МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ИНСТИТУТ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ СИСТЕМ
ФАКУЛЬТЕТ АВТОМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ
КАФЕДРА ЭЛЕКТРОННЫХ ВЫЧИСЛИТЕЛЬНЫХ МАШИН

Направление 09.03.01 - Информатика и вычислительная техника
(код и наименование направления)

Профиль – Программное и аппаратное обеспечение вычислительной техники

Допускаю к защите
Заведующая кафедрой ЭВМ
_____ / Долженкова М.Л. /
(подпись) (Ф.И.О.)

РАЗРАБОТКА ПРИЛОЖЕНИЯ «РАДИО-ТРАССА»

Пояснительная записка выпускной квалификационной работы

ТПЖА 09.03.01.487 ПЗ

Разработал: студент гр.ИВТб-4301-04-00 _____ / Птахова А.М. / _____
(подпись) (Ф.И.О.) (дата)

Руководитель: к.т.н., доцент _____ / Долженкова М.Л. / _____
(подпись) (Ф.И.О.) (дата)

Нормоконтролёр: к.т.н., доцент _____ / Скворцов А.А./ _____
(подпись) (Ф.И.О) (дата)

Киров 2024

РЕФЕРАТ

Птахова А.М.. Разработка приложения «Радио-Трасса». ТПЖА 09.03.01.487 ПЗ: ВКР/ ВятГУ, каф. ЭВМ; рук. Долженкова М.Л. – Киров, 2024. – Гр.ч. 9л. ф.А1; ПЗ 74с., 36 рис., 6 прил.

РАДИОТРАССА, ОБЛАСТЬ ПОКРЫТИЯ РАДИОТРАССЫ, 3D ВИЗУАЛИЗАЦИЯ, QT, C++, ГЕНЕРАЦИЯ КАРТЫ ВЫСОТ, СФЕРИЧЕСКАЯ ТЕОРЕМА КОСИНУСОВ, КОДИРОВАНИЕ ПО МЕТОДУ БЛИЖАЙШЕГО СОСЕДА.

Объект выпускной квалификационной работы – программное обеспечение для расчета и построения радиотрассы и области зоны радиопокрытия, а также для 3D визуализации рельефа местности.

Целью данной выпускной квалификационной работы является предоставление возможности выполнения работ в офлайн режиме.

В данной работе представлены анализ существующих решений и алгоритмов, архитектурно-структурные решения, разработка приложения, тестирование. В результате выполнения дипломного проекта было разработано приложение, позволяющее не только решать необходимые задачи в офлайн режиме, но и выполнять 3D визуализацию рельефа местности, для которой производился расчет радиотрассы. Именно наличие функции построения 3-х мерной модели отличает разработанное приложение от аналогичных решений.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Проникновение интернета в каждую сферу жизни стало неоспоримым фактом в современном мире. С каждым днем растет число пользователей, а, следовательно, и количество передаваемой информации.

Одной из главных задач при передаче данных является сохранение их целостности. Этого можно достичь, если передавать информацию по надежному каналу связи. В свою очередь надежность канала связи подразумевает непрерывность сигнала в различных географических условиях. Для того чтобы оценить возможности создания такого канала на некоторой местности производят расчет радиотрассы.

Расчет радиотрассы включает в себя построение графика высот для выбранной местности, построение линии сигнала с учетом высоты антенн. В результате построения радиотрассы можно увидеть существует ли возможность беспрепятственного прохождения сигнала на данной местности или нет. При необходимости определения границ области, внутри которой отсутствуют помехи, влияющие на непрерывность сигнала, строится область покрытия радиотрассы.

Помимо проведения расчетов радиотрассы, пользователю также необходима возможность визуализации рельефа области, для которой выполняется расчет, с целью получения приблизительной оценки правильности расчетов. Использование 3х-мерной модели рельефа поможет представить пользователю эту информацию в более наглядной форме.

Для решения задачи по определению возможности организации канала связи между объектами возможна разработка приложения, включающего в себя описанные выше функции. Именно разработке такого приложения посвящен данный дипломный проект.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

На первым этапе работы необходимо рассмотреть алгоритмы расчета радиотрассы и области радиопокрытия. На основании рассмотренных алгоритмов определить данные, необходимые для реализации этих алгоритмов. Также требуется сформулировать техническое задание.

1.1 Расчет радиотрассы

Расчет радиотрассы включает в себя построение графика высот для выбранной местности, построение линии сигнала с учетом высоты антенн, вычисление точек 1-ой зоны Френеля и ее построение.

1.1.1 Построение профиля высот

Для того чтобы построить график высот, необходимо выполнить следующие действия:

1) выбрать 2 точки на карте. Эти точки – координаты объектов, между которыми планируется организовать канал связи;

2) вычислить координаты точек, принадлежащих маршруту радиотрассы;

3) определить высоту для каждой точки маршрута радиотрассы, включая точки объектов;

4) построить профиль радиотрассы с учетом высоты точки и отдаленностью этой точки от источника (дистанция в км);

1.1.2 Построение линии сигнала

Для отображения линии связи на графике используются 2 точки:

1) высота 1-ого объекта + высота антенны для 1-ого объекта;

2) высота 2-ого объекта + высота антенны для 2-ого объекта.

Далее эти 2 точки соединяются линией. Пример расчета радиотрассы представлен на рисунке 1.

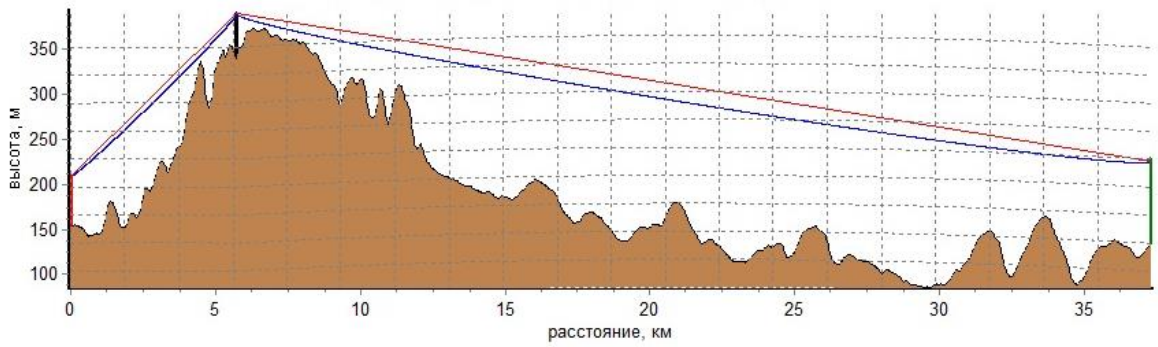


Рисунок 1 – Пример расчета радиотрассы

1.2 Расчет зоны радиопокрытия

Алгоритм расчета зоны радиопокрытия следующий:

1) выбирается точка – источник сигнала;

2) задается радиус области покрытия;

3) вычисляются точки, лежащие на окружности, образуемой областью покрытия;

4) полученные точки разбивают область на фрагменты. Пример разбиения области на фрагменты представлен на рисунке 2;

5) для каждой вычисленной точкой и источником сигнала строится радиотрасса;

6) для каждой построенной радиотрассы определяется её тип: открытая, полуоткрытая, полузакрытая, закрытая;

7) каждый фрагмент области радиопокрытия закрашивается в цвет, соответствующий типу трассы;

8) уменьшается значение радиуса на заданный шаг;

9) повтор п. 3) – 8) до тех пор, пока значение радиуса не станет равным нулю.

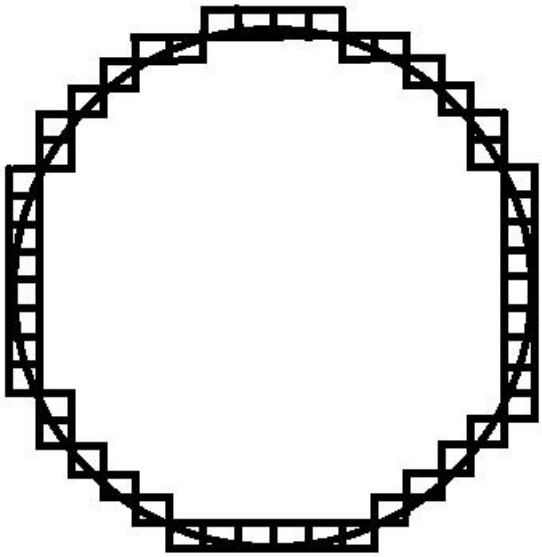


Рисунок 2 – Разбиение области на фрагменты

На основании рассмотренного алгоритма для расчета области радиопокрытия был сделан вывод, что данная задача сводится к многократному решению задачи о построении радиотрассы между 2-мя точками. Следовательно, для их реализации необходимо наличие следующих данных:

1) координаты 2х точек – источника и приемника;

2) массив точек, принадлежащих маршруту радиотрассы;

3) массив тайлов для определения значения высот для каждой точки маршрута;

4) массив высот, описывающих профиль рельефа;

5) массив высот для линии сигнала;

6) массив значений для первой зоны Френеля;

7) массив тайлов и массив высот, которые также необходимы и для выполнения 3D визуализации рельефа местности.

Алгоритмы расчета радиотрассы и области радиопокрытия были рассмотрены. На основании анализа были сделаны выводы о том, какие данные необходимы для их реализации. Можно выполнить обзор аналогов.

1.3 Обзор аналогичных решений

Для решения поставленной задачи уже существуют разработанные приложения и сайты. Но заказчика не устроили их функциональные возможности, поэтому возникла необходимость в разработке приложения. Требования, предъявляемые заказчиком, сформулированы в техническом задании.

1.4 Техническое задание

1.4.1 Наименование приложения

Полное наименование приложения: приложения для расчета и построения радиотрассы.

Сокращенное наименование приложения: «Радио-Трасса».

1.4.2 Краткая характеристика области применения

Приложение предназначено для получения информации о возможности создания канала связи между объектами.

1.4.3 Цели создания приложения

Функциональным назначение приложения является предоставление возможности выполнения работ в офлайн режиме.

1.4.4. Требования к конечному пользователю

Особым требованием к конечному пользователю является базовое понимание принципов расчета и построения радиотрассы.

1.4.5 Требования к программе

Программа должна обеспечивать возможность выполнения перечисленных ниже действий:

1) возможность выбора положение объектов на карте или задания их координат;

2) возможность задания радиуса для области радиопокрытия;

3) расчет и построение радиотрассы;

4) расчет области радиопокрытия;

5) 3D визуализация рельефа местности.

При выполнении расчета радиотрассы данные высот необходимо получать из тайлов, находящихся на сервере карт.

При 3D визуализации рельефа области функция должна реализовывать построение 3D модели на основе имеющихся значений высот для данной области.

В состав технических средств должен входить ПК, включающий в себя:

1) 64-х разрядный процессор с тактовой частотой не меньше 1.0 ГГц;

2) дисплей;

3) не менее 1 Гб оперативной памяти;

4) не менее 100 мегабайт свободного дискового пространства;

5) клавиатура и мышь.

Системные программные средства, используемые программой, должны быть представлены ОС Windows 10;

Программа должна обеспечивать взаимодействие с пользователем по средствам графического интерфейса и предоставлять возможность выполнять наиболее часто используемые операции с помощью сочетаний клавиш на клавиатуре.

1.4.6 Требования к программной документации

Состав программной документации должен включать в себя:

1) техническое задание;

2) руководство пользователя;

3) исходный код.

1.4.7 Стадии и этапы разработки

Разработка должна быть приведена в 3 стадии:

1) разработка технического задания;

2) проектирование;

3) внедрение.

На стадии разработки технического задания должен быть выполнен этап разработки, согласования и разработки технического задания.

На стадии проектирования необходимо выполнить следующие этапы:

1) разработка программы;

2) разработка программной документации;

3) испытание программы.

На этапе внедрения выполняется передача программы заказчику.

Вывод

В результате выполнения анализа предметной области было принято решение о разработке приложения, выполняющего функции для расчета и построения радиотрассы и зоны покрытия радиосигналом, а также реализующего 3D визуализацию для некоторой области.

Данные высот приложение будет получать путем обработки тайлов, полученных от сервера карт. Остальные данные необходимо рассчитать.

2 АРХИТЕКТУРНО – СТРУКТУРНЫЕ РЕШЕНИЯ

На данном этапе работ необходимо определить схему взаимодействия сервера карт с приложением. Создать карту высот для правильной работы приложения. На основании полученной схемы, а также требуемых функций выполнить проектирование приложения.

2.1 Взаимодействие с сервером карт

В результате выполнения первичного анализа сервера карт было выявлено, что сервер карт представляет собой сервер, на котором хранятся карты и клиентскую часть, которая представлена библиотекой с открытым исходным кодом, в которой описаны принципы организации работы с ним.

Для упрощения последующих объяснений введем следующую терминологию: под сервером будет подразумеваться непосредственно сам сервер карт, под клиентом или клиентской частью подразумевается библиотека с открытым исходным кодом, а под сервером карт следует понимать сервер и клиент вместе.

Более детальный анализ сервера карт позволил выявить схему взаимодействия с ним. Схема взаимодействия с сервером карт представлена на рисунке 3.

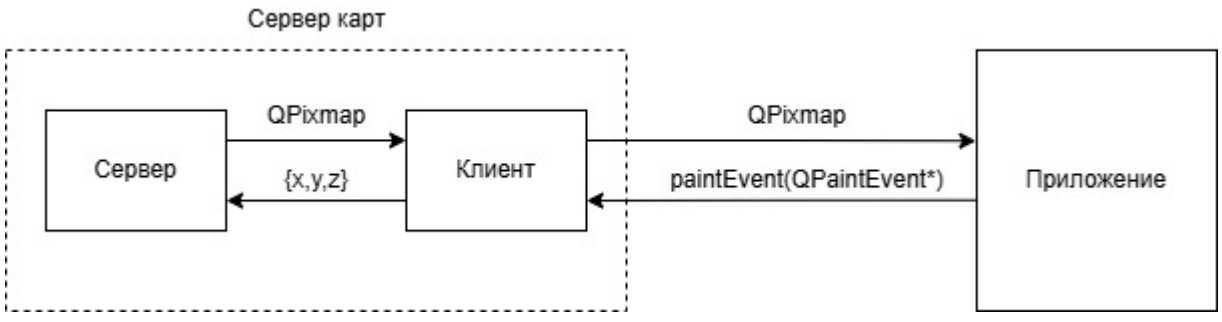


Рисунок 3 – Схема взаимодействия приложения с сервером карт

При определенных взаимодействиях с виджетом внутри приложения: прокрутка колеса мыши, удержание ЛКМ с перемещением курсора, вызывается обработчик события для перерисовки содержимого виджета. Внутри него формируется запрос к клиенту на получение тайлов. В свою очередь клиент вычисляет координаты тайлов, которые необходимо запросить и делает запрос на сервер.

То есть формирование запроса от приложения возникает при таких событиях, как прокрутка колеса мыши, удержание ЛКМ с перемещением курсора. Для решения поставленной задачи данные события не подходят, поэтому необходимо изменить события для формирования запроса на задание 2-х точек объектов или указание точки одного объекта и радиуса области.

При изменении событий для формирования запроса будет изменена клиентская часть сервера карт. Учитывая, что измененная клиентская часть будет необходима только для работы текущего приложения, то было бы правильным решением оставить ее в качестве отдельного модуля внутри самого приложения. В результате этих изменений взаимодействие с сервером карт также изменится. Новая схема взаимодействия приложения с сервером карт представлена на рисунке 4.

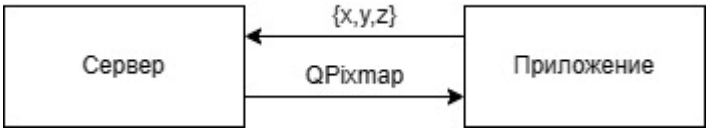


Рисунок 4– Новая схема взаимодействия

2.2 Создание карты

При создании карты следует выполнить обзор аналогов. На основании этого обзора принять решение об используемом формате данных, о

необходимости использования сторонних приложений или программной реализации генерации карты.

2.2.1 Выбор формата данных

В качестве данных для цифровой модели рельефа можно выделить следующие доступные:

1) SRTM;
2) AsterGDEM;
3) Etopo2.

1) SRTM

Наиболее распространенный формат, используется во многих программах.

В качестве формата для записи высот используются файлы с расширением hgt. Используемое здесь слово «HGT» - это просто сокращение от «высота». Каждый файл представляет собой последовательность 16-разрядных целых чисел, задающих высоту каждой ячейки в метрах, расположенных с запада на восток, а затем с севера на юг. Каждый фрагмент данных продолжительностью 3 угловых секунды. Файл содержит 1442401 данных, представляющее сетку размером 1201 ×1201. Формат доработан для разных систем координат.

В старых версиях данные есть только для области между 60 ° с.ш. и 54 ° ю.ш. С 2014 года данные есть для всей области карты.

Обладает высокой точностью за счет того, что съемка производилась при помощи радиолокации.

2) AsterGDEM

Менее распространенный формат, используется в малом числе программ.

ASTER GDEM охватывает поверхность суши между 83° с.ш. и 83° ю.ш. и насчитывает 22,600 фрагментов размером 1°x1°. Распространяется в формате GeoTIFF в географической системе координат (широта/долгота) и разрешением 1 угловая секунда (примерно 30 метров).

Система координат данных WGS84/EGM96.

Точность меньше, чем у srtm, так для использовалась стереофотограмметрическая съемка. И при съемке лесной местности ASTER получает отражение солнечного излучения от кроны дерева, которое затем фотограмметрически обрабатывается для получения ЦМР.

3) Etopo2

Самый нераспространенный формат, потому что является самым неточным из представленных ранее. Его неточность связана с ограничением аппаратных и программных возможностей на момент произведения съемок – 1978г. В последующем доработки не производились.

Результаты сравнения всех рассмотренных данных цифровой модели рельефа сведены в таблицу 1.

