МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «КубГУ»)

Факультет компьютерных технологий и прикладной математики
Кафедра информационных технологий

КУРСОВАЯ РАБОТА

АВТОМАТИЧЕСКАЯ ГЕНЕРАЦИЯ ГОРОДСКОЙ СРЕДЫ НА ОСНОВЕ ДАННЫХ OPENSTREETMAP В UNITY

Работу выполнил _____ Е.Г. Хижний
(подпись)

Направление подготовки 01.03.02 Прикладная математика и информатика

Направленность Системное программирование и компьютерные технологии (Математическое и программное обеспечение вычислительных машин) курс 3

Научный руководитель
канд. тех. наук, доц. _____ А.Н. Полетайкин
(подпись)

Нормоконтролер
ст. преп. _____ А.В. Харченко
(подпись)

Краснодар
2021

РЕФЕРАТ

Курсовая работа содержит страниц, рисунков, источников.

АВТОМАТИЧЕСКАЯ ГЕНЕРАЦИЯ ДОРОЖНОЙ СИТИ И ЗДАНИЙ НА ОСНОВЕ ДАННЫХ OSM В UNITY, OSM, OPENSTREETMAP, UNITY.

В данной курсовой работе рассмотрена автоматическая генерация дорожной сити и зданий на основе данных OSM в Unity.

Целью данной курсовой работы является работы является *решение задачи автоматическая генерация дорожной сити и зданий*, т.е. преобразование набора данных из *OpenStreetMap(OSM)* в реалистичную, чистую и согласованную трехмерную модель дорожной сети и зданий.

В теоретической части изучены принципы формирование данных в OSM, получение и обработка данных из OSM при помощи Overpass-turbo API, автоматическое создание объектов в Unity.

В оригинальной части спроектированы приложения, позволяющие получать данные OSM в виде XML файла и обрабатывать их в Unity. Получение данных и их запись происходит при помощи приложение Windows Form, а обработка полученных данных и формирование трехмерной модели происходит в Unity приложении.

Приложение получения данных из OSM реализовано на языке C# с использование библиотеки GMAP, приложение по обработке данных реализовано на C# с использованием внутренних библиотек Unity.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

С момента появления понятия компьютерной графики в 1950-х годах, она получала постепенное развитие: от простого вывод чисел на экран, до простых приложения, например для рисования при помощи мыши. С созданием в 1970-х годах микропроцессора, случился скачок в развитии вычислительной техники, также в это время интенсивно развивались индустрии компьютерных игр и компьютерной графики. В дальнейшем эти две индустрии стали главной движущей силой развития трехмерной графики.

С помощью трехмерного моделирования можно создавать абсолютно любые объекты, однако моделирование также применимо в реальных задачах, таких как архитектура и городское планирование, моделирования различных ситуаций и так далее. Для данных задач необходимы приближенные к реальным условия, например подробная трехмерная модель города.

В настоящее время трехмерное моделирование города широко используется в таких областях, как городское планирование, моделирование дорожного движения. Инструменты для автоматического моделирования территории представляют большой интерес, так как ручное моделирование городской среды занимает много времени и тратится большое количество человеческих ресурсов. Данные для построения таких моделей можно брать из разных источников, но самым эффективным будет сбор данных из *геоинформационных систем (ГИС).* Само построения трехмерной модели происходит в различных *ПО для создания трёхмерной компьютерной графики*, таких как, например, *игровые движки*.

Решение задачи автоматического построения трехмерной модели на основе данных из реального мира является актуальной проблемой.

Для решения данной задачи необходимо:

а) получить необходимые данные из ГИС;

б) обработать и преобразовать данные;

в) построить трехмерную модель на основе преобразованных данных.

1 Геоинформационная система

Геоинформационные системы (также ГИС — географическая информационная система) — системы, предназначенные для сбора, хранения, анализа и графической визуализации пространственных данных и связанной с ними информации о представленных в ГИС объектах. Другими словами, это инструменты, позволяющие пользователям искать, анализировать и редактировать цифровые карты, а также дополнительную информацию об объектах, например высоту здания, адрес, количество жильцов[1].

Геоинформационная система может включать в свой состав:

− пространственные базы данных (в том числе под управлением универсальных СУБД);

− редакторы растровой и векторной графики;

− различные средства пространственного анализа данных.

Применяются в картографии, геологии, метеорологии, землеустройстве, экологии, муниципальном управлении, транспорте, экономике, обороне и многих других областях. Научные, технические, технологические и прикладные аспекты проектирования, создания и использования геоинформационных систем изучаются геоинформатикой.

Самыми известными ГИС являются:

а) Google Earth и Google Maps;

б) Яндекс карты;

в) 2ГИС;

г) OpenStreetMap.

Первые три сервиса имеют закрытый исходный код, также запросы на получение данных из этих ГИС либо не доступны, либо предоставляются на платной основе, в отличии от OpenStreetMap, где реализован принцип полностью открытых географических данных, которые могут быть использованы кем угодно и как угодно, поэтому в дальнейшем будем рассматривать именно эту ГИС

1.1 OpenStreetMap (OSM)

OpenStreetMap (дословно «открытая карта улиц»), сокращённо OSM — некоммерческий веб-картографический проект по созданию силами сообщества участников — пользователей Интернета подробной свободной и бесплатной географической карты мира[2].

Для создания карт используются данные с персональных GPS-трекеров, аэрофотографии, видеозаписи, спутниковые снимки и панорамы улиц, предоставленные некоторыми компаниями, а также знания человека, рисующего карту. Использование для создания карт информации из несвободных сервисов, подобных Google Maps, без разрешения правообладателя запрещено.

1.2 Типы данных OSM

Существует три базовых типа данных в OSM :

а) *точка (node)* — это минимальный набор данных, который содержит в себе информацию о паре координат: широта, долгота *(lat, lon)* и является базовым в иерархической модели. Это единственный тип данных, который хранит саму географическую информацию — координаты, в виде широты и долготы.

б) *линия (way)* — это совокупность указателей на объекты типа точка (node). Линия, как минимум, состоит из одной точки, т.е. должна содержать как минимум одну ссылку на уже существующий объект типа точка. Порядок перечисления точек в линии важен, он характеризует последовательность точек в линии и направление самой линии;

в) *отношение (relation)* — это совокупность любых объектов, как точек и линий, так и других отношений. В нем указывается не только id объекта, но и его тип. Самое распространённое отношение — отношение типа мультиполигон, которое описывает один замкнутый внешний полигон из n

точек с вырезанным из него замкнутым полигоном тоже из m точек меньшего размера.

Примеры типов данных в формате XML указаны в приложении А.

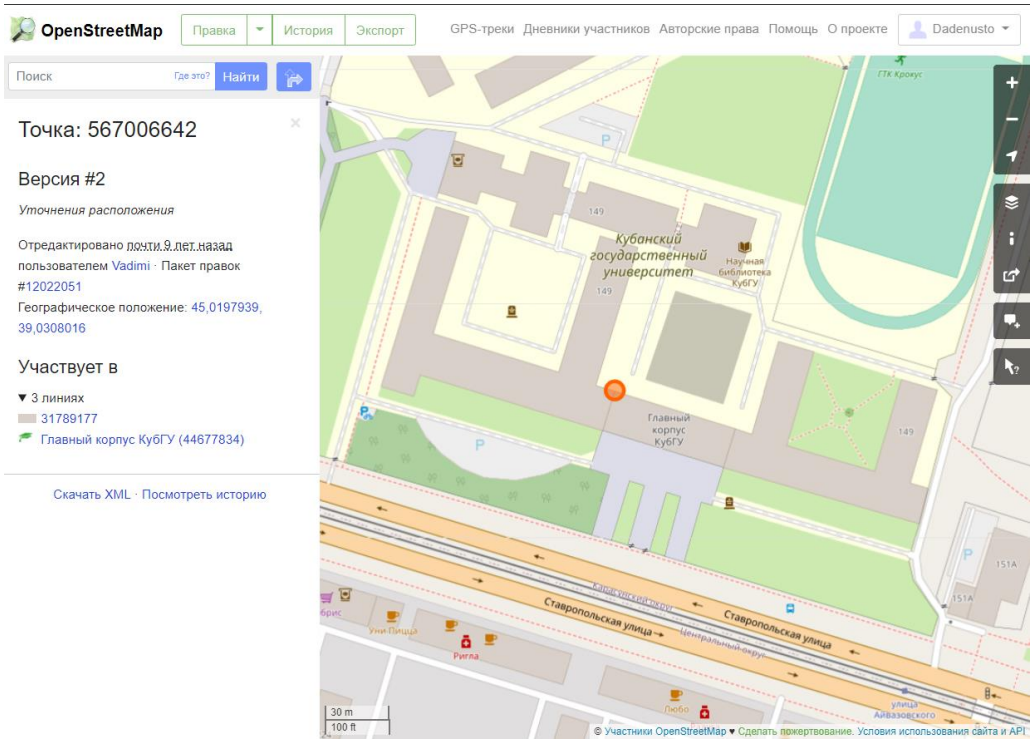

Рисунок 1.1—Пример типа данных node(точка)

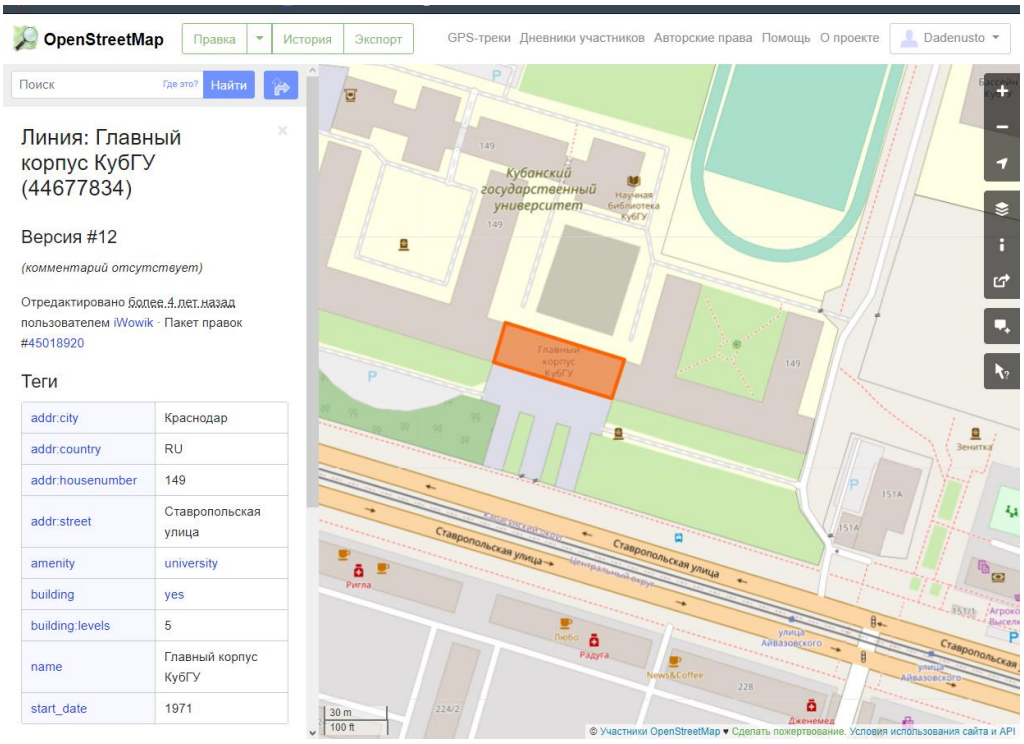

Рисунок 1.2—Пример типа данных way(линия)

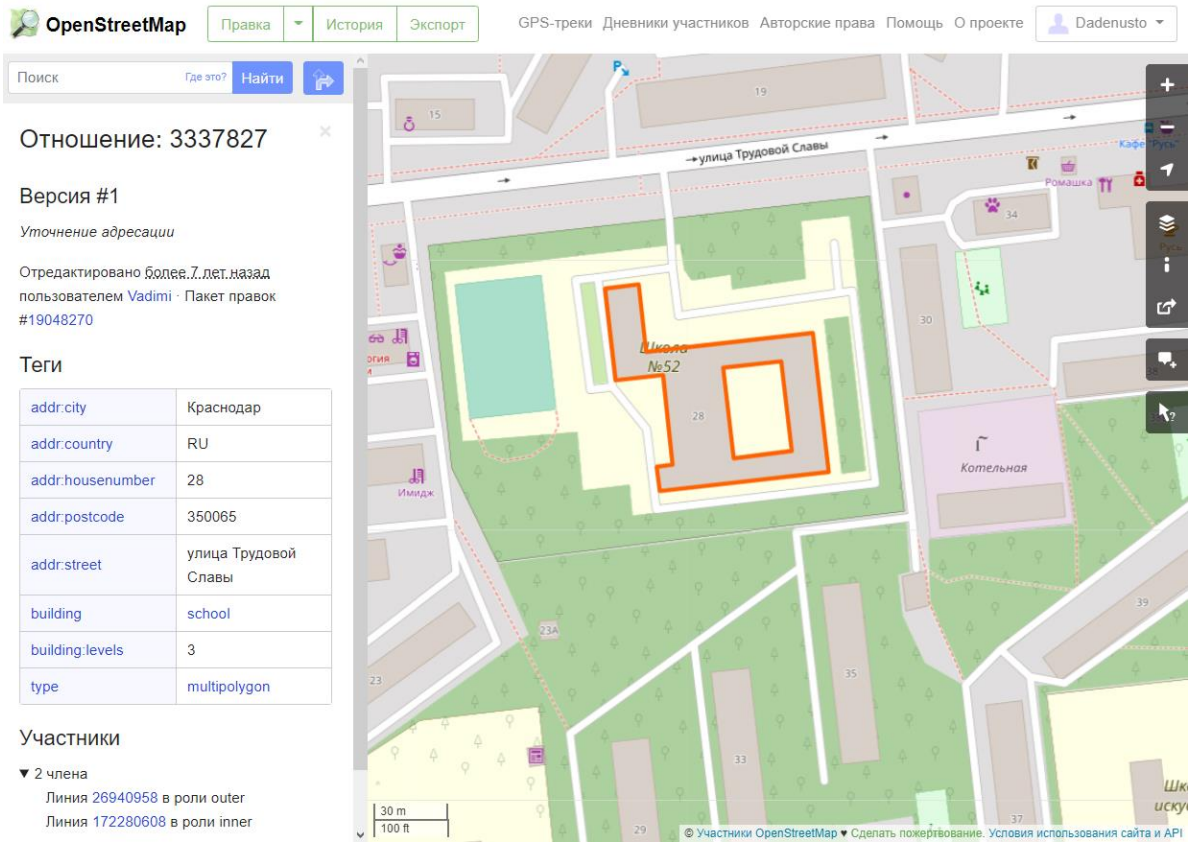

Рисунок 1.3—Пример типа данных relation(отношение)

1.3 Информация об объекте OSM

Тип объекта описывает географические (пространственные) свойства объекта, но указывает на свойства самого объекта, его характеристиках, назначении и прочем. Для это существует информационная часть структуры данных OSM, основанная на принципах тегов, т.е. назначении им определённых меток и указанием свойств этих меток. Теги задаются в виде пары ключ=значение. Схема тегирования в OSM является одним из = главным её архитектурным преимуществом, поскольку позволяет описать фактически любые свойства объекта.

