивается с новым положением модели, полученным от источника (см. рисунок 4).

При значительном отклонении от прогноза, данные положения, рассчитанные для новой точки, считаются недостоверными, им присваивается статус «неверное положение» (BadPos).

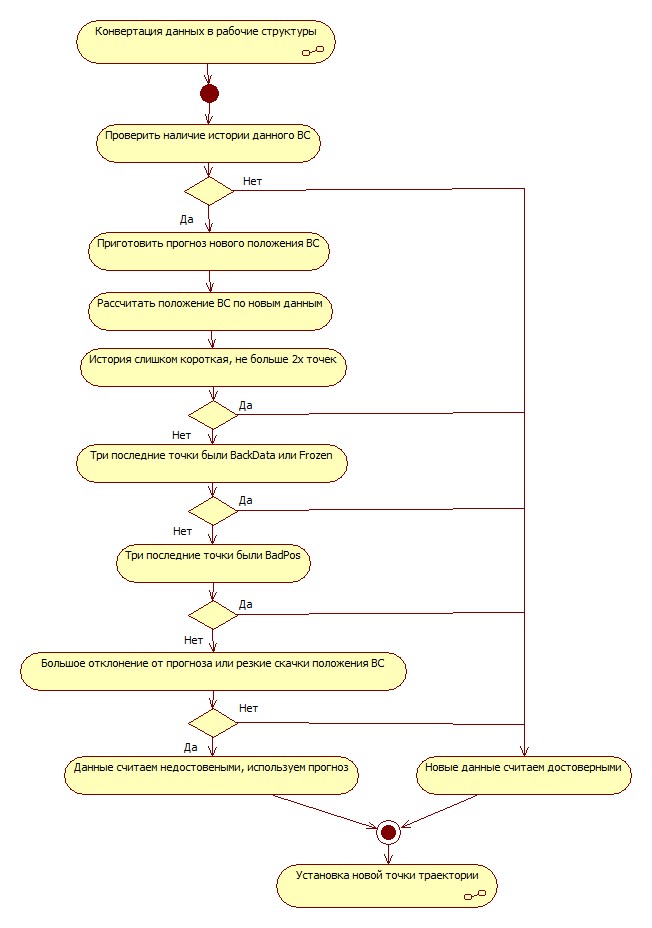


Рис. 4. Прогноз и анализ данных.

Условия, при которых новые данные считаются достоверными, следующие:

* История слишком короткая, 1 или 2 точки.
* Три последние точки в истории были BackData или Frozen
* Три последние точки были BadPos.
* Отличие от прогноза меньше 150 метров.

Условия, при которых новые данные считаются недостоверными, следующие:

* Большая разница между прогнозом и расчетными данными:
  + По высоте: расчетное изменение высоты больше прогнозного в 5 или более раз, при том, что изменение высоты по абсолютной величине больше 200 м.
  + По горизонтали: расчетное изменение положения больше прогнозного в 5 или более раз.

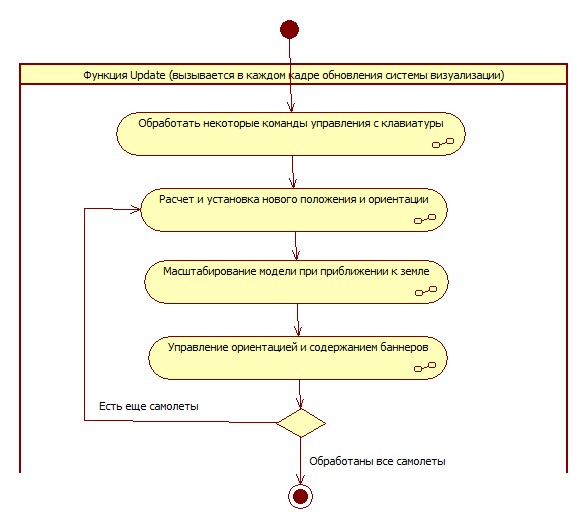
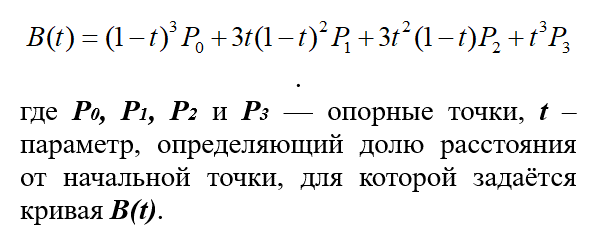


Рис. 5. Положение и размеры моделей и баннеров.

