**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

Федеральное государственное автономное   
образовательное учреждение

Высшего образования

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**

**ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Инженерная школа информационных технологий и робототехники

Направление: «Программная инженерия»

Отчет по УИРС

**«Вывод географических данных в соответствии со стандартом SLD на примере мобильного приложения»**

Выполнил:

Студент группы 8К61 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Исламов Е.Р.

Принял:

Доцент ОИТ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Савельев А.О.

Томск 2020

**Содержание**

[Введение 3](#_Toc30031259)

[Структура геоданных 5](#_Toc30031260)

[Стилизация геоданных. Формат SLD 7](#_Toc30031261)

[Проектирование и разработка логики стилизации геоданных для мобильного приложения 10](#_Toc30031262)

[Оптимизация скорости отрисовки карт 16](#_Toc30031263)

[Заключение 19](#_Toc30031264)

[Список литературы 20](#_Toc30031265)

[Приложение 21](#_Toc30031266)

# Введение

Геоинформационные системы и сервисы получили широкое распространение в современном мире. Такие системы предоставляют возможность оперативного мониторинга и анализа геоданных и востребованы как у рядовых людей, так и при решении задач бизнеса (например, в сфере нефтегазодобычи) [1].

Современный человек регулярно использует картографические сервисы, такие как «2ГИС», «Google Maps» и другие. Обычно такие сервисы обладают такими полезными инструментами, как построение маршрута, мониторинг трафика, поиск географических объектов и так далее. Благодаря таким инструментам, как геопозиционирование (даже в условиях отсутствия интернета), пользование картографическими сервисами стало доступным даже неподготовленному человеку.

Как правило, картографические сервисы можно разделить на две категории:

1. Онлайн-карты
2. Оффлайн-карты

Онлайн-карты предоставляют актуальную географическую информацию посредством сети интернет. Схему взаимодействия с такими сервисами можно описать следующим образом: клиентское приложение устанавливает соединение с сервером геоданных и при каждом перемещении текущего окна (в дальнейшем экстент) окна, клиентское приложение догружает с сервера необходимые участки карты (в дальнейшем тайлы) и рисует полученное изображение на карте.

При этом, можно заметить, что в такой схеме взаимодействия сама логика отрисовки географической информации на тайлах вынесена на сторону сервера, поэтому от клиентской машины не требуется высокая производительность в случае, даже если набор геоданных представляет собой набор сложных объектов. Отрисовка таких карт будет осуществляться быстро, так как она обычно проводится в мощных вычислительных центрах.

В случае оффлайн-карт, основная логика приложения не может быть вынесена на серверную часть, так как серверной части либо может вообще не быть, либо она может быть предназначена для вывода информации с коротким сроком актуальности, такой как состояние траффика на дорогах.

Если требуется отображать сложные данные в большом объеме, то с этим может быть связано несколько ограничений:

1. Клиентское устройство должно обладать достаточным запасом дискового пространства, чтобы хранить геоданные. Если необходимо отображать данные достаточно высокой точности, размер может исчисляться сотнями мегабайт.
2. Клиентское устройство должно обладать достаточно высокой вычислительной мощностью, либо алгоритмы отрисовки должны быть достаточно оптимизированы.

Под сложными данными в этой работе понимаются такие данные, которые могут быть отрисованы с разными стилями в даже рамках одного слоя. Более подробно об этом в главах «Структура геоданных» и «Формат стилизации геоданных SLD».

Данная работа посвящена оффлайн-картам, в частности решению проблемы вывода достаточно больших объемов географической информации с поддержкой произвольной стилизации в рамках одного слоя даже на разных масштабах.

Вывод осуществляется в мобильном приложении. Для непосредственной отрисовки используется библиотека ThinkGeo. Данная библиотека поддерживает вывод сложных данных с помощью фильтров, однако для решения задачи требовалась поддержка стилизации в открытом формате SLD, предоставляющем очень большие возможности для кастомизации карт.

# Структура геоданных

Геоданные не отделяют от обычных данных. Это обычные данные, только с координатами в какой-либо проекции. Операции на таких объектах поддерживают все популярные системы работы с данными, от баз (PostGIS, MySQL, Oracle) до языка R и Python-фреймворков, таких как pandas [2].

Как правило, геоданные разбиты на несколько слоев (файлов) и независимы друг от друга. Разбиение может проводиться, например по типу геометрии, по видовой принадлежности и т.д. Поэтому, при формировании итогового изображения, происходит слияние данных всех слоев в заданном порядке (данные могут накладываться друг на друга). Примерами слоев могут являться «Автомобильные дороги», «Населенные пункты», «Точки заправки» и другие.

Принципиально можно выделить два формата геоданных:

1. Векторные
2. Растровые

Растровые форматы данных используют в тех случаях, когда информация носит непрерывный характер. Растровые форматы (или модели) данных – это матрицы пикселей, где в каждой ячейке хранится одно или несколько числовых значений. Зачастую растровыми данными оказываются либо снимки из космоса (т.н. данные дистанционного зондирования Земли - ДЗЗ), либо результаты таких вычислений, как интерполяция. Распространенные форматы растровых данных – \*. geotiff, \*.tif, \*.jpeg, \*.ecw и так далее [3].

