*ISSN 2782-2001*
*Системы анализа*
*и обработки данных*
*том 103, № 3, 2026, с. 7–22*

*http://journals.nstu.ru/vestnik*
*Analysis and data*
*processing systems*
*Vol. 103, No. 3, 2026, pp. 7–22*

ИНФОРМАЦИОННЫЕ
ТЕХНОЛОГИИ
И ТЕЛЕКОММУНИКАЦИИ

INFORMATION
TECHNOLOGIES
AND TELECOMMUNICATIONS

УДК 004.94

DOI: 10.17212/2782-2001-2026-3-7-22

# Оптимизация рендеринга картографических данных в мобильных геоинформационных системах на фреймворке Flutter*

**А.Б. ГОМБОИН[a], А.М. КОРИКОВ[b]**

*РФ, 634050, г. Томск, пр. Ленина, 40, Томский государственный университет систем управления и радиоэлектроники*

[a] *a.gomboin@mail.ru* [b] *anatolii.m.korikov@tusur.ru*

В настоящей статье рассматривается задача оптимизации рендеринга сложных полигональных объектов в мобильных геоинформационных системах, реализованных на кросс-платформенном фреймворке Flutter. В условиях ограниченных вычислительных ресурсов мобильных устройств ГИС и высоких требований к отзывчивости интерфейса стандартные подходы к визуализации картографических данных, применяемые в библиотеке flutter_map, приводят к критической деградации производительности мобильных ГИС: снижению частоты кадров (FPS – Frames Per Second), росту потребления памяти и нестабильной реакции на пользовательские жесты.

Для обеспечения эффективной работы мобильных ГИС в статье предлагается два метода: отложенная загрузка полигонов для отсрочки пересчета до завершения взаимодействия пользователя с картой (картографическими данными) и кластеризация полигонов по географической близости с последующим геометрическим объединением в составные объекты. Экспериментальное исследование, проведенное на наборе из 5000 полигонов на реальном мобильном устройстве, показало, что комбинированное применение методов оптимизации позволило увеличить средний FPS, снизить потребление оперативной памяти и улучшить оценку пользовательского опыта. Результаты экспериментального исследования подтверждают, что даже в условиях архитектурных ограничений Flutter возможно достижение высокой производительности мобильных ГИС за счет оптимизации логики отображения данных.

Предложенные методы оптимизации рендеринга картографических данных не требуют глубокой модификации существующего кода – они легко интегрируются в архитектуру библиотеки flutter_map и могут быть адаптированы под любые типы полигонов и масштабы, что делает фреймворк Flutter перспективной платформой для разработки профессиональных мобильных ГИС-приложений, в том числе для работы в полевых условиях с минимальными ресурсами.

* *Статья получена 05 октября 2025 г.*

**Ключевые слова**: рендеринг, мобильное приложение, геоинформационные системы (ГИС), мобильная ГИС, Flutter, flutter_map, полигоны, оптимизация, отложенная загрузка полигонов, кластеризация полигонов, комбинирование методов оптимизации, viewport, геопространственные данные, OpenStreetMap

## ВВЕДЕНИЕ

Мобильные приложения, установленные в смартфоне или планшетном компьютере, навигаторе и других подобных переносных устройствах, – это постоянные спутники современного человека, обеспечивающие ему неограниченные возможности и комфорт в повседневной жизни. Особенно полезными являются приложения геоинформационных систем. Они помогают человеку ориентироваться на местности, успешно решают задачи местоопределения и планирования маршрута от старта до финиша, указывают рациональное направление движения, обеспечивают информацией об окружающей среде. Такие приложения полезны во многих областях деятельности – в геологии, добыче нефти, газа и других полезных ископаемых. Трудно переоценить роль мобильных геоинформационных систем (ГИС) для людей, постоянно работающих в различной местности: им нужно постоянно пополнять знания о своем местонахождении и, возможно, в большем объеме, чтобы качественно выполнять свою работу. Как правило, на местности, где ведется поиск и разработка месторождений и/или другая подобная деятельность, покрытия Сети или недостаточно, или она полностью отсутствует, что накладывает ограничения на использование мобильных ГИС, так как они требуют постоянного подключения к интернету. Можно скачать различные карты местности и создать картографическую базу данных (КБД), но пользователям КБД нужна информация, связанная с их непосредственной деятельностью (например, информация о газопроводах, административных зданиях и других объектах, которые отсутствуют на обычных картах местности). Такую картографическую информацию можно загрузить на специальное мобильное приложение, которое позволяет отображать заранее заготовленные данные в специальном формате и представлять их в векторной форме. При использовании таких приложений возникает проблема, связанная с временными задержками работы пользовательского интерфейса, так как пользователям для работы необходимо большое количество картографических данных.

Вышеизложенное доказывает актуальность проблемы оптимизации рендеринга сложных картографических данных, в частности полигональных объектов в мобильном приложении на Flutter [1]. Для ее решения предлагается несколько подходов к оптимизации рендеринга: отложенная загрузка объектов на карту местности и применение кластеризации объектов в один объект.

Экспериментальная оценка показала улучшение скорости рендеринга и снижение нагрузки на память, что значительно повышает пользовательский опыт. Эти результаты предоставляют практические рекомендации для разработчиков, стремящихся сбалансировать визуальное качество и производительность в системах картографии на Flutter.