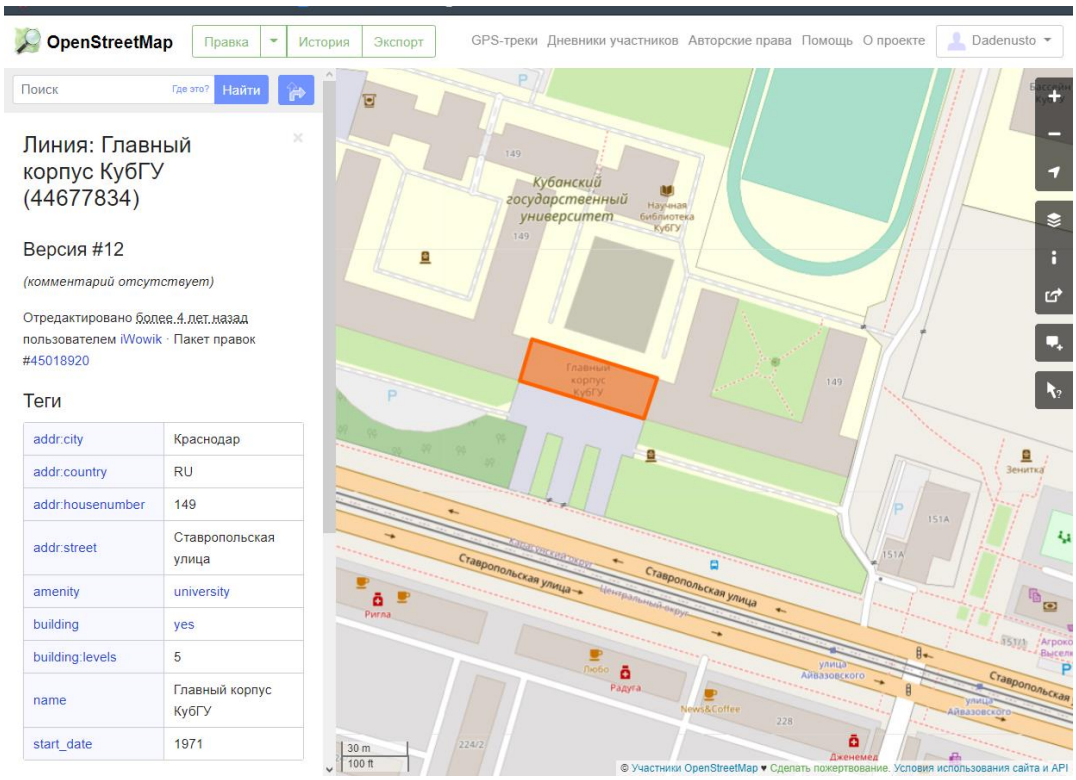

Рисунок 1.4 — Пример информационной части (теги) для данных типа way(линия)

1.3.1 Получение данных из OSM

Для получения данных из OSM необходимо воспользоваться специальным API, разработанным для OSM, под названием *Overpass API*.

Overpass API это доступный только для чтения API, который позволяет извлекать выборочные данные из базы OSM по пользовательскому запросу. Он выступает в качестве базы данных через Интернет: клиент посылает запрос к API и получает обратно набор данных, который соответствует запросу[3].

Для запросов используется публичный сервер https://overpass.openstreetmap.ru с основным адресом для запросов https://overpass.openstreetmap.ru/api/interpreter.

Запросы к Overpass API могут быть в XML или Overpass QL форме.

Полученные данные можно получить в виде JSON или XML файла, либо использовать интерфейс Overpass turbo для создания и отображения запросов Overpass API.

1.3.2 Запрос для получения данных об объекте указанного типа

Overpass XML

```
<osm-script>
  <bbox-query e="7.157" n="50.748" s="50.746"
w="7.154"/>
  <print/>
</osm-script>
```

Overpass QL

```
node(50.746,7.154,50.748,7.15
7);
out body;
```

Рисунок 1.5 — Запрос для получения данных типа node(точка) в указанной рамке

В Overpass xml оператор bbox-query позволяет загрузить все точки, линии или отношения внутри рамки. Задаем:

- s южную границу в десятичных градусах (меньшая широта)
- w западную границу в десятичных градусах (меньшая долгота)
- n северную границу в десятичных градусах (большая широта)
- e восточную границу в десятичных градусах (большая долгота)

В Overpass QL неочевидный порядок координат (s, w, n, e). [4]

Аналогично формируется запрос для way(линия) и relation(отношение).

1.3.3 Запрос для получения данных об объекте по тэгам

Overpass XML

```
<osm-script>
  <query type="way">
    <has-kv k="highway"  />
    <bbox-query e="7.18" n="50.75" s="50.74"
w="7.17"/>
  </query>
  <print limit="" mode="body" order="id"/>
</osm-script>
```

Overpass QL

```
way
  ["highway"]

(50.74,7.17,50.75,7.18)
;
out body;
```

Рисунок 1.6 — Запрос для получения данных типа way(линия) в указанной рамке с тэгом highway(дорога)

В Overpass xml оператор has-kv получает ключ(k) (и значение(v)) и находит все объекты, у которых есть ключ highway.

В Overpass QL ключ задается после типа данных в квадратных скобках[4].

1.3.4 Формирование запроса к Overpass API

Как было указанно в начале главы 1.4, ля запросов используется публичный сервер https://overpass.openstreetmap.ru с основным адресом для запросов https://overpass.openstreetmap.ru/api/interpreter.

Запрос формируется следующим образом:

https://overpass.openstreetmap.ru/api/interpreter?data=[out:type];data_type["key "="value"](s, w, n, e);out geom; где:

а) https://overpass.openstreetmap.ru/api/interpreter — ссылка на сервер для запросов;

б) ?data= — ключевое слово, после которого пишется сам запрос;

в) [out:type] — полученные данные будут в формате type: JSON или XML;

г) data_type — тип данных, необходимый получить: node, way или relation;

д) ["key "="value"] — конкретизировать поиск по тэгам. Значение ключа можно не указывать. Может быть неограниченное количество.

е) (s, w, n, e); — координаты рамки, начиная с нижней, заканчивая правой;

ж) out geom; — выводит отношения (включая way и node)

Пример реального запроса:

https://overpass.openstreetmap.ru/api/interpreter?data=[out:xml]way["building"](50.745,7.17,50.75,7.18);out geom; — все линии типа building в указанной рамке в файле типа XML.

Вывод по данному запросу указан в приложении Б.

2 Трехмерная графика

Трёхмерная графика — раздел компьютерной графики, посвящённый методам создания изображений или видео путём моделирования объектов в трёх измерениях[5].

3D-моделирование — процесс создания трёхмерной модели объекта. Задача 3D-моделирования — разработать зрительный объёмный образ желаемого объекта. При этом модель может как соответствовать объектам из реального мира (автомобили, здания, ураган, астероид), так и быть полностью абстрактной (проекция четырёхмерного фрактала)[5].

Программы для 3D моделирования:

- 3Ds MAX
- 123D
- AutoCAD
- Cinema 4D
- ARCHICAD
- Blender

Для решение поставленной задачи, а именно задачи автоматического построения трехмерной модели на основе данных из реального мира, надо не только построить трехмерную модель, но и также получить возможность взаимодействовать с построенным моделями, чтобы, например, проектировать дорожную обстановку или для архитектурных задач: посмотреть, как будет выглядеть новое здание на выбранной территории. Для это нужно использовать более высокоуровневое ПО, такое, как *игровые движки*.

2.1 Игровой движок

Игровой движок (англ. game engine) — базовое программное обеспечение для разработки компьютерных игр, графики для кино, архитектурных проектов и визуализации. Игровые движки предоставляют

средства разработки, которые могут быть использованы программистами, чтобы упростить их работу, предоставляют инструменты и функциональные возможности для разработки.

При помощи игровых движков, помимо создания игр, можно выполнять многие задачи, некоторые из них:

- демонстрация недвижимости на этапе строительства;
- реконструкция культурных монументов и исторических событий;
- моделирование различных ситуаций, например моделирование дорожной ситуации в городе, моделирование техногенных катастроф и т.д.

Список игровых движков:

- CryEngine 3;
- Source engine SDK;
- Unreal Engine 4
- Unity;

Рассмотрим более подробно Unity, ввиду простоты его освоения, при этом наличие достаточно широких возможностей работы в нем.

2.2 Unity

Unity — межплатформенная среда разработки компьютерных игр, разработанная американской компанией Unity Technologies. Основными преимуществами Unity являются наличие визуальной среды разработки, межплатформенной поддержки и модульной системы компонентов. На нем написаны тысячи игр, приложений, визуализации математических моделей.

В среду разработки Unity интегрирован игровой движок, т.е. возможно протестировать свой проект, не выходя из редактора. Написание сценариев (скриптов) осуществляется на языках программирования C# или JavaScript.

Таким образом, Unity является актуальной платформой, с помощью которой вы можете создавать свои собственные приложения и экспортировать их на различные устройства[6].

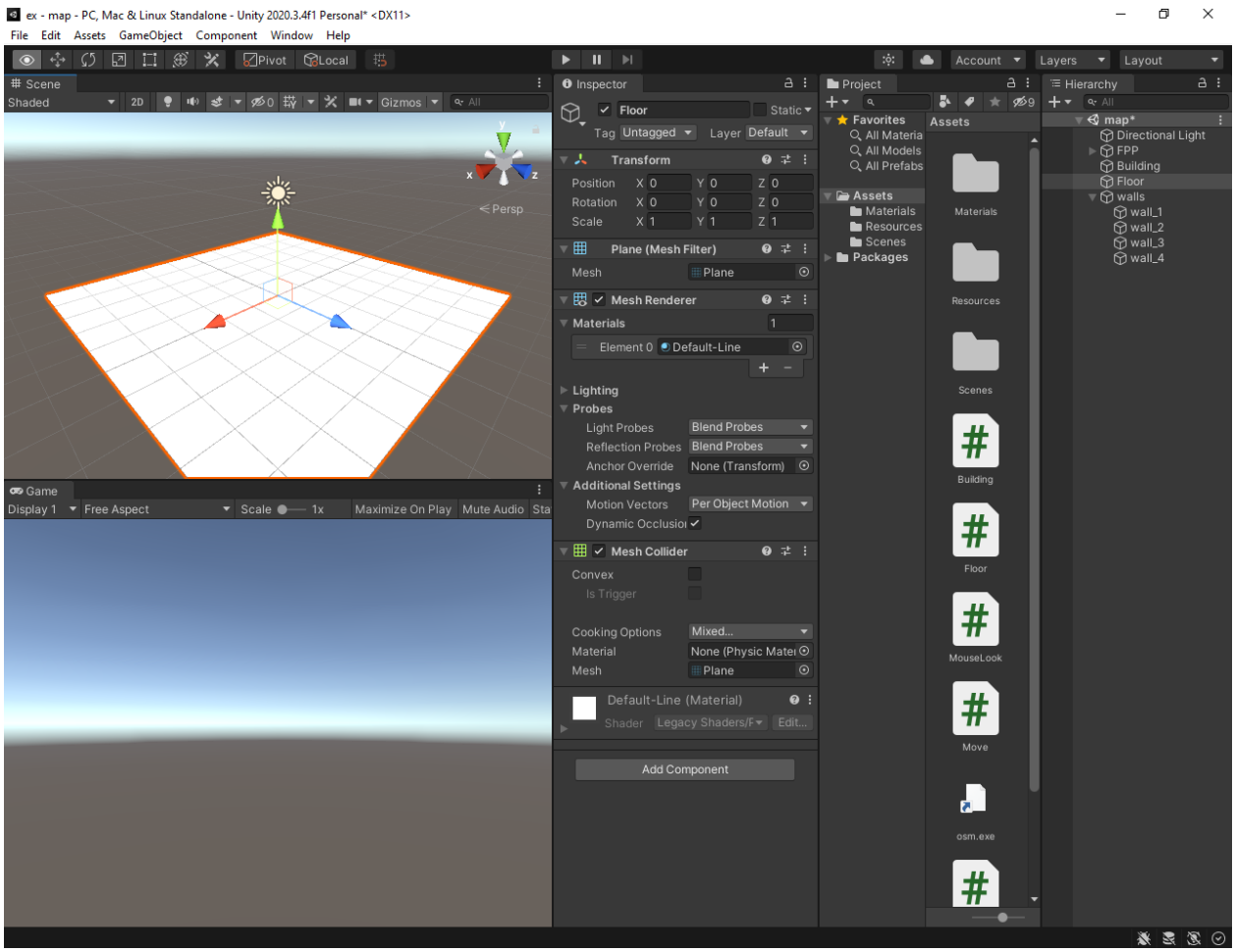

Рисунок 2.1 — окно редактора Unity

2.2.1 Основы Unity

Готовая игра в Unity — это набор сцен, соединенных между собой. Сцены хранятся по пути Assets/Scenes. Сцена содержит объекты игры. Они могут использоваться для создания главного меню, отдельных уровней и для других целей. Можно считать каждый файл сцены отдельным игровым уровнем. В каждой сцене можно разместить объекты окружения, заграждения, декорации и т.д. Сцена может быть в двухмерном (оси x и y) и трехмерном (оси x, y, и z).

Объектами моделирования в Unity являются GameObject (игровой объект). GameObject’ы являются контейнерами, т.е. могут быть чем угодно исходя из желания программиста. В зависимости от того, какой вы хотите создать объект, вы будете добавлять различные комбинации компонентов к GameObject’у. Для данного объекта можно задать координаты расположения, параметры растяжения и поворота вокруг одной из координатных осей. Простыми GameObject в Unity являются куб, цилиндр, сфера.

Также моделирование в Unity можно производить при помощи *mesh (меш). Меш (англ. Mesh)* — это сетка полигонов, из которых состоит любой 3D объект в компьютерной графике. Полигон, в свою очередь, состоит из вершин (вертексов), рёбер (линий, соединяющих две вершины между собой) и граней (плоскости, образуемой между рёбрами). Unity производит так называемую триангуляцию каждого меша и разбивает полигоны 3D объекта на треугольники. Чтобы сделать более детализированный 3D объект необходимо использовать большее количество полигонов. В общем случае mesh в unity задается при помощи двух массивов:

– *vertices* — набор позиций вершин геометрии в трехмерном пространстве с собственным началом координат. Данный массив имеет тип Vector3, в котором хранится три числовых значения, соотвеосвующих значениям координат на осяхх , y, z;

– *triangles* — набор индексов вершин, обрабатываемых по 3 за раз; каждая такая тройка представляет полигон (в данном случае треугольник) модели.

Отображение mesh на сцене (так называемый рендер) происходит при помощи Mesh Renderer.

За внешний вид объектов отвечает параметр Material, к которому привязываются файлы типа mat. Материалы — это определения того, как должна выглядить визуализирована поверхность объекта, включая ссылки на используемые текстуры, цветовые оттенки и многое другое.

Все файлы типа mat хранятся по пути Assets/Materials.

2.2.2 Использование скриптов в Unity

Поведение игровых объектов контролируется с помощью компонентов (Components), которые присоединяются к ним. Unity позволяет создавать свои компоненты, используя скрипты. Они позволяют активировать игровые события, изменять параметры компонентов, и отвечать на ввод пользователя каким вам угодно способом.

Скрипты хранятся в папке Assets.

Unity поддерживает два языка программирования:

– C#;

– UnityScript, язык, разработанный специально для использования в Unity по образцу JavaScript.

