Вечканов А.В., Куц Л.В., Качалин С.В. Сравнение технологий отрисовки групп изображений с использованием фрейморка QT 5. // Проблемы информатики в образовании, управлении, экономике и технике: Сб. статей XVIII Междунар. научно-техн. конф. – Пенза: ПДЗ, 2018. – С. 90-96.

УДК 004.42

СРАВНЕНИЕ ТЕХНОЛОГИЙ ОТРИСОВКИ ГРУПП ИЗОБРАЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ ФРЕЙМВОРКА QT 5

А.В. Вечканов, Л.В. Куц, С.В. Качалин

IMAGE RENDER COMPARISON BY USING QT 5 FRAMEWORK

A.V. Vechkanov, L.V Kuts, S.V. Kachalin

Аннотация. В статье рассмотрены различные методы реализации отрисовки изображений с использованием фреймворка Qt 5. Проведен сравнительный анализ нагрузки на центральный процессор для каждого метода.

Ключевые слова: отрисовка изображений, Qt 5, центральный процессор, графический процессор.

Abstract. This article describes different approaches of image rendering by using Qt 5 framework. There was made a comparative analysis of each approach for comparison purposes of CPU loading.

Keywords: image rendering, Qt 5, CPU, GPU.

Введение

На текущий момент существует потребность в оснащении министерств и ведомств программными компонентами отображения картографических данных в различных представлениях с привязкой к координатам. В данной области есть ряд задач, одной из которых является отображение множества объектов на карте с минимальной нагрузкой на ресурсы электронно-вычислительного устройства. Задачу можно описать как отображение множества объектов поверх фонового изображения высокого разрешения. Ключевым моментом является наличие возможности перемещения всей группы изображений (включая фоновое) – так называемая прокрутка, или скроллинг карты. Перемещение изображения должно выполняться в реальном времени путем перетаскивания карты с помощью мыши (или пальца на сенсорном экране), при этом нагрузка на центральный процессор (ЦП) должна быть минимальной, чтобы обеспечить возможность стабильной фоновой работы других критически важных процессов, работающих в программном обеспечении (ПО).

Различные подходы к реализации поставленной задачи

Для обеспечения наибольшей гибкости и масштабируемости программного компонента принято решение использовать Qt 5. Это современный кроссплатформенный фреймворк для разработки программ-много обеспечения на языке C++ [1], основными преимуществами которого являются подробная документация и удобная кроссплатформенность.

С использованием данного фреймворка поставленную задачу можно решать двумя способами:

- с использованием QtQuick;
- с использованием QtWidgets.

QtQuick – это современная технология создания пользовательского интерфейса, особенностью которой является разделение декларативного описания дизайна интерфейса и императивной логики программирования. На уровне представления приложения реализуются с помощью языка QML [2]. Данная технология предоставляет расширенные возможности при создании интерфейсов для мобильных и встраиваемых приложений, хотя ее также можно использовать и при разработке десктопных приложений.

QtWidgets – это технология для создания так называемых нативных интерфейсов, внешний вид которых не отличается от общепринятых в используемой операционной системе. Данная технология больше ориентирована на создание десктопных приложений [3].

На рис. 1 представлены различные способы реализации поставленной задачи в Qt 5.