В первом разделе настоящей статьи рассматривается постановка задачи, анализируются особенности фреймворка, языка программирования и библио-

теки, применяемые для отображения карты местности и данных на ней, а также анализируется существующий метод кластеризации. Во втором разделе представлены решения по загрузке данных на карту местности в момент их непосредственного использования и объединению полигональных объектов в один объект для уменьшения нагрузки на рендеринг. В третьем разделе представлены результаты экспериментальных исследований.

## 1. ПОСТАНОВКА ЗАДАЧИ

Разработать метод оптимизации рендеринга сложных геопространственных полигонов с помощью инструментов библиотеки flutter_map, обеспечивающий сохранение функциональной точности, интерактивности и визуальной достоверности при одновременном снижении вычислительной нагрузки на рендер-конвейер Flutter, с целью повышения производительности и стабильности работы мобильных приложений ГИС при работе с массивами полигонов.

Современные ГИС, интегрируемые в мобильные и веб-приложения, всё чаще требуют визуализации крупномасштабных, многослойных, высокодетализированных пространственных объектов (в частности, полигонов, представляющих административные границы, зоны экологического мониторинга, территории инфраструктурных объектов, зоны риска и прочие географические сущности). При этом возросшие требования к интерактивности, отзывчивости и кросс-платформенной совместимости вынуждают разработчиков отказываться от традиционных нативных SDK [2] (Software Development Kit – набор инструментов для разработки программного обеспечения (ПО) под конкретную платформу, операционную систему, язык программирования, аппаратное обеспечение, облачные сервисы и фреймворки) в пользу универсальных фреймворков, таких как Flutter.

Однако применение Flutter для реализации картографических компонентов с высокой плотностью полигонов сталкивается с фундаментальными архитектурными ограничениями, обусловленными его декларативной моделью рендеринга. Библиотека flutter_map использует подход, при котором каждый полигон устанавливается как самостоятельный виджет PolygonLayer [2], содержащий собственный CustomPaint [2] и дерево геометрических примитивов. Такая реализация, хотя и обеспечивает гибкость в управлении стилями и событиями, приводит к экспоненциальному росту нагрузки на рендер-конвейер Flutter при увеличении числа полигонов. Это порождает критические деградации производительности: увеличение времени рендеринга, частоту кадров ниже 20 FPS (Frames Per Second – количество кадров в секунду), значительное потребление оперативной памяти (приблизительно 500 МБ и выше), а также нестабильную реакцию на пользовательские взаимодействия: скроллинг, масштабирование, клики.

Таким образом, возникает научно-практическая проблема – отсутствие эффективных методов оптимизации рендеринга сложных, многовершинных полигонов в рамках реактивной архитектуры Flutter, что препятствует внедрению высоконагруженных мобильных приложений ГИС в условиях ограниченных вычислительных ресурсов мобильных устройств.

### 1.1. ФРЕЙМВОРК FLUTTER И ЯЗЫК ПРОГРАММИРОВАНИЯ DART

Flutter – это открытый фреймворк для создания нативных пользовательских интерфейсов, разработанный компанией Google и выпущенный в 2017 году. Его ключевая особенность заключается в использовании единой кодовой базы для генерации приложений на разные операционные системы: iOS, Android, Web, Windows, macOS и Linux, что делает его особенно привлекательным для проектов с высокими требованиями к кросс-платформенности и скорости развертывания. В отличие от традиционных подходов к разработке кросс-платформенных мобильных приложений, Flutter использует собственный высокопроизводительный рендер-движок Skia, который обеспечивает прямую отрисовку UI-элементов на GPU без привлечения нативных компонентов операционной системы. Это позволяет достичь плавности анимаций до 120 FPS даже на устройствах среднего класса, что является критически важным для интерактивных геоинформационных систем [2–4]. В традиционных подходах, таких как WebView-интеграция, пользовательский интерфейс формируется с использованием веб-технологий и отображается внутри встроенного браузерного компонента операционной системы, что сопряжено с накладными расходами на взаимодействие между веб-контекстом и нативной средой. Альтернативный архитектурный паттерн native-bridge, применяемый в React Native и Xamarin, бизнес-логика которых пишется на кросс-платформенных языках JavaScript, C#, для взаимодействия с нативными компонентами операционной системы используют специальный «мост» для передачи сообщений между кросс-платформенным кодом и нативными API, однако асинхронная природа такого взаимодействия может приводить к задержкам при частом обмене данными. Это позволяет достичь плавности анимаций до 120 FPS даже на устройствах среднего класса, что является критически важным для интерактивных геоинформационных систем [2–4].

Основой Flutter является язык программирования Dart, разработанный Google как оптимизированный клиентский язык с поддержкой как компиляции в нативный код AOT – Ahead-of-Time, так и виртуальной машины JIT – Just-in-Time для целей разработки. Язык Dart реализует концепцию реактивного программирования через механизм widgets – декларативных, иммутабельных и рекурсивно-составных единиц интерфейса. Каждый виджет в Flutter представляет собой чистую функцию от состояния State в StatefulWidget или константное описание const Widget в StatelessWidget. Такой подход обеспечивает предсказуемость изменений интерфейса и упрощает трассировку источников перестроения дерева виджетов, но одновременно создает сложности при работе с динамическими, вычислительно тяжелыми элементами, такими как картографические полигоны с тысячами вершин.