Все функции создаются в классе, который наследуется из MonoBehaviour. MonoBehaviour — это базовый класс, от которого наследуются все скрипты.

В скриптинге Unity есть некоторое количество функций события, которые исполняются в заранее заданном порядке по мере исполнения скрипта. Порядок исполнения основных функций описан ниже:

– *Awake* — Первая загрузка сцены. Э функция всегда вызывается до любых функций);

– *Start* — перед первым обновлением кадра. Функция Start вызывается до обновления первого кадра(first frame) только если скрипт включен;

– *Update* — вызывается раз за кадр. Это главная функция для обновлений кадров;

– *OnApplicationQuit*. Эта функция вызывается для всех игровых объектов перед тем, как приложение закрывается.

Пример скриптинга в Unity представлен в приложении В.

3 Генерация городской среды на основе данных OSM В Unity

Как было обозначено ранее, для решения данной задачи необходимо:

а) получить необходимые данные из ГИС;

б) обработать и преобразовать данные;

в) построить трехмерную модель на основе преобразованных данных.

Так как в предыдущих главах были рассмотрены ГИС OSM и Unity как ПО для построения трехмерных моделей, уточним пункты, необходимые для решения исходной задачи:

а) получение данных из OSM;

б) обработка и преобразование данных;

в) построение трехмерную модель на основе преобразованных данных в Unity.

В приложении будут генерироваться 2 типа объектов в городе: дороги (сюда относятся как дороги для автомобилей, так и пешеходные дороги) и здания.

3.1 Получение данных из OSM

Первым этапом для решения поставленной задачи является получение данных из OSM. Как было указанно в главе 1, получить данные можно при помощи Overpass API. Для этого необходимо знать координаты области, в которой необходимо найти объекты и сами объекты, которые надо найти.

Для упрощения получения данных пользователем, создадим приложение. У него должен быть следующий функционал:

а) демонстрация карты, чтобы пользователь знал, какие данные он получает;

б) составление запроса на получение данных и последующая запись этих данных в файл формата XML.

Создадим приложения Windows Forms на C#, в котором будет реализован необходимый функционал.

Подробный код данной программы предоставлен в приложении Г.

Для реализации демонстрации карты воспользуемся GMap.NET.Windows, библиотекой, разработанной специально для работы с картами в Windows Form. При помощи данной библиотеки можно осуществить вывод карты на форму, а также получение необходимых данных для запроса, а именно координаты для рамки.

GMAP имеет множество настроек, например:

а) настройка увеличения карты, максимальное и минимальное значения;

б) выбор начальной позиции карты;

в) настройка маркеров, которые отображаются на карте;

г) выбор того, какая карта будет показана (OSM, Google Maps, Яндекс карты и др.).

3.1.3 Вывод карты на форму

За показ карты отвечает элемент GMapControl, это поле, на котором отображается карта, функция для настройки называется gMapControl1_Load, в ней выставим необходимые настройки, основные из них:

- gmap.MaxZoom = 20 — максимальное приближение;
- gmap.MinZoom = 10 — минимальное приближение;
- gmap.Zoom = 17 — приближение при запуске приложения;
- gmap.MapProvider = GMapProviders.OpenCycleMap — выбор того, какая карта отображается (OpenCycleMap — одна из оболочек OSM);
- gmap.Position = new PointLatLng(45.0196306, 39.0311312) — центр карты при запуске приложения.

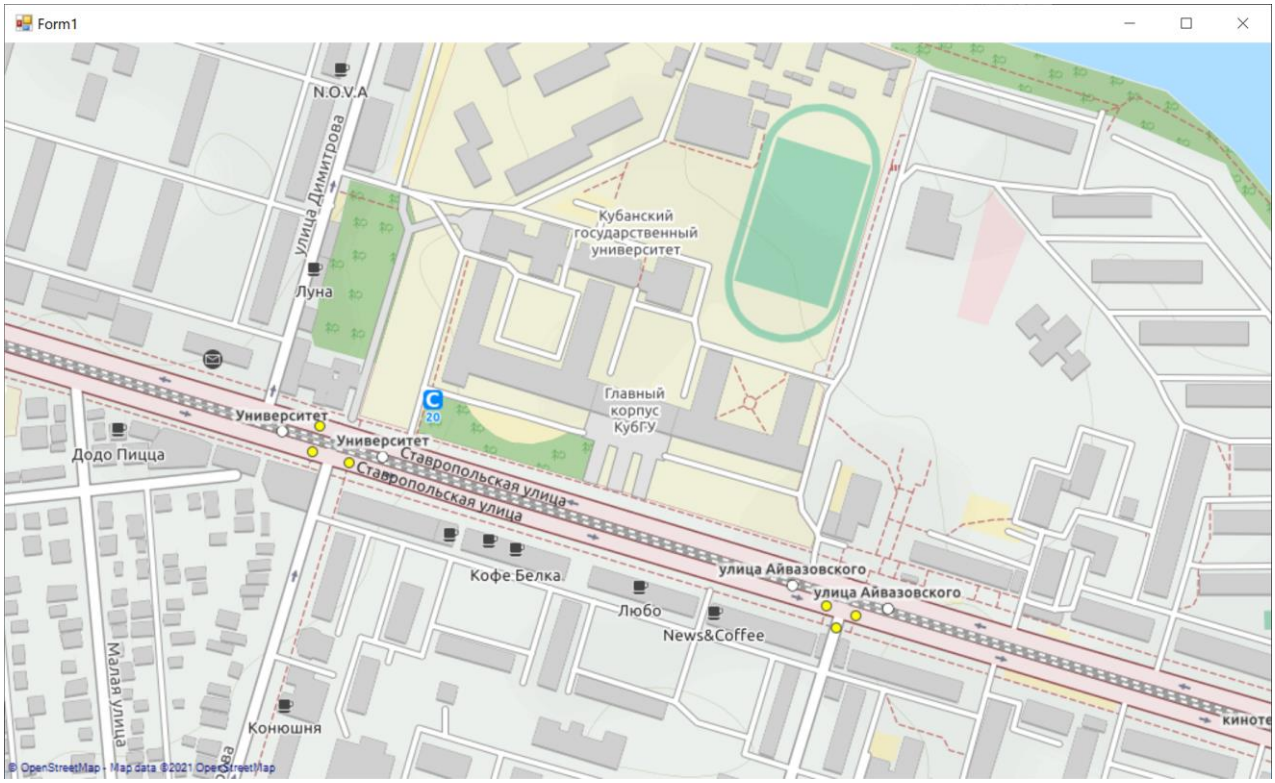

Рисунок 3.1 — Вывод карты на Windows Form

3.1.2 Запрос на получение данных

Следующей функцией будет функция для получения информации о зданиях и дорогах на выбранном участке под названием createCoordinates_Click, вызываться будет при нажатии на кнопку сгенерировать координаты.

Как было описано в главе 1, в OSM существует 3 основных типа: node(точка), way(линия), relation(отношения), для получения координат дорог и домов остановимся чуть подробнее на их составляющих.

Дороги (highway) делятся на дороги для автомобилей и пешеходные дороги (highway = footway). Дороги в OSM являются типом way(линия). Основным параметром для дорог является количество полос lanes, у пешеходных дорог этот параметр отсутствует. Дороги состоят из точек, которые её формируют .

Здания (building) в OSM зачастую являются типом way(линия). Однако некоторые здания, как например здание, показанное на рисунке 1.3, состоит из двух наборов линий: outer (внешние линии) и inner (внутренние линии). Каждая линия является member’ом у relation. Поэтому для зданий будет два запроса, к типу way и relation. Само здание – это точки, составляющие замкнутую фигуру, поэтому первая и последняя точки в зданиях повторяются. Важным параметром является building:levels (этажность здания).

Таким образом, необходимо отправить три запроса для получения данных: way[highway], way[building], relation[building].

Примеры файлов, получаемые по данным запросам, приведены в приложении Д.

Для удобства последующего считывания, для каждого запроса создадим отдельную папку highway, building (для way[building]) и polygon (для relation[building]), в каждой папке информация об каждом объекте будет хранится в отдельном файле формате XML с именем в виде id объекте. При вызове функции проверяется, существуют ли данные папки. Если папки существуют, то очищаем их содержимое, если не существует, то создаем необходимые папки.

Как было указанно в 1.3.4, также необходимы координаты рамки, в которой необходимо найти данные объекты. Для получения данных координат воспользуемся специальными функциями GMAP:

- gmap.ViewArea.Bottom — текущая нижняя широта;
- gmap.ViewArea.Top — текущая верхняя широта;
- gmap.ViewArea.Left — текущая левая долгота;
- gmap.ViewArea.Right — текущая правая долгота.

Запрос выполняется в функции Req со строковым параметром type — запрос, данные по которым надо получить данные, всего будет три запроса, которые были указаны выше, перед вызовом данных функций запускается таймер для подсчета времени выполнения запросов, при успешном выполнении в конце выводится сообщения об успешном выполнении с общем

временем работы, пример на рисунке 3.3. При возникновении ошибки высвечивается соответствующее сообщение.

При формировании запроса к OSM необходимо подключение к сети интернет.

Если здание или дорога находится частично в зоне видимости карты, то мы получим *полную* информацию об данном объекте, то есть о всех точках этого здания, даже если их не видно в области.

Алгоритм действий в Req:

а) формируется строка для запроса sURL, имеющая следующий вид: "https://overpass.openstreetmap.ru/api/interpreter?data=[out:xml][timeout:25];" + type + "(" + gmap.ViewArea.Bottom.ToString("#.#####").Replace(",", ".") + ", " + gmap.ViewArea.Left.ToString("#.#####").Replace(",", ".") + ", " + gmap.ViewArea.Top.ToString("#.#####").Replace(",", ".") + ", " + gmap.ViewArea.Right.ToString("#.#####").Replace(",", ".") + ");out geom;";

б) выбирается папка для записи файлов и переходим в нее;

в) формируется файл *area* формата XML для записи координат рамки, данный файл пригодится в дальнейшем, имеет следующий вид:
"<area bottom = \"" + gmap.ViewArea.Bottom.ToString("#.#####").Replace(",", ".") + "\" left =\" " + gmap.ViewArea.Left.ToString("#.#####").Replace(",", ".") + "\" top=\" " + gmap.ViewArea.Top.ToString("#.#####").Replace(",", ".") + "\" right = \" " + gmap.ViewArea.Right.ToString("#.#####").Replace(",", ".") + "\"/>";

г) в переменную reader формата XmlTextReader записывается полученные данные от запроса sURL. Переменная reader хранит все себя данные, полученные при запросе;

д) каждый отдельный объект записывается в отдельный файл, объекты берутся из переменной reader, каждый отдельный объект начинается и заканчивается тегами way или relation;

е) возвращаемся в исходную папку, где находятся паки с файлами.

Таким образом после выполнения трех функций Req должно сформироваться 3 папки с XML файлами.

3.1.3 Дополнительные функции

Введем объект типа TrackBar для переключения приближения на карте.

Установим для него параметры максимального и минимального значения, равные максимальному и минимальному значению соответственно. Отображаться будет справа

Введем объект типа Label, на котором будет отображаться текущее значение приближения. Отображаться будет в правом верхнем углу.

Если при закрытии приложении оно не обнаруживает папки, то появляется соответствующее сообщение с уведомлением об этом, как показано на рисунке 3.4.

Таким образом приложение для получения данных об объекте будет иметь вид, показанный на рисунке 3.2.

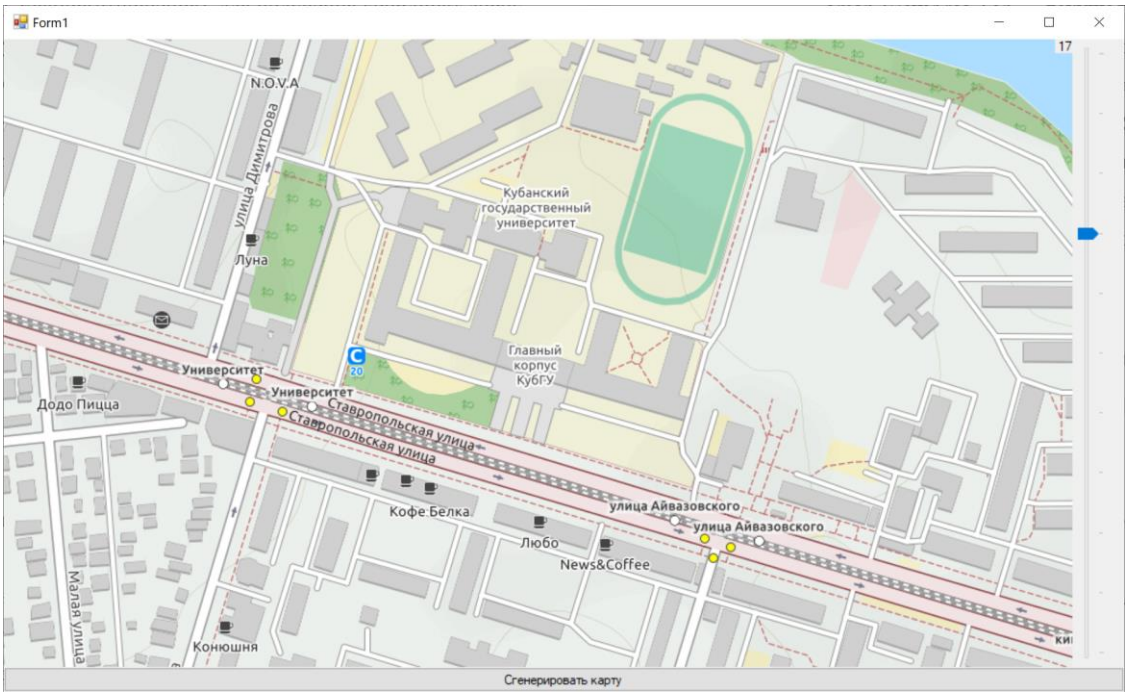

Рисунок 3.2 — Итоговый вид приложения

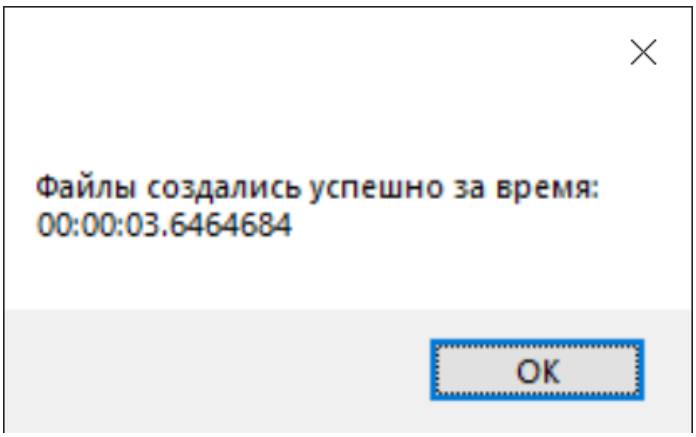

Рисунок 3.3 — Сообщение при успешном создании файлов

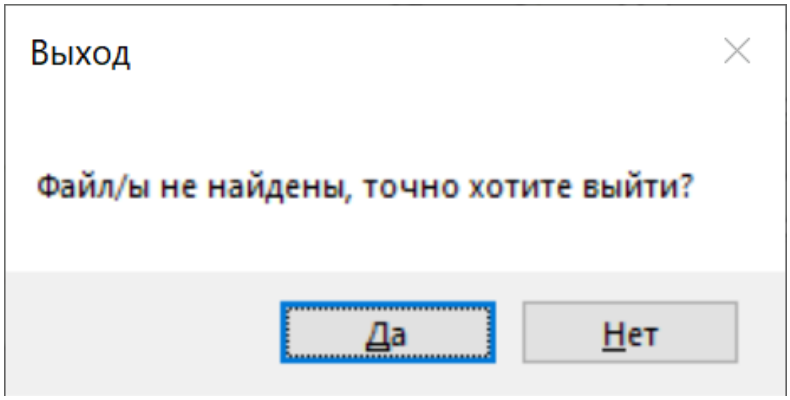

Рисунок 3.4 — Предупреждение о том, что файлы не найдены/не созданы

3.2 Преобразование данных из OSM

Как было описано в главе 3.1, данные об объектах, которые необходимо построить в трехмерной модели, содержат точки, по которым эти объекты строятся. Каждая точка содержит пару чисел: широту и долготу в градусах. Для корректного и более точного отображения переведем данные координаты из градусов в метры на трехмерную координатную плоскость, которая используется в Unity, показанную на рисунке 3.5.

