Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования
«Волгоградский государственный технический университет»
Факультет электроники и вычислительной техники
Кафедра «САПР и ПК»

Лабораторная работа

«Технология OpenStreetMap и библиотека Leaflet»

Методические указания

Волгоград, 2015

1. Цель и задачи

2. Теоретические положения

2.1. Технология OpenStreetMap

2.1.1. Введение

В начале XIX века люди испытывали трудности со временем – не в том, сколько его у них было, а в том, чтобы знать, который сейчас час. Часы, конечно же, уже были придуманы, но в каждом городе было свое время, «местное время», которое синхронизировалось между часовыми башнями города, чаще всего с помощью церковного звона. Позже «Время Железной Дороги» (Railway Time), а затем и Среднее Время по Гринвичу (GMT) постепенно заменили собой все «местное время», и большинство людей сегодня даже не представляют себе, что время – это не некая универсальная вещь.

«Проблема времени» сегодня – это география, и все ищут эталонный источник. Google ежегодно тратит около 1 миллиарда долларов, поддерживая свои карты в актуальном состоянии, и, помимо этого, Google потратила 1.5 миллиарда, купив Waze[3]. И Google далеко не единственная компания, которая пытается купить все, что только можно, связанное с картами, ибо Nokia купила Navteq[4], а TomTom[5] и Tele Atlas[6] работают над слиянием. Все эти компании пытаются стать тем самым эталонным источником информации о том, что находится на земле.

Но раз существуют все эти компании, то для чего же нужен такой проект как OpenStreetMap? Ответ лежит в простой идее о том, что ни одна компания не должна обладать монополией на «место», точно так же как ни одна компания не обладала монополией на «время» в начале XIX века. «Место» является общим ресурсом, и если вы дадите всю власть над ним одной компании, вы дадите ей власть не только сообщать о том, где вы находитесь, но и силу искажать эту информацию. Проще говоря, есть три пункта: кто решает, что показывать на карте; кто решает где вы находитесь и куда направляетесь; а также вопрос личной приватности.

Все эти поставщики карт заинтересованы в том, чтобы собирать информацию о вас, включая способы, с которыми вы вполне можете быть не согласны. Как Google, так и Apple собирают информацию о вашем местоположении, когда вы пользуетесь их сервисами. Они используют эту информацию, чтобы улучшить точность карт, но Google уже объявила, что собирается определять зависимость между тем, что вы ищете и тем, куда направляетесь. С 500 миллионами телефонов на Android это гигантский объем информации о том, какие привычки отдельно взятых людей проявляются когда они просто гуляют, когда едут на работу, идут в больницу, а то и принимают участие в акции протеста.

Однозначно, нельзя игнорировать влияние на общество того факта, что вся эта куча данных находится в одних руках, сколь бы благородными они не представлялись. Компании вроде Foursquare[7] используют игровой процесс, чтобы скрыть то, что по сути является процессом сбора огромного количества данных, и даже Google уже включилась в эту игру с геймификацией в Ingress[8] – игрой, которая накладывает искусственный мир поверх нашего реального и поощряет пользователей собирать информацию о маршрутах и делать фотографии в игровом процессе.

Касательно содержимого карт, OpenStreetMap является и нейтральной и прозрачной. OpenStreetMap является подобной Вики картой, которую может редактировать кто угодно. Если магазин отсутствует на карте, его может добавить как владелец магазина, так и его посетитель. Что касается показа (рендеринга) карты, то любой человек или компания, принимающие участие в создании карты, свободен рендерить ее как ему удобно. Главная карта на OpenStreetMap.org использует ПО рендеринга и стиль со свободной лицензией, которые кто угодно может взять и подправить под свои нужды. Проще говоря, любой, кому необходимо, всегда может создать свои собственные карты, основываясь на данных OSM.

Также, несмотря на то, что самые популярные построители маршрутов для OpenStreetMap лицензированы под FLOSS[9], даже если какая-нибудь компания и выберет другую лицензию, пользователи всегда могут использовать свои построители маршрутов, и сравнив результаты построения, выявить какие-либо подтасовки, если они есть.

И наконец, пользователь волен скачать любую часть или даже всю карту OpenStreetMap для использования в оффлайне. Это означает, что можно использовать данные OpenStreetMap для навигации, вообще не передавая информацию о вашем местоположении на сторону.[1]

OpenStreetMap – некоммерческий веб-картографический проект, который создает и предоставляет свободные географические данные и возможность создавать карты всего мира кому угодно, кто это хочет. Авторы начали этот проект, потому что большинство карт, о которых вы думаете, что они свободны, на самом деле имеют юридические или технические ограничения на их использование, сдерживают людей от их использования творческим, эффективным или неожиданным образами.

Проект предоставляет карту всего мира, которую может редактировать каждый, которая создается практически с чистого листа по GPS-трекам и распространяется под свободной лицензией.

Лицензия OpenStreetMap позволяет свободно (или почти свободно) получать доступ ко всем растровым картам и всем лежащим в их основе пространственным данным, и этот проект направлен на поощрение нового и интересного использования этих данных.[10]

Стив Кост был вдохновлен успехом Википедии и решил, что принцип вики, заложенный в основу Википедии, может применяться и для веб-картографии.

Для реализации своих идей в Великобритании в июле 2004 года он создал проект OpenStreetMap. Это произошло еще до появления Google Maps, и цель OpenStreetMap – получить бесплатную карту мира, опираясь на добровольцев с GPS-устройствами, казалась трудноосуществимой. Единственным образцом для подражания в подобном массовом добровольном сборе данных была Википедия, которая к тому времени уже была серьезным конкурентом коммерческих энциклопедий.

В январе 2007 года Кембридж стал первым полностью отрисованным городом, и к июлю 2007 года, когда состоялась первая международная конференция OSM под названием The State of the Map, в проекте на то время было 9000 зарегистрированных участников. Спонсорами мероприятия были в том числе Google, Yahoo и Multimap.

В январе 2008 была реализована возможность загрузки картографических данных в GPS-устройство для использования велосипедистами. В марте двое основателей анонсировали, что они получили венчурный капитал в 2.4 млн евро для CloudMade, коммерческой компании, которая будет использовать данные OpenStreetMap. 4 марта 2008 года создан русскоязычный раздел на форуме, ставший самым популярным разделом на форуме OSM.