Данная работа сосредоточена на работе с векторными форматами. В векторном формате можно хранить такие геометрические объекты, как точки, линии (полилинии) или полигоны. Каждому геометрическому объекту может быть сопоставлена разнородная атрибутивная информация, например текстовые или числовые данные, такие как наименование объекта, код классификатора, номер региона и др. Наиболее распространенными форматами являются форматы \*.shp, \*.tab, \*.geojson.

Программное взаимодействие с данными в векторном формате происходит как с большим массивом структурированных данных. Условно, такие данные можно представлять в виде большой таблицы объектов. Для оптимизации доступа к объектам, подобно обычным БД, производится индексирование.

На рисунке 1 представлен пример таблицы геоданных (слоя) в формате \*.sqlite. На рисунке 2 представлен пример содержащихся в ней данных (географических объектов).

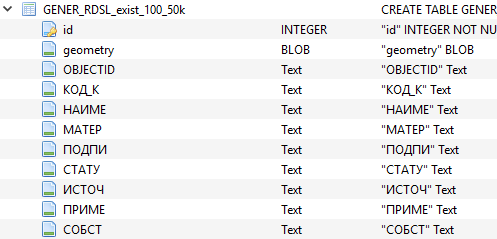


Рисунок - пример слоя

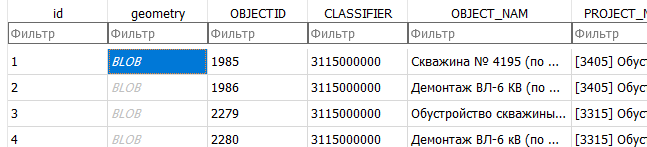


Рисунок - пример данных слоя

# Стилизация геоданных. Формат SLD

Для того, чтобы визуализировать географический объект, требуется считать поле, содержащее геометрию и отрисовать эту геометрию на карте в соответствии с каким-либо набором правил (цвет, толщина линий, прозрачность, сглаживание и т.д.).

В зависимости от требований, набор правил может формироваться как для слоя в целом, тогда отрисовка будет происходить одинаково для всех объектов, так и на основании значений каких-либо свойств конкретного объекта.

Существует много стандартов описания геоданных, такие как GML, MAP и др. В данной работе основное внимание уделено стандарту SLD, так как именно он использовался при описании правил стилизации данных [4].

Структура описания в формате SLD древовидна и ее пример представлен на рис. 3.

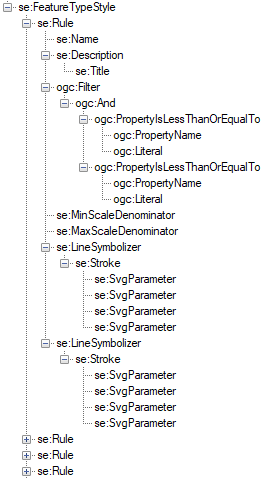


Рисунок – структура описания SLD

Стиль SLD представляет собой текстовый документ в формате XML. Он может применяться к одному или нескольким слоям и содержит набор правил (Rule).

Каждое правило содержит внутри себя наименование группы объектов (Name), фильтр, на основании которого выделять группу (Filter), список параметров отрисовки Symbolizer (на скриншотах представлен LineSymbolizer – параметры отрисовки полилиний, но также есть PointSymbolizer для точек, PolygonSymbolizer для полигонов и TextSymbolizer для текста), а также ограничения масштаба MinScaleDenominator и MaxScaleDenominator [5].

Фильтр может содержать внутри себя как простые правила, например «значение поля XXX должно быть равно YYY», так и сложные (потенциально бесконечные) комбинации правил, которые можно сочетать с помощью правил математической логики (допустимы операции «И», «НЕ», «ИЛИ») [6].

Более подробную информацию касательно стандарта можно узнать на официальном сайте стандарта <https://www.opengeospatial.org/standards/sld> [7].

Пример одного из правил приведен на следующей странице. Согласно этому правилу, при масштабе от 50000 до 100000 на слое выделяется группа данных «Автодороги с усовершенствованным покрытием» на основании значения поля «КОД\_КЛАССИФИКАТОРА» и «МАТЕРИАЛ\_ПОКРЫТИЯ». Так как в приведенном правиле определены только стили линий, то это правило будет накладываться только на полилинии. Так как стилей линий внутри правила два, то отрисовка данных на тайле будет следующей: сначала рисуется черная линия шириной 2.2px, а на ней красная линия шириной 1.4px.

Как следует из примера, стили отрисовки геометрии в рамках одного правила могут комбинироваться (например, несколько стилей линий). Однако, помимо этого, конкретный объект может подходить под фильтры нескольких правил, соответственно, к нему будут применяться несколько правил. Помимо этого, у правила вообще может не быть фильтров, тогда правило будет применяться ко всей геометрии слоя.