Таблица 1 – Результаты сравнения данных цифровой модели рельефа

	SRTM	AsterGDEM	Etopo2
Формат данных	Hgt	GeoTIFF	GeoTIFF
Система координат	Разные	WGS84/EGM96	WGS84
Точность	Высокая	Хорошая	Средняя
Доступность	85°с.ш. и 85°ю.ш	83°с.ш. и 83°ю.ш.	60°с.ш. и 54°ю.ш

В результате сравнения было принято решение об использовании SRTM в качестве данных цифровой модели рельефа. После того, как был выбран формат данных, выполняется генерация карты.

2.2.2 Генерация карты

Любая карта, хранящаяся на сервере, представляет собой набор тайлов для разного увеличения. Пример разбиения карты на тайлы представлен на рисунке 5.

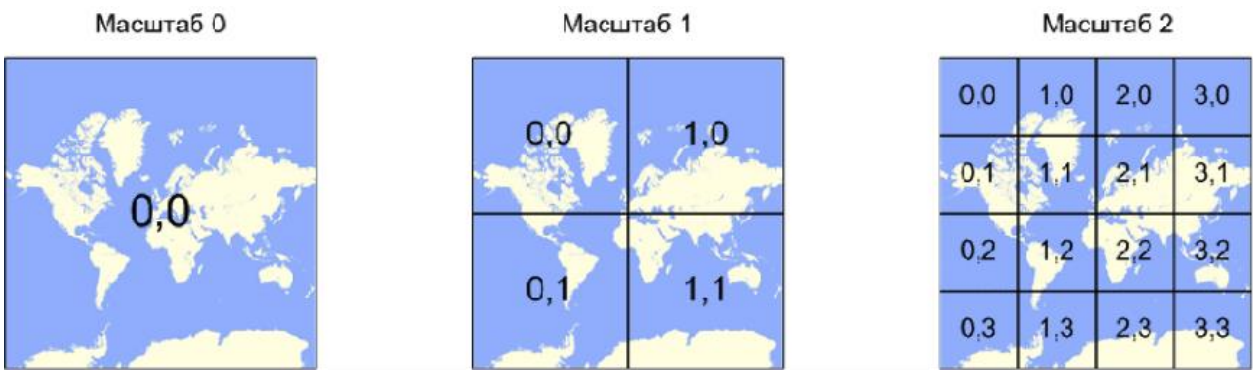


Рисунок 5 – Пример разбиения карты на тайлы

Следовательно, под генерацией карты для использования ее на сервере необходимо понимать создание тайлов карты.

2.2.2.1 Выбор масштаба. Перед тем, как приступить к генерации тайлов необходимо понимать, для какого или для каких масштабов необходимо генерировать тайлы.

Учитывая, что работа будет происходить с использованием файлов данных SRTM, то можно генерировать тайл для каждого значения высоты, то есть для каждых 3” создавать новый тайл. Это возможно выполнить, если масштаб будет равен 17. Достоинством использования такого уровня увеличения является высокая точность при декодировании. Недостатками такого увеличения является большое число тайлов (30 млрд), что требует большого объема памяти для хранения этих данных, а также большого количество времени для их генерации.

Кроме того, использование увеличение равного 17, довольно избыточно. Так как в результате будет генерироваться изображение 256x256, закрашенное

только в один цвет и покрывающее область в 92 м. Пример сгенерированных тайлов представлен на рисунке 6.

Рисунок 6 – Пример сгенерированных тайлов

Можно уменьшить избыточность тайлов, если генерировать тайлы в которых 92м будут приходится на 1 пиксель. То есть каждое значение высоты будет помещаться в 1 пиксель. Это возможно для увеличения равного 11. При таком масштабе возможен компромисс между количеством сгенерированных тайлов и точностью последующего декодирования.

Последующее уменьшение масштаба приведет к снижению точности декодирования. Так как при уменьшении масштаба происходит сжатие изображения и это приведет к «объединению» нескольких соседних пикселей в один.

Так как для выполнения поставленной задачи выводить карту высот для работы пользователю необязательно, а для определения высоты достаточно только увеличения 11. Следовательно, можно сгенерировать тайлы только для

увеличения равного 11.

2.2.2.2 Генерация тайлов. Так как разбиение карты на тайлы является довольно стандартным решением для работы с картами, имеющими высокий уровень увеличения, то существуют приложения, в которых эта функция уже реализована. Как правило, такие приложения уже содержат набор готовых карт. Следовательно, среди этого набора может оказаться и карта высот.

В результате поиска были обнаружены 2 приложения, отвечающие заявленным требованиям (возможность разбиения на тайлы, наличие карты высот): SasPlanet, QGis. Стоит отметить, что оба эти приложения в качестве данных для карты высот используют SRTM.

Далее был выполнен анализ карт высот, которые доступны в приложении. Как итог, карта из приложения SasPlanet не подходит для решения поставленной задачи по 2-м причинам:

1) карта является не полной, так как использует данные от первой версии SRTM;

2) нет сведений о соотношениях между цветом и высотой, а также о методах кодирования высоты цветом.

На рисунке 7 представлена карта, доступная в SasPlanet.

Что касается карты высот, доступной в приложении QGis, то она имеет данные для всей области карты, а также позволяет задавать значения цвета для высот и метод кодирования. Окно выбора цвета для высот и метода кодирования представлены на рисунке 8.

В качестве метода кодирования был использован метод ближайшего соседа.

После того, как заданы цвета для высот и выбран метод кодирования, происходит автоматическое раскрашивание карты. Результат приведен на рисунке 9.

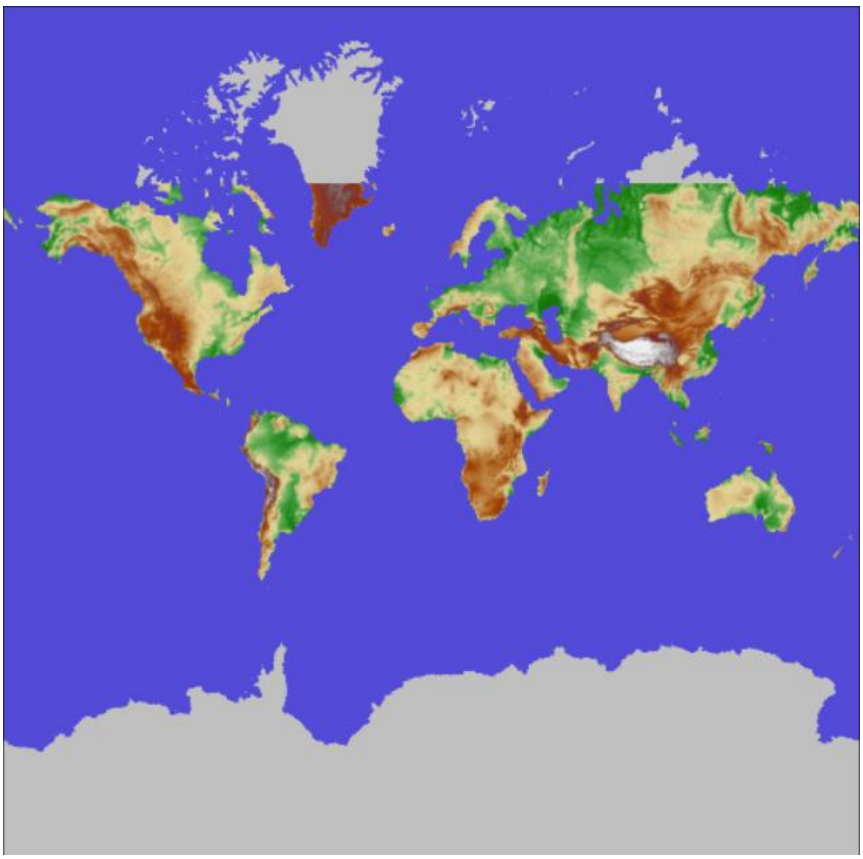


Рисунок 7 - Карта, доступная в SasPlanet

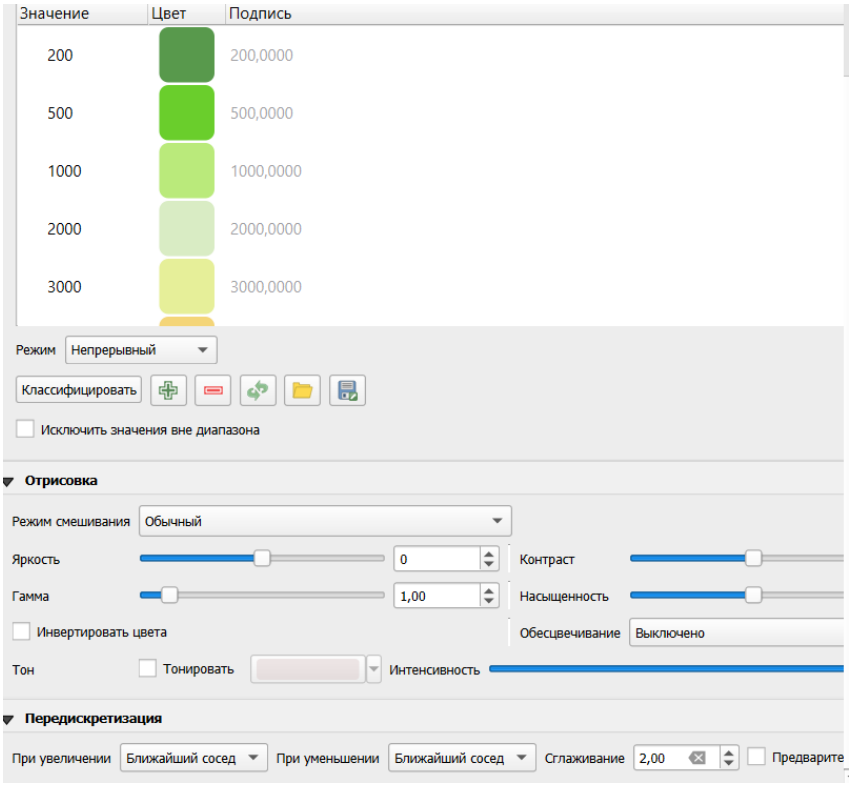


Рисунок 8 – Окно выбора цвета для высот и метода кодирования

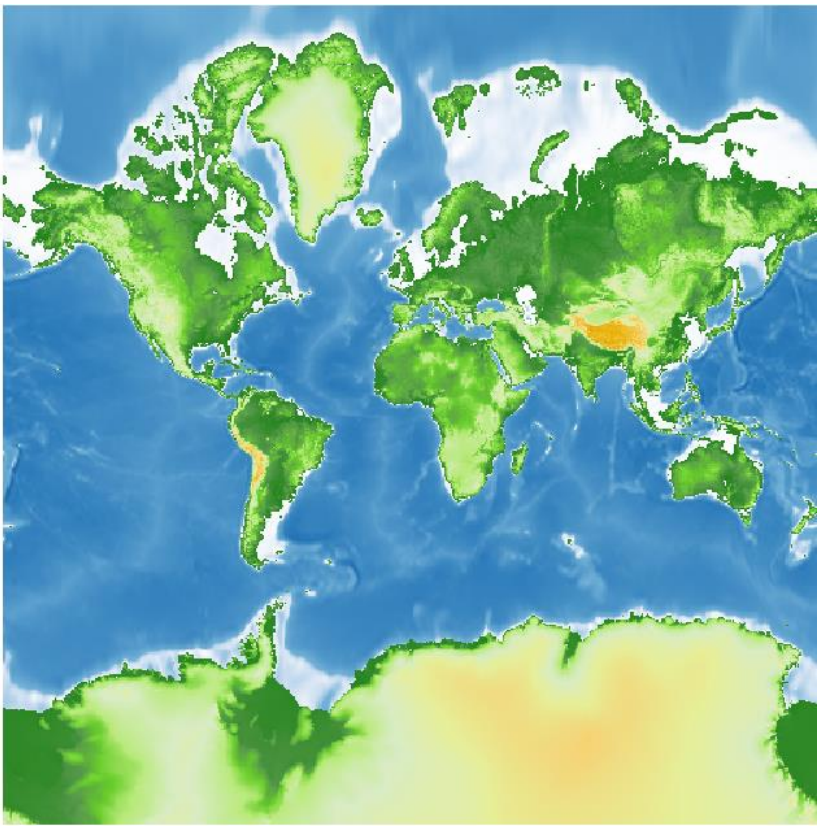


Рисунок 9 – Внешний вид карты после раскраски

После того, как карта готова, можно приступать к разбиению её на тайлы. Для этого, как уже было сказано ранее, в приложении есть специальная функция. При выполнении этой функции начинается генерация 4 194 304 тайла. Каждый тайл генерируется, примерно, 1 секунду. Следовательно, для полного выполнения генерации потребуется 48 дней безостановочной работы. Это довольно много.

Можно попробовать уменьшить количество времени для генерации тайлов, если убрать из генерации тайлы, относящиеся к водному пространству, а потом учесть это при разработке приложения.

К сожалению, QGIS не предоставляет возможность генерации тайлов для некоторой области доступной карты, поэтому необходимо искать альтернативные подходы для решения задачи.

Можно попробовать генерировать тайлы программно. То есть написать скрипт, который бы читал данные из hgt файла, кодировал бы их цветом, вычислял по координатам позицию тайла и пикселя и закрашивал бы этот пиксель в полученный цвет.

Предположим, что обработку каждого пикселя потребуется 1 секунда. Так как тайл имеет размер 256x256, то для полной его генерации потребуется 18 часов. Даже если сделать параллельную обработку 6 тайлов на 6 процессорах, то все равно очень долго. Этот вариант не подходит.

Хотя в QGIS нет возможности сгенерировать тайлы для некоторой области, он позволяет выполнять разбиение на тайлы слоя, который можно создать из файла, содержащих данные высот в допустимом формате. Одним из таких допустимых форматов является hgt файл.

Добавить все имеющиеся файлы высот в QGIS в качестве новых слоев для последующего создания тайлов для каждого из них не предоставляется возможным из-за большого объема этих файлов – 70Гб. При попытке загрузить возникает зависание программы QGIS с последующим завершением работы программы.

Если не получается добавлять все разом, можно добавлять последовательно каждый файл, удаляя предыдущий. Но тут возникает ошибка при создании тайлов: файлы содержат информация для области 1°x1°, а тайлы для масштаба 11 захватывает область размером больше, чем 1°x1°. Для решения этой проблемы достаточно добавлять дополнительные файлы, находящие вокруг необходимого файла. Пример представлен на рисунке 10.

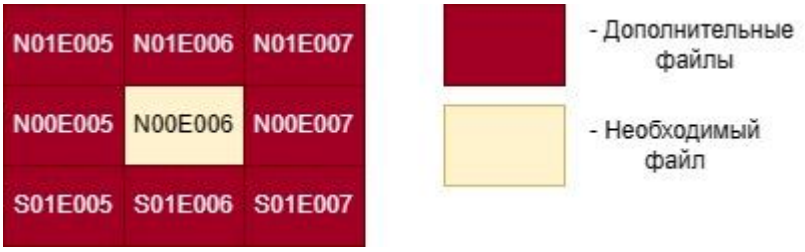


Рисунок 10 – Пример добавления дополнительных файлов

Стоит отметить, что в этом способе генерации карты используется кодирование по методу ближайшего соседа.

Так как добавление, удаление файлов, выбор слоя с последующим разбиением его на тайлы это рутинные задачи, их можно автоматизировать, написав скрипт на python, который будет выполнять все эти функции. Листинг кода этого скрипта приведен в приложении Б.

При таком методе генерации тайла, каждый файл обрабатывается 2 мин. Но из-за того, что таких файлов всего 26 тысяч, то для полной генерации карты потребуется 36 суток безостановочной работы.

Это всё-таки меньше, чем при генерации тайлов для всей области карты. Следовательно, это решение нас устраивает.

2.2.3 Переименование файлов

После того, как все тайлы сгенерированы их необходимо переименовать и переместить в подкаталоги, которые необходимо создать. Это нужно сделать, потому что сервер карт может создавать карту из тайлов только в том случае, если эти тайлы находятся в нужных директориях, иначе он их не сможет обнаружить.

Для примера рассмотрена директория некоторого файла, созданная в QGIS.

…/656/1114.png

656 – координата тайла по x

1114 – координата тайла по y

Этот же файл, но допустимый для сервера карт будет иметь следующую запись:

…/0/x656/1/y1114.png

0 – координата тайла по x деленная на 1024

x656 - координата тайла по x

1 - координата тайла по y деленная на 1024

y1114 - координата тайла по y

Создание папок, переименование файлов и перенесение их в соответствующие папки – это довольно рутинная задача, которая отчасти реализована в total Commander, поэтому можно прибегнуть к написанию скрипта на python. Листинг кода приведен в приложении Б.

2.3 Проектирование приложения

Проектирование приложение выполняется в несколько этапов. Первоначально необходимо разработать структуру приложения. Далее будет рассмотрен процесс работы приложения с целью выявления одинаковой последовательности действий и учета её при дальнейшей разработке. Также необходимо разработать диаграмму потоков данных для выстраивания правильной последовательности действий при формировании запроса тайлов к серверу. Разработаны алгоритмы функционирования.

2.3.1 Разработка структуры приложения

Для демонстрации структуры приложения была разработана диаграмма прецедентов, представленная на рисунке 11.

Как видно из рисунка, пользователь имеет возможность выполнить несколько действий: построить радиотрассу, отобразить область покрытия радиотрассы и выполнить 3D визуализацию рельефа местности, для которой выполнялись расчеты.

Выполнение 3D визуализации включает в себя отображение 3х мерных данных высот, полученных в результате обработки расчетным модулем тайлов, полученных клиентским модулем.