В январе, когда случилось катастрофическое землетрясение на Гаити, тысячи участников проекта приняли участие в составлении и актуализации карты Гаити. Это подняло популярность OpenStreetMap: был введен в широкое использование термин «crisis mapping», многие СМИ написали о проекте. Для использования в OSM свои спутниковые снимки предоставили NOAA, GeoEye, DigitalGlobe, ErosB, CNES / Spot Image, JAXA/ALOS, Google, WorldBank, в навигаторах данные с картой Гаити стали использовать американские спасатели.

После того как произошли разрушительные землетрясение и цунами в Японии и огромное количество домов просто смыло, участники OSM вне Японии по полученным свежим спутниковым снимкам (DigitalGlobe, JAXA/ALOS, MapQuest Open Aerial, Bing, Cnes / SpotImage, Aerial orthophotos from Japanese mapping authority GSI) стали отмечать последствия катастрофы, а сами японцы на местах – источники воды, работающие магазины, телефоны и прочее.[10]

2.1.2. Возможности платформы

Проект OpenStreetMap обладает большими функциональными возможностями. От свободных картографических данных, до разработки программного обеспечения на основе OSM. Стоит отметить большое количество, которые были разработаны на основе или с использованием OpenStreetMap. Среди них: картографические системы, навигационные системы, системы редактирования картографических данных и многие другие.[10]

Выделим главные особенности платформы:

- Проект охватывает всю поверхность земного шара.

- Главной целью проекта является построение не собственно карты, а базы данных, содержащей сведения о точках на земной поверхности. Таким образом, на основе собранных в рамках проекта данных можно создавать карты различного вида и другие сервисы.
- Карты OpenStreetMap двумерные, без отображения высот над уровнем моря, изолиний. Однако существуют проекты, которые отображают рельефные карты, используя данные о высотах из сторонних свободных источников.
- Возможен экспорт карт в форматы PNG, JPEG, SVG, PDF, PostScript. Также существуют проекты по экспорту данных OpenStreetMap в форматы Garmin и ГисРусса.

2.1.3. Формат данных

Значительный объем данных, загружаемых в OSM, выгружается из переносных устройств спутниковой навигации или мониторинга. Для конвертации координат из «сырого» (NMEA) или проприетарных форматов в формат GPX (основан на XML) может использоваться программа GPSBabel. Данные, собранные в формате WGS84 в виде широты/долготы, обычно показываются в проекции Меркатора[14].

В целом, всю структуру данных можно представить схематично на следующем рисунке:

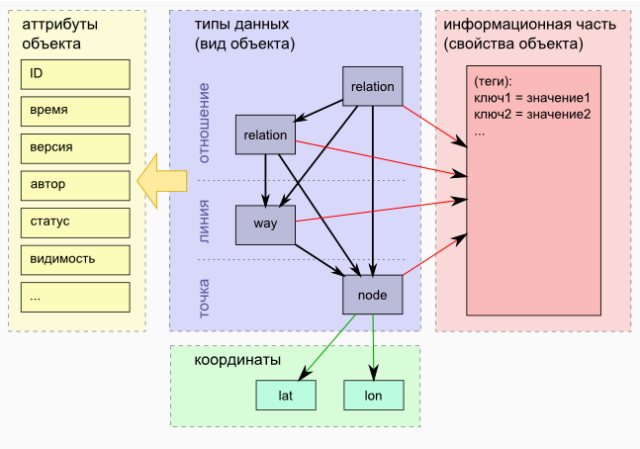

Рисунок 1 — Структура данных OpenStreetMap.

Все данные можно условно разбить на три основные группы:

- типы данных, описывающие в виде иерархической связи сам объект, как некую пространственную сущность, имеющую свой конечный результат – известные координаты всех частей объекта;
- информационная часть – это описательная характеристика объекта, не имеющая к пространственной географической структуре объекта прямого отношения (его название, физические, логические и прочие свойства);

- служебные атрибуты объекта, необходимые для организации процесса хранения и обработки информации в виде набора данных, такие как уникальный идентификатор, состояние объекта в базе, время последней правки объекта в базе и т.д.

2.1.3.1. Базовые типы географических данных в OpenStreetMap

Базовых типов по сути всего три: точка (*node*), линия (*way*) и отношение (*relation*). Сами типы *node*, *way* и *relation* называются так, потому что именно так они были придуманы изначально. Все без исключения объекты в OSM описываются этими тремя типами данных, после чего информационно наполняются комбинациями тегов. Модель данных в OSM строится на иерархической ссылочной структуре, из чего следует, что любой последующий тип данных не содержит информацию содержащуюся в предыдущих типах а образует новую сущность, ссылаясь на некое множество объектов предыдущего типа. Так же следует упомянуть, что любой объект имеет в структуре данных OSM свой идентификатор (*ID*), уникальный в пределах данного типа объектов. Именно по этому идентификатору и происходит ссылка на сам объект. Рассмотрим структуру базовых типов по порядку.

Первый тип: 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. Модель данных OSM оперирует исключительно двухмерными данными в пределах проекции WGS84. В дальнейшем мы будем считать, что координаты – это не информационная составляющая объекта точки, а неотъемлемая часть его структуры. В XML нотации, объект данного типа будет выглядеть так:

```
<node id='19' lat='58.888047127548994' lon='49.747870758186764' />
```

Одна точка с уникальным *id* равным 19 и парой координат. Координаты в OSM используются в десятичной записи, поскольку это гораздо проще обрабатывать чем форматы координат с минутами и секундами. Сама по себе точка может быть самостоятельным объектом, описывающим какой-то точечный объект (геометрический примитив) или не иметь вовсе собственной информационной составляющей, а быть частью другого объекта (линии или отношения). При этом, забегая немного вперед, отметим, что точка одновременно может быть и самостоятельным объектом, несущим уникальную информацию и быть частью другого объекта.

Второй тип данных: линия (*way*) – это совокупность указателей на объекты типа точка (*node*). Как минимум, линия состоит из одной точки, т. е. должна содержать как минимум одну ссылку на уже существующий объект типа точка. Линия из одной точки не противоречит структуре данных OSM,

но противоречит понятиям элементарной геометрии, поэтому правильная линия всегда содержит как минимум ссылки на два существующих объекта типа точка.