Рендеринг в Flutter происходит в три этапа: 1) построение дерева виджетов, 2) создание дерева элементов, 3) построение дерева слоев. На последнем этапе все графические примитивы преобразуются в команды Skia, которые передаются на GPU. Однако при обработке сложных геометрических объектов (например, полигонов с десятками тысяч координат) дерево виджетов может содержать сотни тысяч экземпляров CustomPaint, Path, Polygon и других компонентов, что приводит к экспоненциальному росту времени на layout

(этап конвейера рендеринга, на котором фреймворк вычисляет размеры, координаты и иерархические ограничения для каждого визуального элемента) и paint-фазы (этап, следующий за layout, на котором формируются низкоуровневые графические инструкции для движка рендеринга), вызывая задержки и пропуски кадров [5].

### 1.2. БИБЛИОТЕКА FLUTTER_MAP

Для реализации карты и ее функций для Flutter существуют различные библиотеки, одной из которых является flutter_map [6] – одна из наиболее популярных open-source-альтернатив коммерческим решениям (например, Mapbox GL Native или Google Maps SDK). Flutter_map представляет собой порт JavaScript-библиотеки Leaflet на платформу Dart, адаптированный для работы в рамках реактивной архитектуры Flutter. Она предоставляет модульную структуру, основанную на следующих компонентах:

1) MapController – управление центром, масштабом, поворотом и другими функциями;
2) TileLayer – загрузка и отображение растровых или векторных тайлов XYZ, TMS, WMTS;
3) MarkerLayer – отображение точечных объектов;
4) PolylineLayer и PolygonLayer – отображение линий и замкнутых полигонов.

Полигоны в flutter_map реализованы через класс Polygon, который наследуется от Layer и содержит список координат List<LatLng>, представляющих вершины многоугольника. При каждом обновлении состояния карты, например при изменении масштаба или перемещении, flutter_map пересчитывает положение всех полигонов относительно текущего viewport (прямоугольная область экрана или координатного пространства, в пределах которой осуществляется отрисовка и отображение контента). Flutter_map применяет проекцию Mercator и передает ее в CustomPaint-виджет для отрисовки через Canvas.drawPath().

Однако здесь возникает фундаментальное архитектурное противоречие: flutter_map, будучи реализован как набор виджетов, не учитывает специфику геопространственных данных. Каждый полигон устанавливается как отдельный Widget, что приводит к проблемам.

При отображении большого количества полигонов создается большое количество экземпляров PolygonLayer, каждый из которых содержит свои CustomPaint, Path, Paint, и связанные с ними RenderBox. Это превышает допустимые пределы производительности Flutter, рекомендованные в официальной документации фреймворка: более 1000 виджетов на экране уже считается критическим порогом.

Полигоны рендерятся с одинаковой детализацией на всех масштабах, даже если они находятся за пределами области видимости или имеют слишком мелкие детали, невидимые при текущем zoom-уровне (дискретная или непрерывная величина, определяющая степень приближения / удаления карты относительно исходного масштаба).

Любое изменение состояния (например, изменение цвета одного полигона) вызывает перестроение всего PolygonLayer, поскольку Flutter не имеет встроенной системы дифференциального обновления для сложных геометрических примитивов.

Хранение координат полигонов в виде списков List<LatLng> в Dart-контейнерах без оптимизации памяти (например, через Float32List или ByteData) приводит к фрагментации кучи и увеличению GC-пауз (временные остановки выполнения основного потока приложения, вызванные работой автоматического сборщика мусора среды выполнения Dart). При работе с крупными наборами данных (например, границы регионов) объем потребляемой памяти может достигать 200…500 Мб на устройстве.

Такие проблемы вполне решаемые, а некоторые из них далее детализируются, чтобы показать эффективность разработанных подходов и продемонстрировать оптимизированную обработку больших объемов геоданных в реальном времени.

### 1.3. ГЕОМЕТРИЧЕСКАЯ МОДЕЛЬ ПОЛИГОНОВ: ПРЕДСТАВЛЕНИЕ, ПРОЕКЦИИ И РЕНДЕРИНГ, ПРОБЛЕМЫ РЕАЛИЗАЦИИ ГИС

Полигон в ГИС – это двумерная геометрическая фигура, определяемая замкнутой последовательностью точек или вершин на поверхности земного эллипсоида. В контексте цифровых карт каждая вершина выражается в географических координатах (широте и долготе) в системе WGS84 (World Geodetic System 1984 – всемирная система геодезических параметров Земли 1984 года, в число которых входит система геоцентрических координат) или EPSG:4326 (European Petroleum Survey Group – публичный реестр геодезических данных, систем пространственной привязки, земных эллипсоидов, преобразований координат и связанных с ними единиц измерения) [7]. Для отображения на плоском экране эти координаты должны быть спроецированы в плоскую систему, чаще всего в проекцию Меркатора – EPSG:3857, которая используется большинством картографических онлайн-сервисов: OpenStreetMap, Google Maps, Yandex.Map [7].

Проекция Меркатора сохраняет углы, но искажает площади и расстояния, особенно на высоких широтах. Это требует корректной трансформации координат перед рендерингом. В FlutterMap эта трансформация выполняется методом LatLngToPixel(), который преобразует географические координаты в пиксельные позиции на основе текущего масштаба, центра карты и размера экрана.