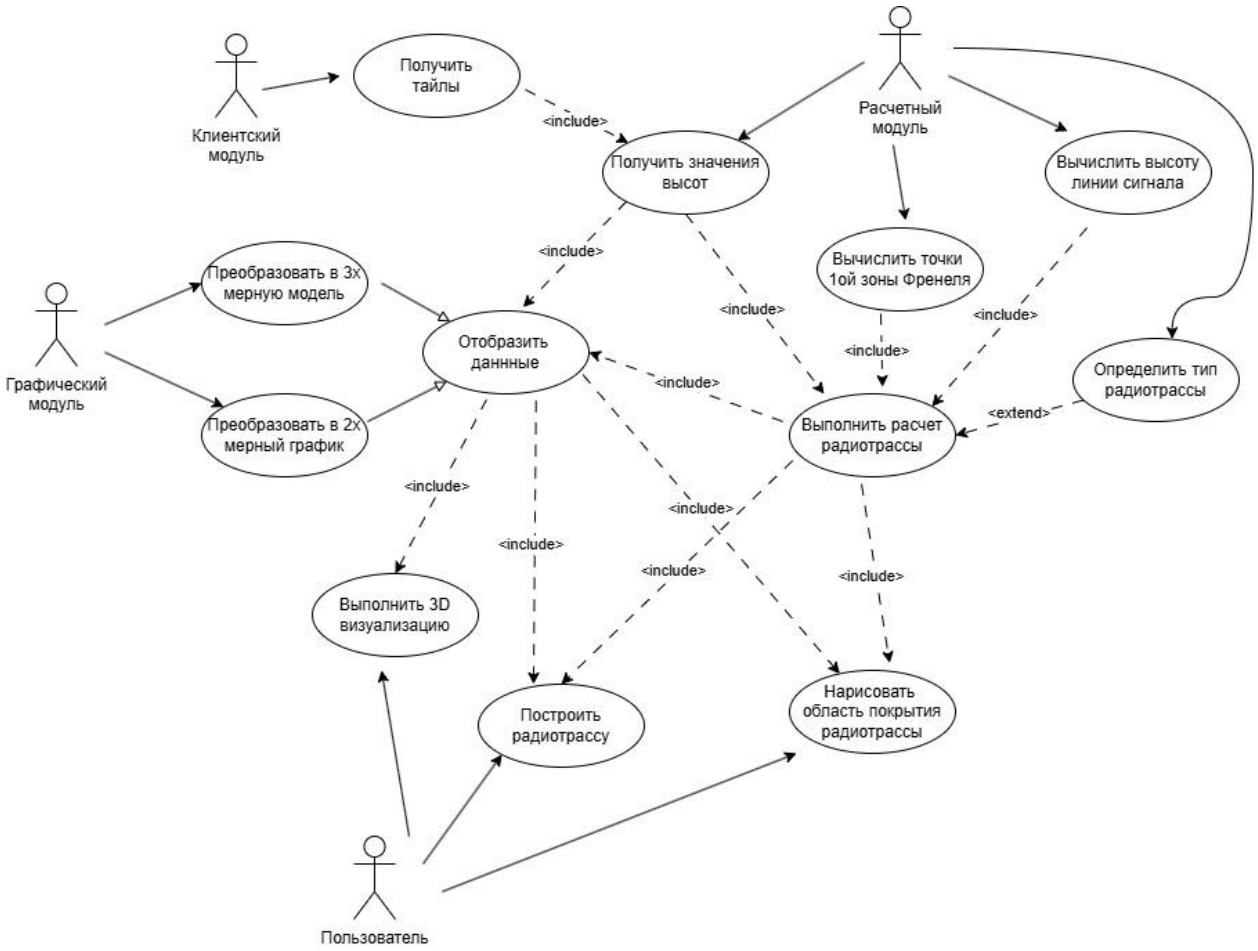


Рисунок 11 – Диаграмма прецедентов

Построение радиотрассы состоит из расчета радиотрассы и отображения полученных результатов в графическом виде. Отображением результатов занимается графический модуль. Расчет радиотрассы состоит из получения значения высот для профиля радиотрассы, вычисление точек зоны Френеля и определением высоты линии сигнала. Всем этим занимается расчетный модуль. Данные для вычисления профиля высот расчетный модуль получает из тайлов, которые были получены клиентским модулем.

Рисование зоны покрытия радиосигналом включает в себя многократное построение радиотрассы с дополнительным расчетом в виде определения типа радиотрассы. Но всеми этими расчетами занимается расчетный модуль. А также многократное закрашивание фрагмента области в цвет согласно вычисленному типу. Этим занимается графический модуль.

2.3.2 Описание процесса работы приложения

Для описания процесса работы приложения была построена диаграмма, описывающая процесс работы приложения и представленная на рисунке 12. Диаграмма необходима для определения последовательности действий при выполнении каждой из функций с целью выявления параллельных и общих частей.

Как видно из рисунка пользователь может выбрать только одно из действий: выполнить построение радиотрассы или 3д визуализации, нарисовать область покрытия радиотрассы. После того, как все необходимые расчеты будут выполнены, полученные результаты функций будет выведены пользователю на экран в некотором виде.

При построении радиотрассы первоначально происходит получение координат объектов, между которыми необходимо организовать канал связи, от пользователя. Далее при помощи этих координат происходит вычисление позиции тайлов с последующим запросом и координат точек маршрута радиотрасы. Эти события являются независимыми, но оба они должны быть выполнены. Расчет профиля высот возможен только в том случае, если до этого был выполнен расчет точек маршрута и получены тайлы. Следовательно, эта функция возможна к выполнению только тогда, когда предыдущие 2 действия выполнились. Расчет линии сигнала связи возможен только после расчета профиля высот, так как для расчета использует данные о высоте рельефа для начальной и конечной точки маршрута. Расчет точек зоны Френеля использует данные высот линии сигнала, поэтому выполняться должен строго после расчеты линии сигнала связи.

Для 3D визуализациии также первоначально необходимо получить координаты объектов или границы области, выполнить расчет позиции тайлов и их запрос. И после этого выполняется преобразование тайлов в 3х мерные данные при помощи специальных алгоритмов.

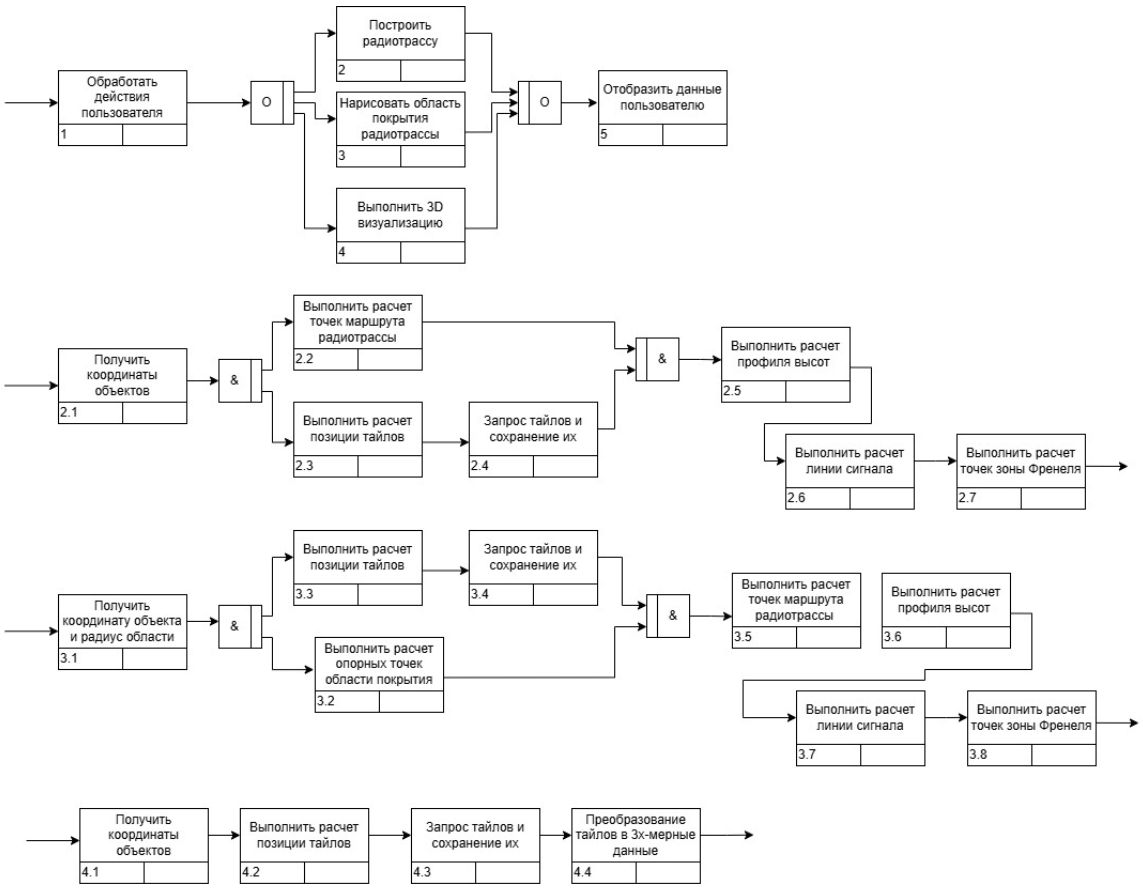


Рисунок 12 – Описание процесса работы приложения

При рисовании зоны покрытия радиосигналом необходимо задать координаты центральной точки и радиус области, для которой будет строиться зона покрытия. После этого происходит вычисление позиции тайлов и их запрос, координат опорных точек, принадлежащих области радиопокрытия для их последующего использования при расчете радиотрассы. Последовательность расчета радиотрассы не изменяется. Добавляется определение типа радиотрассы.

2.3.3 Циркуляция потоков данных

При выполнении описании процесса работы приложения было выявлено, что расчет позиции тайлов и расчет точек маршрута могут выполняться параллельно, то есть полученные от выполнения этих функций данные необходимы вместе для дальнейших вычислений, но при этом сами

они являются не взаимосвязанными. Значит можно сначала запросить тайлы, а потом выполнить расчет координат точек. И тем самым мы сможем уйти от высокой степени связности модулей. Для большей наглядности была разработана диаграмма циркуляции потоков данных при запросе тайлов, представленная на рисунке 13.

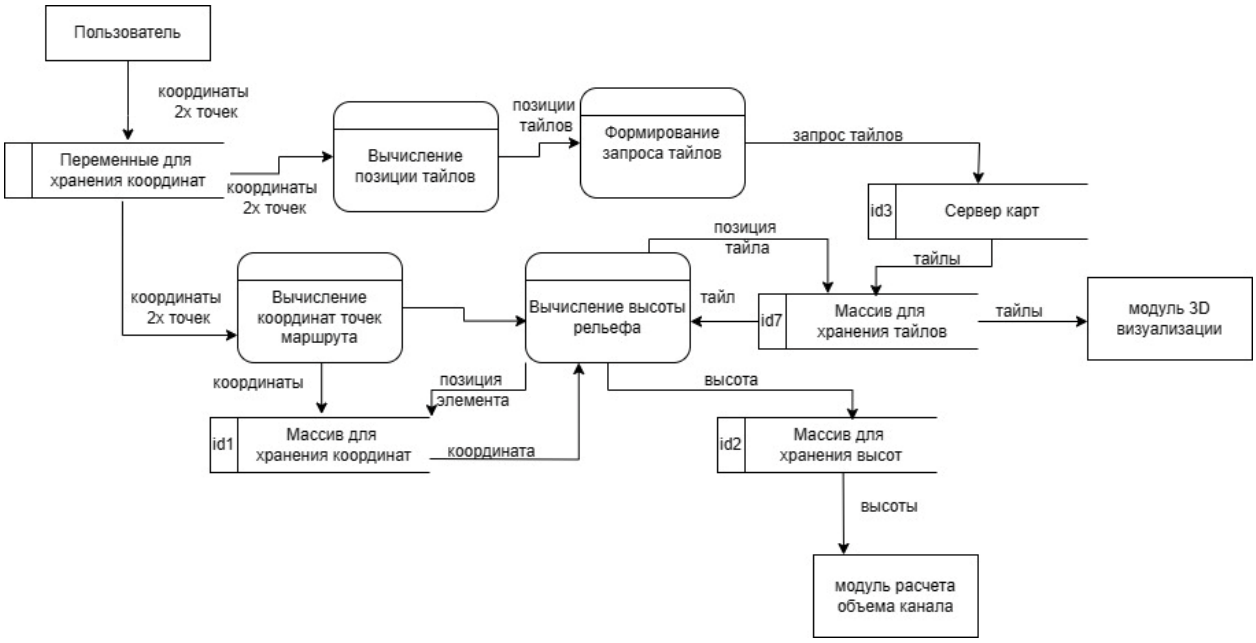


Рисунок 13 – Диаграмма циркуляция потоков данных при запросе тайлов

На рисунке показано, что вычисление позиции тайлов, используемых при расчетах радиотрассы, происходит сразу после того, как пользователь ввел координаты 2х точек. Далее выполняется обращение к серверу карт для запроса всех тайлов. После их получения, они сохраняются в массив.

Когда все тайлы получены и сохранены, выполняется расчет профиля высот. Первоначально происходит вычисление точек маршрута радиотрассы, при расчете которых используются данные 2х точек, которые изначально ввел пользователь. Полученные значения координат точек маршрута записываются в массив. Далее идет вычисление высоты рельефа для каждой из точек. Для этого для каждой точки определяется позиция тайла, выполняется поиск в

массиве тайлов элемента с такими значениями, определяется координаты пикселя внутри тайла и его цвет, полученное значение цвета преобразовывается в значение высоты и записывается в массив высот.

2.3.4 Разработка алгоритмов

2.3.4.1 Расчет радиотрассы. В этом разделе алгоритм расчеты радиотрассы будет рассмотрен более подробно: будут приведены формулы и объяснения необходимые для расчета некоторых данных.

Расчет радиотрассы включает в себя построение графика высот для выбранной местности, построение линии сигнала с учетом высоты антенн, вычисление точек 1-ой зоны Френеля и ее построение.

Построение профиля высот

Для того чтобы построить график высот, необходимо выполнить следующие действия:

1) выбрать 2 точки на карте. Эти точки – координаты объектов, между которыми планируется организовать канал связи;

2) вычислить координаты точек, принадлежащих маршруту радиотрассы. Для вычисления координат точек, находящихся между объектами, можно воспользоваться уравнением прямой. Вычисленная прямая вместе с объектами называется маршрутом радиотрассы. Пример маршрута радиотрассы представлен на рисунке 14.

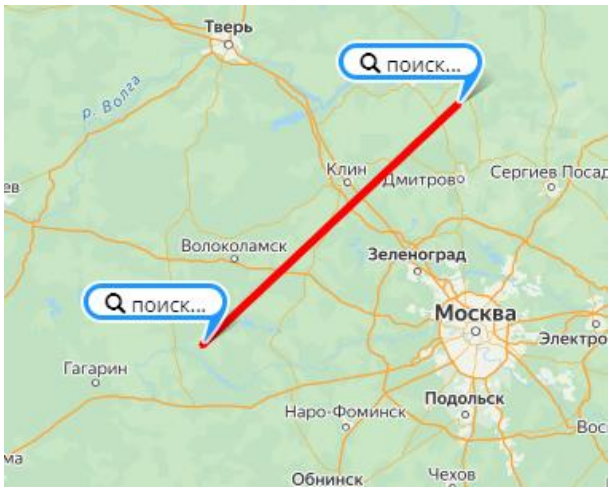


Рисунок 14 – Маршрут радиотрассы

Шаг для вычислений задается из соображений точности, желаемой достичь при расчете радиотрассы: если необходимо достичь высокой точности при построении профиля радиотрассы, то шаг делается небольшим (например, 0.000833); если высокой точности не требуется, тогда и шаг можно брать больше.

Шаг равный 0.000833 был рассчитан следующим образом:

Всего 2^11 = 2048 тайлов.

В каждом тайле 256 пикселей. Следовательно, всего 524 288 пикселей. Так как масштаб карты 11, то на один пиксель приходится 360/524 288 = 0.000686◦ = 2.4719 “

Кроме того, в файле srtm высоты взяты для точек с разницей в 3”.

Получается, что, примерно, на каждый пиксель приходится свое значение высоты. Следовательно, для достижения высокой точности необходимо получать значение высоты для каждой новой координаты с разницей в 3”. Именно поэтому шаг был выбран, как 0.000833;

3) Определить высоту для каждой точки маршрута радиотрассы, включая точки объектов;

4) Построить профиль радиотрассы с учетом высоты точки и отдаленностью этой точки от источника (дистанция в км). Дополнительно стоит отметить, что все высоты откладываются относительно дуги нулевого уровня. Дуга нулевого уровня, в свою очередь, рассчитывается по формуле

$$H(x) = \frac{D^2}{2R} * K * (1 - K), \quad (1)$$

где D — расстояние между объектами, км;

R — радиус Земли — 6371 км;

$K = \frac{X}{D}$ — относительная координата промежуточной точки, 0<K<1;

X — расстояние от 1-ого объекта до промежуточной точки, км.

Расчет точек 1-ой зоны Френеля и её построение

Расчет точек 1-ой зоны Френеля выполняется по следующей формуле

$$r = 17.3 * \sqrt{\frac{1}{f}\frac{D_1 D_2}{D_1 + D_2}}, \quad (2)$$

где f — частота, ГГц;

D_1 — расстояние до нужной вам точки расчета, от первой антенны, км;

D_2 — расстояние до нужной вам точки расчета, от второй антенны, км.

Для отображения зоны Френеля необходимо отложить все вычисленные точки относительно вычисленных точек линии сигнала.

2.3.4.2 Расчет опорных точек. Расчет опорных точек можно производить на основании двух параметров: координаты точки источника сигнала и радиуса зоны покрытия радиосигналом. Дополнительным параметром можно использовать угол поворота радиуса относительно центральной точки. Других параметров нет.

Можно предположить, что расчет опорных точек можно выполнить при помощи следующих формул

$$y = y_0 + R * \sin(a) \quad (3)$$

$$x = x_0 + R * \sin(a), \quad (4)$$

где y_0, x_0 – координаты центра области радиопокрытия;

R – радиус области радиопокрытия;

a – угол поворота радиуса относительно центра.

Но данная формула применима только для плоскости, потому что расчет выполняется по длине отрезка, а не по длине дуги, как это необходимо при работе с географическими координатами. В результате неправильного расчета область видимости приобретает форму овала, что является неверным. На рисунке 15 изображена граница области радиопокрытия при использовании формулы для плоскости.