Правильная XML нотация объекта типа линия будет заключаться в описании всех необходимых точек, после чего следует сама запись о линии, в которой перечисляются все ее точки. В простейшем варианте это будет выглядеть так:

```
<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>
```

Порядок перечисления точек в линии важен, он характеризует последовательность точек в линии и направление самой линии, т. е. у линии всегда есть начало и конец, даже если она замкнутая (в этом случае они просто совпадают). В данном примере мы вначале описали две точки, задав их координаты, а потом описали линию, сославшись на id этих точек. Одна точка может входить в любое количество объектов линий, при этом должна быть описана только один раз, т. е. точка может быть общей для двух линий, в этом случае ссылка на нее содержится в обоих линиях. Таким образом строится цельный граф объектов (чаще всего дорожный граф для расчета роутинга), который представляет из себя совокупность объектов (линий), имеющих связь через их общие члены (точки).

Если мы хотим создать еще одну линию из уже существующих точек 19 и 23, то мы опишем ее так:

```
<way id='48'>
  <nd ref='19' />
  <nd ref='23' />
</way>
```

В нашем случае точки 19 и 23 уже описаны выше, а точка 23 вошла в состав двух линий 24 и 48 и стала общей для них.

Наши линии 24 и 48 можно графически представить в проекции меркартора следующим образом:

Подписи на рисунке – *id* объектов: красные у точек, черные у линий; стрелкой указано направление линии, т. е. обе линии заканчиваются на точке 23.

Следующий тип данных: отношения (*relation*). По сути все объекты кроме точки – уже отношения, однако линии выделены в отдельный тип данных как наиболее распространенные, описывающие основные геометрические

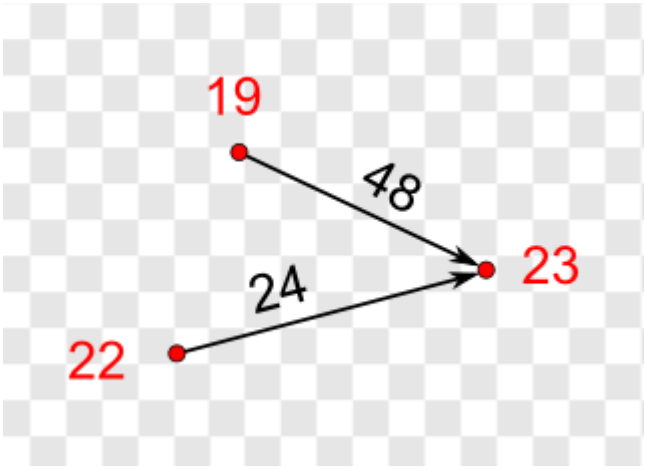

Рисунок 2 — Две линии

примитивы: линии, полилинии и полигоны. Для всех более сложных геометрических объектов, а так же для объектов являющихся не чисто геометрическими, а логическими (коллекции, списки, иерархии взаимосвязей) предназначен универсальный тип данных – отношения.

В целом, описание отношения отличается от линии тем, что линия – это всегда совокупность точек, а отношение – это совокупность любых объектов, как точек и линий, так и других отношений. Следовательно в отношениях указывается не только *id* объекта, но и его тип. В самом минимальном варианте отношение может содержать ссылку только на один объект. Оперируя объектами описанными в примерах выше, можно написать отношение:

```
<relation id='31'>
  <member type='way' ref='24' />
  <member type='node' ref='19' />
</relation>
```

Это фиктивное абстрактное отношение, описывающее, что в него входит два объекта (члены отношения) – точка 24 и линия 19 и более не несущее никакой другой информации. В реальном случае у отношения должен быть указан тип, как тег (информационная составляющая) самого объекта отношения, а у членов отношения должны быть указаны роли в ссылках на объекты.

Ниже приведен пример самого распространенного отношения типа «мультиполигон», которое описывает один замкнутый внешний полигон из трех точек с вырезанным из него замкнутым полигоном тоже из трех точек меньшего размера. О геометрических примитивах (замкнутых и не замкнутых полигонах) и тегах объектов речь пойдет далее, а пока следует обратить внимание на параметры ролей (*role*) у объектов отношения и наличие тега описывающего тип.

```
<node id='1218' lat='58.870941122729505' lon='49.758021019729554' />
<node id='1216' lat='58.8704000725183' lon='49.74703196841415' />
<node id='1215' lat='58.879055860772034' lon='49.74964840920353' />
```

```
<node id='1209' lat='58.86471853452049' lon='49.780522410518245' />
<node id='1207' lat='58.863365649894774' lon='49.72453057762546' />
<node id='1206' lat='58.892035483174' lon='49.74755525657201' />
<way id='1217'>
  <nd ref='1215' />
  <nd ref='1216' />
  <nd ref='1218' />
  <nd ref='1215' />
</way>
<way id='1208'>
  <nd ref='1206' />
  <nd ref='1207' />
  <nd ref='1209' />
  <nd ref='1206' />
</way>
<relation id='1221'>
  <member type='way' ref='1208' role='outer' />
  <member type='way' ref='1217' role='inner' />
  <tag k='type' v='multipolygon' />
</relation>
```

Роль outer говорит о том, что данный объект будет наружным контуром графического объекта, а роль *inner* сообщает о том, что пространство внутри этого объекта необходимо исключить из площади результирующего объекта. Графически наш мультиполигон будет выглядеть так:

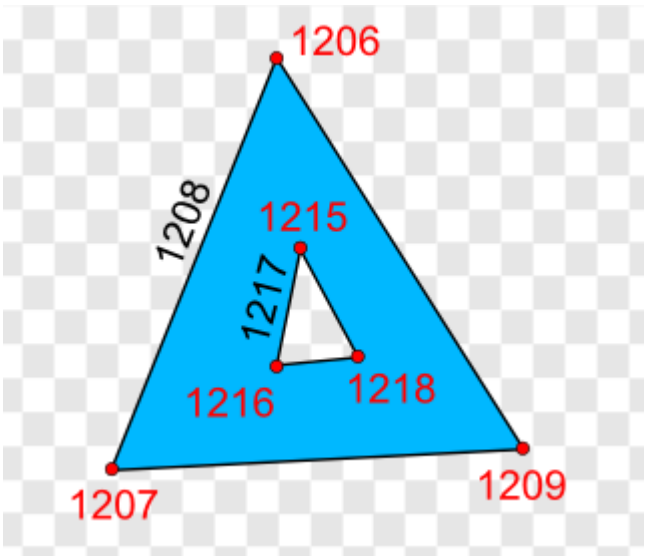

Рисунок 3 — Мультиполигон

Так же как и у линии, у отношения порядок перечисления членов играет роль и учитывается при использовании этого отношения. Например отношением может быть не геометрическая фигура, а маршрут общественного транспорта (логическая схема), тогда в него входят последовательно участки дорог, по которым будет двигаться автобус и список точек – остановки на которых он останавливается, следовательно порядок включения дорог в отношение по-

казывает последовательность прохождения маршрута, а порядок остановок – последовательность их посещения.