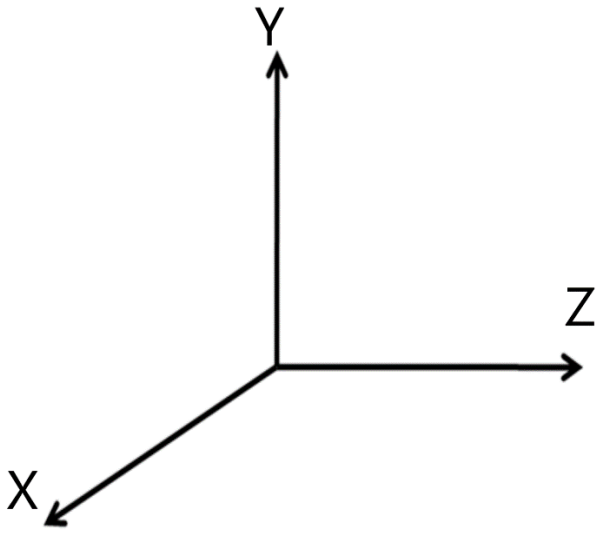

Рисунок 3.5 — Координатная плоскость в Unity

Так как в OSM точки расположены на двумерной плоскости, то для зданий за высоту будем брать этажность здания, умноженную на 3 метра, если этажность не указана, то говорим, что здание имеет один этаж.

Далее преобразуем широту и долготу в метры. Для из файла area.xml мы берём координату нижней широты (bottom) и правой долготы (right).

Широта преобразовывается по следующей формуле:

$$lat_m = (lat_d - bottom) * mer/360, \quad (3.1)$$

где:

– lat_m — широта в метрах;

– lat_d — широта в градусах;

– $bottom$ — нижняя широта в градусах;

– mer — длина меридиана в метрах, равная 40008550 м.

Долгота преобразовывается по следующей формуле:

$$lon_m = (lon_d - right) * \cos(lon_d - right) * lon_e, \quad (3.2)$$

где:

– lon_m — долгота в метрах;

– lat_d — долгота в градусах;

– $right$ — правая долгота в градусах;

– lon_e— длина одного градуса на экваторе в метрах, равная 111321,377778 м.

После данных преобразований координаты на земном шаре преобразовались в координаты на плоскости, где 1 единица деления = 1 метру.

3.3 Создание трехмерной модели в Unity

Для создания трехмерной модели в Unity необходимо иметь данные для построения модели. Как было описано в 3.1, для этого мы используем приложение на windows form (далее *osm.exe*). Перед создание модели будем вызывать osm.exe, находящийся в папке Realese.

Для этого создадим сцену под названием *osm*. Данная сцена будет являться начальным экраном при запуске приложения для создания модели.

На данной сцене размещаем компонент canvas (холст) – пространство для отрисовки UI элементов. На данную панель добавим кнопку Create_map, при нажатии на которую будет выполняться функция ChangeScene, со следующими действиями:

а) инициализируем и запускаем процесс OSM (запускаем приложение osm.exe);

б) как только процесс завершается, проверяем существование необходимых для создание трехмерного макета папок, если они создались, переходим на новую сцену *map*, в противном случае остаемся на сцене osm.

Вид сцены osm представлен на рисунке.

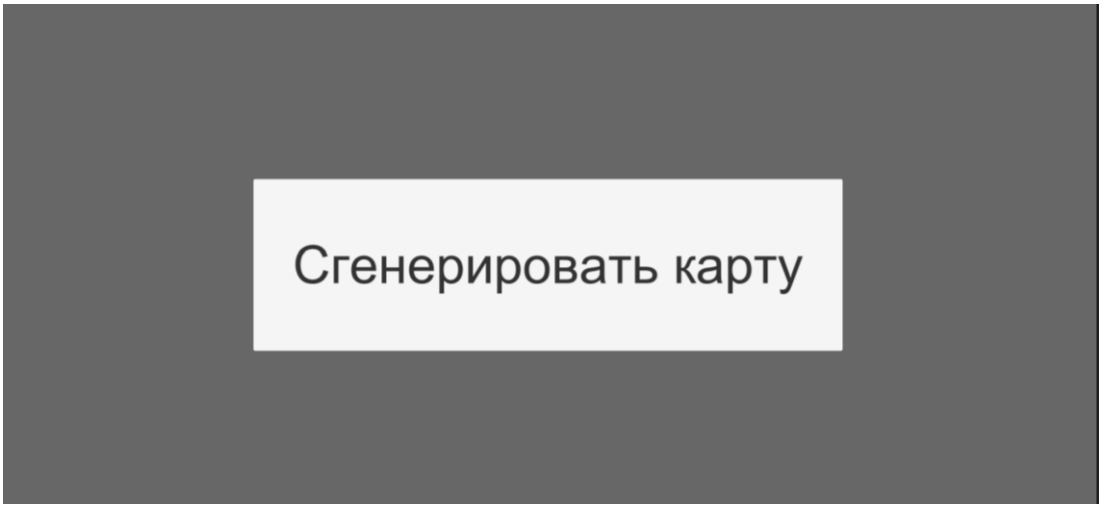

Рисунок 3.6 — Сцена osm

Сцена *map* является сценой для отрисовки трехмерной модели.

В редакторе Unity добавим несколько объектов, а именно:

– fpp — содержит объект mainCamera — камера, предназначена для вывода того, что показывается на сцене, при этом сам fpp также является отдельным объектом. С помощью данного объекта происходит перемещение по макету по осям x и z при помощи скрипта Move, который привязан к fpp, для которого является контролером, а поворот камеры для просмотра на 360 градусов вокруг объекта происходит при помощи скрипта Mouse Look, который привязан к mainCamera;

– floor — объект типа plane: квадрат, расположенный на нижних осях(x и z). Предназначен для отрисовки земли (нулевого уровня);

– walls — 4 объекта типа quad: квадрат, расположенный на осях x и y. Предназначен для отрисовки стен, за которые не может выйти fpp;

Отрисовка объектов будет происходит при помощи скрипта building.cs, который будет активироваться при запуске сцены map.

Весь код building.cs представлен в приложении Е.

Инициализируем переменные, которые понадобятся в дальнейшем.

– double maxX – будет хранить максимальное значение широты среди всех объектов;

– double minX – будет хранить минимальное значение широты среди всех объектов;

– double maxZ – будет хранить максимальное значение долготы среди всех объектов;

– double minZ – будет хранить минимальное значение долготы среди всех объектов;

Данные переменные понадобятся для построения floor;

– GameObject Fl – переменная для построения объекта floor;

– GameObject wall_1–wall4 для построения объектов wall_1–wall4;

– Material materialBuilding – переменная, хранящая информацию об материале для отображения объектов building (здания). Привязан к building.mat, является материалом, который показывает границы материала;

– Material materialRoads – переменная, хранящая информацию об материале для отображения объектов автомобильных дорог. Привязан к roads.mat, является материалом серого цвета;

– Material materialFootway– переменная, хранящая информацию об материале для отображения объектов переходных дорог. Привязан к footway.mat, является материалом светло-бурого цвета.

В функции Start в building.cs происходит следующий алгоритм действий:

а) проверяется, является текущая сцена map, до тех пор, пока текущая сцена не станет map;

б) выполняется функция Buildings для построения объектов зданий, информация о которых хранится в папке buildings;

в) выполняется функция Polygon для построения объектов зданий, информация о которых хранится в папке polygon;

г) выполняется функция Highway для построения объектов зданий, информация о которых хранится в папке highway;

Для построения стен и земли ответим на вопрос: как объекты определяются на координатной сетке?

В пунктах б-в в дополнение к построению объектов также будут находиться переменные maxX, minX, maxZ, minZ. Между данными переменными будут находится все объекты, построенные на карте. Поэтому floor размещаем по оси x в среднем значении minX и maxX, по оси z в среднем значении minZ и maxZ. Если мы растягиваем объект по оси y на 2, то это означает, что и в низ, и вверх добавится длинна 2, поэтому растягиваем по оси x на значение (maxX-minX)/2, по оси y на 0, по оси z на значение(maxZ-minZ)/2;

Таким образом:

д) выставляем объект floor по следующим координатам: x как среднее значение между minX и maxX, y как 0, z как среднее значение между minZ и maxZ, растягиваем по оси x на значение (maxX-minX)/10, по оси y на 0, по оси z на значение(maxZ-minZ)/10;

Деление на 5 происходит потому, что изначально plane растянут на 10 по x и z.

На рисунке 3.7 видно, что wall_1 строится вдоль прямой, расположенной на оси minX, но так как объект типа quad генерируется по прямой minY, построение будет выглядеть следующим образом: wall_1 выставляется в точку, являющуюся серединой отрезка wall_1, т.е. в точку с координатами по x как minX, z как среднее значение между minZ и maxZ. Так как изменение длины происходит по оси Z, следовательно для wall_1 растяжение будет по оси x на значение среднее значение между minZ и maxZ. По оси y разместим в координате 100, растяжение сделаем на 200. На последнем этапе необходимо повернуть стену на 270 градусов по оси y.

Wall_2, wall_3 и wall_4 оправляются аналогично.

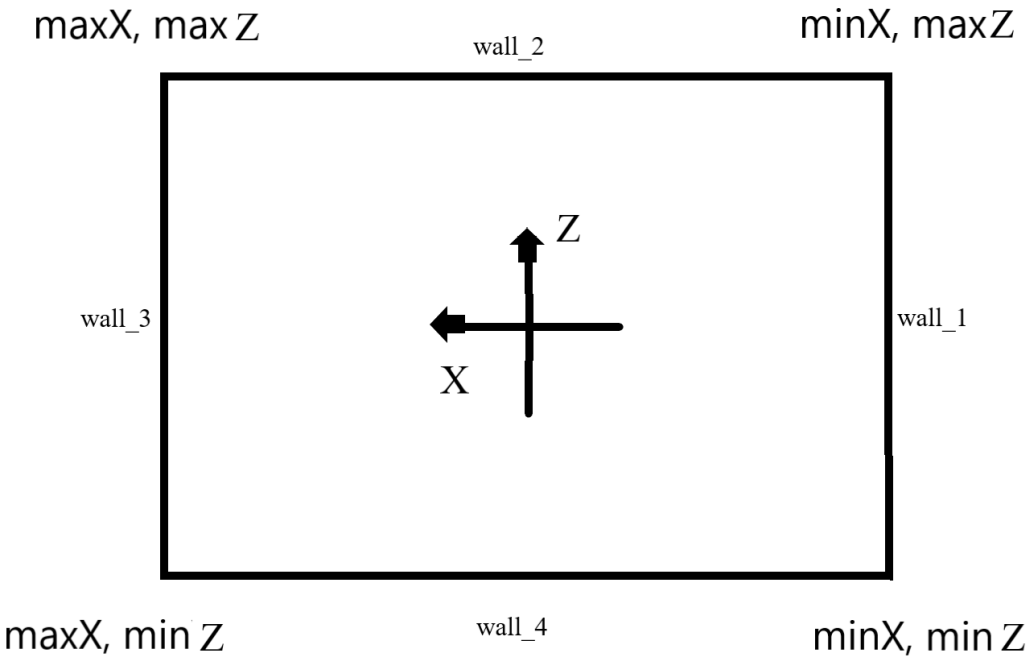

Рисунок 3.7 — Схема построения floor и wall_1— wall4

Таким образом:

е) выставляем объект wall_1 по следующим координатам: x как minX, y как 100, z как среднее значение между minZ и maxZ, растягиваем по оси x на значение maxZ-minZ, по оси y на 200, по оси z на 1;

ж) выставляем объект wall_2 по следующим координатам: x как среднее значение minX и maxX, y как 100, z как maxZ, растягиваем по оси x на значение maxZ-minZ, по оси y на 200, по оси z на 1;

з) выставляем объект wall_3 по следующим координатам: x как maxX, y как 100, z как среднее значение между minZ и maxZ, растягиваем по оси x на значение maxZ-minZ, по оси y на 200, по оси z на 1;

и) выставляем объект wall_4 по следующим координатам: x как среднее значение minX и maxX, y как 100, z как minZ, растягиваем по оси x на значение maxZ-minZ, по оси y на 200, по оси z на 1;

Разберем подробнее построение объектов в пунктах б-в.

3.3.1 Построение зданий

Построение зданий будет выполнятся функцией Buildings, параметром которой является переменная type типа string, отвечающая за то, здания какого типа будут строится: way или relation. В данной функции будем строить только стены домов, так как крыши не будет видно, но при этом их генерация займет некоторое время для генерации и ресурсы для их показа.

Алгоритм действий в функции Buildings:

а) создается массив file_list типа string, в который записываются имена всех файлов, находящихся в папке building за исключением файла area.xml;

б) из файла area.xml считываются переменные right и bottom;

циклом foreach идет считывание информации из каждого файла, а именно координат всех точек, преобразованных по правилам, описанным в пункте 3.2, а также идентификатор объекта, этажность, название улицы и номер дома (если они имеются). Для типа relation данные для построения берем из члена outer;

в) после считывания данных из файла вычисляются переменные maxX, minX, maxZ, minZ;

г) формируется 2 списка точек: points_low – нижние точки, по оси y значение 0) и points_high – верхние точки, по оси y значение равно этажность*3.

д) формируется 2 списка triangles и vertices для создания полигонов по следующему принципу:

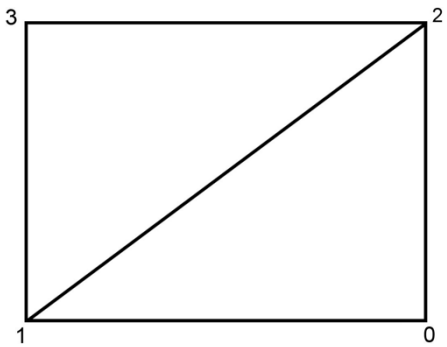

Рисунок 3.8 — Пример для построения стен у зданий

На рисунке 3.8 за точки 0, 1 обозначим две последовательные точки из списка points_low, за точки 2, 3 обозначим две последовательные точки из списка points_high. Необходимо построить 2 треугольника: с точками 0, 1, 2 и 2, 3, 1. Для этого в список vertices добавляем сначала добавляем точки из points_low, затем точки из points_high. Таким образом в данном списке в индексах 0 — vertices.Count/2-1 хранятся точки из points_low, в индексах vertices.Count/2 — vertices.Count-1 хранятся точки из points_high, где vertices.Count — количество элементов в списке vertices. Далее в список triangles заносим тройки индексов точек из vertices, которые формируют треугольники, которые соответствуют двум последовательным точки из points_low и одной точки из points_high, соответствующей точке, которая находится выше первой из двух выбранных точек. Отдельно обрабатывается случай для последней точки, в данном случае в triangles последовательно заносится индекс последней нижней точки, индекс первой нижней точки и индекс последней верхней точки.