Рисунок 15 – Границы области радиопокрытия при использовании формулы для плоскости

Как уже было отмечено ранее, правильный расчет возможен в том случае, если он выполняется по длине дуги. Следовательно, необходимо найти формулу для расчета точки через длину дуги.

Эту формулу можно получить из сферической теоремы косинусов. Так как сферическая теорема косинусов позволяет найти длину дуги между точками, находящимися на сфере, то для получения точки через длину дуги, необходимо решить обратную задачу, воспользовавшись следствием из этой теоремы. Формула для расчета координаты точки следующая [4]

$$\varphi_1 = \arcsin\left(\sin(\varphi_1)\cos\left(\frac{d}{R}\right) + \cos(\varphi_1)\sin\left(\frac{d}{R}\right)\cos(\alpha)\right) \quad (5)$$

$$\lambda_2 = \lambda_1 + atan2\left(sin(\alpha)\, sin\left(\frac{d}{R}\right)\, cos(\varphi_1), cos\left(\frac{d}{R}\right) - sin(\varphi_1) * sin(\varphi_2)\right), (6)$$

где φ_1, λ_1- широта и долгота источника сигнала;

φ_2, λ_2 - широта и долгота приемника сигнала;

α - направление от точки А до точки В (азимут);

R - радиус сферы;

d - длина отрезка по ортодомии.

Граница области радиопокрытия при использовании формулы для сферической поверхности представлена на рисунке 16.

Рисунок 16 - Граница области радиопокрытия при использовании формулы для сферической поверхности

2.3.4.3 Определение типа радиотрассы. Существует 4 типа трассы: открытая, полуоткрытая, полузакрытая и закрытая [5]. Тип определяется согласно тому, как профиль высот пересекает область сигнала вместе с зоной Френеля:

1) если профиль выше верхней границы зоны Френеля – закрытая;

2) если профиль выше линии сигнала, но не пересекает верхнюю границу зоны Френеля – полузакрытая;

3) если профиль ниже линии сигнала и пересекает нижнюю границу зоны Френеля – полуоткрытая;

4) если профиль ниже нижней границы зоны Френеля – открытая;

Весь процесс определения типа изображен на схеме алгоритма, представленной на рисунке 17.

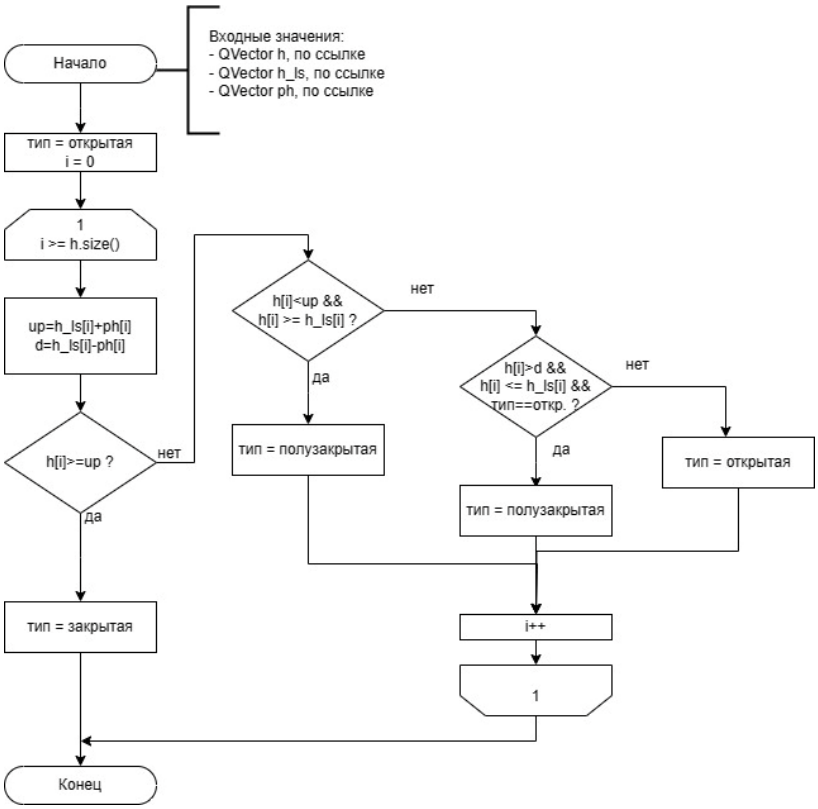


Рисунок 17 – Схема алгоритма определения типа трассы

В качестве входных параметров передаются массивы профиля высот, высот линии сигнала и точек зоны Френеля. Далее происходит определение нижней и верхней границы зоны Френеля для каждой точки профиля высот. Для определения верхней границы производится сложение значение высоты линии сигнала со значением расстояния зоны Френеля в этой точке, для нижней – вычитание. Далее происходит сравнение высоты профиля и верхней границы зоны Френеля. Если профиль отказывается выше, то трассе присваивается закрытый тип и алгоритм заканчивает свою работу. Если же нет, не пересекает, то выполняется следующая проверка. И так до тех пор, пока тип трассы не станет закрытым или пока не будут рассмотрен весь профиль высот.

2.3.4.5 Декодирование цвета в высоту. Для декодирования высоты по значению цвета был использован алгоритм по методу ближайшего соседа. Этот метод был использован из-за того, что карта генерировалась с его использованием. Так как генерация карты была выполнена при помощи сторонней программы, в которой этот алгоритм был четко задан, то и декодировать придется с его использованием.

Этот метод подразумевает, что выбираются несколько значений высот, которые будут являться границами. Для каждого такого значения задается цвет. Далее считывается значение высоты из файла. Для того, чтобы вычислить значение цвета, соответствующей этой высоте, необходимо выполнить следующие действия:

1) определить между какими заданными границами высотами располагается полученная высота;

2) вычислить коэффициент «отдаленности» по формуле

$$p = \frac{h - h_1}{h_2 - h_1}, \tag{7}$$

где h – полученная высота;

h_1–нижняя граница интервала высот;

h_2 – верхняя граница интервала высот;

3) вычислить значения цвета по формуле

$$r = (1 - p) * r_1 + p * r_2 \tag{8}$$

$$g = (1 - p) * g_1 + p * g_2 \tag{9}$$

$$r = (1 - p) * b_1 + p * b_2, \tag{10}$$

где r_2, g_2, b_2– цвет для верхней границы интервала высот;

r_1, g_1, b_1 – цвет для нижней границы интервала высот.

Следовательно, для декодирования нужно выполнить последовательность действий в обратном порядке.

2.3.4.6 Создание изображения из массива тайлов. Массив тайлов – одномерный массив. Но при переложении на плоскость тайл должен иметь 2-х мерные координаты. Поэтому изображение можно представить, как двумерный массив. Следовательно, необходимо разработать алгоритм для преобразования номера тайла в массиве в пиксельные координаты начала тайла внутри изображения.

Процесс формирования массива тайлов представлен на рисунке 18.

0	3	6	9
1	4	7	10
2	5	8	11

Рисунок 18 – Процесс формирования массива тайлов

Более подробно процесс формирования массива тайлов рассмотрен в 3 разделе пояснительной записки.

Следовательно, для определения координат тайла внутри изображения необходимо вычислить номер столбца и строки, где должен располагаться тайл.

Для этого можно воспользоваться следующей формулой

$$numberRow = \frac{numberTile}{diff\_y} \quad (11)$$

$$numberStr = numberTile - numberRow * diff\_y, \quad (12)$$

где $diff\_y$ – кол-во тайлов в одном столбце;

$numberRow$ – номер столбца тайла внутри изображения;

$numberStr$ – номер строки тайла внутри изображения;

$numberTile$ – номер тайла в массиве тайлов

Вычислить $diff\_y$ можно по следующей формуле

$$diff\_y = tiles[tiles.size - 1].pos_y - tiles[0].pos_y + 1, \quad (13)$$

где $tiles$ – массив тайлов.

А для получения координат достаточно умножить полученные значения столбца и строки на 256 – размер тайла в пикселях.

Вывод

В результате выполнения текущего этапа работ была спроектирована структура приложения, состоящая из нескольких модулей: клиентского, расчетного и графического. Клиентский модуль необходим для реализации взаимодействия с сервером, так как в ходе анализа сервера карт было выяснено, что существующая клиентская часть не подходит для решения поставленной задачи. Разработанные диаграммы, описывающие процесс работы приложения и циркуляции потоков данных при выполнении запроса тайла, показывают невысокую степени связности между модулями. Для расчетного модуля были разработаны алгоритмы, выполняющие декодирование высоты по цвету пикселя, расчет опорных точек для области радиопокрытия и т.д.

3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

В ходе выполнения программной реализации необходимо разработать классы и методы для клиентского и расчетного модулей [1].

3.1 Клиентская часть

Для того, чтобы изменить событие формирования запроса, а также вынести клиентскую часть в модуль приложения, необходимо выполнить детализацию структуру клиента и связи приложения с ним.

3.1.1 Детализация структуры клиента

Под детализацией следует понимать рассмотрение последовательности формирования запроса и получения ответа. Для наглядности была разработана диаграмма кооперацій, представленная на рисунке 19.

Класс mapWidget является расширением класса mapControl, являющегося наследником от класса QWidget, для возможности отрисовки карты на виджете и взаимодействия с ней.

Все необходимые методы для обработки действий пользователя при взаимодействии с картой описаны в классе mapControl. Также в этом классе сделано переопределение метода paintEvent, позволяющее выполнять отрисовку изображения (карты) на виджете (mapWidget).

Получается, что при определенных манипуляциях пользователя с картой, нарисованной на mapWidget, возникает событие paintEvent, принадлежащее классу mapControl. В свою очередь mapControl, обрабатывая это событие, создает экземпляр класса QPainter и делает обращение к классу layerManager, передавая в качестве параметра ссылку на созданный экземпляр класса QPainter.

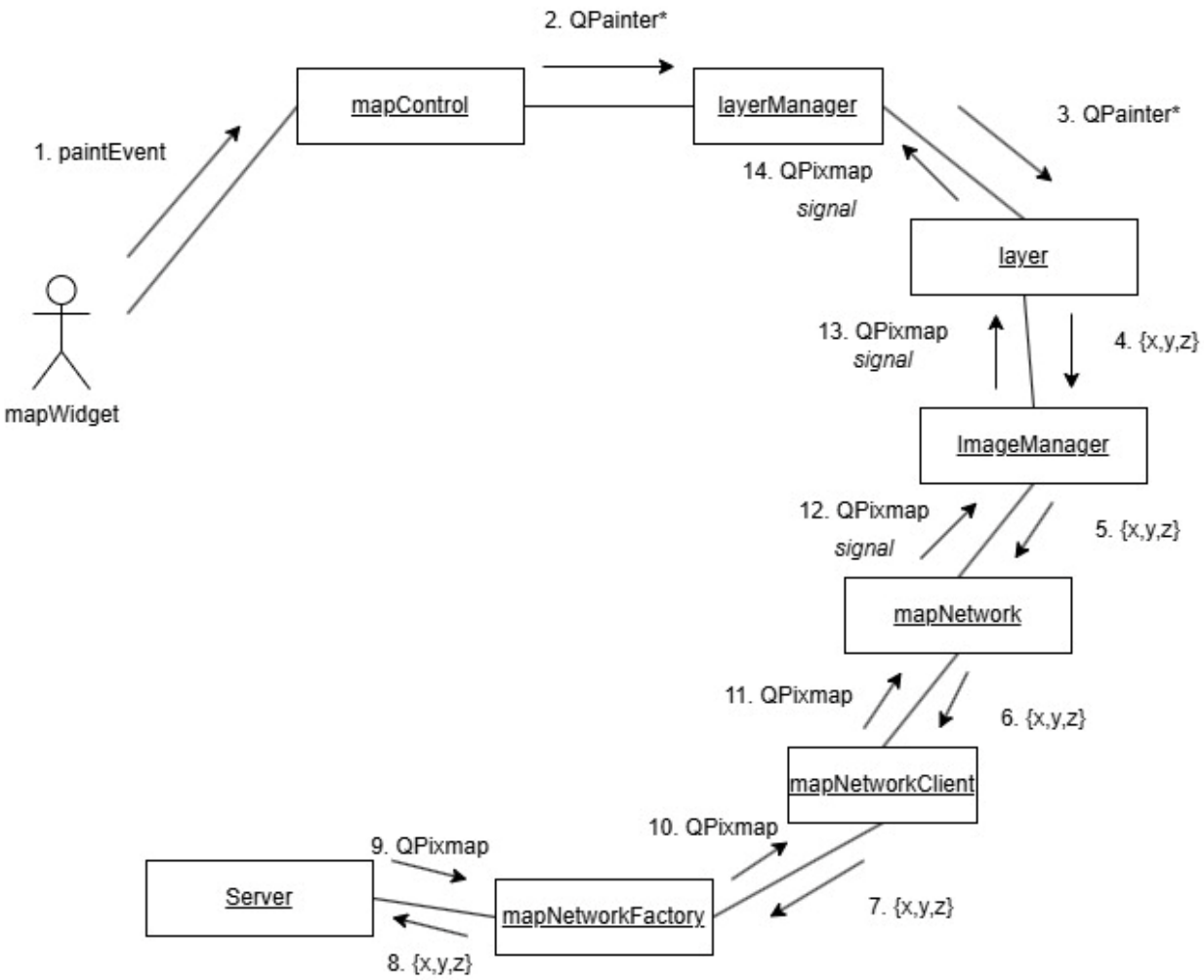


Рисунок 19 – Диаграмма кооперацій

В layerManager происходит управление работой всеми слоями карты. Это необходимо в случае, если карта состоит из нескольких слоев. Например, гибридная карта создается из 2-х слоев: снимков спутника и osm-карты, где обозначены границы государств и подписаны названия городов.

При получении управления layerManager создает еще один экземпляр класса QPainter и делает обращение к классу layer, передавая в качестве параметров ссылку на экземпляр класса и координаты центра карты, которые хранятся и рассчитываются в layerManager.

Класс layer занимается отрисовкой изображения на слое. И при обращении к нему он делает обращение к классу imageManager для получения

изображения. Но перед тем, как выполнить обращение происходит расчет позиции тайлов через координаты центра карты и координаты позиции курсора. Последние получают из класса mapAdapter. Полученные значение позиции тайлов передается в качестве параметров при обращении к imageManager.

ImageManager выполняет всю работу, связанную с обработкой изображений, полученных от сервера, а также выполняет запрос к mapNetwork, передавая позицию тайла в параметрах.

Через классы mapNetworkClient и mapNetworkFactory происходит обращение к серверу с целью получения тайла с определенными координатами.

После того, как ответ от сервера получен, класс layer принимает изображение в качестве возвращаемого результата при выполнении обращения к imageManager.

При получении изображения layer вычисляет координаты начала тайла внутри изображения, отображающего ту часть карты, которая должна быть видна пользователю. Также выполняется отрисовка полученного тайла на изображении части карты согласно вычисленным координатам. Отрисовка происходит при помощи экземпляра класса QPainter, ссылку на которой была получена от layerManager.

Запрос тайлов от layer к imageManager выполняется до тех пор, пока не будут получены все тайлы, позиции которых были вычислены.

Далее LayerManager получает изображение с нарисованном на нем тайлами. Далее происходит отрисовка полученного изображения части карты пользователю. Это происходит через экземпляр класса QPainter, переданный по ссылке от класса mapControl.

MapControl в конце выполнения обращения получает изображение, отрисованное на виджете и доступное пользователю.

В результате выполнения детализации клиентской части сервера карт стало очевидно, что существующая клиентская часть подходит для задачи отображения области карты для предоставления пользователю возможности выбора точек, где будут располагаться объекты, но не походит для задачи получения высоты для координаты, так как не имеет алгоритма для преобразования координаты в позицию тайла, а также не имеет возможности сохранения позиции тайла для последующей фильтрации изображений, полученных от сервера. Следовательно, необходимо выполнить разработку клиентского модуля с учетом выявленных требований к нему.

3.1.2 Разработка клиентского модуля

Так как в рисовании изображения на виджете нет необходимости, то делать вложенные запросы с получением изображения в качестве возвращаемого значения, как это было сделано в клиентской части, не нужно. Можно передавать изображения через сигналы, описанные внутри каждого класса.

В классе imageManager необходимо добавить методы для сохранения позиции тайла для последующего выделения нужного тайла путем сравнения сохраненных значений координат тайла с теми, которые пришли вместе с изображением. Кроме того, необходимо добавить метод, реализующий обработку пустых изображений. О необходимости добавления этого метода было рассказано в главе, посвященной генерации карты. Для добавления новых методов необходимо создать новый класс imageManager_req, отнаследовав его от imageManager.

Далее необходимо изменить алгоритм определения позиции тайла в классе layer, чтобы он мог работать с географическими координатами.

Алгоритм преобразования координат в позицию тайла заключается в преобразовании географических координат сначала в меркаторские, а затем в пиксели.

Преобразование в меркаторские осуществляется по формуле. Формула зависит от вида проекции. Вид проекции в свою очередь зависит от того, как представляют Землю: в виде сферы, в виде эллипса или в виде геоида [2]. Существует еще одна проекция, в которой Землю представляют в виде сферы, но в которой упрощена формула расчета – web Mercator. Именно ей необходимо воспользоваться. Формула для преобразования географических координат в меркаторские следующая

$$y = \frac{1 - \frac{\log\left(\frac{\tan\left(\frac{\pi}{4} + lat\right)}{2}\right)}{\pi}}{2} \quad (14)$$

$$x = \frac{lon + 180}{360}, \quad (15)$$

где lat – значения широты в градусах;

lon – значения долготы в градусах.

Получается, что в методе класса Layer, который выполнял обращение к ImageManager необходимо изменить функция расчета позиции тайла, добавить вызов метода сохранения позиции тайла у ImageManager. Кроме того, для обращения ко всем методам необходимо создать экземпляр класса наследника ImageManager. Следовательно, для класса Layer также необходимо создавать новый дочерний класс, в котором выполнять переопределение метода.