Сложность рендеринга возрастает, когда полигоны становятся многосвязными с внутренними «дырами» или содержат тысячи вершин (например, границы городов с учетом микрорайонов, береговых линий, лесных массивов). В таких случаях стандартный алгоритм отрисовки через Canvas.drawPath() становится неэффективным по следующим причинам:

1) каждый вызов drawPath() требует копирования всей последовательности вершин в буфер GPU;
2) отсутствует автоматическое сглаживание для малых масштабов;
3) не применяются алгоритмы упрощения геометрии (например, Douglas-Peucker);

4) не реализуется culling – отсечение невидимых полигонов за пределами view frustum (объем видимости виртуальной камеры).

Кроме того, Flutter не предоставляет встроенного механизма для батчинга – пакетной обработки геометрических примитивов. В отличие от WebGL (Web-based Graphics Library – кросс-платформенный API для отображения 3D-графики в браузере) или OpenGL ES (Open Graphics Library for Embedded Systems – программный интерфейс к графическому оборудованию, который позволяет указывать объекты и операции для создания высококачественных графических изображений, в частности цветных изображений трехмерных объектов), где можно отправить сотни полигонов одним вызовом glDrawElements(), Flutter вынуждает разработчика использовать либо CustomPaint, либо сторонние пакеты типа flutter_svg или flutter_map_3d, которые не решают проблемы производительности, а лишь добавляют дополнительный уровень абстракции.

### 1.4. АНАЛИЗ СУЩЕСТВУЮЩЕГО МЕТОДА КЛАСТЕРИЗАЦИИ

Если использовать популярные онлайн-карты (Google Maps, «Яндекс.Карты»), то можно заметить, что при уменьшении масштаба отдельные метки (например, кафе или АЗС) начинают сливаться в цифры: «12», «47», «203». Это кластеризация маркеров – один из самых известных и отлаженных приемов в геовизуализации. Она работает просто и эффективно. Система берет все точки в текущем viewport, группирует их поблизости (например, в ячейки сетки), заменяет каждую группу на одну метку с числом объектов внутри. Для точек это идеально. Геометрия тривиальна – одна пара координат, объединение – просто подсчет, визуализация – одна иконка и текст. Но важно отметить, что всё это работает только с точками. Эта технология реализована как для библиотеки flutter_map с помощью отдельного пакета flutter_map_marker_cluster [8], так и для других библиотек. Заметим, что эти пакеты не работают с полигонами, и готовые решения для работы с полигонами отсутствуют, поэтому кластеризация полигонов в мобильных ГИС сохраняет свою актуальность и в настоящее время.

## 2. МЕТОДЫ ОПТИМИЗАЦИИ РЕНДЕРИНГА ПОЛИГОНАЛЬНЫХ ОБЪЕКТОВ В FLUTTER MAP

### 2.1. ОТЛОЖЕННАЯ ЗАГРУЗКА И ОТРИСОВКА ПОЛИГОНОВ

Отложенная загрузка – стратегия, при которой объекты обрабатываются и отображаются только в момент их фактической необходимости вместо их предварительной обработки в полном объеме [9]. В контексте визуализации полигонов на карте это означает, что рендеринг происходит только для тех объектов, которые находятся в текущей области видимости пользователя – так называемом viewport, что также называют экстентом.

Отложенная загрузка основана на простом, но эффективном принципе: не отрисовывать то, что пользователь не видит. При перемещении или масштабировании карты viewport динамически изменяется, и система должна пересчи-

тывать, какие полигоны попадают в его границы. Все полигоны, находящиеся за пределами viewport, исключаются из процесса построения дерева виджетов и рендеринга, что позволяет существенно снизить нагрузку на поток, отвечающий за рендеринг пользовательского интерфейса, и GPU (Graphics Processing Unit – специализированный графический процессор, который обрабатывает данные, связанные с отображением изображений, видео и 3D-графики).

В рамках архитектуры Flutter, где каждый виджет – это объект в памяти, а каждый Polygon в flutter_map порождает собственный CustomPaint и RenderObject, отложенная загрузка становится критически важной при работе с тысячами полигонов. Без нее даже на мощных устройствах возникают задержки, пропуски кадров и повышенное потребление памяти.

В качестве примера нами реализовано приложение с подключенной библиотекой flutter_map. Это приложение принимает данные в виде файлов с расширением sqlite, сохраняющим данные по объектам, которые должны быть приведены на карте. Эти данные получаем SQL-запросом (Structured Query Language – язык структурированных запросов для работы с реляционными базами данных) к загруженному файлу. Далее эти данные кешируются в оперативную память и хранятся в переменной для удобного доступа к ним [10].

Данные, которые поместили в оперативную память, можем передавать в библиотеку flutter_map постепенно, по ходу движения пользователя по карте. Для этого необходимо знать границы экрана и соотнести их с границами на карте. В библиотеке имеется инструмент, который дает нужную информацию по границам. Библиотека передает их в виде объекта LatLngBounds. Благодаря этим данным можно получать только необходимые полигоны, а не целиком все сразу, и уменьшить нагрузку на рендеринг.

На рис. 1 представлен фрагмент кода, показывающий процесс выбора полигонов, которые находятся на экране. Также стоит отметить, что отображаются полигоны, имеющие несколько точек на экране. Точное количество пересчитывается в пропорции от общего числа внешних точек полигона, чтобы не засорять рендеринг.

В рассматриваемом коде имеется переменная visiblePolygonsController, которая является типом Stream, для потоковой передачи данных flutter_map. В этой библиотеке также есть встроенный инструмент для работы с потоковыми данными, благодаря которому легко можно передавать сразу на карту необходимые полигоны, и она будет своевременно предупреждена о новых данных.