<se:Rule>

<se:Name>Автодороги с усовершенствованным покрытием (усовершенствованные шоссе)</se:Name>

<se:Description>

<se:Title>Автодороги с усовершенствованным покрытием (усовершенствованные шоссе)</se:Title>

</se:Description>

<ogc:Filter xmlns:ogc="http://www.opengis.net/ogc">

<ogc:And>

<ogc:PropertyIsLessThanOrEqualTo>

<ogc:PropertyName>КОД\_КЛАССИФИКАТОРА</ogc:PropertyName>

<ogc:Literal>1321100000</ogc:Literal>

</ogc:PropertyIsLessThanOrEqualTo>

<ogc:PropertyIsLessThanOrEqualTo>

<ogc:PropertyName>МАТЕРИАЛ\_ПОКРЫТИЯ</ogc:PropertyName>

<ogc:Literal>АСФАЛЬТ</ogc:Literal>

</ogc:PropertyIsLessThanOrEqualTo>

</ogc:And>

</ogc:Filter>

<se:MinScaleDenominator>50000</se:MinScaleDenominator>

<se:MaxScaleDenominator>100000</se:MaxScaleDenominator>

<se:LineSymbolizer>

<se:Stroke>

<se:SvgParameter name="stroke">#000000</se:SvgParameter>

<se:SvgParameter name="stroke-width">2.2</se:SvgParameter>

<se:SvgParameter name="stroke-linejoin">bevel</se:SvgParameter>

<se:SvgParameter name="stroke-linecap">square</se:SvgParameter>

</se:Stroke>

</se:LineSymbolizer>

<se:LineSymbolizer>

<se:Stroke>

<se:SvgParameter name="stroke">#ff0000</se:SvgParameter>

<se:SvgParameter name="stroke-width">1.4</se:SvgParameter>

<se:SvgParameter name="stroke-linejoin">bevel</se:SvgParameter>

<se:SvgParameter name="stroke-linecap">square</se:SvgParameter>

</se:Stroke>

</se:LineSymbolizer>

</se:Rule>

# Проектирование и разработка логики стилизации геоданных для мобильного приложения

Мобильное приложение представляет собой геоинформационную систему, основной функцией которого является показ картографической информации в режиме оффлайн.

Интерфейс приложения приведен на рисунке 4.

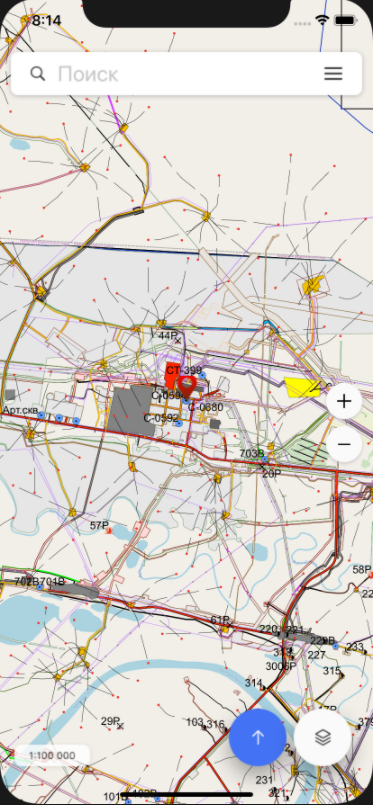


Рисунок – интерфейс приложения

В рамках данной работы была разработана логика стилизации географической информации, отображаемой на карте, в соответствии с описанием в формате SLD. На рисунке можно увидеть разные виды геометрии: полигоны, линии, точки. Также можно увидеть текстовую информацию.

Все данные, выводимые на карте, разделяются слои. Общее число слоев – 69. При этом существует иерархия, называемая деревом слоев (рис. 5).

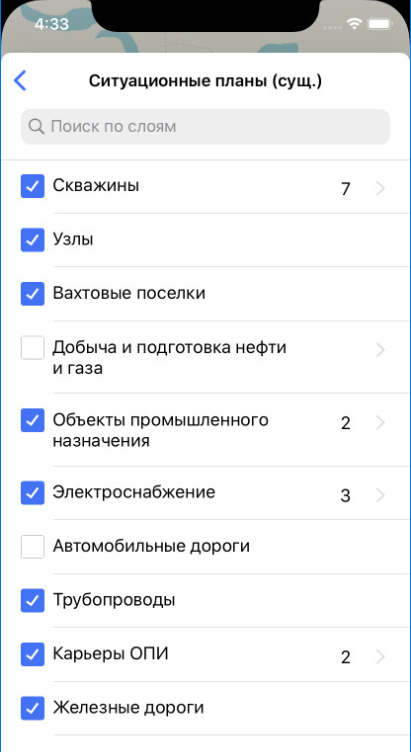


Рисунок – дерево слоев.

Дерево слоев служит для 2 вещей:

1. Управление порядком вывода и активностью слоев
2. Сопоставление файлов с данными слоев и файлов стилей.

Фрагмент XML-представления файла иерархии (дерева) слоев приведен ниже. Из него следует, что каждый из слоев (LocalSource>ShpLayer) может ссылаться на стиль в формате SLD (StylePath).