Аналогично формируем треугольник, состоящий из индексов двух последовательных точек из points_high и индекса точки из points_low, находящейся ниже второй точки.

ж) обращаемся к функции CreateCube, параметрами которой являются: списки vertices и triangles, а также две переменные addr_street и addr_house_number типа string, в которых передается название улицы и номер дома соответственно. В данной функции происходит построение здания с указанными параметрами на сцене map.

3.3.2 Построение дорог

Построение зданий будет выполнятся функцией Highway.

Алгоритм действий в функции Highway:

а) создается массив file_list типа string, в который записываются имена всех файлов, находящихся в папке highway за исключением файла area.xml;

б) из файла area.xml считываются переменные right и bottom;

в) циклом foreach идет считывание информации из каждого файла, а именно координат всех точек, преобразованных по правилам, описанным в пункте 3.2, а также количество полос и название улицы, на которой расположена дорога (если имя улицы существует) или проезд, если highway является типом service, или тротуар, если это пешеходная дорога;

г) после считывания данных из файла вычисляются переменные maxX, minX, maxZ, minZ;

д) формируется список точек: points – точки, составляющие дорогу (по оси y значение 0.1, если автомобильная дорога; 0.11, если пешеходная);

е) формируется 2 списка triangles и vertices для создания полигонов по следующему принципу:

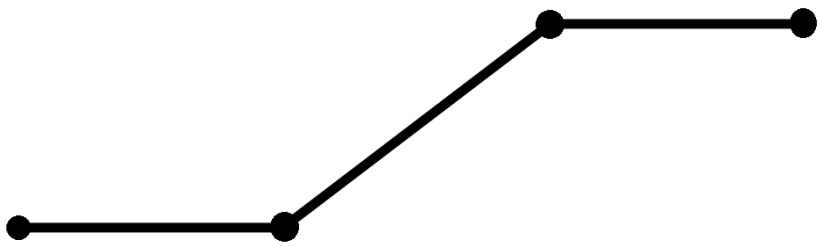

Рисунок 3.8 — Пример дороги

Как было сказано ранее, дороги в OSM представляются на карте в виде точек, без указания ширины в координатах. Для построения дороги с шириной воспользуемся следующим алгоритмом:

а) построим 4 точки для двух последовательных координат дорог, которые будут составлять край дороги, если бы дорога состояла только из этих точек, как показано на рисунке 3.9.

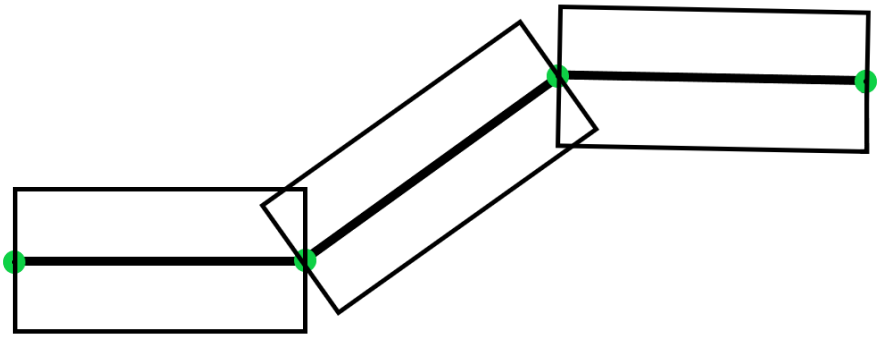

Рисунок 3.9 — Дорога, построенная пунктом а

б) для двух соседних участков найдем точки пересечения краев при помощи метода Гауса, показано на рисунке 3.10, красными точками обозначены найденные точки, серыми изначальные;

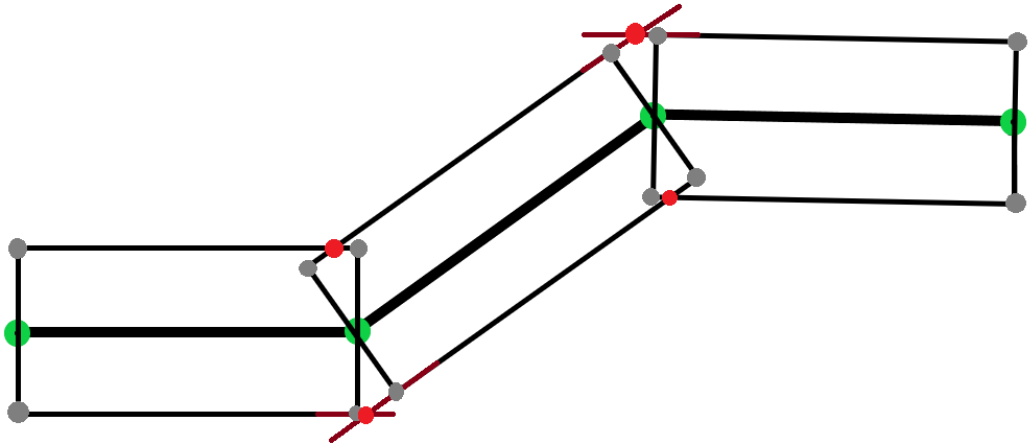

Рисунок 3.10 — Дорога, построенная пунктом б

в) формируется 2 списка triangles и vertices для создания полигонов по следующему принципу: берем 2 последовательных участка, в vertices заносим две начальные у первого участка, общие для конечного первого и начального второго и конечные и второго и формируем triangles, как показано на рисунке 3.11;

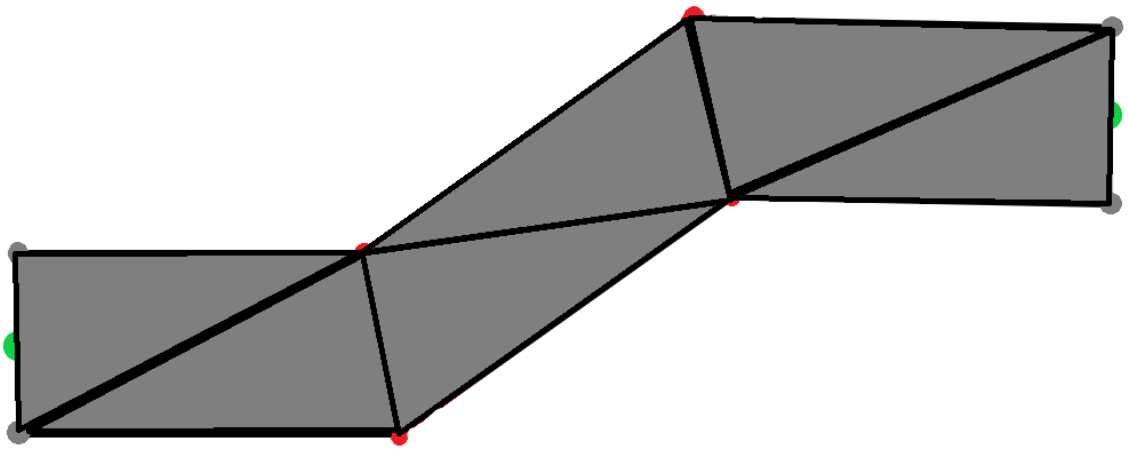

Рисунок 3.10 — Дорога, построенная пунктом в

г) обращаемся к функции CreateLines, параметрами которой являются: списки vertices и triangles, а также переменные lines типа double,в котором передается количество полос и street_name типа string, в котором передается имя улицы, на которой расположена дорога или ее тип. В данной функции происходит построение дорог с указанными параметрами на сцене map.

3.4 Пример работы приложения

а) запускаем приложение Unity;

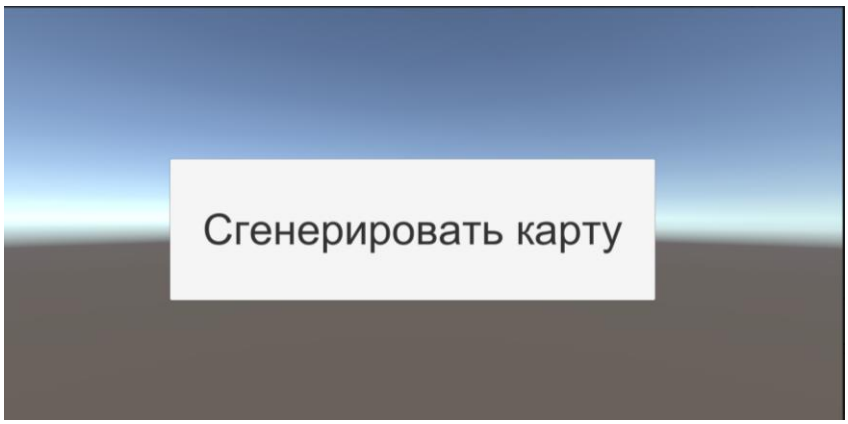

Рисунок 3.11 — Сцена osm

б) нажимаем на кнопку *Сгенерировать карту*, отрывается osm.exe;

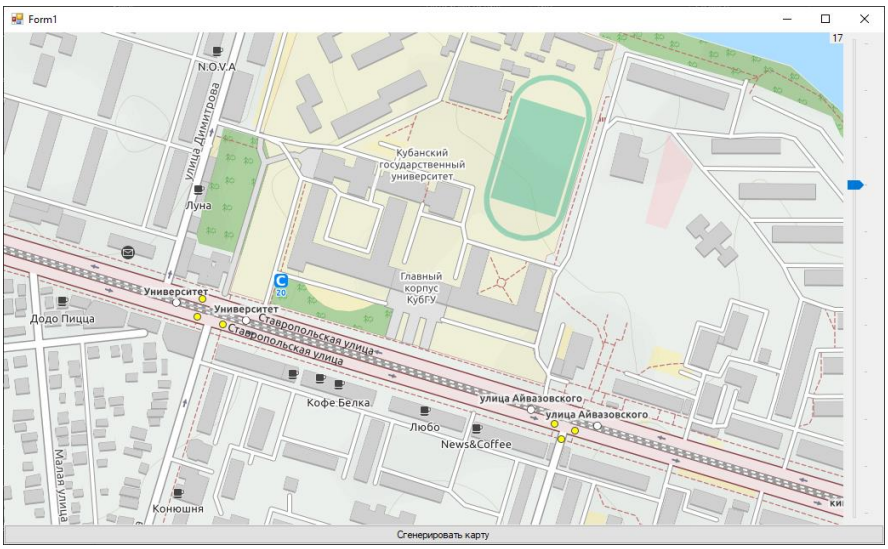

Рисунок 3.12 — приложение OSM

в) выбираем участок, необходимый для генерации и нажимаем на кнопку *Сгенерировать карту*, дожидаемся окончания генерации карты, закрываем приложение;

г) на следующей сцене создается трехмерная модель на основе выбранных данных, по которой можно перемещаться.

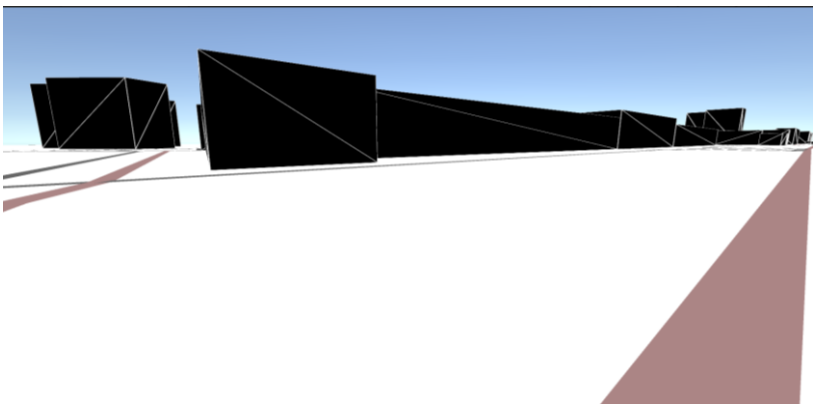

Рисунок 3.13 — сгенерированная модель

Таким образом построилась трехмерная модель по заданным параметрам.

ЗАКЛЮЧЕНИЕ

Цель данной курсовой работы – *решение задачи автоматическая генерация дорожной сити и зданий* выполнена.

В теоретической изучены принципы формирование данных в OSM, получение и обработка данных из OSM при помощи Overpass-turbo API, автоматическое создание объектов в Unity.

В оригинальной части предложен проект приложения, позволяющего получать данные OSM в виде XML файла и обрабатывать их в Unity. Получение данных и их запись происходит при помощи приложение Windows Form, а обработка полученных данных и формирование трехмерной модели происходит в Unity приложении.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Браун Л. А. История географических карт. — Москва: Центрполиграф, 2006. — 479 с. — ISBN 5-9524-2339-6 [История ГИС от древности до XX века] (дата обращения: 8.05.2021).

2 Wikipedia [Электронный ресурс] / Свободная энциклопедия – OpenStreetMap – Электрон. текстовые дан. – Режим доступа: https://ru.wikipedia.org/wiki/OpenStreetMap (дата обращения: 8.05.2021).

3 OpenStreetMap Wiki [Электронный ресурс] – RU:Overpass API – Электрон. текстовые дан. – Режим доступа: https://wiki.openstreetmap.org/wiki/RU:Overpass_API (дата обращения: 9.05.2021).

4 OpenStreetMap Wiki [Электронный ресурс] – RU:Overpass API/Language Guide / Материалы по языку запросов – Электрон. текстовые дан. – Режим доступа: https://wiki.openstreetmap.org/wiki/RU:Overpass_API/Language_Guide (дата обращения: 9.05.2021).

5 Wikipedia [Электронный ресурс] / Свободная энциклопедия – Трёхмерная графика – Электрон. текстовые дан. – Режим доступа: https://ru.wikipedia.org/wiki/Трёхмерная_графика (дата обращения: 10.05.2021).

6 Wikipedia [Электронный ресурс] / Свободная энциклопедия – Трёхмерная графика – Электрон. текстовые дан. – Режим доступа: https://ru.wikipedia.org/wiki/Трёхмерная_графика (дата обращения: 10.05.2021).

7 Unity User Manual 2020.3 (LTS) [Электронный ресурс] / Документация по Unity – Электрон. текстовые дан. – Режим доступа: https://docs.unity3d.com/Manual/index.html (дата обращения: 10.05.2021).