Для такой реализации использовался также паттерн дебаунсер, так как он помогает сгладить рывки интерфейса, снизить нагрузку на процессоры устройства и избежать лишних вычислений при частых изменениях viewport, например, при быстром скроллинге или масштабировании карты.

Дебаунсер – это программный паттерн, который откладывает выполнение функции до тех пор, пока не пройдет определенный интервал времени без новых вызовов [11–13]. Если на этом интервале поступает новый вызов – таймер сбрасывается и выполнение вновь откладывается.

Этот паттерн помогает в ситуациях, когда при перемещении карты во flutter_map событие изменения позиции может срабатывать десятки раз

в секунду, особенно при перемещении по карте или различных скроллингах. Паттерн начинает свою работу после того, как пользователь остановился на каком-то месте на карте, и если в течение определенного времени не было изменений в карте, то подгружаются данные.

```
/// Метод возвращает список полигонов, которые находятся в зоне экрана
///
/// [bounds] - текущие границы видимой области карты
List<Polygon> getVisiblePolygons(LatLngBounds bounds) {
  // Фильтруем исходный список полигонов, оставляя только те,
  // которые попадают в заданные географические границы
  return listPolygones.where((polygon) {
    return _isPolygonInBounds(polygon, bounds);
  }).toList();
}

/// Проверяет, находится ли полигон в пределах заданных границ карты
///
/// Возвращает `true`, если хотя бы одна вершина полигона
/// попадает в область [bounds]
bool _isPolygonInBounds(Polygon polygon, LatLngBounds bounds) {
  // Проходим по всем точкам (вершинам) полигона
  for (final point in polygon.points) {
    // Если хотя бы одна точка находится внутри границ,
    // считаем весь полигон видимым
    if (bounds.contains(point))
      return true;
  }
  // Если ни одна точка не попала в границы, полигон невидим
  return false;
}

/// Обновляет поток видимых полигонов и уведомляет подписчиков
void updateVisiblePolygons(LatLngBounds bounds) {
  // Получаем отфильтрованный список полигонов для текущей области
  final visiblePolygons = getVisiblePolygons (bounds);
  // Отправляем обновленный список в StreamController
  _visiblePolygonsController.add(visiblePolygons);
}
```

*Рис. 1.* Фрагмент кода для выбора полигонов

*Fig. 1.* Code snippet for selecting polygons

## 2.2. КЛАСТЕРИЗАЦИЯ ПОЛИГОНОВ

При визуализации больших объемов геопространственных данных на мобильных устройствах, особенно в условиях ограниченных вычислительных ресурсов, возникает необходимость не только отбирать объекты для отображения, как при ленивой загрузке, но и сокращать их количество за счет семантического или геометрического объединения. Для этого применяется метод кластеризации и агрегации полигонов – подход, при котором множество отдельных полигонов, близко расположенных или логически связанных, объединяются в один составной объект, отображаемый на карте как единое целое [14–18].

Этот метод особенно эффективен на мелких масштабах, когда детализация отдельных объектов теряется, а их визуальное скопление приводит к перегруженности карты и снижению читаемости. Агрегация позволяет сохранить смысловую нагрузку данных, одновременно снижая нагрузку на рендер-конвейер Flutter.

Кластеризация делится на две части: геометрическое объединение и семантическая агрегация.

Геометрическое объединение – физическое слияние пересекающихся или близко расположенных полигонов в один составной полигон с общей границей.

Семантическая агрегация – объединение полигонов по признаку принадлежности к одной категории, региону или типу (например, все полигоны зон застройки в районе, один полигон района с меткой «количество входящих в него полигонов»).

В обоих случаях результат – уменьшение количества рендер-объектов, что напрямую снижает количество виджетов Polygon в дереве Flutter, число вызовов Canvas.drawPath(), объем памяти, занимаемый геометрией и стилями, а также нагрузку на графический процессор и поток пользовательского интерфейса.

В отличие от точечных объектов, для которых кластеризация реализована во многих библиотеках, для полигонов готовых решений нет, что делает эту задачу особенно актуальной.

В реализованном нами приложении использована следующая логика: группирование полигонов по географической близости; группирование по идентичным семантическим признакам (к примеру, лес, озеро, промышленная зона и т. д.).

Реализация представляет собой объединение по расстоянию между полигонами: берутся координаты каждого полигона и вычисляется площадь, самый большой полигон остается как полигон, который будет отображаться; задается расстояние между полигонами. Если расстояние меньше 10 пикселей, то полигоны объединяются в один. Для семантического объединения используем данные объектов, которые заранее известны.

На рис. 2, *а* показано, как были отрисованы полигоны до применения кластеризации; на рис. 2, *б* представлены полигоны после применения кластеризации; рис. 2, *б* визуально представляет суть кластеризации полигонов в мобильном приложении. Видно, что полигоны скрыты за самым большим полигоном, но информация о их существование не теряется.




*а* *б*

*Рис. 2.* Полигоны до применения кластеризации (*а*), после применения кластеризации (*б*)

*Fig. 2.* Polygons before clusterization (*a*), after clusterization is applied (*b*)

## 3. ЭКСПЕРИМЕНТАЛЬНЫЕ ИССЛЕДОВАНИЯ ПОДХОДОВ ОПТИМИЗАЦИИ РЕНДЕРИНГА

Для оценки эффективности предложенных методов оптимизации – ленивой загрузки и кластеризации полигонов – была проведена серия экспериментов на симуляторе мобильного устройства Pixel 8 Pro с версией андроид 13 в условиях, максимально приближенных к эксплуатационным: отсутствие подключения к Wi-Fi, фоновые процессы, запущенные приложения.