<ParentLayer>

<Name>Общая информация</Name>

<LocalSource>

<Name>Горные отводы месторождений</Name>

<IsActive>true</IsActive>

<ShpLayer>

<Path>Общая информация/Горные\_отводы\_месторождений.shp</Path>

<StylePath>Общая информация/Горные\_отводы\_месторождений.sld</StylePath>

</ShpLayer>

</LocalSource>

<LocalSource>

<Name>Границы районов РФ</Name>

<IsActive>true</IsActive>

<ShpLayer>

<Path>Общая информация/Границы\_районов\_РФ.shp</Path>

<StylePath>Общая информация/Границы\_районов\_РФ.sld</StylePath>

</ShpLayer>

</LocalSource>

</ParentLayer>

Как говорилось ранее, для вывода геоданных используется библиотека ThinkGeo, предназначенная для разработки геоинформационных систем. Библиотека предоставляет возможность вывода карты, предлагает набор встроенных стилей для слоев, обладает возможностью тайлового кэширования и др.

Несмотря на то, что библиотека предлагает ряд встроенных стилей, они недостаточно гибкие. В частности, в базовой реализации невозможно проводить древовидную фильтрацию.

В связи с этим, была разработана система классов, описывающих формат SLD, а также обладающая методами самопреобразования в объекты стилей библиотеки. Часть классов переопределяет поведение стандартной библиотеки, расширяя ее классы.

Для корректного отображения карты при формировании слоев требуется следовать следующей логике:

1. Провести десериализацию файла стиля в сущность ILayerStyle
2. Конвертировать сущность в группу объектов Style (библиотечных)
3. Задать объект Style в качестве стиля слоя

Разработанная система классов представлена с помощью диаграммы классов на рисунке 14.

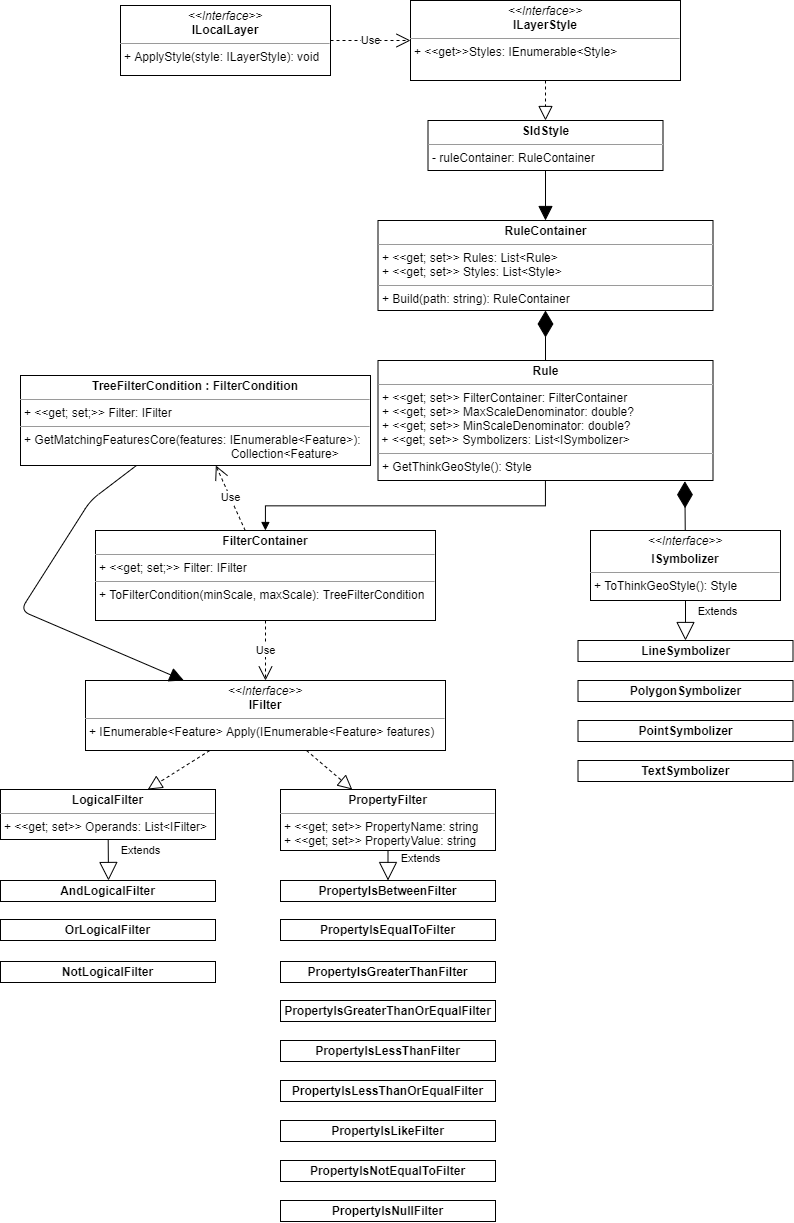


Рисунок – диаграмма классов

Как видно из диаграммы, в целом система классов соответствует структуре SLD (стр.7, рис. 3): класс SldStyle включает в себя набор правил RuleContainer, а каждое правило Rule включает в себя ограничения масштаба MinScaleDenominator и MaxScaleDenominator, дерево фильтров FilterContainer, а также список Symbolizers – объектов, включающих правила отрисовки линий, полигонов, точек или текста.