ПРИЛОЖЕНИЕ А

Пример типов данных в OSM

```
<!--тип данных node(точка)-->
<node id='19' lat='58.888047127548994' lon='49.747870758186764' />

<!--тип данных way(линия)-->
<node id='23' lat='58.875047918145675' lon='49.785240674006126' />
<node id='22' lat='58.86687448573524' lon='49.737090974777324' />

<way id='24'>
  <nd ref='22' />
  <nd ref='23' />
</way>

<!--тип данных relation(отношение)-->
<relation id='31'>
  <member type='way' ref='24' />
  <member type='node' ref='19' />
</relation>
```

ПРИЛОЖЕНИЕ Б

Пример вывода запроса к Overpass API

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="Overpass API 0.7.56.9 76e5016d">
<note>The data included in this document is from www.openstreetmap.org.
The data is made available under ODbL.</note>
<meta osm_base="2021-05-10T12:43:47Z"/>

  <way id="92528411">
    <bounds minlat="50.7453630" minlon="7.1700759" maxlat="50.7454797"
maxlon="7.1702704"/>
    <nd ref="3256465605" lat="50.7454797" lon="7.1700942"/>
    <nd ref="3256465599" lat="50.7454316" lon="7.1700759"/>
    <nd ref="3256465598" lat="50.7454284" lon="7.1700966"/>
    <nd ref="3256465595" lat="50.7453868" lon="7.1700808"/>
    <nd ref="3256465592" lat="50.7453630" lon="7.1702361"/>
    <nd ref="3256465604" lat="50.7454527" lon="7.1702704"/>
    <nd ref="3256465605" lat="50.7454797" lon="7.1700942"/>
    <tag k="addr:city" v="Bonn"/>
    <tag k="addr:country" v="DE"/>
    <tag k="addr:housenumber" v="2"/>
    <tag k="addr:postcode" v="53229"/>
    <tag k="addr:street" v="Am Tiergarten"/>
    <tag k="building" v="residential"/>
    <tag k="building:levels" v="2"/>
    <tag k="roof:levels" v="1"/>
    <tag k="roof:shape" v="gabled"/>
    <tag k="source:outline" v="ALK NRW"/>
  </way>
  <way id="170341558">
    <bounds minlat="50.7450377" minlon="7.1703408" maxlat="50.7451789"
maxlon="7.1705415"/>
    <nd ref="1814985839" lat="50.7450592" lon="7.1703408"/>
    <nd ref="1814985979" lat="50.7451789" lon="7.1703808"/>
    <nd ref="1814985962" lat="50.7451575" lon="7.1705415"/>
    <nd ref="1814985820" lat="50.7450377" lon="7.1705016"/>
    <nd ref="1814985839" lat="50.7450592" lon="7.1703408"/>
    <tag k="addr:city" v="Bonn"/>
    <tag k="addr:country" v="DE"/>
    <tag k="addr:housenumber" v="15"/>
    <tag k="addr:postcode" v="53229"/>
    <tag k="addr:street" v="Paul-Langen-Straße"/>
    <tag k="building" v="yes"/>
    <tag k="source" v="HiRes aerial imagery"/>
  </way>
  <way id="170341566">
    <bounds minlat="50.7450475" minlon="7.1702881" maxlat="50.7451178"
maxlon="7.1703572"/>
```

```
    <nd ref="1814985913" lat="50.7451178" lon="7.1703094"/>
    <nd ref="1814985903" lat="50.7451114" lon="7.1703572"/>
    <nd ref="1814985822" lat="50.7450475" lon="7.1703359"/>
    <nd ref="1814985834" lat="50.7450539" lon="7.1702881"/>
    <nd ref="1814985913" lat="50.7451178" lon="7.1703094"/>
    <tag k="building" v="yes"/>
  </way>
  <way id="170341586">
    <bounds minlat="50.7450049" minlon="7.1700488" maxlat="50.7451846"
maxlon="7.1702913"/>
    <nd ref="1814985793" lat="50.7450049" lon="7.1702411"/>
    <nd ref="1814985815" lat="50.7450232" lon="7.1701038"/>
    <nd ref="1814985895" lat="50.7451044" lon="7.1701309"/>
    <nd ref="1814985907" lat="50.7451154" lon="7.1700488"/>
    <nd ref="1814985983" lat="50.7451846" lon="7.1700719"/>
    <nd ref="1814985946" lat="50.7451553" lon="7.1702913"/>
    <nd ref="1814985793" lat="50.7450049" lon="7.1702411"/>
    <tag k="addr:city" v="Bonn"/>
    <tag k="addr:country" v="DE"/>
    <tag k="addr:housenumber" v="13"/>
    <tag k="addr:postcode" v="53229"/>
    <tag k="addr:street" v="Paul-Langen-Straße"/>
    <tag k="building" v="yes"/>
    <tag k="source" v="HiRes aerial imagery"/>
  </way>

</osm>
```

ПРИЛОЖЕНИЕ В

Скрптинг в Unity

```
using UnityEngine;
using System.Collections;

// класс MainPlayer, наследуемый от MonoBehaviour
public class MainPlayer : MonoBehaviour {


    // Функция Awake, вызывается раньше всех функций
    void Awake () {
       Debug.Log("I am here!");
    }

    // Функция Start, вызывается 1 раз после Awake
    void Start () {
       Debug.Log("I am alive!");
    }

    // Функция Update, вызывается каждый раз при обновлении кадра
    void Update () {
       Debug.Log("I am update!");
    }

    // Функция OnApplicationQuit, вызывается при закрытии игры
    void OnApplicationQuit () {
       Debug.Log("Bye");
    }

}
```

ПРИЛОЖЕНИЕ Г

Код приложения windows form

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Threading;
using System.Windows.Forms;
using System.IO;
using System.Xml;
using System.Net;

// Библиотеки для карты
using GMap.NET;
using GMap.NET.MapProviders;
using GMap.NET.WindowsForms;
using GMap.NET.WindowsForms.Markers;
using GMap.NET.WindowsForms.ToolTips;
namespace osm
{
    public partial class Form1 : Form
    {
        // Класс точка - координаты
        public class CPoint
        {
            public double x { get; set; }
            public double y { get; set; }
            public string place { get; set; }
            public string building { get; set; }
            public string street { get; set; }
            public string town { get; set; }

            public string Image;

            public CPoint() { }
            public CPoint(double _x, double _y)
            {
                x = _x;
                y = _y;
            }

            public CPoint(double _x, double _y, string _place)
            {
                x = _x;
                y = _y;
                place = _place;
            }

            public CPoint(double _x, double _y, string _building, string _street, string
_town)
            {
                x = _x;
                y = _y;
                building = _building;
                street = _street;
                town = _town;
            }

            public CPoint(double _x, double _y, string _building, string _street, string
_town, string _image)
            {
```

```
                x = _x;
                y = _y;
                building = _building;
                street = _street;
                town = _town;
                Image = _image;
            }
        }
        public Form1()
        {
            InitializeComponent();
        }
private void gMapControl1_Load(object sender, EventArgs e)
{
    // Настройки для компонента GMap
    gmap.Bearing = 0;
    // Перетаскивание левой кнопки мыши
    gmap.CanDragMap = true;
    // Перетаскивание карты левой кнопкой мыши
    gmap.DragButton = MouseButtons.Left;

    gmap.GrayScaleMode = true;

    // Все маркеры будут показаны
    gmap.MarkersEnabled = true;
    // Максимальное приближение
    gmap.MaxZoom = 20;
    // Минимальное приближение
    gmap.MinZoom = 10;
    // Курсор мыши в центр карты
    gmap.MouseWheelZoomType = MouseWheelZoomType.MousePositionWithoutCenter;

    gmap.MouseWheelZoomEnabled = false;
    // Отключение нигативного режима
    gmap.NegativeMode = false;
    // Разрешение полигонов
    gmap.PolygonsEnabled = true;
    // Разрешение маршрутов
    gmap.RoutesEnabled = true;
    // Скрытие внешней сетки карты
    gmap.ShowTileGridLines = false;
    // При загрузке 10-кратное увеличение
    gmap.Zoom = 17;
    // Убрать красный крестик по центру
    gmap.ShowCenter = false;

    // Чья карта используется

    gmap.MapProvider = GMapProviders.OpenCycleMap;
    GMaps.Instance.Mode = AccessMode.ServerOnly;
    gmap.Position = new PointLatLng(45.0196306, 39.0311312);

    // Добавление маркеров на карту
    // Инициаизируем список маркеров

    label1.Text = gmap.Zoom.ToString();
    trackBar.Maximum = 20;
    trackBar.Minimum = 10;
    trackBar.Value = (int)gmap.Zoom;
}
        private void createCoordinates_Click(object sender, EventArgs e)
        {
            // проверяем существование папок
            if (Directory.Exists("building"))
            {
```

```
                DirectoryInfo dir = new DirectoryInfo("building");
                foreach (FileInfo f in dir.GetFiles())
                    f.Delete();
            }
            else
                Directory.CreateDirectory("building");


            if (Directory.Exists("highway"))
            {
                DirectoryInfo dir = new DirectoryInfo("highway");
                foreach (FileInfo f in dir.GetFiles())
                    f.Delete();
            }
            else
                Directory.CreateDirectory("highway");

            if (Directory.Exists("polygon"))
            {
                DirectoryInfo dir = new DirectoryInfo("polygon");
                foreach (FileInfo f in dir.GetFiles())
                    f.Delete();
            }
            else
                Directory.CreateDirectory("polygon");
            try
            {
                // создаем и запускаем таймер
                Stopwatch stopwatch = new Stopwatch();
                stopwatch.Start();
                // обращемся к функции Req
                Req("way[building");
                Req("relation[building");
                Req("way[highway");
                // останавливаем таймер
                stopwatch.Stop();
                // выводим сообщение об успешном выполнении
                MessageBox.Show("Файлы создались успешно за время:\n" +
stopwatch.Elapsed.ToString());
            }
            catch
            {
                // если возникла ошибка
                MessageBox.Show("Ошибка!");
            }
        }

void Req(string type)
{
    // инцилизируем строку для запроса
    string sURL =
"https://overpass.openstreetmap.ru/api/interpreter?data=[out:xml][timeout:25];" + type +
"](" + gmap.ViewArea.Bottom.ToString("#.#####").Replace(",", ".") + ", " +
gmap.ViewArea.Left.ToString("#.#####").Replace(",", ".") + ", " +
gmap.ViewArea.Top.ToString("#.#####").Replace(",", ".") + ", " +
gmap.ViewArea.Right.ToString("#.#####").Replace(",", ".") + ");out geom;";
    // строка для начала файла
    string xml = "<?xml version=\"1.0\" encoding=\"UTF - 8\"?>\n";
    //переходим в необходимую папку
    //если relation
    if (type == "relation[building")
    {
        Directory.SetCurrentDirectory("polygon");
    }
    // если way building или way highway
```

```
    else
    {
        Directory.SetCurrentDirectory(type.Substring(4));
    }
    // инициализация файла area.xml
    string area = "<area bottom = \"" +
gmap.ViewArea.Bottom.ToString("#.#####").Replace(",", ".") + "\" left =\" " +
gmap.ViewArea.Left.ToString("#.#####").Replace(",", ".") + "\" top=\" " +
gmap.ViewArea.Top.ToString("#.#####").Replace(",", ".") + "\" right = \" " +
gmap.ViewArea.Right.ToString("#.#####").Replace(",", ".") + "\"/>";
    File.WriteAllText("area.xml", area);
    bool way = false;
    string req_id = "";
    // создаем файлы в папке
    using (var webClient = new WebClient())
    {
        // Выполняем запрос по адресу и получаем ответ в виде строки
        XmlTextReader reader = new XmlTextReader(sURL);
        while (reader.Read())
        {
            switch (reader.NodeType)
            {
                case XmlNodeType.Element: // формируем данные для записи в файл
                    if (!(reader.Name.Equals("note") || reader.Name.Equals("bounds") ||
reader.Name.Equals("meta") || reader.Name.Equals("osm")))
                    {
                        xml += ("<" + reader.Name);
                        while (reader.MoveToNextAttribute())
                        {
                            xml += (" " + reader.Name + "=\"" +
reader.Value.Replace("\"", "'").Replace("&", "and") + "\"");
                            if (reader.Name.Equals("id") ||
reader.Name.Equals("generator") || (reader.Name.Equals("role")))
                                xml += (">\n");
                            else if (reader.Name.Equals("lon") ||
reader.Name.Equals("v"))
                                xml += ("/>\n");
                            if (reader.Name.Equals("id"))
                                req_id = reader.Value;
                        }
                    }
                    break;
                case XmlNodeType.EndElement:  // завершаем формирование данных для записи
                    if (!(reader.Name.Equals("note") || reader.Name.Equals("bounds") ||
reader.Name.Equals("meta") || reader.Name.Equals("osm")))
                        xml += ("</" + reader.Name + ">\n");
                    if (reader.Name.Equals("way") || reader.Name.Equals("relation"))
                        way = true;
                    break;
            }
            // записываем содержимое в файла
            if (way)
            {
                File.WriteAllText(req_id + ".xml", xml);
                xml = "<?xml version=\"1.0\" encoding=\"UTF - 8\"?>\n";
                way = false;
            }
        }
    }
    // возвращаемся в дерекорию выше, где находятся папки с файлами
    Directory.SetCurrentDirectory("..\\");
}
        private void Form1_FormClosing(Object sender, FormClosingEventArgs e)
        {
```

```
            if (Directory.Exists("highway") && Directory.Exists("polygon") &&
Directory.Exists("building"))
            {
                DialogResult dialog = MessageBox.Show("Действительно выйти?", "Выход",
MessageBoxButtons.YesNo);
                if (dialog != DialogResult.Yes)
                    e.Cancel = true;
            }
            else
            {
                DialogResult dialog = MessageBox.Show("Файл/ы не найдены, точно хотите
выйти?", "Выход", MessageBoxButtons.YesNo);
                if (dialog != DialogResult.Yes)
                    e.Cancel = true;
            }
        }

        private void trackBar_Scroll_1(object sender, EventArgs e)
        {
            gmap.Zoom = trackBar.Value;
            label1.Text = trackBar.Value.ToString();
        }
    }
}
```