В качестве тестового набора данных использовался массив из 5000 полигонов, представляющих условные участки промышленной застройки. Каждый полигон содержал в среднем 8–12 вершин. Полигоны были сгруппированы в плотные кластеры, примерно по 50–200 объектов на квадратный километр, при этом имитировались реальные условия размещения инфраструктурных объектов. Также запускалось приложение без использования группировки в кластеры и без использования отложенной загрузки. Результаты экспериментального исследования представлены в таблице.

**Результаты оптимизации**

**Optimization results**

| Показатель | Базовая реализация | Оптимизированная реализация | Улучшение |
|---|---|---|---|
| FPS (средний) | 18…25 | 52…58 | в 2,5…3 раза |
| Пиковое потребление RAM | 580 МБ | 190 МБ | –67 % |
| Время отклика на жесты | 300…500 мс | 40…80 мс | в 6…7 раз |
| Количество рендер-объектов | 5000 | 85…120 (кластеры) | в 40…60 раз |

Экспериментальное исследование подтвердило высокую эффективность комбинированного применения ленивой загрузки и кластеризации полигонов по географической близости в мобильных ГИС-приложениях на Flutter с использованием библиотеки flutter_map.

Достигнуто трехкратное увеличение FPS и снижение потребления памяти на 67 %. Интерфейс стал отзывчивым и плавным даже на устройствах среднего класса. Приложение стало энергоэффективным и стабильным при длительной работе.

Результаты показывают, что даже в условиях архитектурных ограничений Flutter – декларативный рендеринг, отсутствие батчинга на уровне фреймворка – можно добиться высокой производительности за счет оптимизации логики отображения данных, а не переписывания ядра приложения.

## ЗАКЛЮЧЕНИЕ

В условиях стремительного расширения сферы применения мобильных ГИС и постоянно возрастающих требований к их производительности, отзывчивости и энергоэффективности разработка оптимизированных методов визуализации картографических данных становится не только актуальной научно-технической задачей, но и ключевым фактором успешного внедрения профессиональных решений в полевых условиях применения ГИС. Особенно это актуально для отраслей, где стабильность и скорость работы мобильных приложений ГИС напрямую влияют на эффективность выполнения задач: геологии, нефтегазовой разведки, логистики, экологического мониторинга и т. п.

В данной работе решена научно-практическая задача оптимизации рендеринга сложных полигональных объектов в мобильном приложении на Flutter с учетом архитектурных особенностей фреймворка и ограничений мобильных устройств. Разработаны и экспериментально проверены два ключевых метода.

Отложенная загрузка полигонов с применением дебаунсера позволила исключить из рендеринга невидимые объекты и сгладить частоту пересчетов при активном взаимодействии с картой местности.

Кластеризация и агрегация полигонов по географической близости обеспечила снижение количества рендер-объектов за счет семантически корректного объединения соседних полигонов в составные объекты.

Экспериментальная оценка показала значительное улучшение ключевых метрик производительности: средний FPS увеличился, потребление

оперативной памяти снизилось, время отклика на пользовательские жесты уменьшилось.

Результаты исследования подтверждают, что даже в условиях декларативной архитектуры Flutter, где каждый виджет – это отдельный объект, а рендеринг основан на перестройке дерева, можно добиться высокой производительности за счет логики управления данными, а не переписывания фреймворка или перехода на нативную разработку. Это делает Flutter перспективной платформой для создания высоконагруженных ГИС-приложений, особенно в условиях кросс-платформенности и ограниченных ресурсов.

Предложенные методы не требуют глубокой модификации существующего кода – они легко интегрируются в архитектуру flutter_map и могут быть адаптированы под любые типы полигонов и масштабы.

## СПИСОК ЛИТЕРАТУРЫ

1. *Гомбоин А.Б., Кориков А.М.* Параллельная обработка географических данных в мобильном приложении «Мобильная ГИС» // Доклады ТУСУР. – 2024. – Т. 27, № 2. – С. 51–56. – DOI: 10.21293/1818-0442-2024-27-2-51-56.

2. Official Flutter Documentation. – URL: https://docs.flutter.dev (accessed: 01.09.2025).

3. Dart: Language tour и performance best practices. – URL: https://dart.dev/guides/language/language-tour (accessed: 01.09.2025).

4. *Колтыгин Д.С., Седельников И.А., Крумин О.К.* Разработка кросс-платформенного приложения для управления робототехническим комплексом с помощью Flutter // Системы анализа и обработки данных. – 2025. – № 1 (97). – С. 7–26. – DOI: 10.17212/2782-2001-2025-1-7-26.

5. Google. Flutter Performance Best Practices. – URL: https://docs.flutter.dev/perf/rendering (accessed: 01.09.2025).

6. Flutter Map documentation. – URL: https://github.com/fleaflet/flutter_map (accessed: 01.09.2025).

7. *Dhonju H.K., Walsh K.B., Bhattarai T.* Web mapping for farm management information systems: A review and Australian orchard case study // Agronomy. – 2023. – Vol. 10. – P. 2563.