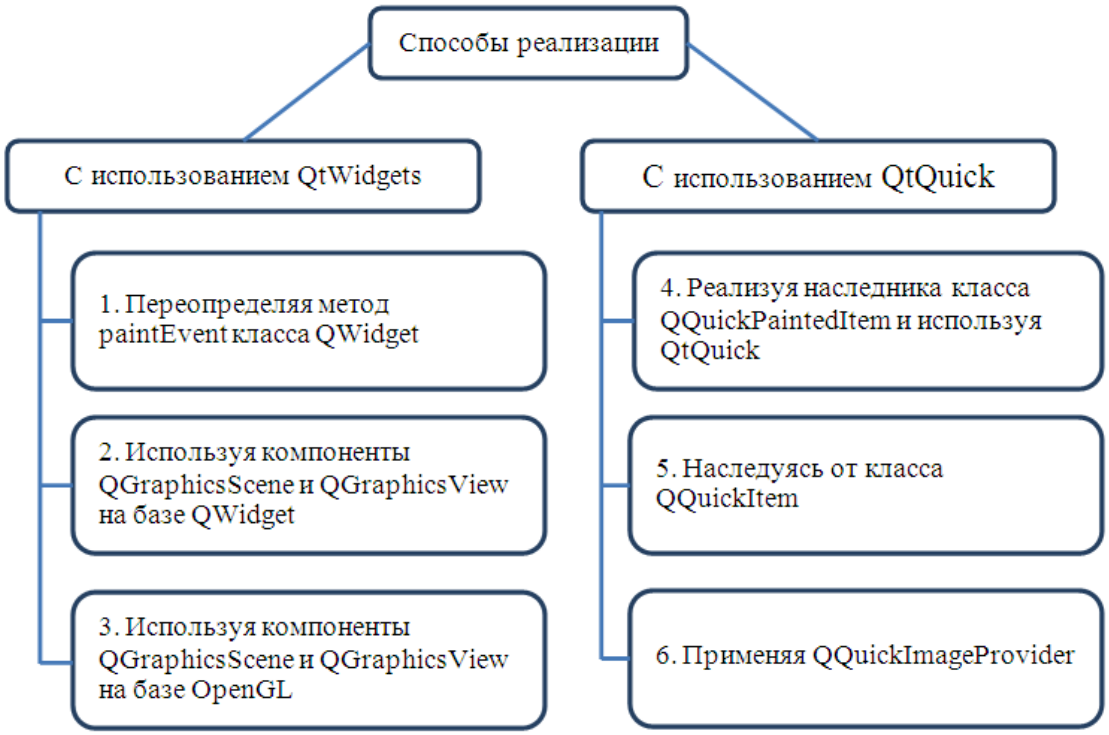

Рис. 1. Способы реализации

Первый подход к рендеру карты заключается в переопределении метода QWidget::paintEvent у главного окна [4]. В данном методе с помощью QPainter сначала на фоне отрисовывается карта, а затем уже цели поверх карты. Плюсом данного подхода является простота его реализации, к минусам можно отнести то, что отрисовка происходит на ЦП без использования графической карты (GPU).

В следующем подходе на графической сцене QGraphicsScene карта и цели отображаются с помощью QGraphicsPixmapItem. Все это выводится на виджет QGraphicsView [5]. Данный метод несколько сложнее предыдущего в реализации, но использование подсистемы QGraphics дает большую гибкость при разработке. Данная подсистема неплохо оптимизирована и может отображать десятки тысяч объектов [6].

Если предыдущий подход расширить за счет использования OpenGL для рендера внутри QGraphicsView, мы получаем еще один метод решения поставленной

задачи. Плюсом данного подхода является задействование графической карты, которая может взять на себя часть работы и снизить нагрузку на ЦП.

Четвертый подход очень похож на первый за исключением того, что в нем используется QtQuick. Поэтому отрисовка происходит не в главном окне, а в методе QQuickPaintedItem::paint у элемента, который отображается на главном окне [7]. Данный подход, пожалуй, наиболее прост и удобен в реализации при использовании QtQuick.

Используя наследника класса QQuickItem, в котором мы переопределяем метод QQuickItem::updatePaintNode, получаем еще один способ рендера нашей карты. При этом нам приходится работать с SceneGraphics примитивами, такими как QSGGeometryNode, QSGTextureMaterial и QSGTexture [8]. В отличие от предыдущего подхода данный метод наиболее сложен в реализации на QtQuick.

В последнем подходе необходимо создать наследника класса QQuickImageProvider, который будет возвращать результирующие изображения нашей карты с целями, которые уже будут отображаться в QML с использованием Image – примитива для отображения изображений [9]. При этом перемещение этого изображения будет производиться средствами QML. Данный подход является скорее экспериментальным, так как его использование для поставленной задачи весьма затруднительно. Трудности связаны в первую очередь с обновлением карты, поскольку примитив Image по умолчанию кеширует изображение, вследствие чего прямое обновление данных из QQuickImageProvider в Image является затруднительным.

Описание тестового приложения

Для выбора наиболее оптимального варианта реализации необходимо для каждого из вышеперечисленных способов разработать тестовое приложение, реализующее поставленную задачу. После этого необходимо провести практические испытания этих приложений, замерив нагрузку на ЦП. Полученную информацию можно проанализировать с целью выявления наиболее оптимального подхода к реализации описанной задачи.

Тестовое приложение должно выполнять следующие функции:

– загружать и отображать изображения карты;

– загружать и отображать изображения целей поверх карты,

– производить имитацию перемещения всех изображений (прокрутки карты).

Для исключения влияния человеческого фактора на результаты необходимо, чтобы тестовая программа производила имитацию действий пользователя (автоматически прокручивать карту согласно заданному алгоритму). Внешний вид тестового приложения представлен на рис. 2. Стоит понимать, что использование подобной автоматизации приведет к увеличению нагрузки на ЦП за счет паразитного потребления ресурсов (overhead) [10]. Данный overhead необходимо учитывать для более точного измерения нагрузки на центральный процессор.

Для снижения нагрузки на ЦП применяется технология кеширования: у картографической подсистемы запрашивается изображение карты намеренно большего размера, чем то, что будет отображаться в окне. Данный подход уменьшает число обращений к картографической подсистеме, что снижает нагрузку на центральный процессор за счет потребления дополнительной оперативной памяти.

Простая перерисовка изображения при его перемещении относительно окна менее затратна, чем рендер нового изображения при обращении к картографической подсистеме.