Так как LayerManager взаимодействует с методом класса Layer, который был переопределен, необходимо переопределять и метод класса LayerManager. Раз происходит изменение метода, то из него можно убрать отрисовку изображения при помощи QPainter, так как для задачи этого не требуется.

По той же причине, по которой было выполнено переопределение layerManager, необходимо выполнить наследование класса mapControl, в котором выполнить переопределение метода обращения к классу Layermanager.

Так как напрямую изменять методы в классах нельзя, то необходимо выполнить наследование от классов и переопределять методы внутри созданных классов.

Диаграмма классов представлена на рисунке 20.

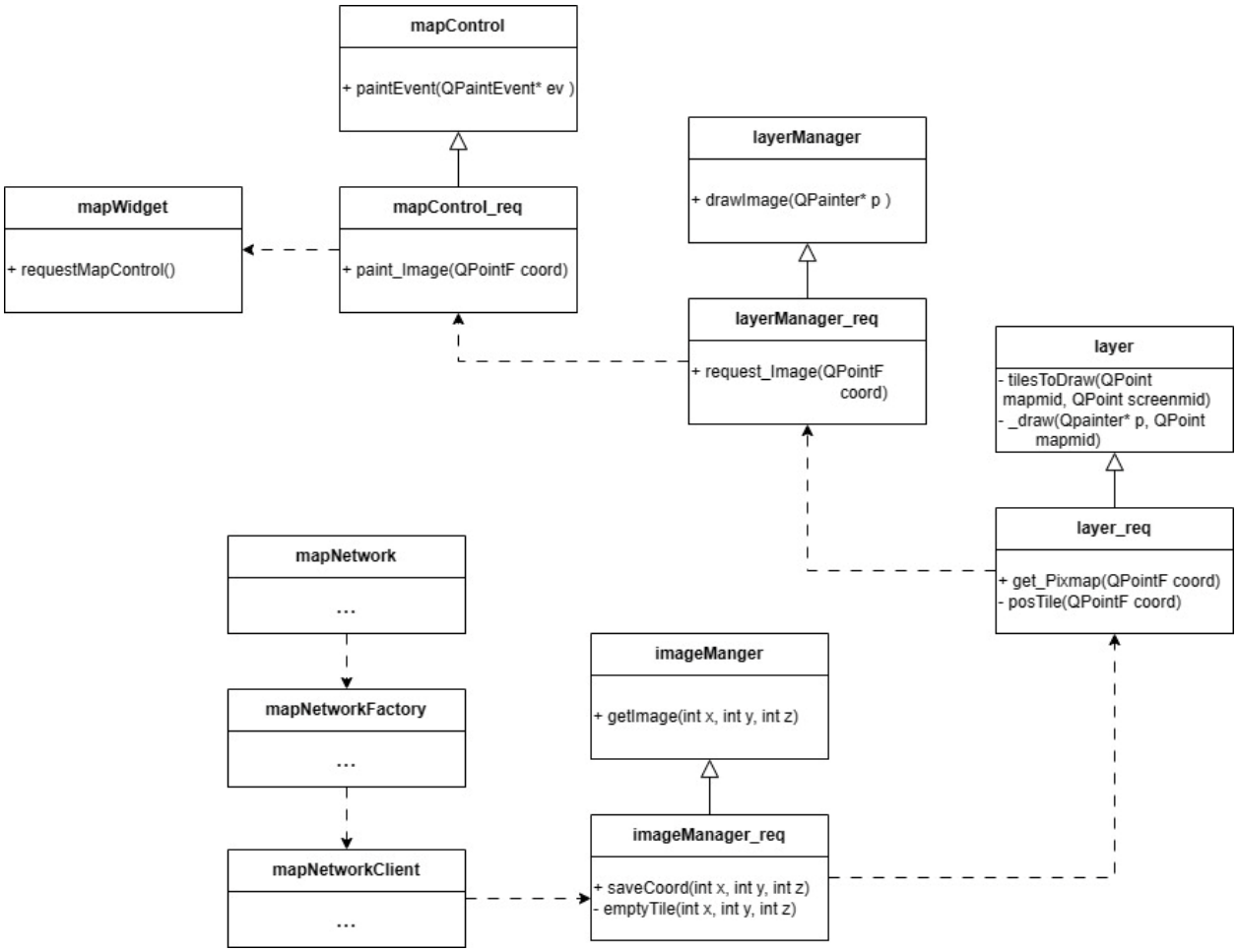


Рисунок 20 – Диаграмма классов

Диаграмма кооперaций, описывающая новую последовательность действий при запросе тайла представлена на рисунке 19, где курсивом выделены изменения в последовательности.

Листинг кода клиентского модуля приведен в приложении В.

3.1.3 Тестирование

При анализе работы сервера карт, было упомянуто, что клиентская часть имеет графический интерфейс, позволяющий просматривать позиции тайла. Воспользуемся этой функцией для проверки корректности алгоритма преобразования географических координат в позицию тайла, а также корректности формирования запроса и получения ответа.

Проверка заключается в следующем: в программе задаются какие-то географические координаты; далее они преобразуются в запрос на сервер, проходя через всю последовательность действий; сервер в качестве результата вернет изображение, которое можно сохранить, как x_y.jpg. После этого можно сравнить координаты тайла, указанные в клиентской части, с теми которые получились, также можно визуально сравнить изображения на предмет сходства.

Пример выполнения проверки для 25° 53’ 15” ю.ш. и 113° 44’ 53” з.д. представлен на рисунке 21.

а) тайл, полученный от сервера с помощью разработанного алгоритма

б) тайл, который должен был получиться

Рисунок 21 – Результат выполнения проверки

3.2 Расчет радиотрассы

3.2.1 Разработка классов для расчета радиотрассы

Для выполнения расчета радиотрассы были разработаны классы.

Диаграмма классов представлена на рисунке 22.

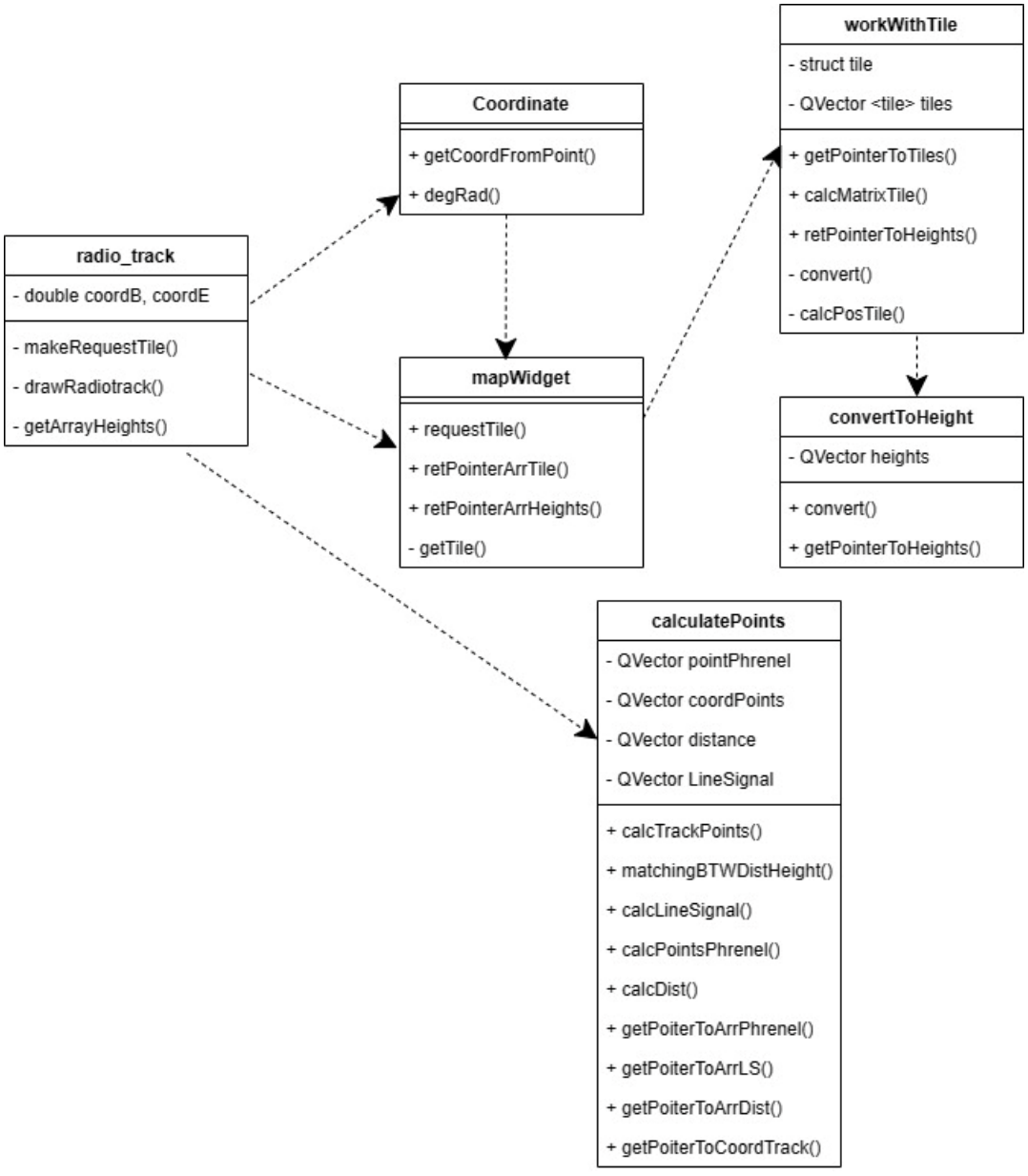


Рисунок 22 – Диаграмма классов

Как видно из рисунка, все методы, а также переменные для хранения данных, относящихся к расчетам различных точек, вынесены в отдельный класс – calculatePoint. Всё что касается работы с координатами: выделение градусов, минут и секунд из дробного значения; перевод градусы в радианы и тд, также вынесены в отдельный класс. Вся работа с тайлами тоже вынесена в отдельный класс. Помимо методов для работы с тайлами в класс была

добавлена структура tile, имеющая следующие описание:

```
struct tile
{
   int pos_x;
   int pos_y;
   QPixmap pix_map:
};
```

Более подробно стоит рассмотреть функцию calcMatrixTile, выполняющую расчет массива тайлов. Схема алгоритма для этой функции представлена на рисунке 23.

Как видно из рисунка в качестве входных данных используются географические координаты объектов - источника и приемника. Далее эти координаты передаются в некоторую функцию, которая переводит их в позицию тайла для каждого объекта.

При вычислении min, max позиции тайлов происходит сравнение вычисленных позиций с последующим присваиванием min значения источнику, а max – приемнику.

Например, в результате перевода позиция тайла для источника равна 1214 по x и 741 по y, а для приемника – 1218 по x, 738 по y. Этот пример приведен на рисунке 24.

После сравнения позиций тайлов станет известно, что минимальная позиция тайла по x равна 1214, а по y – 738. Эти значения позиций получит источник, а другие – приемник, рисунок 25.

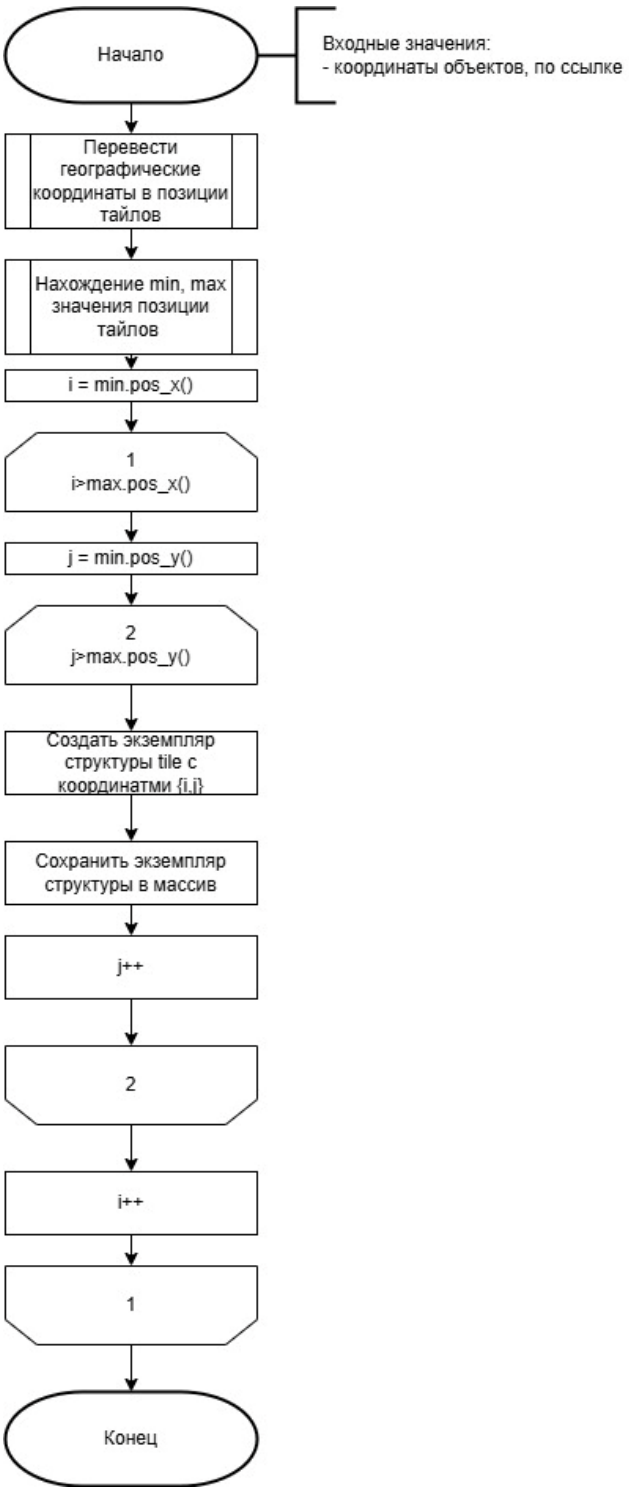


Рисунок 23 – Схема алгоритма для функции calcMatrixTile

Рисунок 24 – Координаты позиций тайлов объектов до сравнения

Рисунок 25 - Координаты позиций тайлов объектов после сравнения

Далее выполняется перебор всех позиций тайлов в диапазоне, ограниченном значениями источника и приемника, при помощи цикла. Новые позиции тайлов сохраняются в экзмепляр структуры, который сохраняется в массив.

Листинг кода, содержащий описание функции приведен ниже:

```
void workWithTile::calcMatrixTiles(const QPointF& coord_begin,
                                   const QPointF& coord_end, int zoom)
{
// перевод географических координат точки в позицию тайла
   QPoint coord_tile_begin = convertCoordDisplayToPosTile(coord_begin, zoom);
    QPoint coord_tile_end = convertCoordDisplayToPosTile(coord_end, zoom);
//нахождение минимальной позиции тайла
    findMinCoord(coord_tile_begin, coord_tile_end);
//перебор всех позиций тайлов в пределах вычисленной
```

```
for (auto i = coord_tile_begin.x(); i <= coord_tile_end.x(); i++)
  {
    for (auto j = coord_tile_begin.y(); j <= coord_tile_end.y(); j++)
    {
      tile new_tile;
      new_tile.pos_x = i;
      new_tile.pos_y = j;
      saveTile(new_tile); }}}
```

3.2.2 Тестирование

Оценка погрешности будет выполняться в 3 этапа:

1) сравнение профилей радиотрассы, один из которых построен в разработанном приложении, а другой – в radio mobile;

2) сравнение высоты некоторой точки маршрута радиотрассы, полученной в приложении с высотой этой же точки, но полученной в radio mobile;

3) сравнение высоты некоторой точки маршрута радиотрассы, полученной в приложении с высотой этой же точки, но заданной в файле hgt.

3.2.2.1 Сравнение профилей. На рисунке 26 представлен профиль радиотрассы, построенный в разработанном приложении

На рисунке 27 представлен профиль радиотрассы, построенный в radio mobile.

Стоит отметить, что при построении радиотрассы в radio mobile выполняется сглаживание, из-за чего может сложиться впечатление о несхожести профилей, но если попробовать применить сглаживание к профилю, построенному в приложении, как показано на рисунке 28, то можно сделать вывод о схожести профилей.