Непосредственная фильтрация географических объектов происходит в методе GetMatchingFeaturesCore() класса TreeFilterCondition, который является расширением библиотечного класса FilterCondition.

Часть кода программы приведена в приложении.

На рисунке 7 приведен пример работы программы для полилиний при единственном включенном слое – слое автодорог. Из рисунка видно, что некоторые дороги являются красными, некоторые – оранжевыми. Также есть черные и серые дороги. Кроме того, можно заметить, что некоторые дороги являются скругленными, некоторые – пунктирными. Это можно задавать в параметрах SLD.



Рисунок – результат работы на примере слоя с линиями

На рисунке 8 можно увидеть результат работы программы для полигонов, а на рисунке 9 – для точек. Изображение точечных объектов может задаваться с помощью картографических или обычных шрифтов.

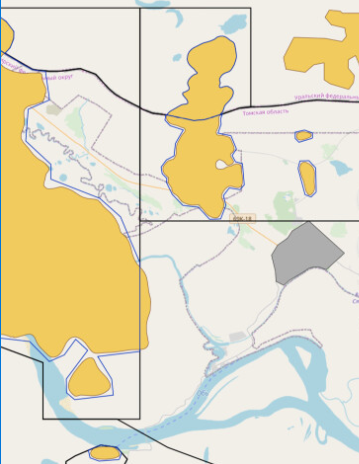


Рисунок 8 – результат работы на примере слоя с полигонами



Рисунок 9 – результат работы на примере слоя с точками

Увидеть результат работы программы при условии включения всех слоев одновременно можно увидеть, вернувшись к рис. 4 на стр. 10.

# Оптимизация скорости отрисовки карт

В связи с тем, что объем выводимых данных оказался слишком большим, первоначальная реализация оказалась недостаточно производительной. Основная проблема заключалась в том, что при каждой отрисовке тайла, каждый объект каждого слоя проходил через метод GetMatchingFeaturesCore() класса TreeFilterCondition, в котором осуществлялась древовидная фильтрация. При максимальном масштабе время полной отрисовки карты составляло около 90 секунд.

Низкая производительность была связана с тем, что фильтрация осуществлялась во время выполнения и для ее повышения было принято решение вынести логику древовидной фильтрации на этап подготовки данных.

Решение заключалось в формировании трехуровневого кэша:

1. Кэша параметров отрисовки
2. Кэша стилей
3. Кэша слоя

Пример кэша параметров отрисовки приведен ниже:

<Symbolizer Id="1">

<PolygonSymbolizer>

<Fill>

<SvgParameter name="fill">#f9f4d5</SvgParameter>

</Fill>

</PolygonSymbolizer>

</Symbolizer>

<Symbolizer Id="2">

<PointSymbolizer>

<Graphic>

<Mark>

<Fill>

<SvgParameter name="fill">#000000</SvgParameter>

</Fill>

<MarkIndex>137</MarkIndex>

<OnlineResource href="RN\_UZsitplans1" />

</Mark>

<Size>12</Size>

</Graphic>

</PointSymbolizer>

</Symbolizer>

Кэш стилей содержит в себе наименования группы объектов, номера стилей, а также ссылки на параметры отрисовки кэша предыдущего уровня.

<Style Id="1" Name="Скважины (полигоны)">

<SymbolizerId>0</SymbolizerId>

</Style>

<Style Id="2" Name="Скважины (точки)">

<SymbolizerId>1</SymbolizerId>

</Style>

Кэш слоя формируется отдельно для каждого слоя. Для его формирования нужно один раз провести фильтрацию всех данных слоя в соответствии с фильтрами SLD. В результате, получается набор групп объектов, с указанием стиля, который нужно применить.

<Scale Scale="100000">

<FeatureGroups StyleId="1">

<FeatureId>1</FeatureId>

<FeatureId>2</FeatureId>

<FeatureId>3</FeatureId>

<FeatureId>4</FeatureId>

</Scale>

<Scale Scale="200000">

<FeatureGroups StyleId="2">

<FeatureId>1</FeatureId>

<FeatureId>2</FeatureId>

<FeatureId>3</FeatureId>

<FeatureId>4</FeatureId>

</Scale>

Вышеприведенные примеры позволяют отображать объекты одного и того же слоя на масштабе 100,000 как полигоны, а на масштабе 200,000 – как точки.

На основании данных кэшей формируются хэш-таблицы соответствия групп объектов с их стилями, а также таблицы принадлежности объекта к группе объектов.

Таким образом, вместо применения древовидной фильтрации, достаточно проверить по хэш-таблицам, нужно ли применять текущий стиль к текущему объекту, а это гораздо более быстрая операция.

В результате применения такого подхода скорость отрисовки карты на максимальном масштабе (5,000,000) составила около 20 секунд. На часто используемых масштабах (50,000 – 500,000) отрисовка происходит в течение 1-3с в зависимости от количества данных на карте. Такая скорость достаточна для удобного пользования приложением.