ПРИЛОЖЕНИЕ Д

Примеры файлов, возвращаемых Req

```
<!--way building-->
<?xml version="1.0" encoding="UTF - 8"?>
<way id="38198344">
     <nd ref="450080435" lat="45.0555808" lon="38.9700458"/>
     <nd ref="450080439" lat="45.0558334" lon="38.9710479"/>
     <nd ref="450080443" lat="45.0556830" lon="38.9711239"/>
     <nd ref="450080446" lat="45.0554304" lon="38.9701218"/>
     <nd ref="450080435" lat="45.0555808" lon="38.9700458"/>
     <tag k="addr:city" v="Краснодар"/>
     <tag k="addr:country" v="RU"/>
     <tag k="addr:housenumber" v="71"/>
     <tag k="addr:street" v="Гаражная улица"/>
     <tag k="building" v="apartments"/>
     <tag k="building:levels" v="20"/>
</way>

<!--relation building-->
<?xml version="1.0" encoding="UTF - 8"?>
<relation id="3723721">
     <member type="way" ref="38235451" role="outer">
       <nd lat="45.0540534" lon="38.9699055"/>
       <nd lat="45.0545986" lon="38.9696474"/>
       <nd lat="45.0546460" lon="38.9698244"/>
       <nd lat="45.0541088" lon="38.9700710"/>
       <nd lat="45.0540303" lon="38.9702704"/>
       <nd lat="45.0541455" lon="38.9706931"/>
       <nd lat="45.0545875" lon="38.9706996"/>
       <nd lat="45.0545991" lon="38.9708926"/>
       <nd lat="45.0541561" lon="38.9708795"/>
       <nd lat="45.0540407" lon="38.9708084"/>
       <nd lat="45.0538739" lon="38.9702465"/>
     </member>
     <member type="way" ref="279984888" role="outer">
       <nd lat="45.0538739" lon="38.9702465"/>
       <nd lat="45.0538877" lon="38.9701510"/>
       <nd lat="45.0539066" lon="38.9700254"/>
       <nd lat="45.0539554" lon="38.9699891"/>
       <nd lat="45.0540534" lon="38.9699055"/>
     </member>
```

```
    <tag k="addr:city" v="Краснодар"/>
    <tag k="addr:country" v="RU"/>
    <tag k="addr:housenumber" v="67"/>
    <tag k="addr:postcode" v="350020"/>
    <tag k="addr:street" v="Гаражная улица"/>
    <tag k="building" v="apartments"/>
    <tag k="building:levels" v="22"/>
    <tag k="type" v="multipolygon"/>
</relation>

<!--way highway-->
<?xml version="1.0" encoding="UTF - 8"?>
<way id="26939547">
    <nd ref="295167674" lat="45.0554084" lon="38.9668426"/>
    <nd ref="560168309" lat="45.0554300" lon="38.9669389"/>
    <nd ref="450080503" lat="45.0555757" lon="38.9675704"/>
    <nd ref="3812858559" lat="45.0557042" lon="38.9681467"/>
    <nd ref="4513279442" lat="45.0557364" lon="38.9682858"/>
    <nd ref="4513279445" lat="45.0558471" lon="38.9687515"/>
    <nd ref="4513279446" lat="45.0558855" lon="38.9689131"/>
    <nd ref="4513279449" lat="45.0560420" lon="38.9695118"/>
    <nd ref="295167673" lat="45.0564138" lon="38.9711896"/>
    <tag k="highway" v="tertiary"/>
    <tag k="name" v="Морская улица"/>
    <tag k="name:en" v="Morskaya Street"/>
    <tag k="name:ru" v="Морская улица"/>
    <tag k="postal_code" v="350020"/>
    <tag k="source" v="AlexAR Mapping Project"/>
</way>
```