Объект *relation* может быть членом другого *relation*, при этом уровень вложенности и иерархия вверх ничем не ограничивается. Структурное ограничение заключается в том, что объект *relation* не может быть членом самого себя, т. е. содержать ссылок на самого себя. Рекурсия в структуре типов данных в OSM недопустима, хотя конечно ничто не мешает создать такой объект и даже вполне успешно воткнуть его в базу данных.

Определив три базовых типа объектов надо ввести понятие начального и конечного объекта. Начальным объектом является любой объект, который входит в состав любого другого объекта, т. е. является дочерним по отношению как минимум к одному объекту, но сам при этом ни включает ни одного другого объекта. В случае OSM это всегда точка. Либо другое его определение – начальный объект несет в своей структуре (не в информационной части!) только географические координаты и не содержит ссылок на другие объекты.

Конечным объектом является максимальный по иерархии родительский объект, который не является дочерним по отношении ни к одному другому объекту, т. е. не входит в состав ни одного другого объекта. Это может быть любой из трех перечисленных типов: точки, линии, отношения. Отдельный точечный объект, никуда не входящий, состоящий из одного объекта типа точка, не является начальным объектом, поскольку не является началом иерархии объектов, но является конечным объектом, поскольку на нем заканчивается пространственное описание объекта.[2]

2.1.3.2. Информационная схема объектов

Тип объекта описывает географические (пространственные) свойства объекта, но ничего не говорит о свойствах самого объекта, его характеристиках, назначении и прочем. Для это существует информационная часть структуры данных OSM, основанная на принципах тегирования объектов, т. е. назначении им определенных меток и указанием свойств этих меток. Теги задаются в виде пары *ключ* = *значение*, что в нотации XML для нашей линии 24 выглядит так:

```
<way id='24'>
  <nd ref='22' />
  <nd ref='23' />
  <tag k='highway' v='primary' />
</way>
```

В данном случае мы добавили свойство нашей линии, а именно указали тег *highway* со значением *primary*, что в принятой схеме тегирования обозначает что наша линия является основной дорогой (дорога класса ниже магистрали, но выше второстепенной). Тегов у любого объекта может сколько угодно много, что позволяет задать все его основные свойства и описать все

второстепенные параметры, а так же в произвольной форме дополнить объект любой информацией. Сама схема тегирования в OSM является одновременно самым главным ее архитектурным преимуществом, поскольку позволяет описать фактически любые свойства объекта, т. к. реально никто не ограничивает вас в выборе новых тегов для новых свойств объектов; и одновременно самым больным ее местом, поскольку любая свобода в выборе способов обозначения всегда порождает религиозные войны различных групп пользователей, так и не сошедшихся во мнении, как обозначать тот или иной спорный объект.

Если мы немного расширим информационное описание наших двух линий 24 и 48 из первой части, то можем получить что-то вроде:

```
<node id='23' lat='58.87753645355202' lon='49.79290110146539'>
  <tag k='highway' v='traffic_signals' />
</node>
<node id='22' lat='58.87456113991739' lon='49.73690926857261' />
<node id='19' lat='58.89362576054878' lon='49.7492065402827' />
<way id='48'>
  <nd ref='19' />
  <nd ref='23' />
  <tag k='embankment' v='yes' />
  <tag k='highway' v='secondary' />
  <tag k='incline' v='up' />
  <tag k='lanes' v='2' />
  <tag k='maxspeed' v='60' />
  <tag k='name' v='улица Пожарского' />
</way>
<way id='24'>
  <nd ref='22' />
  <nd ref='23' />
  <tag k='highway' v='primary' />
  <tag k='lanes' v='6' />
  <tag k='lit' v='yes' />
  <tag k='name' v='проспект Минина' />
  <tag k='oneway' v='yes' />
  <tag k='ref' v='M84' />
</way>
```

Линия 48 у нас стала «улицей Пожарского», с ограничением скорости в 60 км/час, количеством полос равным двум, имеющей положительный градиент уклона в сторону от точки 19 к точке 23, являющейся второстепенной дорогой и поднятой относительно уровня земли на насыпь. А линия 24 у нас основная дорога (классом выше чем secondary) с 6-тю полосами движения, имеющей стационарное освещение и одностороннее движение разрешенное в направлении от точки 22 в сторону точки 23, носит название «проспект Минина» и является частью федеральной трассы М84. Обе дороги имеют общую точку 23, которая является перекрестком со светофором. [2]

2.1.3.3. Геометрические примитивы

Основная или как минимум наиболее частая задача для любых географических пространственных данных – получение графического представления объектов, описанных этими данными. Проще говоря рендеринг самих карт, схем, планов. Сами алгоритмы, правила, стили и методы рендеринга карт – это задача уже прикладного софта, но все сводится к отрисовке основных геометрических примитивов, которые получаются из объектов трех типов данных, перечисленных в 2.1.3.1..

Итак, из чего формируются основные геометрические примитивы.

Точка (*point*) – это один объект типа *node*. Положению ее в заданной проекции на карте соответствует ее пространственное положение в географических координатах. Одна пара координат *lat/lon* транслируется в координаты x/y карты с учетом проекции.

Линия (*line*) – это, как мы уже сказали, кратчайшее растояние между двумя точками, соответствует объекту типа *way*, содержащему два объекта node. Любые два *node*, поскольку мы оперируем плоским пространством, а следовательно расстояние между любыми двумя точками всегда будет прямой линией.

Полилиния (*polyline*) – это связанная последовательность сегментов, где каждый сегмент представляет из себя одну линию, связанную своим концом с началом следующего сегмента. Вся последовательность является единым цельным объектом. Соответствует объекту *way* содержащему три и более *node*. Полилиния может быть объектом *relation*, содержащим последовательно включенные объекты *way*, где каждый следующий объект *way* начинается с объекта *node*, которым закончился предыдущий *way*. Полилиния может быть либо объектом *way*, содержащим *node*, либо *relation*, содержащим *way*, т. е. не может быть *relation* содержащим и *way* и *node* одновременно, однако может быть объектом *relation*, содержащим одновременно *way* и другие *relation*, содержащие только объекты *way*.