# Заключение

В результате выполненной работы были изучены основные форматы географических данных (векторный и растровый), характеристики и особенности структуры геоданных, способы их хранения, а также способы их стилизации.

Также был реализован программный код, позволяющий выводить векторные данные на мобильных устройствах в условиях отсутствия доступа к интернету в соответствии со стандартом SLD.

Код написан с использованием платформы Xamarin и следует стандарту .NET Standard 2.0, поэтому он будет одинаково выполняться как на устройствах с ОС iOS, так и Android.

# Список литературы

1. Панамарева Олеся Николаевна. Геоинформационные системы и программные технологии в управлении сложными территориально-экономическими процессами // Вестник государственного университета морского и речного флота им. адмирала С.О. Макарова. 2012. №4 (16). URL: <https://cyberleninka.ru/article/n/geoinformatsionnye-sistemy-i-programmnye-tehnologii-v-upravlenii-slozhnymi-territorialno-ekonomicheskimi-protsessami>.
2. Что такое геоданные? [Электронный ресурс] // Теплица социальных технологий. Образование.

URL: https://te-st.ru/2018/07/17/what-means-geodata/

1. Справочник по ГИС (QGIS) и открытым геоданным для архитектора [Электронный ресурс] // ГИС-справочник

URL: <https://gis4arch.blogspot.com/2019/01/blog-post.html>

1. Владимир Андрианов, DATA+. Стандарты в ИПД //  ArcReview №2 (37) за 2006 г. URL: <https://www.esri-cis.ru/news/arcreview/detail.php?ID=1590&SECTION_ID=44>
2. Introduction to SLD [Электронный ресурс] // GeoServer 2.16.x User Manual URL: <https://docs.geoserver.org/stable/en/user/styling/sld/introduction.html#introduction-to-sld>
3. Серверы WMS и фильтрация данных [Электронный ресурс] // GISLAB. Географические информационные системы и дистанционное зондирование URL: <https://gis-lab.info/qa/wms-filter.html>
4. Styled Layer Descriptor [Электронный ресурс] // Open Geospatial Consortium  
   URL: <https://www.opengeospatial.org/standards/sld>

# Приложение

/// <summary>

/// Локальный слой.

/// </summary>

public interface ILocalLayer

{

/// <summary>

/// Есть ли стили.

/// </summary>

bool HasStyles { get; }

/// <summary>

/// Слой ThinkGeo.

/// </summary>

Layer Layer { get; }

/// <summary>

/// Метаданные слоя.

/// </summary>

LayerMeta LayerMeta { get; set; }

/// <summary>

/// Максимальный масштаб.

/// </summary>

int MaxScale { get; set; }

/// <summary>

/// Минимальный масштаб.

/// </summary>

int MinScale { get; set; }

/// <summary>

/// Путь к стилю.

/// </summary>

string StylePath { get; }

/// <summary>

/// Применить стили по умолчанию.

/// </summary>

void ApplyDefaultStyles();

/// <summary>

/// Применить стиль.

/// </summary>

/// <param name="style">Стиль.</param>

void ApplyStyle(ILayerStyle style);

/// <summary>

/// Построить стиль.

/// </summary>

/// <param name="path">Путь к стилю.</param>

/// <returns>Стиль.</returns>

ILayerStyle BuildStyle(string path);

/// <summary>

/// Подготовить слой. Должен вызываться перед использованием других методов.

/// </summary>

void Prepare();

}

/// <summary>

/// Стиль слоя.

/// </summary>

public interface ILayerStyle

{

/// <summary>

/// Стили ThinkGeo.

/// </summary>

IEnumerable<ThinkGeoStyle> Styles { get; }

/// <summary>

/// Обновить масштаб или метаданные.

/// </summary>

/// <param name="meta">Метаданные.</param>

/// <param name="newScale">Масштаб.</param>

void Update(LayerMeta meta, int newScale);

}

/// <summary>

/// Стиль слоя.

/// </summary>

public class SldStyle : ILayerStyle

{

/// <summary>

/// Конструктор.

/// </summary>

/// <param name="path">Путь к SLD.</param>

/// <param name="layerMeta">Метаданные слоя.</param>

public SldStyle(string path, LayerMeta layerMeta)

{

var rules = RuleContainer.Build(path, layerMeta);

Styles = rules.Styles;

}

/// <summary>

/// Список стилей ThinkGeo.

/// </summary>

public IEnumerable<ThinkGeoStyle> Styles { get; }

/// <summary>

/// Обновить масштаб и метаданные стиля.

/// </summary>

/// <param name="meta">Метаданные.</param>

/// <param name="newScale">Новый масштаб.</param>

public void Update(LayerMeta meta, int newScale)

{

IterateOverFilterCondition(x =>

{

x.LayerMeta = meta;

x.UpdateScale(newScale);

});

}

/// <summary>

/// Пройти по условиям фильтрации стиля.

/// </summary>

/// <param name="action">Действие.</param>

private void IterateOverFilterCondition(Action<TreeFilterCondition> action)

{

Styles?