ПРИЛОЖЕНИЕ Е

Код building.cs

```
using System;
using UnityEngine;
using System.IO;
using System.Xml;
using System.Collections.Generic;
using UnityEngine.SceneManagement;

public class Building : MonoBehaviour
{
    public double maxX;
    public double minX;
    public double maxZ;
    public double minZ;
    public GameObject Fl;
    public GameObject wall_1;
    public GameObject wall_2;
    public GameObject wall_3;
    public GameObject wall_4;
    public Material materialBuilding;
    public Material materialRoads;
    public Material materialFootway;
    void Start(){
        while (true){
            if(SceneManager.GetActiveScene().name=="map")
                break;
        }
        // вызов функций для генерации зданий и дорог
        Buildings("way");
        Buildings("relation");
        Highway();

        Fl.transform.localPosition =  new Vector3(Convert.ToSingle(maxX+minX)/2, 0, C
onvert.ToSingle(maxZ+minZ)/2);
        Fl.transform.localScale = new Vector3(Convert.ToSingle(maxX-
minX)/10, 0 ,Convert.ToSingle(maxZ-minZ)/10);

        wall_1.transform.localPosition =  new Vector3(Convert.ToSingle(minX), 100f, -
Convert.ToSingle(maxZ+minZ)/2);
        wall_1.transform.localScale = new Vector3(Convert.ToSingle(maxZ-
minZ), 200f ,1f);
        wall_1.transform.localRotation = Quaternion.Euler(0, 270, 0);

        wall_2.transform.localPosition =  new Vector3(Convert.ToSingle(maxX+minX)/2,
100f, -Convert.ToSingle(maxZ));
        wall_2.transform.localScale = new Vector3(Convert.ToSingle(maxX-
minX), 200f ,1f);
        wall_2.transform.localRotation = Quaternion.Euler(0, 180, 0);

        wall_3.transform.localPosition =  new Vector3(Convert.ToSingle(maxX), 100f, -
Convert.ToSingle(maxZ+minZ)/2);
        wall_3.transform.localScale = new Vector3(Convert.ToSingle(maxZ-
minZ), 200f ,1f);
        wall_3.transform.localRotation = Quaternion.Euler(0, 90, 0);
```

```
        wall_4.transform.localPosition =  new Vector3(Convert.ToSingle(maxX+minX)/2,
100f, -Convert.ToSingle(minZ));
        wall_4.transform.localScale = new Vector3(Convert.ToSingle(maxX-
minX), 200f ,1f);
        wall_4.transform.localRotation = Quaternion.Euler(0, 0, 0);
   }

    // функция для постороения зданий
    void Buildings(string type){
        // если тип relation, то считываем файлы из папки polygon, иначе из папки bui
ldings
        string[] file_list = new string[]{};
        if(type.Equals("relation"))
            file_list = Directory.GetFiles("polygon");
        else if (type.Equals("way"))
            file_list = Directory.GetFiles("building");
        List<string> f_a = new List<string>();
        for(int i = 0; i<file_list.Length-1;i++)
            f_a.Add(file_list[i]);
        file_list = f_a.ToArray();
        // инициализируем дополнительные переменные
        double bootom = 0;
        double right = 0;
        double lat = 0;
        double lon = 0;
        // считываем данные из area.xml
        FileStream file_area = new FileStream("building\\area.xml", FileMode.OpenOrCr
eate,  FileAccess.ReadWrite, FileShare.None);
        StreamReader read_area = new StreamReader(file_area);
        XmlTextReader reader = new XmlTextReader(read_area);
        while (reader.Read())
        {
            switch (reader.NodeType)
            {
                case XmlNodeType.Element:
                    if (reader.Name.Equals("area"))
                    {
                        while (reader.MoveToNextAttribute())
                        {
                            if (reader.Name.Equals("bottom"))
                                bootom = Double.Parse(reader.Value.Replace(".", ","))
;
                            else if (reader.Name.Equals("right"))
                                right = Double.Parse(reader.Value.Replace(".", ","));
                        }
                    }
                    break;
                case XmlNodeType.EndElement:
                    break;
            }
        }
        List<(int, double[][,], int, string, string)> building = new List<(int, doubl
e[][,], int, string, string)>();
        // считываем все файлы из файлов в папке
        foreach (string f in file_list){
            FileStream file = new FileStream(f, FileMode.OpenOrCreate,  FileAccess.Re
adWrite, FileShare.None);
            StreamReader read = new StreamReader(file);
            reader = new XmlTextReader(read);
            List<double[,]> coor = new List<double[,]>();
            int id_bulding = 0;
            lat = 0;
```

```
            lon = 0;
            int level = 1;
            string addr_street = "";
            string addr_house_number = "";
            bool o = false;
            while (reader.Read())
            {
                switch (reader.NodeType)
                {
                    case XmlNodeType.Element:
                        // если строим здания типа relation
                        if(type =="relation"){
                            if (reader.Name.Equals("relation"))
                            {
                                while (reader.MoveToNextAttribute())
                                    id_bulding = Int32.Parse(reader.Value);
                            }
                            else if(reader.Name.Equals("member")){
                                while (reader.MoveToNextAttribute())
                                {
                                    if (reader.Name.Equals("role"))
                                    {
                                        if(reader.Value.Equals("outer"))
                                            o = true;
                                        else
                                            o = false;
                                    }
                                }
                            }
                            else if (reader.Name.Equals("nd") && o)
                            {
                                while (reader.MoveToNextAttribute())
                                {
                                    if (reader.Name.Equals("lat"))
                                        lat = (Double.Parse(reader.Value.Replace(".",
 ",")) - bootom)*40008550/360;
                                    else if (reader.Name.Equals("lon"))
                                    {
                                        lon = Double.Parse(reader.Value.Replace(".",
 ","))-right;
                                        double len = Math.Cos(lon) * 111321.377778;
                                        lon *= len;
                                    }
                                }
                                double[,] c = { { lat, lon } };
                                coor.Add(c);
                            }
                        }
                        // если строим здания типа way
                        else if(type =="way"){
                            if (reader.Name.Equals("way"))
                            {
                                while (reader.MoveToNextAttribute())
                                    id_bulding = Int32.Parse(reader.Value);
                            }
                            else if (reader.Name.Equals("nd"))
                            {
                                while (reader.MoveToNextAttribute())
                                {
                                    if (reader.Name.Equals("lat"))
                                        lat = (Double.Parse(reader.Value.Replace(".",
 ",")) - bootom)*40008550/360;
```

```
                                else if (reader.Name.Equals("lon"))
                                {
                                    lon = Double.Parse(reader.Value.Replace(".",
","))-right;
                                    double len = Math.Cos(lon) * 111321.377778;
                                    lon *= len;
                                }
                            }
                            double[,] c = { { lat, lon } };
                            coor.Add(c);
                        }
                    }
                    // считываем дополнительную информацию
                    else if (reader.Name.Equals("tag"))
                    {
                        bool b_l = false;
                        bool a_s = false;
                        bool a_h = false;
                        while (reader.MoveToNextAttribute())
                        {
                            if (reader.Name.Equals("k"))
                            {
                                if (reader.Value.Equals("building:levels"))
                                    b_l = true;
                                else if(reader.Value.Equals("addr:street"))
                                    a_s = true;
                                else if(reader.Value.Equals("addr:housenumber"))
                                    a_h = true;
                            }
                            if (reader.Name.Equals("v"))
                            {
                                if (a_h){
                                    addr_house_number = reader.Value;
                                    a_h=false;
                                }

                                else if (a_s){
                                    addr_street = reader.Value;
                                    a_s = false;
                                }
                                else if (b_l){
                                    level = Int32.Parse(reader.Value);
                                    b_l=false;
                                }
                            }
                        }
                    }
                    break;
                // завершаем чтение из файла
                case XmlNodeType.EndElement:
                    if(reader.Name.Equals("way")|| reader.Name.Equals("relation")
)
                    {
                        double[][,] arr = coor.ToArray();
                        (int, double[][,], int, string, string) bu = (id_bulding,
 arr, level, addr_street, addr_house_number);
                        // нахадим значениния для построения земли
                        for (int j=0;j<bu.Item2.Length;j++){
                            if (bu.Item2[j][0, 0] > maxX)
                                maxX = bu.Item2[j][0, 0];
                            else if (bu.Item2[j][0, 0] < minX)
                                minX = bu.Item2[j][0, 0];
```

```
                                if (bu.Item2[j][0, 1] > maxZ)
                                    maxZ = bu.Item2[j][0, 1];
                                else if (bu.Item2[j][0, 1] < minZ)
                                    minZ = bu.Item2[j][0, 1];
                            }
                            List<Vector3> vertices = new List<Vector3>();
                            List<int> triangles = new List<int>();

                            List<Vector3> points_high = new List<Vector3>();
                            List<Vector3> points_low = new List<Vector3>();
                            // формируем список нижних точек здания
                            for (int j= 0; j < bu.Item2.Length; j++){
                                points_low.Add(new Vector3(Convert.ToSingle(bu.Item2[
j][0, 0]),0, -Convert.ToSingle(bu.Item2[j][0, 1])));;
                            }
                            // формируем список нижних точек здания
                            for (int j = 0; j < bu.Item2.Length; j++){
                                points_high.Add(new Vector3(Convert.ToSingle(bu.Item2
[j][0, 0]), Convert.ToSingle(bu.Item3*3), -
Convert.ToSingle(bu.Item2[j][0, 1])));
                            }
                            Vector3[] vertices_low = points_low.ToArray();
                            Vector3[] vertices_high = points_high.ToArray();
                            // формируем списки triangles и vertices
                            for (int l=0; l < vertices_low.Length-
1;l++){
                                triangles.Add(l+1);
                                triangles.Add(l);
                                triangles.Add(vertices_low.Length+l);

                                triangles.Add(l);
                                triangles.Add(l+1);
                                triangles.Add(vertices_low.Length+l);
                            }
                            triangles.Add(0);
                            triangles.Add(vertices_low.Length-1);
                            triangles.Add(vertices_low.Length);
                            triangles.Add(vertices.Count);
                            triangles.Add(vertices_low.Length);
                            triangles.Add(vertices_low.Length-1);
                            for (int l=0; l < vertices_low.Length-1;l++){
                                triangles.Add(vertices_low.Length+vertices_low.Length
-1-l);
                                triangles.Add(vertices_low.Length+vertices_low.Length
-2-l);
                                triangles.Add(vertices_low.Length-l-1);
                                triangles.Add(vertices_low.Length+vertices_low.Length
-2-l);
                                triangles.Add(vertices_low.Length+vertices_low.Length
-1-l);
                                triangles.Add(vertices_low.Length-l-1);
                            }
                            triangles.Add(vertices_low.Length+vertices_low.Length-1);
                            triangles.Add(vertices_low.Length-1);
                            triangles.Add(vertices_low.Length);
                            triangles.Add(vertices_low.Length+vertices_low.Length-1);
                            triangles.Add(vertices_low.Length);
                            triangles.Add(vertices_low.Length-1);
                            foreach (Vector3 V  in vertices_low)
                                vertices.Add(V);
                            foreach (Vector3 V  in vertices_high)
                                vertices.Add(V);
```

```
                            // производим отображение на сцену
                            CreateCube(triangles, vertices, bu.Item4, bu.Item5);
                            building.Add(bu);
                            level = 1;
                            coor.Clear();
                        }
                        break;
                    }
                }
            }
        }
        // функция для постороения дорог
        void Highway()
        {
            // считывание файлов из папки highway
            string[] file_list = Directory.GetFiles("highway");
            List<string> f_a = new List<string>();
                for(int i = 0; i<file_list.Length-1;i++)
                    f_a.Add(file_list[i]);
            file_list = f_a.ToArray();
            // инициализация дополнительных переменных
            double bootom = 0;
            double right = 0;
            double lat = 0;
            double lon = 0;
            // считывание даных из area.xml
            FileStream file_area = new FileStream("highway\\area.xml", FileMode.OpenOrCre
ate,  FileAccess.ReadWrite, FileShare.None);
            StreamReader read_area = new StreamReader(file_area);
            XmlTextReader reader = new XmlTextReader(read_area);
            while (reader.Read())
            {
                switch (reader.NodeType)
                {
                    case XmlNodeType.Element:
                        if (reader.Name.Equals("area"))
                        {
                            while (reader.MoveToNextAttribute())
                            {
                                if (reader.Name.Equals("bottom"))
                                    bootom = Double.Parse(reader.Value.Replace(".", ","))
;
                                else if (reader.Name.Equals("right"))
                                    right = Double.Parse(reader.Value.Replace(".", ","));
                            }
                        }
                        break;
                    case XmlNodeType.EndElement:
                            break;
                }
            }
            // считываем все файлы из файлов в папке
            foreach (string f in file_list){
                FileStream file = new FileStream(f, FileMode.OpenOrCreate,  FileAccess.Re
adWrite, FileShare.None);
                StreamReader read = new StreamReader(file);
                reader = new XmlTextReader(read);
                // инициализация вспомогательных переменных
                List<double[,]> coor = new List<double[,]>();
                int id_highway = 0;
                lat = 0;
                lon = 0;
```

```
                double lanes = 1;
                string street_type = "";
                while (reader.Read())
                {
                    switch (reader.NodeType)
                    {
                        case XmlNodeType.Element:
                        // считывание информации о точках
                            if (reader.Name.Equals("way"))
                            {
                                while (reader.MoveToNextAttribute())
                                    id_highway = Int32.Parse(reader.Value);
                            }
                            else if (reader.Name.Equals("nd"))
                            {
                                while (reader.MoveToNextAttribute())
                                {
                                    if (reader.Name.Equals("lat"))
                                        lat = (Double.Parse(reader.Value.Replace(".", ","
)) - bootom)*40008550/360;
                                    else if (reader.Name.Equals("lon"))
                                    {
                                        lon = Double.Parse(reader.Value.Replace(".", ",")
) - right;
                                        double len = Math.Cos(lon) * 111321.377778;
                                        lon *= len;
                                    }
                                }
                                double[,] c = { { lat, lon } };
                                coor.Add(c);
                            }
                            // считывание дополнительной информации
                            else if (reader.Name.Equals("tag"))
                            {
                                bool lan = false;
                                bool s_n = false;
                                bool f_w = false;
                                while (reader.MoveToNextAttribute())
                                {
                                    if (reader.Name.Equals("k"))
                                    {
                                        if (reader.Value.Equals("lanes"))
                                            lan = true;
                                        else if(reader.Value.Equals("name:ru"))
                                            s_n=true;
                                        else if(reader.Value.Equals("highway"))
                                            f_w = true;
                                    }
                                    if (reader.Name.Equals("v"))
                                    {
                                        if (s_n){
                                            street_type = reader.Value;
                                            s_n=false;
                                        }
                                        else if (lan){
                                            lanes = Double.Parse(reader.Value);
                                            lan=false;
                                        }
                                        else if(f_w && reader.Value.Equals("footway")){
                                            street_type = "Тротуар";

                                            lanes = 0.5;
```

```
                                    }
                                    else if(f_w && reader.Value.Equals("service")){
                                        street_type = "Проезд";

                                        lanes = 0.5;
                                    }
                                }
                            }
                        }
                        break;
                    // завершение считывания данных из файла
                    case XmlNodeType.EndElement:
                        double[][,] arr = coor.ToArray();
                        // необходимые данные для построения объекта
                        (int, double[][,], double, string) bu = (id_highway, arr, lan
es, street_type);
                        // нахадим значениния для построения земли
                        for (int j=0; j < bu.Item2.Length;j++){
                            if (bu.Item2[j][0, 0] > maxX)
                                maxX = bu.Item2[j][0, 0];
                            else if (bu.Item2[j][0, 0] < minX)
                                minX = bu.Item2[j][0, 0];
                            if (bu.Item2[j][0, 1] > maxZ)
                                maxZ = bu.Item2[j][0, 1];
                            else if (bu.Item2[j][0, 1] < minZ)
                                minZ = bu.Item2[j][0, 1];
                        }
                        // список для хранения точек
                        List<Vector3> points = new List<Vector3>();
                        for (int i=0; i < bu.Item2.Length; i++){
                            // если тратуар, ставим высоту по у 0.11, в противоном сл
учае 0.1
                            if (bu.Item4.Equals("Тратуар"))
                                points.Add(new Vector3(Convert.ToSingle(bu.Item2[i][0
, 0]), 0.11f, -Convert.ToSingle(bu.Item2[i][0, 1])));
                            else
                                points.Add(new Vector3(Convert.ToSingle(bu.Item2[i][0
, 0]), 0.1f, -Convert.ToSingle(bu.Item2[i][0, 1])));
                        }
                        // формируем списки triangles и vertices
                        List<int> triangles = new List<int>();
                        List<Vector3> vertices = new List<Vector3>();
                        // строим по по правилам в 3.3.2
                        if (points.Count!=2)
                        {
                            for (int  i = 0;  i < points.Count-2;  i+=2)
                            {
                                float a1 = (float)Math.Sqrt(Math.Pow(points[i+1].x-
points[i].x, 2) + Math.Pow(points[i+1].z-points[i].z, 2));
                                float d1 = (float)bu.Item3*3/2;
                                float c1 = (float)Math.Sqrt(Math.Pow(a1, 2) + Math.Po
w(d1, 2));

                                float a2 = (float)Math.Sqrt(Math.Pow(points[i+2].x-
points[i+1].x, 2)+Math.Pow(points[i+2].z-points[i+1].z, 2));
                                float d2 = (float)bu.Item3*3/2;
                                float c2 = (float)Math.Sqrt(Math.Pow(a2, 2) + Math.Po
w(d2, 2));

                                Vector3 e1 = new Vector3((points[i+1].x-
points[i].x)/a1, points[i].y ,(points[i+1].z-points[i].z)/a1);
```

```
                                Vector3 e2 = new Vector3((points[i+2].x-
points[i+1].x)/a2, points[i].y ,(points[i+2].z-points[i+1].z)/a2);

                                Vector3 v2 = new Vector3(points[i].x+e1.z*d1, e1.y, p
oints[i].z-e1.x*d1);
                                Vector3 v3 = new Vector3(points[i].x-
e1.z*d1, e1.y, points[i].z+e1.x*d1);

                                Vector3 v4_1 = new Vector3(points[i+1].x+e1.z*d1, e1.
y, points[i+1].z-e1.x*d1);
                                Vector3 v5_1 = new Vector3(points[i+1].x-
e1.z*d1, e1.y, points[i+1].z+e1.x*d1);
                                Vector3 v4_2 = new Vector3(points[i+1].x+e2.z*d2, e2.
y, points[i+1].z-e2.x*d2);
                                Vector3 v5_2 = new Vector3(points[i+1].x-
e2.z*d2, e2.y, points[i+1].z+e2.x*d2);

                                Vector3 v6 = new Vector3(points[i+2].x+e2.z*d2, e2.y,
 points[i+2].z-e2.x*d2);
                                Vector3 v7 = new Vector3(points[i+2].x-
e2.z*d2, e2.y, points[i+2].z+e2.x*d2);

                                // находим общую точку для двух последовательных крае
в
                                Vector3 v4 = Gaus(v2, v4_1, v4_2, v6);
                                Vector3 v5 = Gaus(v3, v5_1, v5_2, v7);

                                triangles = new List<int>();
                                vertices = new List<Vector3>();

                                vertices.Add(v2);
                                vertices.Add(v3);
                                vertices.Add(v4);
                                vertices.Add(v5);
                                vertices.Add(v6);
                                vertices.Add(v7);

                                triangles.Add(0);
                                triangles.Add(1);
                                triangles.Add(2);

                                triangles.Add(2);
                                triangles.Add(3);
                                triangles.Add(1);

                                triangles.Add(1);
                                triangles.Add(0);
                                triangles.Add(2);

                                triangles.Add(3);
                                triangles.Add(2);
                                triangles.Add(1);

                                triangles.Add(2);
                                triangles.Add(3);
                                triangles.Add(4);

                                triangles.Add(4);
                                triangles.Add(5);
                                triangles.Add(3);

                                triangles.Add(2);
```

```
                                triangles.Add(4);
                                triangles.Add(3);

                                triangles.Add(5);
                                triangles.Add(4);
                                triangles.Add(3);
                                CreateLine(triangles, vertices, bu.Item3, bu.Item4);
                                // отдельно обрабатываем последний случай, если дорог
а имеет четное количесво точек
                                if (points.Count % 2==0&& i == points.Count-3)
                                {
                                    a1 = (float)Math.Sqrt(Math.Pow(points[points.Coun
t-2].x-points[points.Count-3].x, 2) + Math.Pow(points[points.Count-2].z-
points[points.Count-3].z, 2));
                                    d1 = (float)bu.Item3*3/2;
                                    c1 = (float)Math.Sqrt(Math.Pow(a1, 2) + Math.Pow(
d1, 2));

                                    a2 = (float)Math.Sqrt(Math.Pow(points[points.Coun
t-1].x-points[points.Count-2].x, 2)+Math.Pow(points[points.Count-1].z-
points[points.Count-2].z, 2));
                                    d2 = (float)bu.Item3*3/2;
                                    c2 = (float)Math.Sqrt(Math.Pow(a2, 2) + Math.Pow(
d2, 2));

                                    e1 = new Vector3((points[points.Count-2].x-
points[points.Count-3].x)/a1, points[points.Count-3].y ,(points[points.Count-2].z-
points[points.Count-3].z)/a1);
                                    e2 = new Vector3((points[points.Count-1].x-
points[points.Count-2].x)/a2, points[points.Count-3].y ,(points[points.Count-1].z-
points[points.Count-2].z)/a2);

                                    v2 = new Vector3(points[points.Count-
3].x+e1.z*d1, e1.y, points[points.Count-3].z-e1.x*d1);
                                    v3 = new Vector3(points[points.Count-3].x-
e1.z*d1, e1.y, points[points.Count-3].z+e1.x*d1);

                                    v4_1 = new Vector3(points[points.Count-
2].x+e1.z*d1, e1.y, points[points.Count-2].z-e1.x*d1);
                                    v5_1 = new Vector3(points[points.Count-2].x-
e1.z*d1, e1.y, points[points.Count-2].z+e1.x*d1);

                                    v4_2 = new Vector3(points[points.Count-
2].x+e2.z*d2, e2.y, points[points.Count-2].z-e2.x*d2);
                                    v5_2 = new Vector3(points[points.Count-2].x-
e2.z*d2, e2.y, points[points.Count-2].z+e2.x*d2);

                                    v6 = new Vector3(points[points.Count-
1].x+e2.z*d2, e2.y, points[points.Count-1].z-e2.x*d2);
                                    v7 = new Vector3(points[points.Count-1].x-
e2.z*d2, e2.y, points[points.Count-1].z+e2.x*d2);

                                    v4 = Gaus(v2, v4_1, v4_2, v6);
                                    v5 = Gaus(v3, v5_1, v5_2, v7);

                                    triangles = new List<int>();
                                    vertices = new List<Vector3>();

                                    vertices.Add(v4);
                                    vertices.Add(v5);
                                    vertices.Add(v6);
                                    vertices.Add(v7);
```

```
                                triangles.Add(0);
                                triangles.Add(1);
                                triangles.Add(2);

                                triangles.Add(2);
                                triangles.Add(3);
                                triangles.Add(1);

                                triangles.Add(1);
                                triangles.Add(0);
                                triangles.Add(2);

                                triangles.Add(3);
                                triangles.Add(2);
                                triangles.Add(1);

                                CreateLine(triangles, vertices, bu.Item3, bu.Item
4);
                            }
                        }
                    }
                    // если дорога состоит из ожного участка
                    else if(points.Count==2)
                    {;
                        float a2 = (float)Math.Sqrt(Math.Pow(points[points.Count-
1].x-points[points.Count-2].x, 2)+Math.Pow(points[points.Count-1].z-
points[points.Count-2].z, 2));
                        float d2 = (float)bu.Item3*3/2;
                        float c2 = (float)Math.Sqrt(Math.Pow(a2, 2) + Math.Pow(d2
, 2));

                        Vector3 e2 = new Vector3((points[points.Count-1].x-
points[points.Count-2].x)/a2, points[points.Count-2].y ,(points[points.Count-1].z-
points[points.Count-2].z)/a2);

                        Vector3 v1 = new Vector3(points[points.Count-
2].x+e2.z*d2, e2.y, points[points.Count-2].z-e2.x*d2);
                        Vector3 v2 = new Vector3(points[points.Count-2].x-
e2.z*d2, e2.y, points[points.Count-2].z+e2.x*d2);

                        Vector3 v3 = new Vector3(points[points.Count-
1].x+e2.z*d2, e2.y, points[points.Count-1].z-e2.x*d2);
                        Vector3 v4 = new Vector3(points[points.Count-1].x-
e2.z*d2, e2.y, points[points.Count-1].z+e2.x*d2);


                        triangles = new List<int>();
                        vertices = new List<Vector3>();

                        vertices.Add(v1);
                        vertices.Add(v2);
                        vertices.Add(v3);
                        vertices.Add(v4);

                        triangles.Add(0);
                        triangles.Add(1);
                        triangles.Add(2);

                        triangles.Add(2);
                        triangles.Add(3);
```

```
                            triangles.Add(1);

                            triangles.Add(1);
                            triangles.Add(0);
                            triangles.Add(2);

                            triangles.Add(3);
                            triangles.Add(2);
                            triangles.Add(1);

                            CreateLine(triangles, vertices, bu.Item3, bu.Item4);
                            lanes = 1;
                            coor.Clear();
                        }
                    break;
                }
            }
        }
    }
    // функция для отрисовки зданий
    void CreateCube(List<int> triangles, List<Vector3> vertices, string addr_street =
"", string addr_house_number = "") {
        GameObject gObject = new GameObject();
        if(addr_street.Equals("") && addr_house_number.Equals(""))
            gObject.name ="Здание";
        else
            gObject.name = addr_house_number + " " + addr_street;
        MeshFilter mf = gObject.AddComponent<MeshFilter>();
        Mesh mesh = mf.mesh;
        Renderer rend = gObject.AddComponent<MeshRenderer>();
        MeshCollider mc = gObject.AddComponent<MeshCollider>();
        mc.sharedMesh = mf.mesh;
        mc.convex = true;
        mesh.vertices = vertices.ToArray();
        mesh.triangles = triangles.ToArray();
        rend.material = materialBuilding;
        mesh.Optimize();
        mesh.RecalculateNormals();
    }
    // функция для отрисовки домов
    void CreateLine(List<int> triangles, List<Vector3> vertices, double lines, string
street_name){
        GameObject gObject = new GameObject();
        // если у дороги нет улицы, называем дорога
        if(street_name.Equals(""))
            gObject.name = "Дорога";
        else
            gObject.name = street_name;
        MeshFilter mf = gObject.AddComponent<MeshFilter>();
        Mesh mesh = mf.mesh;
        Renderer rend = gObject.AddComponent<MeshRenderer>();
        rend.material = materialRoads;
        if (street_name.Equals("Тротуар"))
            rend.material = materialFootway;
        else
            rend.material = materialRoads;
        mesh.vertices = vertices.ToArray();
        mesh.triangles = triangles.ToArray();
        mesh.Optimize();
        mesh.RecalculateNormals();
    }
```

```
    // метод гауса
    public Vector3 Gaus(Vector3 v1, Vector3 v2, Vector3 v3, Vector3 v4){
        float s = 0;
        int n = 2;
        float[,] a = new float[n, n];
        float[] b = new float[n];
        float[] x = new float[n];
        a[0, 0] = v1.z-v2.z;
        a[0, 1] = v2.x-v1.x;
        a[1, 0] = v3.z-v4.z;
        a[1, 1] = v4.x-v3.x;
        b[0] = -v1.x*v2.z + v1.z*v2.x;
        b[1] = -v3.x*v4.z + v3.z*v4.x;
        for (int k = 0; k < n - 1; k++)
            {
                for (int i = k + 1; i < n; i++)
                {
                    for (int j = k + 1; j < n; j++)
                        a[i, j] = a[i, j] - a[k, j] * (a[i, k] / a[k, k]);
                    b[i] = b[i] - b[k] * a[i, k] / a[k, k];
                }
            }
            for (int k = n - 1; k >= 0; k--)
            {
                s = 0;
                for (int j = k + 1; j < n; j++)
                    s = s + a[k, j] * x[j];
                x[k] = (b[k] - s) / a[k, k];
            }
        return new Vector3(x[0], (float)0.1, x[1]);
    }
    // функции для выхода из приложения
    public void Update() {
        if (Input.GetKeyDown(KeyCode.Escape)) {
            QuitGame();
        }
    }
    void QuitGame () {
        Application.Quit ();
    }
}
```