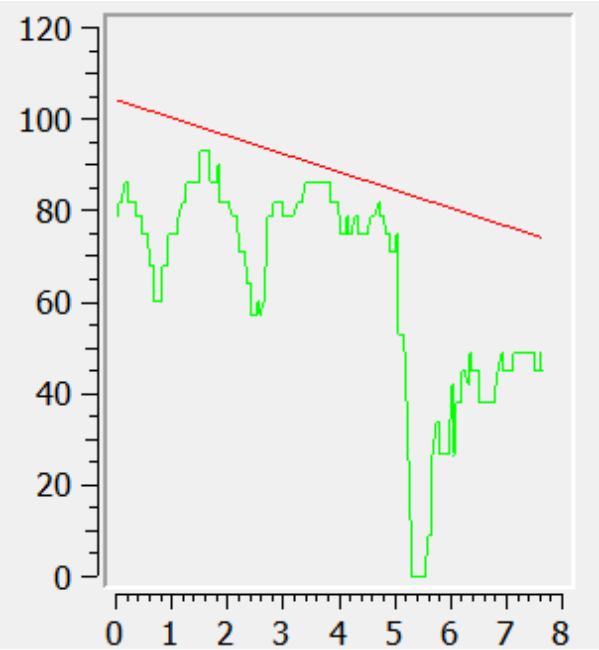


Рисунок 26 – Профиль радиотрассы, построенный в приложении

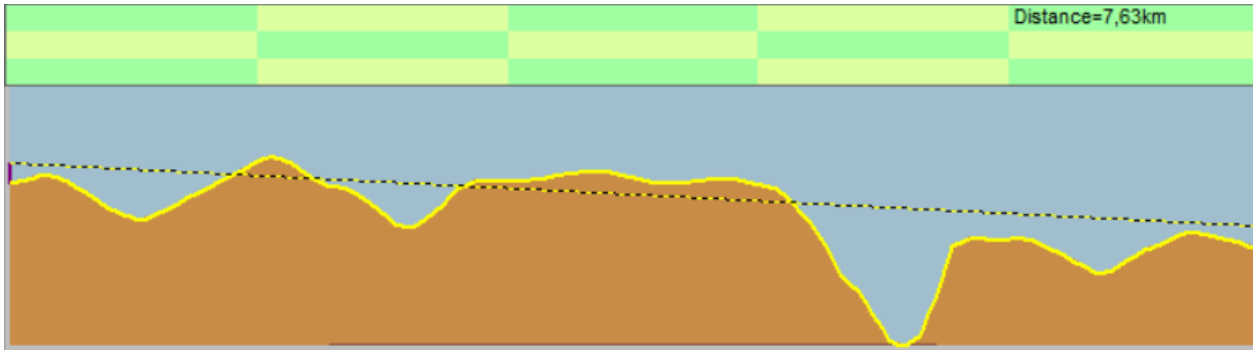


Рисунок 27 – Профиль радиотрассы, построенный в radio mobile

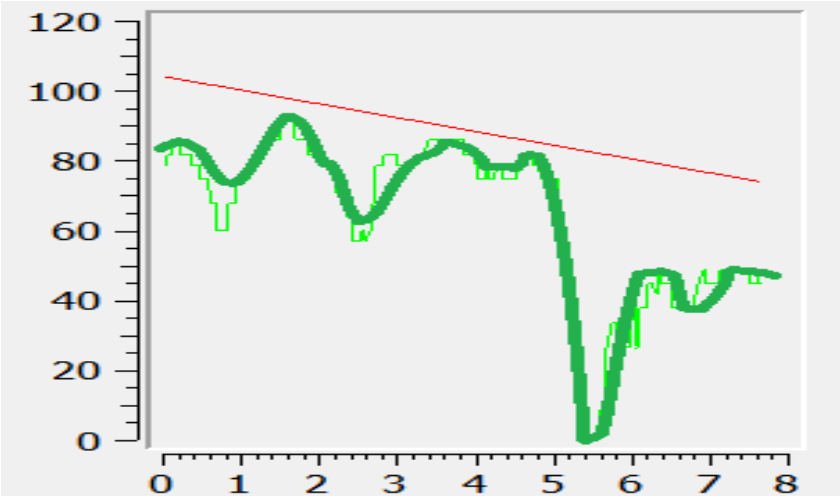


Рисунок 28 – Применение сглаживания к профилю радиотрассы

3.2.2.2 Сравнение высот. Значение высоты будет определяться для точки с координатами 44° 34’ 52” с.ш. и 33° 30’ 17” з.д.

Значение высоты, полученное в приложении: 86 м

Значение высоты, полученное в radio mobile: 84 м

Значение высоты, полученное из файла: 87 м

3.3 Область радиопокрытия

Расчет зоны покрытия радиосигналом является ничем иным, как многократным расчетом радиотрассы для разных координат точек приемника. В разработанном алгоритме запрос тайлов происходит для каждого нового расчета радиотрассы. Такой подход приведет к дублированию тайлов при вычислении области радиопокрытия, поэтому стоит сделать перегрузку уже имеющейся функции для расчета массива тайлов, изменив набор параметров и отредактировав саму реализацию.

Для определения позиций тайлов необходимо вычислить географические координаты самых крайних точек области: верхней, нижней, левой и правой. Затем перевести полученные координаты позиций тайлов, сравнить полученные значения координат для левой и верхней точек, а также для правой и нижней точек с целью нахождения максимальных и минимальных значений. В результате нахождения min и max левая точка будет содержать в себе значение левого верхнего угла матрицы тайлов, а правая точка – значение правого нижнего угла матрицы тайлов. После определения диапазона значений тайла по x и по y происходит перебор всех позиций тайлов, входящих в этот диапазон, с последующим сохранением значений в структуру.

Код приведен ниже:

```
void workWithTile::calcMatrixTiles(const QPointF& coord_right,
                                   const QPointF& coord_up,
```

```
                                const QPointF& coord_left,
                                const QPointF& coord_down, int zoom)
{
    QPoint coord_tile_left = convertCoordDisplayToPosTile(coord_left, zoom);
    QPoint coord_tile_right = convertCoordDisplayToPosTile(coord_right, zoom);
    QPoint coord_tile_up = convertCoordDisplayToPosTile(coord_up, zoom);
    QPoint coord_tile_down = convertCoordDisplayToPosTile(coord_down, zoom);
    findMinCoord(coord_tile_left, coord_tile_up);
    findMinCoord(coord_tile_right, coord_tile_down);

    for (int i = coord_tile_left.x(); i <= coord_tile_down.x(); i++)
    {
        for (int j = coord_tile_left.y(); j <= coord_tile_down.y(); j++)
        {
            tile new_tile;
            new_tile.pos_x = i;
            new_tile.pos_y = j;
            saveTile(new_tile);
        } }}
```

Ранее уже было сказано, что закраска фрагмента области подразумевает не только саму закраску фрагмента, но и первоначальный расчет координат границ этого фрагмента.

Так как закраска фрагмента связана с определением типа трассы, определение которого в свою очередь связано с расчетом радиотрассы. А расчет радиотрассы включает в себя определение точек маршрута, то можно предположить, что границы фрагмента можно получать из координат последней и предпоследней точек маршрута радиотрассы. Пример получения границ фрагмента представлен на рисунке 29.

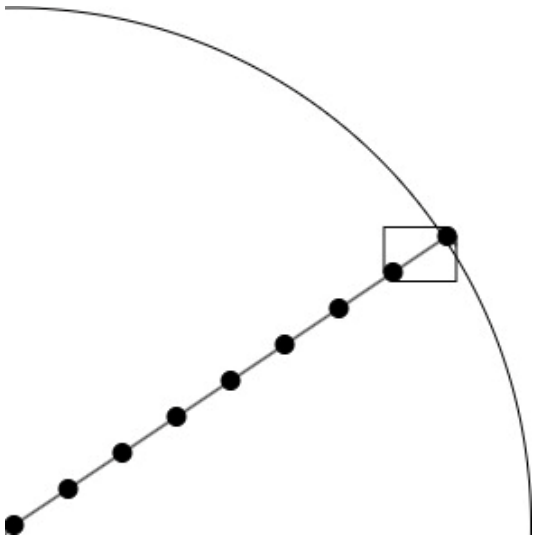


Рисунок 29 – Пример получения границ фрагмента

Результат применения данного алгоритма для расчета области радиопокрытия в радиусе 10 км представлен на рисунке 30.

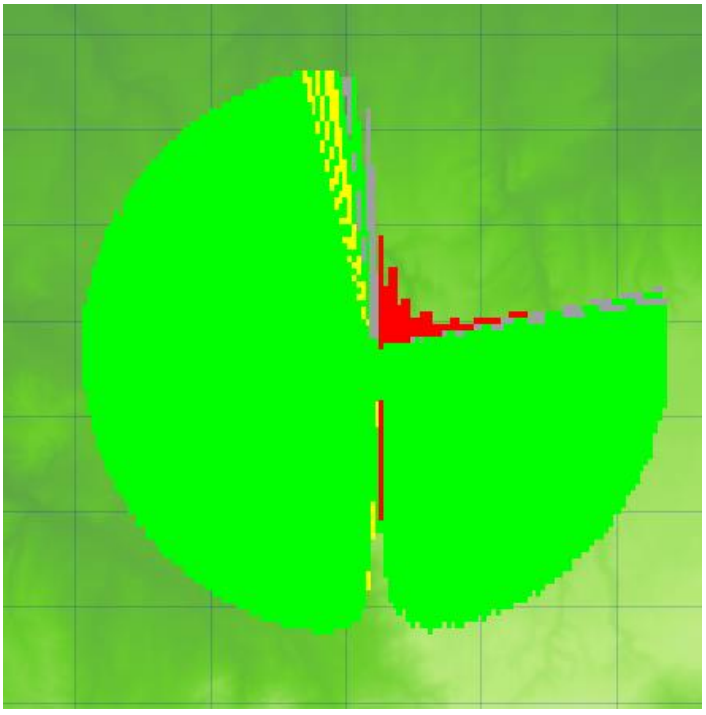


Рисунок 30 – Результат применения алгоритма для области в радиусе 10 км

Из рисунка видно, что при небольшом значении радиуса область закрашивается равномерно. Но при увеличении значения радиуса область уже

начинает закрашиваться неравномерно: появляются просветы.

Пример закраски области с просветами при радиусе в 40 км представлен на рисунке 31.

Появление просветов связано с тем, что вычисленные опорные точки и точки маршрута для этих точек разбивают область покрытия на неодинаковые фрагменты. Для решения проблемы с появлением просветов при закраске, необходимо выполнить разбиение области на одинаковые фрагменты. Следовательно, предложенные первоначально алгоритм не подходит и его требует изменить.

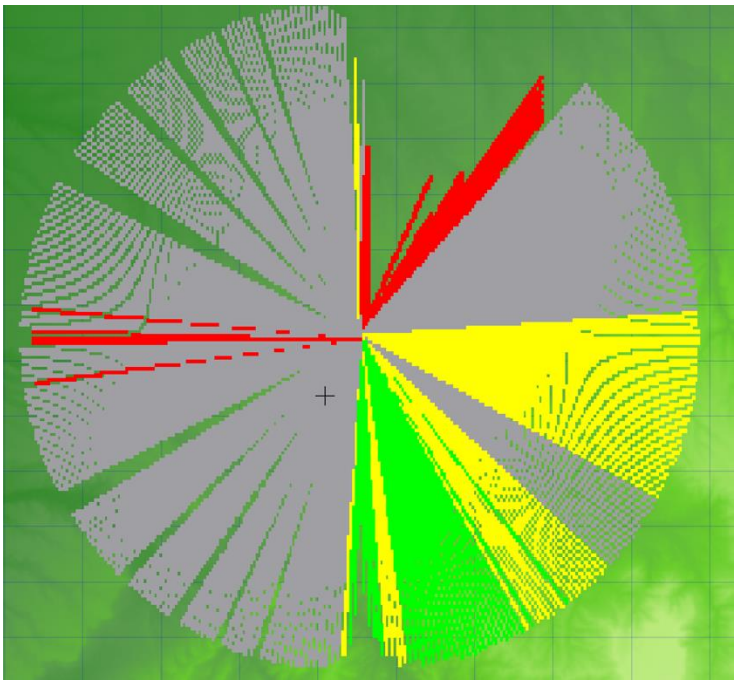


Рисунок 31 - Пример закраски области с просветами при радиусе в 40 км

Тогда алгоритм разбиения области на одинаковые фрагменты следующий:

1) вычисление координат самих крайних опорных точек;

2) определение минимальных и максимальных координат среди координат крайних точек;

3) задание шага;

4) проход по циклу от минимальных до максимальных значений координат с заданным шагом и получение координат границ фрагмента.

Расчет радиотрассы и определения ее типа происходит в момент, когда происходит расчет границ фрагмента. Результат работы алгоритма при разделении области на одинаковые фрагменты приведен на рисунке 32.

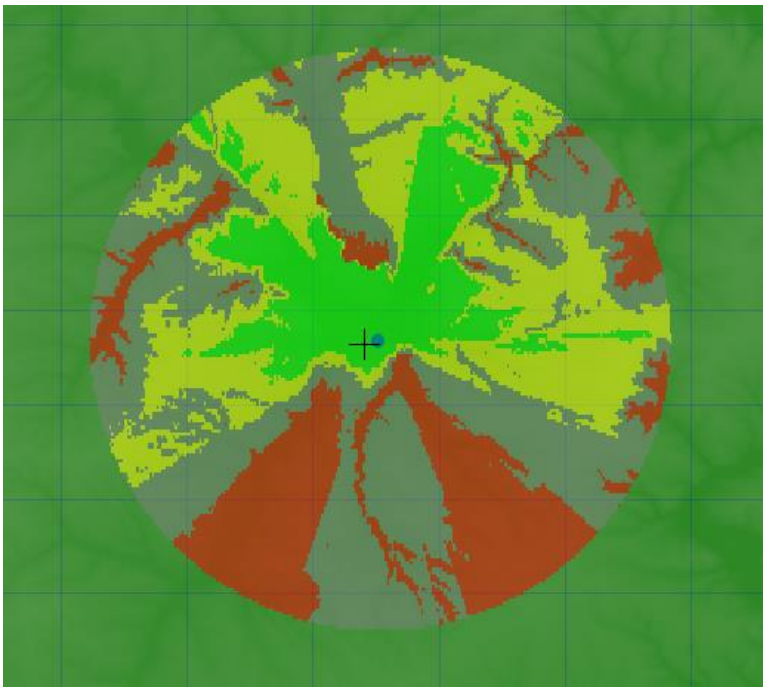


Рисунок 32 - Результат работы алгоритма при разделении области на одинаковые фрагменты

Диаграмма классов представлена на рисунке 33.

Были добавлен новые классы, содержащий в себе методы для расчета и отрисовки зоны покрытия радиосигналом на карте.

Листинг кода приведен в приложении Г.

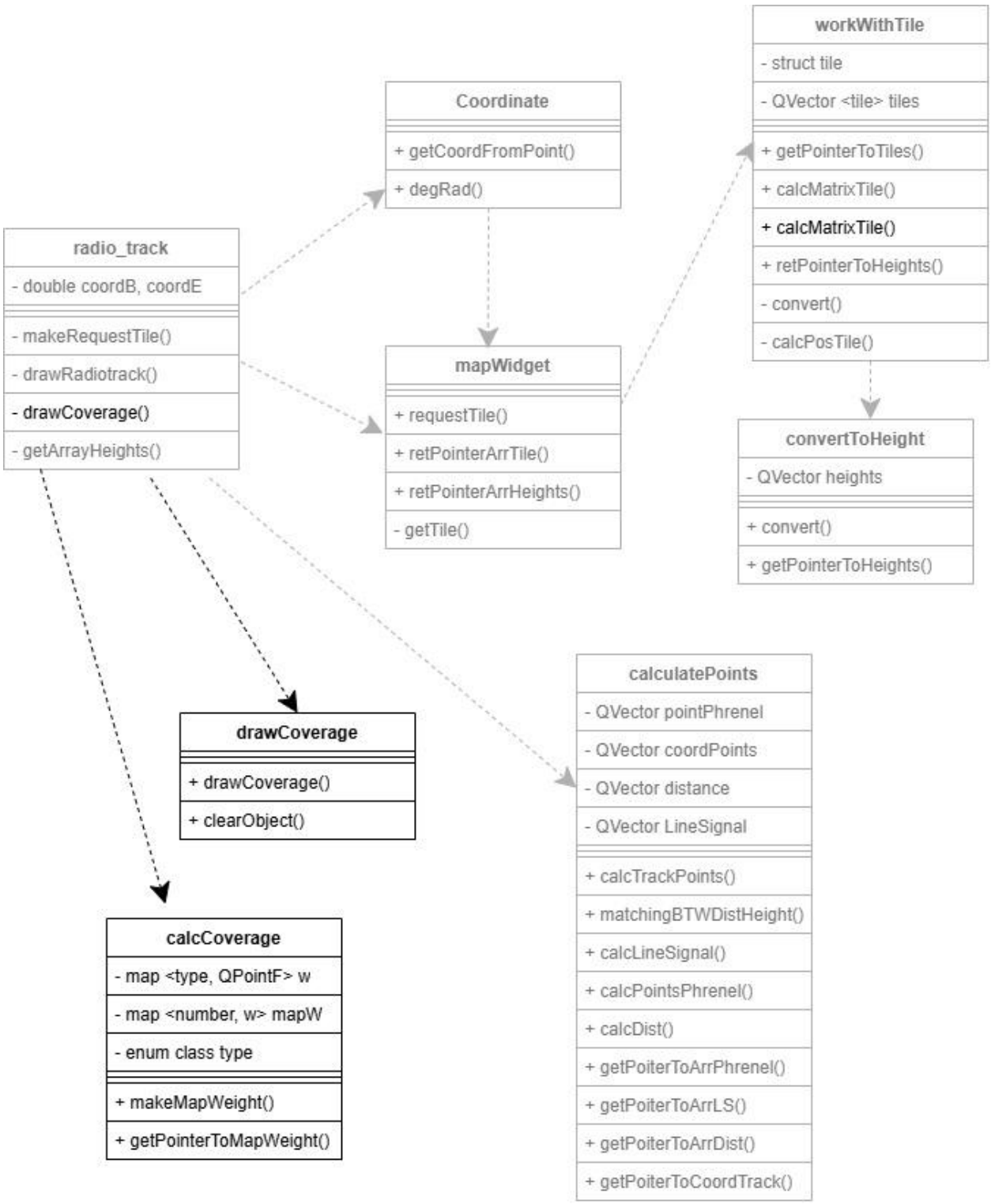


Рисунок 33 – Диаграмма классов

3.4 3D визуализация

Все реализованные ранее части приложения разрабатывалась под Qt [3]. И было бы намного удобнее реализовать новый метод также с использованием Qt. Удобство использование этого фреймворка обусловлена удобством и простой дальнейшей поддержки этого проекта.

Задача построения 3D модели для какого-то набора данных является довольно распространенной задачей. Следовательно, во фреймворке Qt должны быть уже реализованы модули, позволяющие выполнять 3D визуализацию.

Фреймворк, действительно, содержит 2 модуля, позволяющие выполнить задачу: Qt Data Visualization, Qt OpenGL

Что касается последнего, то с Qt 5 данный модуль считается устаревшим, а со следующей версии он был убран. Так как разрабатываемое приложение планируется поддерживать и дальше, то использование этого модуля для разработки является нецелесообразным.

Анализ классов для данного модуля производился при помощи рассмотрения диаграммы классов, представленной на рисунке 34.