.OfType<FilterStyle>()

.SelectMany(x => x.Conditions.OfType<TreeFilterCondition>())

.ForEach(action);

}

}

/// <summary>

/// Контейнер стилей, основанных на правилах.

/// </summary>

[XmlRoot("FeatureTypeStyle")]

public class RuleContainer

{

/// <summary>

/// Список правил.

/// </summary>

[XmlElement("Rule")]

public List<Rule> Rules { get; set; }

/// <summary>

/// Получить стили ThinkGeo.

/// </summary>

public List<Style> Styles => Rules.Select(x => x.GetThinkGeoStyle()).ToList();

/// <summary>

/// Построить контейнер из файла.

/// </summary>

/// <param name="path">Путь к файлу.</param>

/// <param name="layerMeta">Метаданные слоя.</param>

public static RuleContainer Build(string path, LayerMeta layerMeta)

{

var doc = XDocument.Load(path);

RemoveNamespacePrefix(doc.Root);

var xElement = doc.XPathSelectElement("/StyledLayerDescriptor/NamedLayer/UserStyle/FeatureTypeStyle");

var reader = xElement.CreateReader();

var serializer = new XmlSerializer(typeof(RuleContainer));

var ruleContainer = (RuleContainer)serializer.Deserialize(reader);

ruleContainer.Rules.ForEach(x => x.LayerMeta = layerMeta);

return ruleContainer;

}

/// <summary>

/// Стиль, применяемый по правилу (фильтру и масштабу).

/// </summary>

public class Rule

{

/// <summary>

/// Список фильтров.

/// </summary>

[XmlElement("Filter")]

public FilterContainer FilterContainer { get; set; }

/// <summary>

/// Метаданные слоя.

/// </summary>

[XmlIgnore]

public LayerMeta LayerMeta { get; set; }

/// <summary>

/// Минимальный масштаб отображения.

/// </summary>

public double? MaxScaleDenominator { get; set; }

/// <summary>

/// Максимальный масштаб отображения.

/// </summary>

public double? MinScaleDenominator { get; set; }

/// <summary>

/// Наименование элемента.

/// </summary>

public string Name { get; set; }

/// <summary>

/// Список стилей точек, линий, полигонов и др.

/// </summary>

[XmlElement("PointSymbolizer", typeof(PointSymbolizer))]

[XmlElement("LineSymbolizer", typeof(LineSymbolizer))]

[XmlElement("PolygonSymbolizer", typeof(PolygonSymbolizer))]

[XmlElement("TextSymbolizer", typeof(TextSymbolizer))]

public List<ISymbolizer> Symbolizers { get; set; }

/// <summary>

/// Получить стиль ThinkGeo.

/// </summary>

/// <returns>Стиль.</returns>

public Style GetThinkGeoStyle()

{

if (FilterContainer == null)

return new CompositeStyle(Symbolizers?.Select(x => x.ToThinkGeoStyle()));

var style = new FilterStyle();

style.Conditions.Add(FilterContainer.ToFilterCondition(LayerMeta, MinScaleDenominator, MaxScaleDenominator));

FilterContainer?

.Filter?

.GetRequiredColumnNames()

.Distinct()

.ForEach(style.RequiredColumnNames.Add);

Symbolizers?

.Select(x => x.ToThinkGeoStyle())

.ForEach(style.Styles.Add);

return style;

}

}

/// <summary>

/// Древовидное условие фильтрации объектов карты.

/// </summary>

public class TreeFilterCondition : FilterCondition

{

/// <summary>

/// Фильтр (дерево или узел).

/// </summary>

private readonly IFilter \_filter;

/// <summary>

/// Максимальный масштаб.

/// </summary>

private readonly int \_maxScale;

/// <summary>

/// Минимальный масштаб.

/// </summary>

private readonly int \_minScale;

/// <summary>

/// Текущий масштаб.

/// </summary>

private int \_currentScale = GispScales.DefaultScale;

/// <summary>

/// Конструктор.

/// </summary>

/// <param name="filter">Фильтр.</param>

public TreeFilterCondition(IFilter filter, LayerMeta layerMeta, double? minScale, double? maxScale)

: base("OBJECTID", ">0")

{

\_filter = filter;

LayerMeta = layerMeta;

var minSc = minScale ?? GispScales.MinScale - 1;

var maxSc = maxScale ?? GispScales.MaxScale + 1;

\_minScale = (int)Math.Round(minSc);

\_maxScale = (int)Math.Round(maxSc);

}

/// <summary>

/// Метаданные слоя.

/// </summary>

public LayerMeta LayerMeta { get; set; }

/// <summary>

/// Обновить масштаб.

/// </summary>

/// <param name="newScale">Новый масштаб.</param>

public void UpdateScale(int newScale)

{

\_currentScale = newScale;

}

// Есть баг: создаются пустые бесполезные колонки, судя по всему при добавлении стиля.

// Ищем по КОД\_К, но почему-то еще добавляется КОД\_К1, КОДК\_2, КОДК\_3 и тд. Причина пока неясна.

/// <summary>

/// Отфильтровать объекты на карте.