Виды полилиний:

Полигон (*polygon*) – это замкнутая полилиния, у которой последняя точка совпадает с первой. В типах данных OSM соответствует объекту *way* с несколькими (три и более) объектами *node*, при этом количество членов объекта *way* всегда больше на единицу, т. к. первый объект *node* повторяется дважды: в начале и в конце списка. Так же полигон может быть собран в виде *relation*, в который последовательно включаются *way*, образующие совместно замкнутый контур, т. е. началу каждого объекта *way* соответствует конец одного другого объекта *way*. В отличии от полигона в *way*, полигон в *relation* не дублирует последний объект в перечислении с первого, поскольку для *way* это необходимо в связи с тем, что он ссылается на объекты *node* и только по факту дублирования *node* как первого и последнего элемента списка членов можно судить о том что это замкнутый полигон, а не линейный объект полили-

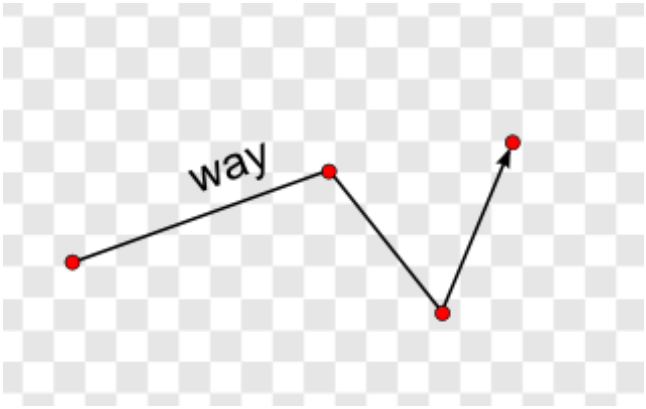

Рисунок 4 — polyline = way(node1,node2,node3,node4)

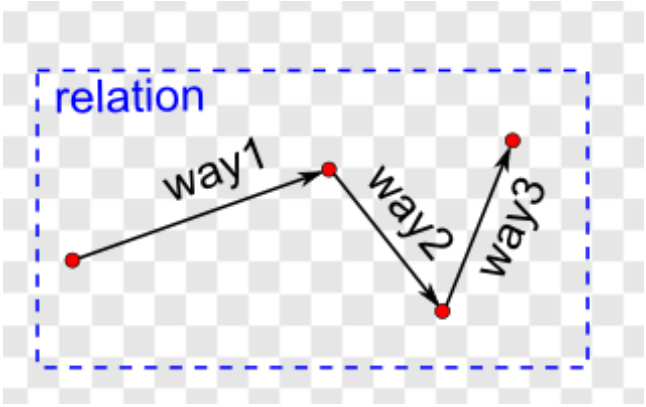

Рисунок 5 — polyline = relation(way1(node1,node2), way2(node2,node3), way3(node3,node4))

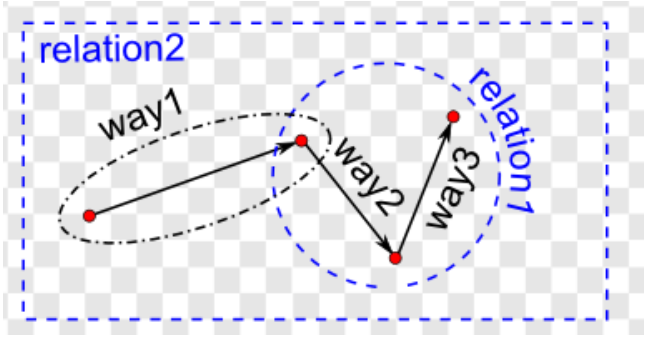

Рисунок 6 — polyline = relation2(way1(node1,node2), relation1(way2(node2,node3), way3(node3,node4)))

ния. В полигоне собранном в виде *relation* последний объект *node* последнего включенного объекта *way* или *relation* содержащего *way* соответствует первому объекту node первого включенного *way*.

В случае полигона описываемого в виде *relation*, обязательно указывается тег *type* = *multipolygon* на самом объекте *relation*. Таким образом мы определяем, что речь идет о площадном геометрическом, а не о линейном объекте.

Виды полигонов:

Составные объекты – это объекты которые невозможно описать одним примитивом *way*, всегда строятся на базе объектов *relation* у которого *type* = *multipolygon*. Полигоны и полилинии описанные в виде *relation* всегда можно упростить до одного объекта *way*, в то время как составной объект в самом

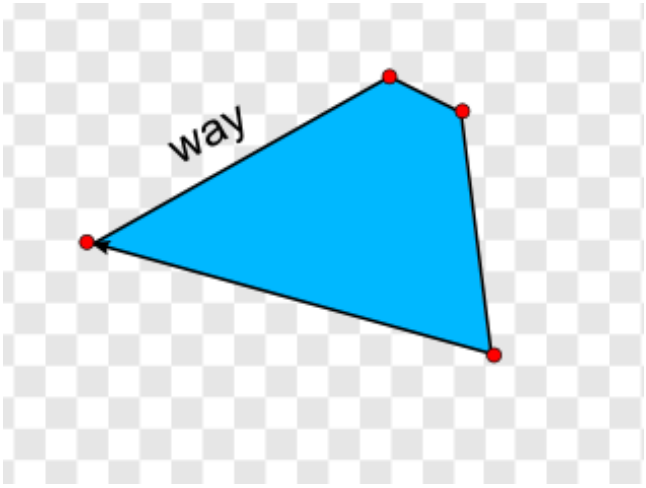

Рисунок 7 — polygon = way(node1,node2,node3,node4,node1)

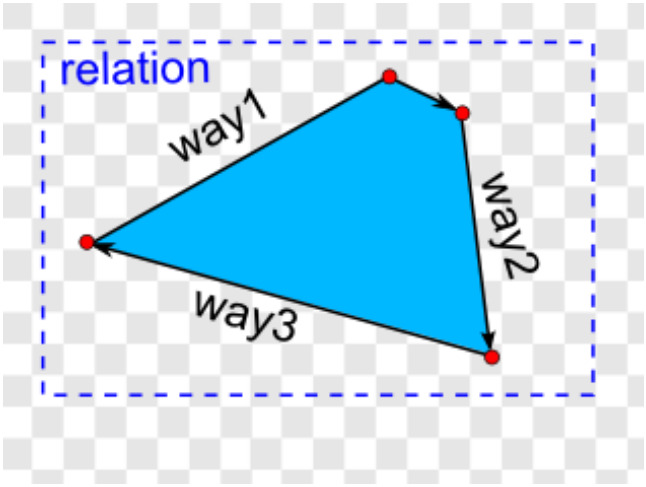

Рисунок 8 — polygon = relation(way1(node1,node2,node3), way2(node3,node4), way3(node4,node1))