**Расчет нового положения модели**

Полученные от источника данные дают точки на траектории, отстоящие на ~5 секунд полета. При скорости самолета ~250 км/час (посадочная скорость Boeing 737 или A320), расстояние между двумя последовательными точками будет ~350 метров.

Для обеспечения плавного движения самолетов применяется сглаживание. На каждом цикле обновления сцены (не менее 30 раз в секунду) программа рассчитывает новое положение самолета, строя его по последним четырем известным точкам траектории (полученным от источника или прогнозным) используя для интерполяции функцию кривых Безье 3-й степени.



B(t) – вектор положения, Р0, P1, Р2 и P3 – последние четыре точки траектории, t – время перелета из точки Р0 в точку P3, нормированное от 0 до 1.

Движение начинается, когда в истории полета известны 4 точки. Новые данные поступают с периодом ~ 5 секунд. Таким образом, помещая самолет в точку Р0, мы двигаем его с отставанием на ~20 секунд от времени получения последних данных, что считаем приемлемой платой за сглаживание движения.

На каждом цикле обновления сцены программа определяет текущее время, нормирует его на время перелета Р0 – P3, вычисляет новую точку на кривой Безье и перемещает в нее самолет.

Речь идет только о горизонтальных координатах, для положения по вертикали достаточно линейной интерполяции к последней известной точке (также можно назвать кривой Безье первой степени). В сцене Unity эти координаты соответствуют осям X и Z (ось X направлена на восток, ось Z – на север, ось Y - вверх).

Кроме того, нос самолета должен быть постоянно направлен в сторону движения, то есть по касательной к кривой Безье. Для этого на каждом шаге также вычисляем производную z'x.

Уравнение Безье задано параметрически, поэтому

Отсюда находим угол наклона касательной относительно оси X и устанавливаем по нему модель самолета по курсу:

Разобраться и написать про выбор знака.

Для полной красоты мы можем вычислить также радиус кривизны траектории и поставить ему в соответствие угол крена модели самолета.

Радиус кривизны определяется формулой:

где

Радиус кривизны траектории называется в авиации радиусом разворота. Он определяется скоростью самолета *V* и его углом крена *β (*g – ускорение свободного падения)

Таким образом, угол крена:

Однако, через ~5 секунд после начала движения программа получает новую точку траектории. После этого, группа опорных точек для кривой Безье меняется. Новые точки определяются так: новая точка Р0 нов. = текущему положению B (чтобы самолет не переместился в начале движения по новой кривой), новые точки P1 нов. = Р2, Р2 нов. = P3, а P3 нов. = новой последней точке траектории.

Все довольно просто, но есть два «но»:

* В точке перехода со «старой» кривой Безье на «новую», движение не должной иметь излома, иначе самолет рывком станет на новый курсовой угол.
* В точке перехода со «старой» кривой Безье на «новую» радиус кривизны не должен меняться, иначе крен самолета изменится также рывком.

К счастью эти обе проблемы можно решить, найдя подходящие координаты для новой точки P1. Конечно, тогда эта точка не будет соответствовать данным, полученным от источника, которые означают что самолет реально был в точке с координатами P1. Но не будем переживать: двигаясь по кривой Безье наша модель самолета все равно прошла бы мимо точки P1, так ее и не посетив – в этом суть интерполяции.

Итак, мы имеем два отрезка кривых Безье. Первый начинается в точке Р0 и строится по опорным точкам Р0, Р1, Р2 и Р3. Второй начинается в некой точке Р0 нов., которая является одной из точек, принадлежащих первому отрезку и определяется временем t нов., (доля пути из точки Р0 в Р3, которую наша модель «прошла» к моменту получения новых данных) и строится по новым опорным точкам точка Р0 нов., P1 нов., Р2 нов. и P3 нов.. В точке Р0 нов., которая является общей для обеих кривых, должны совпадать их углы наклона касательных и радиусы кривизны – это обеспечит нужную для визуализации плавность движения. Угол наклона касательной в точке кривой определяется первой производной функции в этой точке. Вместо радиусов кривизны можно потребовать равенства вторых производных – это не изменит сути (см. ниже «Отступление»), но немного упростит выкладки. То есть, в точке Р0 нов. должны быть равны первая и вторая производные для обеих кривых.