8. Flutter Map Marker Cluster. – URL: https://github.com/lpongetti/flutter_map_marker_cluster (accessed: 01.09.2025).

9. Optimizing Flutter apps for performance: A practical guide. – URL: https://medium.com/flutter/optimizing-flutter-apps-for-performance-7d7b3b45a1a2 (accessed: 01.09.2025).

10. *Silva Y.N., Almeida I., Queiroz M.* SQL: From traditional databases to big data // Proceedings of the 47th ACM Technical Symposium on Computing Science Education. – ACM, 2016. – P. 413–418.

11. *Yang L., Lan Q.* Front-end performance optimisation: An exploration of debounce and throttling strategies in image rendering // 2024 3rd International Conference on Cloud Computing, Big Data Application and Software Engineering. – IEEE, 2024. – P. 504–508.

12. *Ganssle J.G.* A guide to debouncing / The Ganssle Group. – Baltimore, MD, US, 2004. – 22 p.

13. *Нуждин Д.Г.* Оптимизация производительности мобильных приложений: стратегии и лучшие практики // Вестник науки. – 2023. – № 11, т. 3. – С. 850–862. – DOI: 10.24412/2712-8849-2023-1168-850-862.

14. *Douglas D.H., Peucker T.K.* Algorithms for the reduction of the number of points required to represent a digitized line or its caricature // The Canadian Cartographer. – 1973. – Vol. 10. – P. 112–122.

15. *Полищук Ю.М., Кочергин Г.А.* Использование геоинформационных систем для комплексного анализа пространственных данных на основе многомерной кластеризации // Геоинформатика. – 2011. – № 2. – С. 11–15.

16. *Колесенков А.Н.* Современные подходы к обработке данных при построении геоинформационных систем экологического мониторинга // Известия Тульского государственного университета. Технические науки. – 2016. – № 9. – С. 103–112.

17. *Hu Z., Li B., Luo J.* Flutter: Scheduling tasks closer to data across geo-distributed datacenters // IEEE INFOCOM 2016 – The 35th Annual IEEE International Conference on Computer Communications. – IEEE, 2016. – P. 1–9.

18. Development of a platform-independent renderer for the rendering of openstreetmap indoor maps in Flutter / J. Richter, R. Thomas, D. Lange, T. Graichen, U. Heinkel // Proceedings of the 17th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP). – 2022. – Vol. 1. – P. 147–154.

*Гомбоин Александр Булатович*, аспирант кафедры автоматизированных систем управления Томского государственного университета систем управления и радиоэлектроники. Основное направление научных исследований – мобильные геоинформационные системы и их оптимизация. Имеет 10 научных публикаций. E-mail: a.gomboin@mail.ru

*Кориков Анатолий Михайлович*, доктор технических наук, профессор, заслуженный деятель науки РФ, академик МАН ВШ и МАИ, профессор кафедры автоматизированных систем управления Томского государственного университета систем управления и радиоэлектроники (ТУСУР), заслуженный профессор ТУСУР. Основное направление научных исследований – автоматизация и оптимизация систем обработки информации и управления. Имеет более 300 научных публикаций, в том числе 12 монографий. E-mail: anatolii.m.korikov@tusur.ru

*Gomboin Alexander B*., PhD student at the Department of Automated Control Systems, Tomsk State University of Control Systems and Radio Electronics. The main area of his scientific research is mobile geoinformation systems and their optimization. He has 10 scientific publications. E-mail: a.gomboin@mail.ru

*Korikov Anatoly M.,* Doctor of Engineering Sciences, Professor, Professor at the Automated Control Systems Department in Tomsk State University of Control Systems and Radioelectronics (TUSUR), Honored Scientist of the Russian Federation, Academician of the Higher School. The main field of his scientific research is automation and optimization of information processing and control systems. He has more than 300 scientific publications, including 12 monographs. E-mail: anatolii.m.korikov@tusur.ru

DOI: 10.17212/2782-2001-2026-3-7-22

## *Optimization of cartographic data rendering in mobile geoinformation systems on the Flutter framework*[*]

*A.B. GOMBOIN*[a], *A.M. KORIKOV*[b]

*Tomsk State University of Control Systems and Radio Electronics, 40 Lenin Avenue, Tomsk, 634050, Russian Federation*

[a] *a.gomboin@mail.ru* [b] *anatolii.m.korikov@tusur.ru*

**Abstract**

This article discusses the problem of optimizing the rendering of complex polygonal objects in mobile geographic information systems (GIS) implemented on the Flutter cross-platform framework. In conditions of limited computing resources of mobile GIS devices and high

[*] *Received 05 October 2025.*

requirements for interface responsiveness, standard approaches to visualization of cartographic data used in the flutter_map library lead to critical degradation of mobile GIS performance: a reduced frame rate (FPS – Frames Per Second), an increased memory consumption and an unstable response to user gestures.

To ensure an efficient operation of mobile GIS, the article proposes two methods: lazy loading of polygons to delay recalculation until the user's interaction with the map (cartographic data) is completed, and clustering polygons by geographical proximity followed by geometric integration into composite objects. An experimental study conducted on a set of 5,000 polygons using a real mobile device showed that the combined use of optimization methods made it possible to increase average FPS, reduce RAM consumption, and improve user experience assessment. The results of the experimental study confirm that even under the conditions of Flutter's architectural limitations, it is possible to achieve high performance of mobile GIS by optimizing the logic of data display.

The proposed methods for optimizing the rendering of cartographic data do not require deep modification of the existing code – they are easily integrated into the architecture of the flutter_map library and can be adapted to any type of polygon and scale, which makes the Flutter framework a promising platform for developing professional mobile GIS applications, including for work in the field with minimal resources.