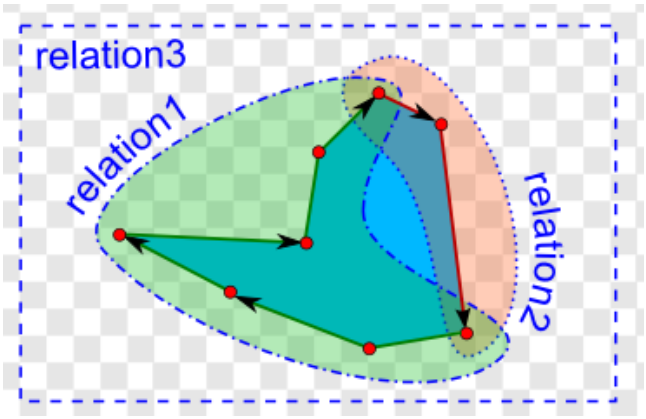

Рисунок 9 — polygon = relation3(relation1(way1(node1,node2,node3), way2(node3,node4), way3(node4,node5), way4(node5,node6,node7)), relation2(way5(node7,node8), way6(node8,node1)))

упрощенном случае дает минимум два объекта *way*. Например это площадная фигура (полигон) из которой математически вычтена другая фигура (полигон меньшего размера). Пример мультиполигона, являющегося составным объектом, с иллюстрацией был приведен на рисунке 3, в описании объектов *relation*.

Мультиполигоны для составных объектов, так же как и полигоны и полилинии могут собираться из простых way или из других *relation*, состоящих из любого количества *way*. Для такого мультиполигона обязательно указание роли для каждого входящего члена, будь то *way* или *relation*. Как минимум должен быть один член с ролью *outer*. Именно объект или объекты с этой ролью задают главный (внешний) геометрический контур результирующего объекта. Объектов с ролью *inner* может и не быть в частном случае, но тогда такой мультиполигон – это обычный полигон, просто описанный избыточно. Объекты с ролью *inner* указывают какие участки, находящиеся внутри внешнего контура, не являются частью результирующей фигуры. Например это поляна в лесу или внутренний двор дома, ограниченный со всех сторон стенами этого дома.

Все объекты с одной ролью в одном мультиполигоне должны собраться в один или несколько замкнутых не пересекающихся по границам контуров.[1]

2.2. Библиотека Leaflet

2.2.1. Введение

Leaflet является современный открытым проектом написанная на JavaScript библиотека для отображения мобильных интерактивных карт. Она разработана Владимиром Агафонкиным и командой. При весе всего около 33 КБ JS, он имеет все функции, которые могут понадобиться большинству разработчиков для отображения интернет-карт.

Leaflet задумана как библиотека, одинаково хорошо работающая и на десктопных браузерах, и на мобильных устройствах (iPhone/iPad, Android) – очень быстрая, легковесная, с простым API, красивым и понятным ООП-кодом. В отличии от OpenLayers[13] авторы не пытаемся впихнуть в нее все фичи, о которых только можно помыслить, раздувая код до немыслимых размеров – только самое основное, минимальный набор, который удовлетворяет нужды 99% применений карт в онлайне (тайлы, маркеры, векторы, попапы), но реализовывая их максимально лучшим образом.

Библиотека Leaflet разработана с упором на простоту и производительность, а также для удобного использования где угодно. Она работает эффективно на всех основных настольных и мобильных платформ из коробки, пользуясь HTML5 и CSS3 на современных браузерах. Функциональность библиотеки может быть расширена с помощью огромного количества плагинов, имеющие хорошо документированный и простой API, красивый и легко читаемый исходных код.[12]

2.2.2. Возможности библиотеки

Доступные слои:

- Слой тайлов («плиток» карты)

- Маркеры
- Всплывающие элементы
- Векторные элементы: линии, многоугольники (полигоны), круги, круглые маркеры
- GeoJSON слои
- Накладываемые изображения / слои
- Слой WMS
- Группы слоев

Поддержка браузеров:

- Firefox 3.6+
- Chrome
- Safari 5+
- Opera 11.11+
- IE 7–9
- IE 6 (не отлично, но достаточно)
- Safari for iOS 3/4/5+
- WebKit for Android 2.2+, 3.1+, 4+

Все доступные возможности библиотеки Leaflet описаны в разделе *Features* по ссылке http://leafletjs.com/features.html. Также на официальном сайте можно получить подробную справку по работе с библиотекой (API, плагины, примеры) http://leafletjs.com/. Рассмотрим в данном методическом пособии несколько важных моментов.

3. Пример выполнения лабораторной работы

3.1. Задача

3.1.1. Подготовка HTML-страницы

Для начала в заголовке html-файла подключаем библиотеку Leaflet:

```
<script src="http://cdn.leafletjs.com/leaflet-0.7/leaflet.js"></script>
```

Для нормального отображения карты и управляющих элементов необходимо подключить таблицу стилей:

```
<link rel="stylesheet" href="http://cdn.leafletjs.com/leaflet-0.7/leaflet.css" />
```

В качестве контейнера карты создадим в теле html-страницы пустой блок с идентификатором *map*:

```
<div id="map"></div>
```

Основной код программы будем создавать отдельно в файле *main.js*, подключим его:

```
<script src="main.js"></script>
```

Окончательно, получаем используемую html-страницу:

```
<html>
  <head>
    <title>Example</title>
    <link rel="stylesheet" href="http://cdn.leafletjs.com/leaflet-0.7/leaflet.css" />
    <script src="http://cdn.leafletjs.com/leaflet-0.7/leaflet.js"></script>
    <style>
      body { margin: 0; padding: 0; }
      #map { position: absolute; left: 0; top: 0; bottom: 0; width: 100%; }
    </style>
  </head>
  <body>
    <div id="map"></div>
    <script src="main.js"></script>
  </body>
</html>
```

3.2. Создание карты

Перейдем непосредственно к созданию фигуры на карте. Для начала нам, собственно, необходимо отрисовать карту в нашем подготовленном блоке map. Это делается простой командой:

```
var map = L.map('map');
```

где 'map' – идентификатор нашего блока.

Чтобы загружалась какая-либо конкретная область карты необходимо задать начальные координаты и масштаб (метод setView([широта, долгота], масштаб)):

```
map.setView([48.7819, 44.7777], 14);
```

или

```
var map = L.map('map').setView([48.7819, 44.7777], 14);
```

После этого создадим слой, содержащий изображения фрагментов карты:

```
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(map);
```

Метод addTo(map) добавляет наш слой на карту map.