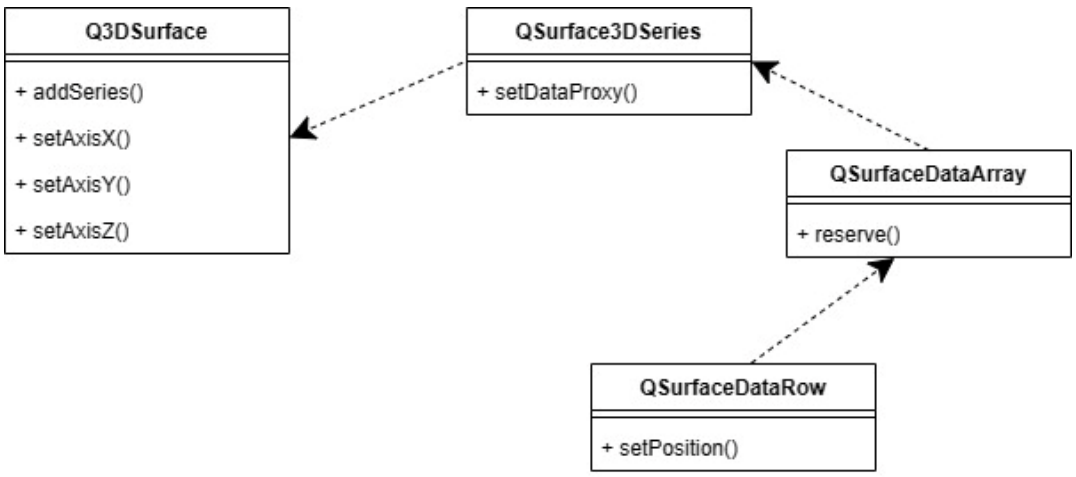


Рисунок 34 – Диаграмма классов модуля Qt Data Visualization

Для задания системы координат используются методы setAxisX, setAxisY, setAxisZ. Для рисования 3D модели используется метод addSeries.

Для отрисовки данных их необходимо сформировать и передать обратно в класс Q3DSurface. Для этой передачи данных был реализован класс QSurface3DSeries, содержащий метод setDataProxy. А для формирования данных используется класс QSurfaceDataArray.

Так как любая 3х-мерная модель состоит из набора 3х-мерных данных, то в модуле был разработан класс QSurfaceDataArray, который сохраняет весь набор данных в 3х-мерный массив для дальнейшего использования. Кроме того, был разработан класс QSurfaceDataRow для создания 3х мерной точки, которая впоследствии записывается в 3х-мерный массив.

Диаграмма разработанных классов с использованием классов модуля представлена на рисунке 35.

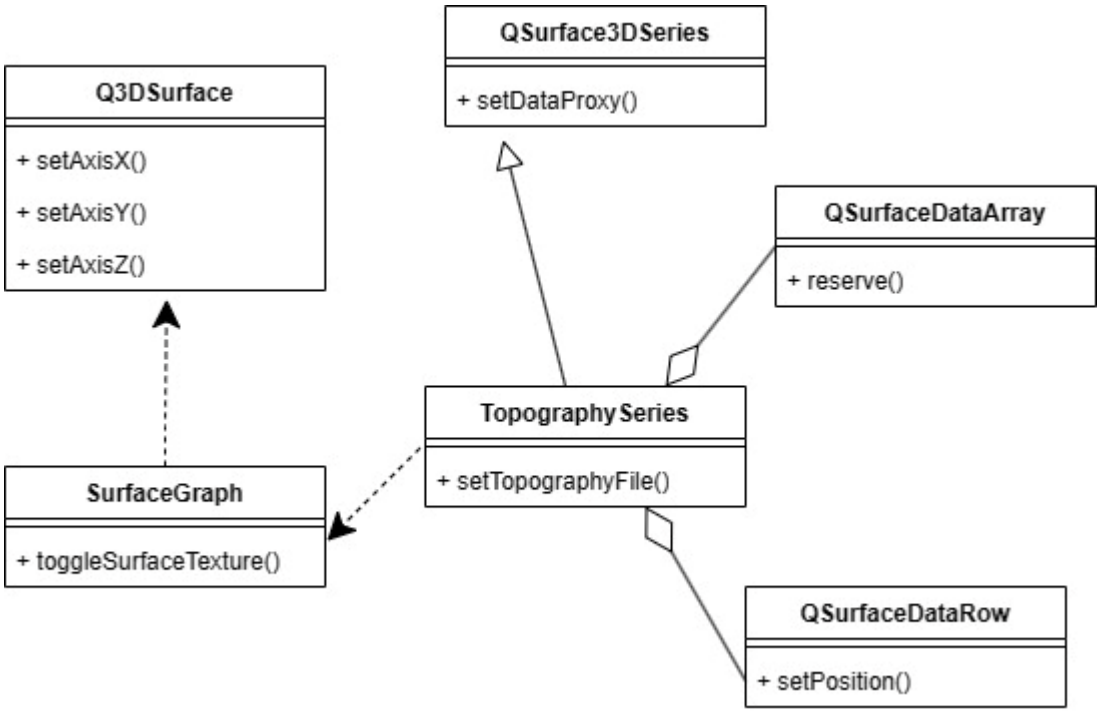


Рисунок 35 - Диаграмма разработанных классов с использованием классов модуля

Данные для 3х-мерной модели формируются за счет преобразования каждого тайла, используемого при расчетах, в 3х-мерный массив. То есть координаты пикселя вместе с номером тайла в массиве преобразовываются в координаты X, Z. А значение цвета пикселя, конвертируется в высоту и будет являться координатой Y.

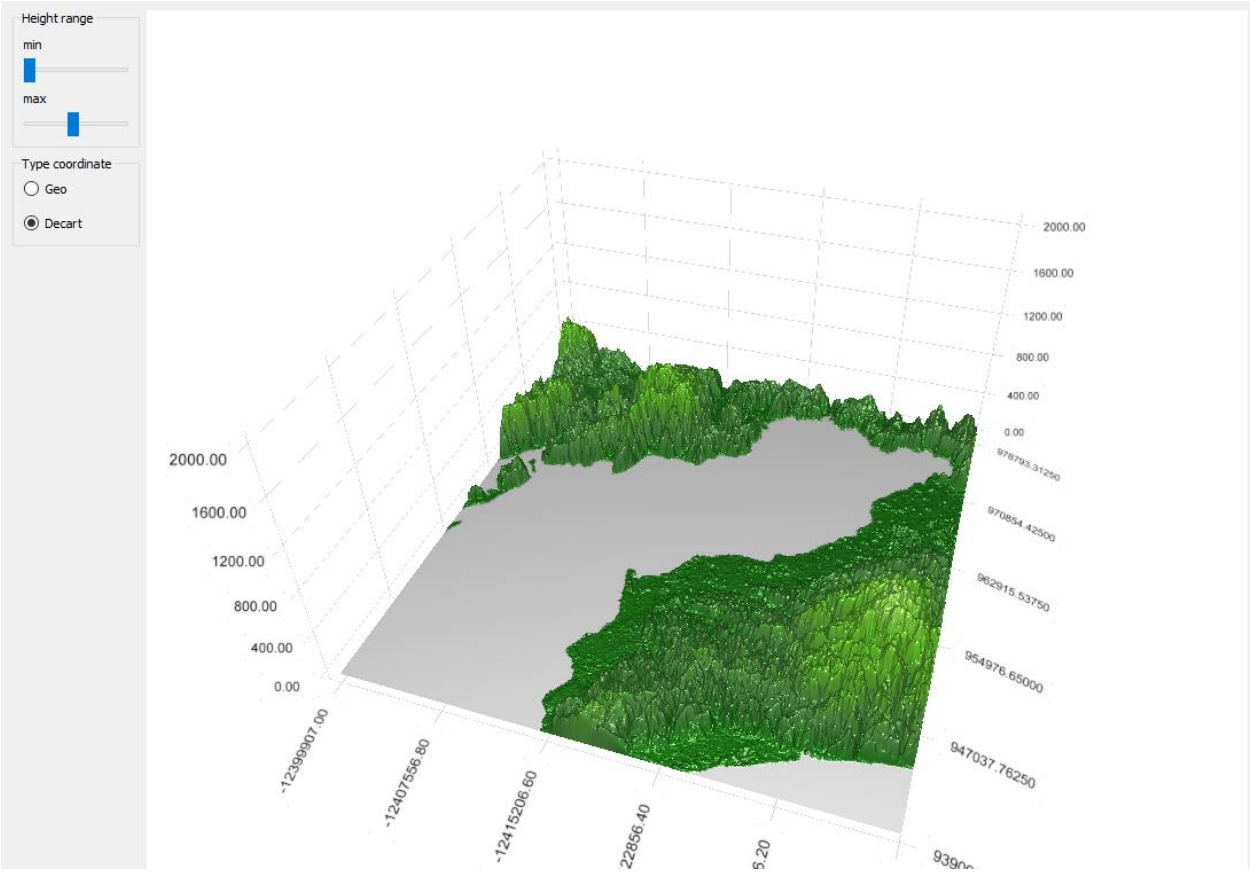
Для формирования данных был разработан класс TopographySeries, который является наследником от класса QSurface3DSeries и в котором добавлен метод setTopographyFile, который и занимается получением данных из тайлов.

Для создания 3х-мерной точки используется класс QSurfaceDataRow, экземпляр которого создается внутри TopographySeries и в который записывается значение полученная в ходе конвертации значений точка. А для сохранения этой точки в массив используется QSurfaceDataArray, экземпляр которого создается внутри TopographySeries.

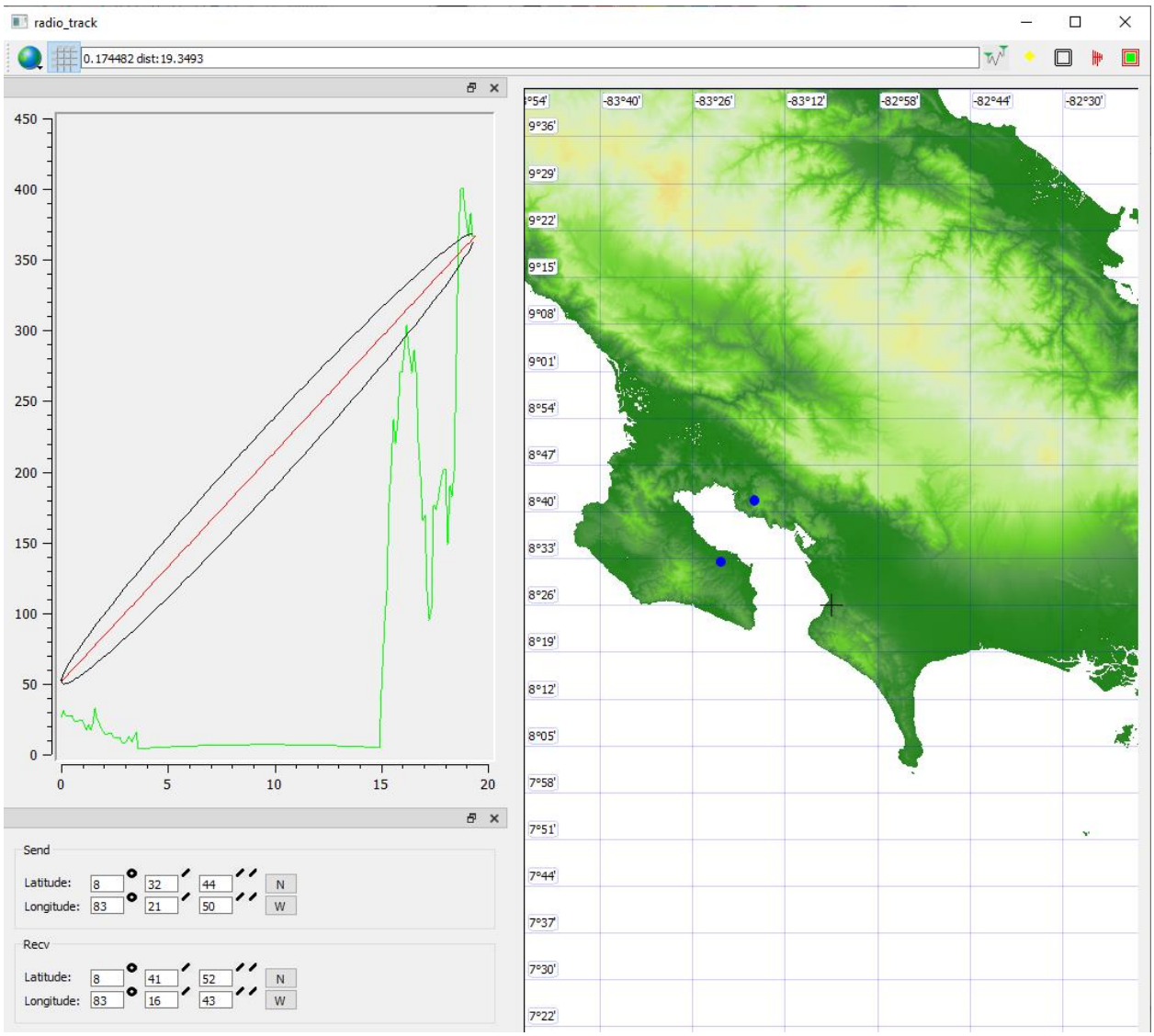
Далее набор данных передается для отрисовки, но так как QSurface3DSeries был изменен через создание наследника - нового класса, то вместо Q3DSurface был разработан новый класс SurfaceGraph, который вызывает методы TopographySeries для получения и формирования данных для 3D модели. Создание координатных осей остается без изменений, поэтому методы для их создания вызываются из экземпляра класса Q3DSurface.

Листинг кода приведен в приложении Г.

Результат построения 3D модели рельефа для области представлен на рисунке 36.



а) результат построения 3D модели рельефа для области



б) область, для которой выполнялось 3D моделирование

Рисунок 36 – Пример работы приложения

Вывод

В результате выполнения данного этапа работ были разработаны классы и методы для клиентского и расчетного модулей.

ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломного проекта был проведён анализ предметной области, определены алгоритмы для расчета радиотрассы и области радиопокрытия, а также составлено расширенное техническое задание. Основные функции приложения: расчет и построения радиотрассы и области радиопокрытия, выполнение 3D визуализации рельефа местности.

В результате выполнения анализа был выявлен набор данных, необходимый для работы функций приложения. На основании технического задания был сделан вывод о том, что данные высот необходимо получать из тайлов, полученных от сервера карт, а остальные данные рассчитываются.

Следующим этапом работ было изучение сервера карт. Изучение включало в себя определение структуры сервера карт и принципов взаимодействия с ним. На основании этого анализа было выявлено, что на сервере карт нет карты, из которой можно было бы получать значения высот, а также существующие принципы взаимодействия для запроса тайлов на сервер не подходят для решения поставленной задачи. В результате выполнения анализа сервера карт было принято решение о создании собственной карты высот и изменение клиентской части с целью изменения событий для запроса тайлов.

При генерации карты высот первоначально выполнялся обзор аналогов, состоящий из выбора формата данных, определения масштаба карты и сравнения программ для автоматической генерации карты. Главная проблема, с которой приходилось постоянно сталкиваться во время выполнения данного этапа работ – значительные временные затраты, связанная с необходимостью генерации большого количества тайлов. В результате сравнения аналогов был выявлен оптимальный способ, позволивший сгенерировать карту высот.

Проектирование структуры приложения включало в себя проектирование клиентского модуля с учетом необходимых изменений в

клиентской части сервера карт, выявленных в ходе детализации её и разбора последовательности при запросе тайлов.

Помимо клиентского модуля в приложении существуют еще расчетный и графический модули. Первоначально для правильного проектирования этих модулей и взаимодействия между ними было выполнено описание процесса работы приложения с последующим построением диаграммы циркуляции потоков данных при запросе тайлов. Эти действия выполнялись для того, чтобы удостовериться, что между модулями будет отсутствовать высокая степень связности. Для расчетного модуля в ходе выполнения этапа проектирования была разработаны алгоритмы функционирования.

После завершения этапа проектирования была выполнена программная реализация. В ходе ее выполнения были разработаны классы для расчетного и клиентского модулей. А также была выполнена разработка классов для 3D визуализации. В ходе выполнения данного этапа работ производилось тестирование для некоторых алгоритмов или функций.

В результате выполнения всех этапов работы было разработано приложение, отвечающее требованиям, указанным в техническом задании.

Разработанное приложение позволяет выполнить расчет радиотрассы для конкретных 2-х объектов, или выполнить расчет радиотрасс для некоторой области, или выполнить 3D визуализацию рельефа для области, для которой производился расчет. Визуальная составляющая приложения позволяет увидеть возможность организации надежного канала связи между объектами при выполнении построении радиотрассы, или увидеть границы области, внутри которой можно рассчитывать на хороший сигнал, при построении области радиопокрытия, или оценить правильность расчетов при выполнении 3D визуализации. Этот проект отличается от аналогов наличием 3D визуализации рельефа местности, участвующей в расчетах. Это позволяет улучшить восприятие информации для пользователя.

ПРИЛОЖЕНИЕ А

(обязательное)

Авторская справка

Я, Птахова Анастасия Муслимовна, автор выпускной квалификационной работы «Разработка приложения «Радио-Трасса» сообщаю, что мне известно о персональной ответственности автора за разглашение сведений, подлежащих защите законами РФ о защите объектов интеллектуальной собственности.

Одновременно сообщаю, что:

1. При подготовке к защите выпускной квалификационной работы не использованы источники (документы, отчёты, диссертации, литература и т.п.), имеющие гриф секретности или «Для служебного пользования» ФГБОУ ВО «Вятский государственный университет» или другой организации.

2. Данная работа не связана с незавершёнными исследованиями или уже с завершёнными, но ещё официально не разрешёнными к опубликованию ФГБОУ ВО «Вятский государственный университет» или другими организациями.

3. Данная работа не содержит коммерческую информацию, способную нанести ущерб интеллектуальной собственности ФГБОУ ВО «Вятский государственный университет» или другой организации.

4. Данная работа не является результатом НИР или ОКР, выполняемой по договору с организацией.

5. В предлагаемом к опубликованию тексте нет данных по незащищённым объектам интеллектуальной собственности других авторов.

6. Использование моей дипломной работы в научных исследованиях оформляется в соответствии с законодательством РФ о защите интеллектуальной собственности отдельным договором.