**Keywords**: rendering, map, mobile application, GIS, mobile GIS, Flutter, flutter_map, polygons, optimization, lazy loading of polygons, clusterization of polygons, combination of optimization methods, viewport, geospatial data, OpenStreetMap

## REFERENCES

1. Gomboin A.B., Korikov A.M. Parallel'naya obrabotka geograficheskikh dannykh v mobil'nom prilozhenii «Mobil'naya GIS» [Parallel processing of geographical data in a mobile application]. *Doklady TUSUR = Proceedings of TUSUR University*, 2024, vol. 27, no. 2, pp. 51–56. DOI: 10.21293/1818-0442-2024-27-2-51-56.

2. *Official Flutter Documentation*. Available at: https://docs.flutter.dev (accessed 01.09.2025).

3. *Dart: Language tour u performance best practices*. Available at: https://dart.dev/guides/language/language-tour (accessed 01.09.2025).

4. Koltygin D.S., Sedelnikov I.A., Krumin O.K. Razrabotka kross-platformennogo prilozheniya dlya upravleniya robototekhnicheskim kompleksom s pomoshch'yu Flutter [Development of a cross-platform application for controlling a robotic complex using Flutter]. *Sistemy analiza i obrabotki dannykh = Analysis and Data Processing Systems*, 2025, no. 1 (97), pp. 7–26. DOI: 10.17212/2782-2001-2025-1-7-26.

5. Google. *Flutter Performance Best Practices*. Available at: https://docs.flutter.dev/perf/rendering (accessed 01.09.2025).

6. *Flutter Map documentation*. Available at: https://github.com/fleaflet/flutter_map (accessed 01.09.2025).

7. Dhonju H.K., Walsh K.B., Bhattarai T. Web mapping for farm management information systems: A review and Australian orchard case study. *Agronomy*, 2023, vol. 10, p. 2563.

8. *Flutter Map Marker Cluster*. Available at: https://github.com/lpongetti/flutter_map_marker_cluster (accessed 01.09.2025).

9. *Optimizing Flutter apps for performance: A practical guide*. Available at: https://medium.com/flutter/optimizing-flutter-apps-for-performance-7d7b3b45a1a2 (accessed 01.09.2025).

10. Silva Y.N., Almeida I., Queiroz M. SQL: From traditional databases to big data. *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, 2016, pp. 413–418.

11. Yang L., Lan Q. Front-end performance optimisation: An exploration of debounce and throttling strategies in image rendering. *2024 3rd International Conference on Cloud Computing, Big Data Application and Software Engineering*. IEEE, 2024, pp. 504–508.

12. Ganssle J.G. *A guide to debouncing*. Ganssle Group. Baltimore, MD, US, 2004. 22 p.

13. Nuzhdin D.G. Optimizatsiya proizvoditel'nosti mobil'nykh prilozhenii: strategii i luchshie praktiki [Optimization of mobile application performance: strategies and best practices]. *Vestnik nauki*, 2023, no. 11, vol. 3, pp. 850–862. DOI: 10.24412/2712-8849-2023-1168-850-862. (In Russian).

14. Douglas D.H., Peucker T.K. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 1973, vol. 10, pp. 112–122.

15. Polishchuk Yu.M., Kochergin G.A. Ispol'zovanie geoinformatsionnykh sistem dlya kompleksnogo analiza prostranstvennykh dannykh na osnove mnogomernoi klasterizatsii [Use of geoinformation systems for complex analysis of spatial data based on multidimensional clusterization]. *Geoinformatika*, 2011, no. 2, pp. 11–15. (In Russian).

16. Kolesenkov A.N. Sovremennye podkhody k obrabotke dannykh pri postroenii geoinformatsionnykh sistem ekologicheskogo monitoringa [Modern approaches to the processing of data in the construction of information systems of environmental monitoring]. *Izvestiya Tul'skogo gosudarstvennogo universiteta. Tekhnicheskie nauki = News of the Tula state university. Technical sciences*, 2016, no. 9, pp. 103–112.

17. Hu Z., Li B., Luo J. Flutter: Scheduling tasks closer to data across geo-distributed datacenters. *IEEE INFOCOM 2016 – The 35th Annual IEEE International Conference on Computer Communications*. IEEE, 2016, pp. 1–9.

18. Richter J., Thomas R., Lange D., Graichen T., Heinkel U. Development of a platform-independent renderer for the rendering of openstreetmap indoor maps in Flutter. *Proceedings of the 17th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP)*, 2022, vol. 1, pp. 147–154.

Для цитирования:

*Гомбоин А.Б., Кориков А.М.* Оптимизация рендеринга картографических данных в мобильных геоинформационных системах на фреймворке Flutter // Системы анализа и обработки данных. – 2026. – № 3 (103). – С. 7–22. – DOI: 10.17212/2782-2001-2026-3-7-22.

For citation:

Gomboin A.B., Korikov A.M. Optimizatsiya renderinga kartograficheskikh dannykh v mobil'nykh geoinformatsionnykh sistemakh na freimvorke Flutter [Optimization of cartographic data rendering in mobile geoinformation systems on the Flutter framework]. *Sistemy analiza i obrabotki dannykh = Analysis and Data Processing Systems*, 2026, no. 3 (103), pp. 7–22. DOI: 10.17212/2782-2001-2026-3-7-22.