Результат:

Рисунок 10 — Отображение карты

3.3. Маркеры, круги и всплывающие сообщения

Допустим, нам дан набор данных из 10 точек, который нужно отобразить на карте:

```
data = [[48.79481, 44.74777],
        [48.78028, 44.75485],
        [48.76463, 44.78363],
        [48.76719, 44.78686],
        [48.77923, 44.81202],
        [48.78289, 44.80787],
        [48.79229, 44.77988],
        [48.78893, 44.76422],
        [48.79407, 44.75757],
        [48.79667, 44.75634]];
```

Отобразить точку можно различными способами, здесь мы рассмотрим два основных: круг и маркер.

Обычный маркер создается командой L.marker([широта, долгота]). Создадим маркеры для всех точек:

```
for (i in data) {
    L.marker(data[i]).addTo(map);
}
```

Результат можно увидеть на рисунке 11.

Для того, чтобы при нажатии на маркер всплывало сообщение с его координатами, привяжем к маркеру так называемый *popup*:

```
for (i in data) {
    marker = L.marker(data[i]).addTo(map);
    marker.bindPopup(data[i][0] + ', ' + data[i][1]);
}
```

Или:

```
for (i in data) {
    L.marker(data[i]).addTo(map).bindPopup(data[i][0] + ', ' + data[i][1]);
}
```

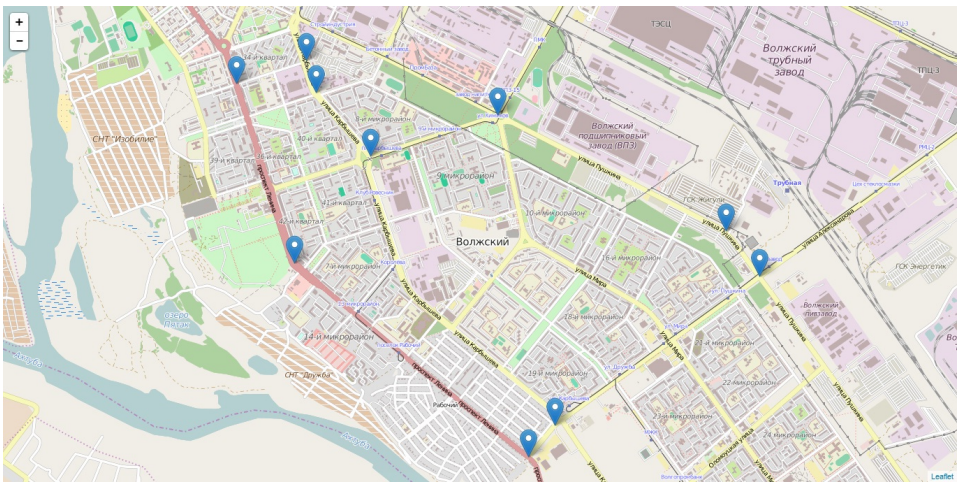

Рисунок 11 — Создание маркеров

Команда bindPopup(html_text) позволяет при нажатии на маркер отображать текст, оформленный с помощью html-тегов, записанный в переменной html_text. Стандартно, на карте одновременно может отображаться не больше одного сообщения, поэтому при вызове нового сообщения старое автоматически закрывается. Таким образом, после нажатия на маркер получаем следующую картину:

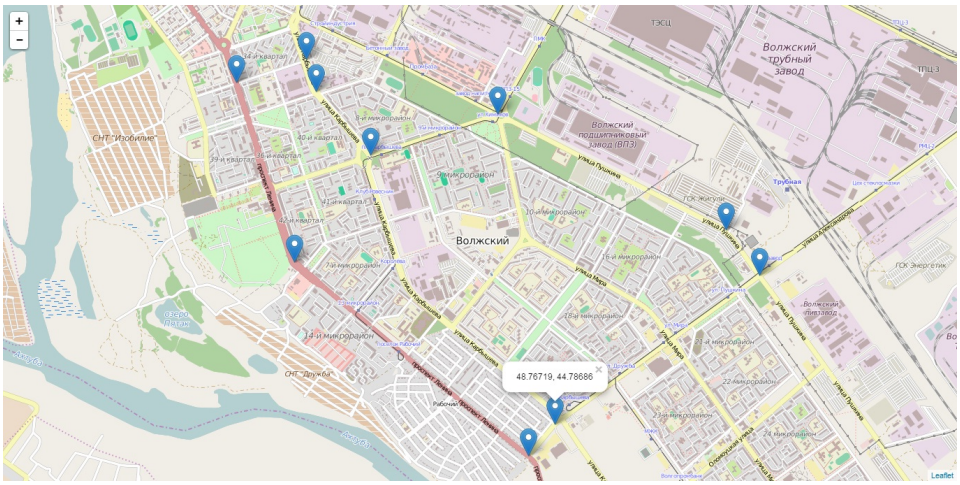

Рисунок 12 — Привязка сообщений к маркерам

Теперь вместо маркеров отобразим точки кругами. Для создания круга используется команда L.circle([широта, долгота], радиус). Отобразим круги для всех точек:

```
for (i in data) {
    L.circle(data[i], 50).addTo(map);
}
```

Применяя уже знакомый метод bindPopup, привяжем сообщение с координатами на нажатие круга:

```
for (i in data) {
    L.circle(data[i], 50).addTo(map).bindPopup(data[i][0] + ', ' + data[i][1]);
}
```

Полученный результат:

Рисунок 13 — Создание кругов

Привяжем теперь всплывающее сообщение о координатах к нажатию на карту в любом месте. Для этого нам придется отлавливать событие клика и вызывать функцию отображения всплывающего сообщения.

Чтобы при нажатии левой кнопкой мыши на карту выполнялась функция onClick необходимо вызвать метод on карты map:

```
map.on('click', onClick);
```

Здесь 'click' – это название события. Теперь при клике на карту будет вызвана функция onClick, которой передадут так называемый *event object* в качестве аргумента. В нашем случае это будет *MouseEvent*.