Автор: Птахова А.М. «___» _____2024 г. _____

подпись

Сведения по авторской справке подтверждаю: «___»_____2024 г.

Заведующая кафедрой ЭВМ: М.Л. Долженкова _____

Подпись

ПРИЛОЖЕНИЕ Б

(обязательное)

Листинг кода для генерации карты

```
import os
import pyautogui
import time
import shutil


def generateFilename(lat, lon):
    filename = "E://srtm/"
    if (lat<0):
        filename +="S"
    else:
        filename += "N"

    if (abs(lat)<10):
        filename = filename + "0"+str(abs(lat))
    else:
        filename = filename + str(abs(lat))

    if (lon<0):
        filename +="W"
    else:
        filename += "E"

    if (abs(lon)<10):
        filename = filename + "00"+str(abs(lon))
    elif (abs(lon)<100):
        filename = filename + "0"+str(abs(lon))
    else:
        filename += str(abs(lon))

    filename+=".hgt"

    return filename

def getFilename(lat, lon):



   base_filename = generateFilename(lat, lon)
    filename = []

    lat_list = []
    lon_list = []

    lat_list.append(lat-1)
    lat_list.append(lat)
    lat_list.append(lat+1)

    lon_list.append(lon-1)
    lon_list.append(lon)
    lon_list.append(lon+1)


    for it_lat in range(len(lat_list)):
        for it_lon in range(len(lon_list)):
            filename_gen = generateFilename(lat_list[it_lat], lon_list[it_lon])
            print(filename_gen )
            if (os.path.isfile(filename_gen)):
```

```
            filename.append(filename_gen.split('/')[3])

filename.sort()

with open ('test.txt', 'w', encoding ='utf-8') as f:
    f.write(str(len(filename)))


for i in range(len(filename)):
    #вызов функции добавления слоя
    pyautogui.hotkey('ctrl', 'Shift', 'r')

    # Ждем немного, чтобы QGIS успел обработать нажатие клавиш
    time.sleep(1)
    #диалог открытия файла
    pyautogui.moveTo(1590, 350)
    pyautogui.click()

    time.sleep(1)


  #ввод имени файла
    pyautogui.moveTo(1000, 700)
    pyautogui.write(filename[i], interval=0.25)

    flag=False

    recycle_filename ="E://srtm/"+filename[i]+".aux.xml"
    if (os.path.isfile(recycle_filename)):

        flag=True
        #time.sleep(2)

    #выбор файла
    pyautogui.move(125,30)
    pyautogui.click()
    #pyautogui.move(100, 500)


    #добавление слоя
    pyautogui.moveTo(1440,945)
    pyautogui.click()

    if (not flag):
        time.sleep(5)
    else:
        print("yes")
        time.sleep(2)

    time.sleep(1)

    #закрытие вкладки
    pyautogui.move(-100,0)
    pyautogui.click()

    #скрытие описание слоя
    pyautogui.moveTo(90, 480)
    pyautogui.click()

    #открытие свойств слоя
    pyautogui.move(100, 0)
    pyautogui.doubleClick()


  #открытие меню выбора палитры
    pyautogui.moveTo(670, 200)
    pyautogui.click()
```

```
        pyautogui.sleep(1)

#выбор псевдоцвета
                pyautogui.move(0, 30)
                pyautogui.click()

                time.sleep(1)

                #открытие диалога выбора палитры цвета
                pyautogui.moveTo(820, 840)
                pyautogui.click()

                time.sleep(1)

                #ввод имени файла с палитрой
                pyautogui.move(0, -180)
                pyautogui.write('color_height.txt', interval=0.25)

                #выбор этого файла
                pyautogui.hotkey('Enter')

                #применение палитры
                pyautogui.moveTo(970, 1040)
                pyautogui.click()

                pyautogui.sleep(1)

            #вызов метода разбиения на тайлы
            pyautogui.hotkey('Shift', 'T')

            #открытие меню выбора файла для разбиения на тайлы

            pyautogui.moveTo(1000, 525)
            pyautogui.click()

            ind = filename.index(base_filename.split('/')[3])
            print("ind:", ind)
            #выбор файла
            print("offset:", ind*25)
            pyautogui.move(0, ind*25)
            pyautogui.click()

            #time.sleep(1)
            #процесс разбиения на тайлы
            pyautogui.moveTo(1100, 820)
            # pyautogui.hotkey('enter')
            pyautogui.click(clicks=2, interval=0.5)
            # time.sleep(1)

            pyautogui.moveTo(1050, 560)
            pyautogui.click()
            #ждем когда процесс разбиения закончится
            time.sleep(6)

            pyautogui.moveTo(1200, 820)
            pyautogui.click()


        if __name__ == "__main__":

            # Переключаемся на окно QGIS
            pyautogui.getWindowsWithTitle('QGIS')[0].minimize()
            pyautogui.getWindowsWithTitle('QGIS')[0].maximize()
            start = time.time()
```

```
for j in range(22, 41, 1):

        lat =48

        lon =j
        filename_ = generateFilename(lat, lon)
        if (os.path.isfile(filename_)):

            with open('test.txt', 'r', encoding='utf-8') as f:
                data = int(f.read(1))

            print(data)
            for i in range(data):
                pyautogui.moveTo(190, 480)
                pyautogui.click()

                pyautogui.hotkey('Ctrl', 'd')
                pyautogui.hotkey('Enter')

            getFilename(lat, lon)



            #time.sleep(1)



    end = time.time()-start

    print(end)


import os

import pyautogui
import time

def renamefile(x):
        #переход в директорию
        pyautogui.moveTo(1200, 820)
        pyautogui.click()

        pyautogui.write('cd '+str(x), interval=0.25)
        pyautogui.hotkey('Enter')

        #открытие папки
        pyautogui.moveTo(1590, 350)
        pyautogui.click()

        time.sleep(1)

        #выбор всех файлов
        pyautogui.hotkey('Ctrl', 'a')

        #кнопка группового переименования
        pyautogui.moveTo(1000, 525)
        pyautogui.click()
        pyautogui.hotkey('Enter')

        #создание каталога
        pyautogui.hotkey('F7')
        pyautogui.write(str(0), interval=0.25)
        pyautogui.click()
```

```
            #выделение всех файлов, снятие выделение с папки
            pyautogui.hotkey('Ctrl', 'a')
            pyautogui.move(0, -25)
            pyautogui.click()


            #перемещение файлов
            pyautogui.hotkey('F6')


            pyautogui.moveTo(1200, 820)
            pyautogui.click()

            pyautogui.write('cd ', interval=0.25)
          pyautogui.hotkey('Enter')

    if __name__ == "__main__":

        # Переключаемся на окно Total Commander
        pyautogui.getWindowsWithTitle('Total Commander')[0].minimize()
        pyautogui.getWindowsWithTitle(''Total Commander ')[0].maximize()

        #имя каталога (x)
        for x in range(1148, 1260):
           renameFile(x)
```

ПРИЛОЖЕНИЕ В

(обязательное)

Листинг кода клиентского модуля

```
#pragma once
#include <core/mapcontrol.h>

#include "layermanager_req.h"

namespace qmapcontrol
{
class MapControl_req : public MapControl
{
    Q_OBJECT
public:
    MapControl_req(QSize size, QWidget* parent = NULL, MouseMode = Panning);
    ~MapControl_req();

    void paint_Image(QPointF coordinate);
    void addLayer(Layer_req* layer);
signals:
    void tile_load(const QPixmap pixmap);

private:
    LayerManager_req* layermanager_req;
};
}  // namespace qmapcontrol


#include "mapcontrol_req.h"

namespace qmapcontrol
{
MapControl_req::MapControl_req(QSize size, QWidget* parent, MouseMode mousemode)
    : MapControl(size, parent, mousemode)
{
    layermanager_req = new LayerManager_req(this, size);

    connect(layermanager_req, &LayerManager_req::tile_load, this,
            &MapControl_req::tile_load, Qt::QueuedConnection);
}

void MapControl_req::paint_Image(QPointF coordinate)
{
    layermanager_req->request_Image(coordinate);
}

void MapControl_req::addLayer(Layer_req* layer)
{
    layermanager_req->addLayer(layer);
}

MapControl_req::~MapControl_req()
{
    delete layermanager_req;
}
}  // namespace qmapcontrol
```

ПРИЛОЖЕНИЕ Г

(обязательное)

Листинг кода остальных модулей приложения

```
#pragma once

#include <QActionGroup>
#include <QToolBar>
#include <QStatusBar>
#include <QLabel>
#include <QMenu>
#include <QLineEdit>

#include "core/mapnetworkfactory.h"
#include "extentions/MapExtentionManager.h"
#include "extentions/lengthmeasuremapextention.h"
#include "extentions/layerswitchmapextention.h"
#include "extentions/coordinatenetmapextention.h"
#include "extentions/CursorPosMapExtention.h"

#include "ObjectDrawController.h"
#include "BeginPosition.h"
#include "endCoveragePos.h"
#include "drawCoverage.h"

#include "ToDegree.h"
#include "PelengCaption.h"

#include "core/imagemanager.h"
#include "core/maploader.h"

#include <core/mapcontrol.h>
#include "mapcontrol_req.h"

#include "color_height.h"
#include "commonType.h"

namespace track
{
class MapWidget : public qmapcontrol::MapControl
{
    Q_OBJECT
signals:
    //void MouseMoveEvent(QMouseEvent* evnt, QPointF coordinate);
    void MouseDoubleClickEvent(QMouseEvent* evnt, QPointF coordinate);
    void converting_finished(QVector<int> height);

public:
    MapWidget(QWidget* parent);
    void addMapButtonOnToolbar(QToolBar* toolBar);
    void setActionToRuler(QAction* rulerAction);
    void setStatusLabel(QLabel* statusLabel);
    void registerDrawController(const ObjectDrawControllerPtr& controller);
    void redrawControllerData(const ObjectDrawControllerPtr& controller);
    void setControllerVisible(const ObjectDrawControllerPtr& controller,
                              bool isVisible);
    void unregisterDrawController(const ObjectDrawControllerPtr& controller);
    ~MapWidget();
```

```
void drawPoint(QPointF coordinate);
    void drawCirclePoint(QPointF coordinate);
    void drawVisibleCoverage(weight& mapWeight);
    void requestMapControl();
    QLineEdit* edit_coord;

    void redrawControllerDataForCoverage();
    earth_math::earth_point findNetPoint(
        const earth_math::earth_point& ep) const;
    /*
        * Метод сохраняет координаты позиции тайла в массив coord
        * @param координаты позиции тайла
        */
    void initial_arr(double x, double y);

    /*
        * Метод сохраняет координаты в переменные
        */
    void saveCoord_px(int x, int y);

    void clear_heights();

    void addTextOnEdit(QString text);
    void init_arr_tiles(int coord_tile_x, int coord_tile_y);
private:
    QVector<QPoint> coord_tile;
    qmapcontrol::MapNetworkFactory netFactory_;
    QStringList loadedMaps_;
    QActionGroup mapLayersGroup_;
    qmapcontrol::MapExtentionManager extentionManager_;
    QToolBar* mapSwitchToolbar_ = nullptr;
    struct MapExtentions
    {
        qmapcontrol::LayerSwitchMapExtention* switchMapExt;
        qmapcontrol::LengthMeasureMapExtention* lenMeasureExt;
        qmapcontrol::CursorPosMapExtention* cursorPosExt;
        qmapcontrol::CoordinateNetMapExtention* coordinateNetExt;
    } mapExtentions_{};
    struct MapLayers
    {
        qmapcontrol::Layer* lengthMeasureLayer;
        qmapcontrol::Layer* coordinateNetLayer;
    } layers_{};
    bool firstMapLoaded_ = false;
    QMap<QString, qmapcontrol::Layer*> controllersLayers_;
    virtual void initMap();
    virtual void createExtentions();
    virtual void createLayers();

    void getColorPoint(const QPixmap pixmap);

    QMenu* selectedMenu;

    std::shared_ptr<BeginPosition> beginPositionController_;
    std::shared_ptr<BeginPosition> endPositionController_;
    std::shared_ptr<endCoveragePos> endCoveragePosController_;
    std::shared_ptr<drawCoverage> drawCoverage_;

    uint8_t count = 0;

    qmapcontrol::MapControl_req* mapcontrol_req;
    QVector<QPointF> coord;
    QVector<QPoint> coord_px;
```

```
    converter_toHeight* converter;
    QVector<int> value_height;

    earth_math::degree_abstract netStep_ =
        earth_math::degree_abstract::from_seconds(7);

    bool isPointNear(const earth_math::earth_point& arcA,
                     const earth_math::earth_point& arcB,
                     const earth_math::earth_point point) const;
private slots:
    void tile_get(const QPixmap pixmap);
};
}  // namespace track


MapWidget::MapWidget(QWidget* parent)
    : MapControl(QSize(100, 100))
    , netFactory_(this)
    , mapLayersGroup_(this)
    , extentionManager_(false, this)

{
    mapcontrol_req = new qmapcontrol::MapControl_req(QSize(100, 100));

    connect(mapcontrol_req, &qmapcontrol::MapControl_req::tile_load, this,
            &MapWidget::tile_get);

    converter = new converter_toHeight();

    MapWidget::initMap();
}

void MapWidget::saveCoord_px(int x, int y)
{
    coord_px.append({x, y});
}

void MapWidget::getColorPoint(const QPixmap pixmap)
{
    //координаты начала тайла в пикселях
    int coordBeginTilePix_x = coord[0].x() * 256;
    int coordBeginTilePix_y = coord[0].y() * 256;

    //координаты точки в пикселях
    int coordPointPix_x = coord_px[0].x();
    int coordPointPix_y = coord_px[0].y();

    //пиксельные координаты точки внутри тайла
    int coordPointIntoTile_x = coordPointPix_x - coordBeginTilePix_x;
    int coordPointIntoTile_y = coordPointPix_y - coordBeginTilePix_y;

    QImage image(pixmap.toImage());

#ifdef DEBUG
    QString name = "test_" + QString::number(coord[0].x()) + "_"
                   + QString::number(coord[0].y()) + ".jpg";
    QImage img(name, "grb");
    img.setPixel(coordPointIntoTile_x, coordPointIntoTile_y, qRgb(255, 0, 0));
    img.save("text.jpg", "JPG");
#endif  // DEBUG

    QColor color(image.pixel(coordPointIntoTile_x, coordPointIntoTile_y));
    int red = color.red();
```

```
    int green = color.green();
    int blue = color.blue();

    int height = converter->convert(red, green, blue);

    value_height.push_back(height);
}
```

```
#include "ObjectDrawController.h"
#include "earth_point.hpp"
#include <optional>
namespace track
{
class drawCoverage : public ObjectDrawController
{
public:
    drawCoverage() = default;
    void drawObjects(qmapcontrol::Layer* layer) override;
    void clearObjects() override;
    ~drawCoverage() override = default;
    QString controllerName() override;
    void setOwnPoint(earth_math::earth_point own_point, int weight);
    std::optional<earth_math::earth_point> ownPoint() const;
private:
    std::optional<earth_math::earth_point> own_point_{};
    uint8_t count;
    QBrush drawBrush;
    int weight_;
};
}  // namespace track
```

```
#include "drawCoverage.h"
#include <geometries/region.h>
using namespace track;
void drawCoverage::drawObjects(qmapcontrol::Layer* layer)
{
    if (own_point_)
    {
        count++;
        drawBrush.setStyle(Qt::BrushStyle::SolidPattern);
        switch (weight_)
        {
            case 1:
            {
                drawBrush.setColor(Qt::red);
                break;
            }
            case 2:
            {
                drawBrush.setColor(Qt::gray);
                break;
            }
            case 3:
            {
                drawBrush.setColor(Qt::yellow);
                break;
            }
            case 4:
            {
drawBrush.setColor(Qt::green);
```

```
                break;
            }
            default:
                break;
        }

        layer->addGeometry(new qmapcontrol::Region(
            QRectF(own_point_.value().longitude().to_degree(),
                   own_point_.value().latitude().to_degree(), 0.0021, 0.0021),
            drawBrush, "", Qt::NoPen));
    }
}

void drawCoverage::clearObjects()
{
    own_point_ = std::nullopt;
}

QString drawCoverage::controllerName()
{
    return "drawCoverage";
}

void drawCoverage::setOwnPoint(earth_math::earth_point own_point, int weight)
{
    own_point_ = own_point;
    weight_ = weight;
    //own_point2_ = own_point2;
}

std::optional<earth_math::earth_point> drawCoverage::ownPoint() const
{
    return own_point_;
}
```

ПРИЛОЖЕНИЕ Д

(справочное)

Список сокращений и обозначений

Понятие	Значение
Маршрут радиотрассы	Прямая между 2мя объектами, между которыми необходимо установить связь
Профиль высот	Высоты точек, принадлежащих маршруту радиотрассы
Тайл	Фрагмент географической карты определенного размера (256x256 пикселей)
Сервер карт	ПО, состоящее из сервера и клиента
Клиент (Клиентская часть)	Существующие классы для работы с сервером
Клиентский модуль	Разработанные в дипломном проекте классы для работы с сервером
Дуга нулевого уровня	Дуга земной кривизны
Опорные точки	Границы области радиопокрытия
Ортодомия	Кратчайшее расстояние между двумя точками на поверхности Земли
Меркаторские координаты	Координаты точек на цилиндрической прямоугольной проекции Меркатора

ПРИЛОЖЕНИЕ Е

(справочное)

Библиографический список

1. Документация по C++ [Электронный курс] – режим доступа: https://learn.microsoft.com/ru-ru/cpp/cpp/?view=msvc-160

2. "Map Projections: A Reference Manual" by L.P. Lee [Текст], Taylor & Francis Inc., 1900. – 352 с.

3. Документация по Qt [Электронный курс] – режим доступа https://doc.qt.io/all-topics.html

4. Дьяков Б.Н. Геодезия. Общий курс: Учеб. пособие для вузов. [Текст] - Новосибирск: Изд-во Новосиб. ун-та, 1993. - 171 с.

5. Определение типа радиотрассы [Электронный курс] - режим доступа https://habr.com/ru/articles/257731/