Вторая производная для функции, заданной параметрически, определяется формулой:

Координаты Р0, Р1, Р2 и Р3 программе известны. На каждом шаге она вычисляет новую точку на первом отрезке кривой Безье. В момент времени t нов. программа получает координаты следующей точки траектории и начинает готовить данные для расчета второго отрезка.

Пусть первая производная для первого отрезка в момент t нов. будет *D*1, а вторая производная – *D*2. Их значения можно получить по формулам, приведенным выше.

В начале движения по второму отрезку нормированное время сбрасывается, то есть производные в точке Р0 нов, для второго отрезка вычисляются для времени t=0.

Получим

и

Для упрощения записи мы опустили индексы *нов.* при координатах, так как все координаты имеют эти индексы, а других координат в формулах нет.

Эти два уравнения содержат две неизвестных: X1 и Z1. Это координаты точки P1 (до опускания индексов мы написали бы P1 нов.), применение которых вместо полученных от источника данных позволит обеспечить гладкость движения модели. Решим эти уравнения.

Перепишем последнее равенство в немного измененном виде:

Сделаем полезную подстановку x1=(X1-X0), x2=(X2-X1), z1=(Z1-Z0), z2=(Z2-Z1) (как бы перенесем начало координат в точку X0, Z0) Это также полезно с точки зрения программирования, так как позволит избежать потери точности при работе с числами разных порядков (расстояния между точками Р0, Р1, Р2 и Р3 ~ сотни метров, а если самолет движется близко к направлению север-юг или восток-запад, то изменение одной координаты может составить десятки или даже единицы метров, при том, что абсолютные значения координат на границе отображаемой зоны ~ 50 километров).

Уравнения примут вид:

Подставим *z*1=*D*1*x*1 во второе уравнение

Тогда

Подставляя полученные значения в качестве координат точки перехода от старой траектории к новой Р1(*X*1,*Z*1) получим равенство первых и вторых производных в этой точке. С учетом произведенной подстановки «перенос начала координат в точку Р0» получим (уже в абсолютных координатах):

Таким образом модель самолета будет плавно двигаться по меняющимся с каждой новой порцией данных кривым Безье, держа нос по курсу и кренясь в сторону поворота.

**Отступление.**

Вернемся немного назад и решим ту же задачу еще раз, не заменяя условие равенства радиусов кривизны для двух отрезков кривых в точке перехода на условие равенства вторых производных.

Производная в момент перехода:

Радиус кривизны:

После преобразований и подстановок получим:

Новое выражение для *X*1 похоже на полученное ранее. Выделим отличие: в старом мы имеем под корнем , а в новом:

Посмотрим, эквивалентны ли эти выражения.

Умножим на :

Но , а в той же переходной точке, для которой мы вычисляем и R. То есть, мы можем записать:

Это значит, что выражение для *X*1 полученное первым (через вторую производную ) и вторым (через радиус ) эквивалентны. Иначе говоря, при поиске точки P1, требование равенства радиусов кривизны можно заменить на требование равенства вторых производных.

**Отступлению конец.**

**Практическая реализация выявляет проблему**

Алгоритм расчета новой опорной точки кривой Безье P1 был реализован в программе и в большинстве случаев работал нормально: самолеты плавно шли по рассчитанным траекториям, держа нос по курсу и правдоподобно кренясь в разворотах. Однако оказалось, что в ряде случаев алгоритм дает следующий результат: расчетная новая точка P1 отстоит от точки P0 намного дальше исходной точки P1, может быть даже дальше, чем точка P2 или P3. В результате самолет в точке P0 резко увеличивает скорость, стремясь за то же самое время пройти значительно больший путь. Также, если успевает залететь достаточно далеко, он выполняет резкие развороты почти на 180°.