Теперь напишем функцию вызова всплывающего сообщения. Для начала вне функции определим объект типа L.popup, который и будет являться нашим сообщением. В функции onClick мы тогда будем просто менять координаты вызова и текст сообщения.

```
var popup = L.popup();
function onClick(e) {
  lat = e.latlng.lat.toFixed(5);
  lng = e.latlng.lng.toFixed(5);
  popup.setLatLng(e.latlng)
       .setContent('<b>' + lat + ', ' + lng + '</b>')
       .openOn(map);
}
```

Разберем тело функции построчно. Первыми двумя строчками мы задаем переменным lat и lng некоторые значения. e.latlng – это объект, содержащий в себе координаты точки, в нашем случае – точки клика пользователя на карте; e.latlng.lat – широта, e.latlng.lng – долгота. Таким образом, переменные lat и lng хранят в себе координаты точки клика на карте с 5 знаками после запятой. Следующей строчкой мы задаем координаты нашего всплывающего сообщения методом setLatLng. Затем задаем *контент* – текст сообщения. Мы выводим координаты клика, причем полужирного начертания. Последней строчкой, как не трудно догадаться, мы показываем наше сообщение на карте.

Результат:

Рисунок 14 — Всплывающее сообщение при клике на карту

3.4. Ломаная и область

При необходимости выделить область на карте или проложить маршрут используют объекты *polygon* и *polyline*.

Для создания ломаной по нашим точкам, необходимо сделать следующее:

```
L.polyline(data).addTo(map);
```

Теперь у нас есть ломаная, соединяющая по порядку точки из нашего набора. Немного *приукрасим* ее: изменим цвет и толщину. Для этого после набора данных data функции надо передать объект свойств ломаной; за цвет отвечает свойство color, за толщину – weight:

```
L.polyline(data, {color: 'black', weight: 3}).addTo(map);
```

Таким же образом можно менять стиль кругов, изображающих наши точки:

```
L.circle(data[i], 50, {color: '#f00'}).addTo(map);
```

После этого круги у нас будут красного цвета, а линия – черная и узкая (рис. 15).

Для выделения области (полигона) необходимо создать объект *polygon*:

```
L.polygon(data, {color: 'green'}).addTo(map);
```

После этого область, задаваемая нашими точками, окрасится в зеленый цвет. Если вместо нашего массива координат data передать массив массивов точек [data, data1, data2, ...], то перекрываемые области будут вырезаться из полигона, а неперекрываемые – добавляться.

Привяжем к нашему полигону какое-нибудь сообщение:

```
L.polygon(data, {color: 'green'}).addTo(map)
  .bindPopup('<i>I\'m so green</i>');
```

Рисунок 15 — Создание линии

Результат ожидаем:

Рисунок 16 — Создание полигона

Полный текст программы (логическая переменная markers отвечает за отображение точек в виде маркеров вместо кругов, логическая переменная polyline отвечает за отрисовку ломаной вместо полигона):

```
var map = L.map('map').setView([48.7819, 44.7777], 14);
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png').addTo(map);
map.on('click', onClick);

var popup = L.popup();
function onClick(e) {
  lat = e.latlng.lat.toFixed(5);
  lng = e.latlng.lng.toFixed(5);
  popup.setLatLng(e.latlng)
       .setContent('<b>' + lat + ', ' + lng + '</b>')
       .openOn(map);
}

data = [[48.79481, 44.74777],
        [48.78028, 44.75485],
        [48.76463, 44.78363],
        [48.76719, 44.78686],
        [48.77923, 44.81202],
```

```
        [48.78289, 44.80787],
        [48.79229, 44.77988],
        [48.78893, 44.76422],
        [48.79407, 44.75757],
        [48.79667, 44.75634]];

markers = false;
for (i in data) {
  if (markers) marker = L.marker(data[i]);
  else marker = L.circle(data[i], 50, {color: '#f00'}).addTo(map);
  marker.addTo(map).bindPopup(data[i][0] + ', ' + data[i][1]);
}

polyline = false;
if (polyline) {
  L.polyline(data, {color: 'black', weight: 3}).addTo(map);
} else {
  L.polygon(data, {color: 'green'}).addTo(map)
    .bindPopup('<i>I\'m so green</i>');
}
```

4. Задания на выполнение лабораторной работы

1. Написать js-скрипт, который случайно расставляет точки на карте в заданном диапазоне координат, задавая каждой точке уникальный 6-значный hex-идентификатор. Каждую точку отобразить на карте кружком, цвет которого является значением идентификатора.

2. Написать функцию рисования фигуры:

   - треугольник,
   - звезда,
   - ломаная,
   - «бублик», то есть многоугольник с вырезанной областью внутри.

   Для ломаной и внешней границы многоугольника используйте координаты точек, полученные в 1-ом задании.

3. Изобразить точки из 1-го задания маркерами, по нажатию на который выводится сообщение, содержащее идентификатор точки и ее координаты.

5. Контрольные вопросы

1. Назовите базовые типы географических данных в OpenStreetMap. Кратко расскажите о них. Для чего они нужны?
2. Чем или почему проект OpenStreetMap лучше многих других?
3. Зачем нужен тип данных отношение (*relation*)?
4. Зачем нужно теггирование объектов? Для чего оно используется?
5. Преимущества и недостатки библиотеки Leaflet.

6. Литература

Список литературы

[1] Зачем миру нужен OpenStreetMap http://habrahabr.ru/post/217291/

[2] Структура данных проекта OpenStreetMap, заглянем под юбку сервису http://habrahabr.ru/post/146503/

[3] Приложение дорожной навигации https://www.waze.com/

[4] Ведущий в мире производитель цифровых карт и других данных для геоинформационных систем www.navteq.com

[5] Нидерландская компания, поставщик устройств для автомобильной и персональной GPS-навигации www.tomtom.com

[6] Глобальным поставщик навигационных карт для автомобильных навигационных систем, геоинформационных и локально-информационных систем www.tomtom.com

[7] Социальная сеть с функцией геопозиционирования, предназначенная в основном для работы с мобильными устройствами. https://foursquare.com/

[8] Многопользовательская онлайн-игра, созданная Niantic Labs в Google для Android- и iOS-устройств https://www.ingress.com/

[9] The Free-Libre / Open Source Software (FLOSS) License http://www.dwheeler.com/essays/floss-license-slide.html

[10] https://ru.wikipedia.org/wiki/OpenStreetMap

[11] http://wiki.openstreetmap.org/wiki/RU:Map_Features

[12] Open-Source библиотека предназначенная для отображения карт на вебсайтах http://leafletjs.com/

[13] Библиотека с открытым исходным кодом, написанная на JavaScript, предназначенная для создания карт http://openlayers.org/

[14] Проекция Меркатора https://ru.wikipedia.org/wiki/Проекция_Меркатора