/// </summary>

/// <param name="features">Объекты карты.</param>

/// <returns>Отфильтрованный список.</returns>

protected override Collection<Feature> GetMatchingFeaturesCore(IEnumerable<Feature> features)

{

if (\_currentScale > \_maxScale || \_currentScale < \_minScale)

return new Collection<Feature>();

var result = new Collection<Feature>(\_filter.Apply(features, LayerMeta).ToList());

return result;

}

}

/// <summary>

/// Логический фильтр.

/// </summary>

public abstract class LogicalFilter : IFilter

{

/// <summary>

/// Список операндов(фильтров) фильтра.

/// </summary>

[XmlElement("And", typeof(AndLogicalFilter))]

[XmlElement("Or", typeof(OrLogicalFilter))]

[XmlElement("PropertyIsNotEqualTo", typeof(PropertyIsNotEqualToFilter))]

[XmlElement("Not", typeof(NotLogicalFilter))]

[XmlElement("PropertyIsBetween", typeof(PropertyIsBetweenFilter))]

[XmlElement("PropertyIsEqualTo", typeof(PropertyIsEqualToFilter))]

[XmlElement("PropertyIsGreaterThan", typeof(PropertyIsGreaterThanFilter))]

[XmlElement("PropertyIsGreaterThanOrEqualTo", typeof(PropertyIsGreaterThanOrEqualToFilter))]

[XmlElement("PropertyIsLessThan", typeof(PropertyIsLessThanFilter))]

[XmlElement("PropertyIsLessThanOrEqualTo", typeof(PropertyIsLessThanOrEqualToFilter))]

[XmlElement("PropertyIsLike", typeof(PropertyIsLikeFilter))]

[XmlElement("PropertyIsNull", typeof(PropertyIsNullFilter))]

public List<IFilter> Operands { get; set; }

/// <inheritdoc />

public abstract IEnumerable<Feature> Apply(IEnumerable<Feature> features, LayerMeta layerMeta);

/// <inheritdoc />

public IEnumerable<string> GetRequiredColumnNames()

{

return Operands.SelectMany(x => x.GetRequiredColumnNames());

}

}

/// <summary>

/// Фильтр по свойству объекта.

/// </summary>

public abstract class PropertyFilter : IFilter

{

/// <summary>

/// Наименование свойства.

/// </summary>

[XmlIgnore]

private string \_propertyName;

/// <summary>

/// Наименование свойства.

/// </summary>

[XmlElement("PropertyName")]

public string PropertyName

{

get => \_propertyName;

set

{

var maxLen = Regex.IsMatch(value, @"\p{IsCyrillic}")? 5 : 10;

var val = value.Length > maxLen? value.Substring(0, 5) : value;

\_propertyName = val.ToUpperInvariant();

}

}

/// <summary>

/// Значение свойства.

/// </summary>

[XmlElement("Literal")]

public string PropertyValue { get; set; }

/// <inheritdoc />

public abstract IEnumerable<Feature> Apply(IEnumerable<Feature> features, LayerMeta layerMeta);

/// <inheritdoc />

public IEnumerable<string> GetRequiredColumnNames()

{

return new List<string> { PropertyName };

}

}

/// <summary>

/// Значение свойства меньше либо равно указанному.

/// </summary>

[XmlRoot("PropertyIsLessThanOrEqualTo")]

public class PropertyIsLessThanOrEqualToFilter : SingleValuedPropertyFilter

{

/// <inheritdoc />

protected override bool FilterFunc<T>(T first, T second)

{

return first.CompareTo(second) <= 0;

}

}

/// <summary>

/// Фильтр вида "Логическое И".

/// Находит пересечение множеств, полученных в результате применения операндов.

/// </summary>

[XmlRoot("And")]

public class AndLogicalFilter : LogicalFilter

{

/// <inheritdoc />

public override IEnumerable<Feature> Apply(IEnumerable<Feature> features, LayerMeta layerMeta)

{

return Operands

.Select(operand => operand.Apply(features, layerMeta))

.Aggregate(features, (current, opFiltered) => current.Intersect(opFiltered))

.Distinct();

}

}

/// <summary>

/// Стиль, который можно преобразовать в стиль ThinkGeo.

/// </summary>

public interface ISymbolizer

{

/// <summary>

/// Преобразовать в стиль ThinkGeo.

/// </summary>

/// <returns>Стиль ThinkGeo.</returns>

Style ToThinkGeoStyle();

}

/// <summary>

/// Стиль полигонов.

/// </summary>

public class PolygonSymbolizer : ISymbolizer

{

/// <summary>

/// Стиль заливки.

/// </summary>

public Fill Fill { get; set; }

/// <summary>

/// Стиль линий.

/// </summary>

public Stroke Stroke { get; set; }

/// <inheritdoc />

public Style ToThinkGeoStyle()

{

var stroke = Stroke?.ToGeoPen() ?? new GeoPen(GeoColors.Black, 0.5f);

var fill = Fill?.ToGeoSolidBrush() ?? new GeoSolidBrush(GeoColor.FromArgb(0, GeoColors.Black));

return new AreaStyle(stroke, fill);

}