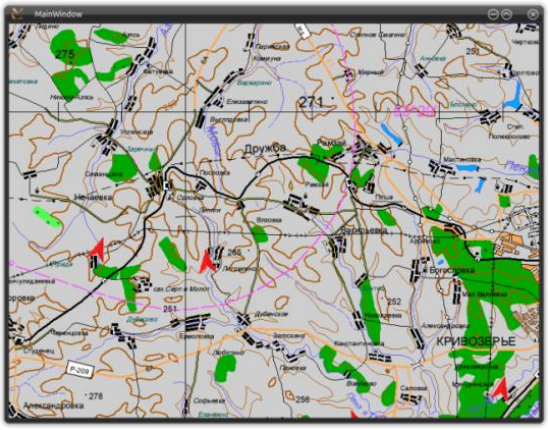

Рис. 2. Внешний вид тестового приложения

Методика проведения испытаний

Для исключения влияния различных факторов на результат изме-рения нагрузки на ЦП необходимо провести многократные замеры нагрузки с интервалом в 1 с в течение 30 с. Среднее арифметическое получен-ных результатов покажет достаточно верный результат использования ресурсов центрального процессора. Испытания проводились на двух электронно-вычислительных машинах (ЭВМ):

– с наличием графического процессора (ГП);
– без наличия ГП (табл. 1, 2).

Таблица 1

Результаты испытаний на ЭВМ с ГП

Используемый подход	Общая нагрузка на ЦП, %	Overhead нагрузка автоматизации на ЦП, %	Реальная нагрузка на ЦП, %
QWidget	9.58	1.32	8.26
QGraphicsScene – QWidget	8.32	1.27	7.05
QGraphicsScene – OpenGL	8.81	1.31	7.50
QQuickPaintedItem	34.78	1.60	33.18
QQuickItem	3.87	3.60	0.17
QQuickImageProvider	4.12	1.95	2.17

Таблица 2

Результаты испытаний на ЭВМ без ГП

Используемый подход	Общая нагрузка на ЦП, %	Overhead нагрузка автоматизации на ЦП, %	Реальная нагрузка на ЦП, %
QWidget	13.05	1.27	11.78
QGraphicsScene - QWidget	10.94	1.65	9.29
QGraphicsScene - OpenGL	77.70	3.55	74.15
QQuickPaintedItem	67.68	2.50	65.18
QQuickItem	82.10	81.04	1.06
QQuickImageProvider	68.41	3.05	65.36

Выводы

Полученные результаты практических испытаний, представленные в табл. 1, 2, позволяют оценить общую тенденцию потребления ресурсов электронно-вычислительной машины каждым из вышеописанных подходов при отрисовке картографических данных. Проанализировав полученные результаты, делаем вывод, что отрисовка карты с помощью QGraphicsScene и QGraphicsView является достаточно универсальным методом, который показывает приемлемые результаты вне зависимости от наличия или отсутствия графического процессора. Чуть худший результат показывает самый простой в реализации метод – переопределение метода paintEvent у класса QWidget.

Однако если на ЭВМ установлен графический процессор, то менее ресурсоемким является применение наследника QQuickItem. Данный метод меньше всех нагружает процессор, но следует учитывать, что его реализация несколько сложнее всех остальных. Использование указанного метода без наличия ГП само по себе дает увеличенную нагрузку на ЦП, так как приходится эмулировать работу графической подсистемы, что прекрасно видно по завышенным результатам измерения overhead нагрузки автоматизации.

Методы, ориентированные на использование ГП, показывают ожидаемо хороший результат при наличии графического процессора и совершенно неприемлемы для использования при его отсутствии.

Библиографический список

1. Qt [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Qt (дата обращения: 04.07.2018).

2. Qt Quick [Электронный ресурс]. URL: https://www1.qt.io/ru/qt-quick (дата обращения: 04.07.2018).

3. Qt Widgets [Электронный ресурс]. URL: https://doc.qt.io/qt-5/qwidgets-index.html (дата обращения: 04.07.2018).

4. QWidget [Электронный ресурс]. URL: https://doc.qt.io/qt-5/qwidget.html#paintEvent (дата обращения 04.07.2018).

5. QGraphicsScene [Электронный ресурс]. URL: https://doc.qt.io/qt-5/qgraphicsscene.html (дата обращения: 04.07.2018).

6. 40000 Chips [Электронный ресурс]. URL: https://doc.qt.io/qt-5/qtwidgets-graphicsview-chip-example.html (дата обращения: 04.07.2018).

7. QQuickPaintedItem [Электронный ресурс]. URL: https://doc.qt.io/qt-5/qquickpainteditem.html (дата обращения: 04.07.2018).

8. QQuickItem [Электронный ресурс]. URL: https://doc.qt.io/qt-5/qquickitem.html (дата обращения 04.07.2018).

9. QQuickImageProvider [Электронный ресурс]. URL: https://doc.qt.io/qt-5/qquickimageprovider.html (дата обращения: 04.07.2018).

10. Overhead (computing) [Электронный ресурс]. URL: https://wikipedia.org/wiki/Overhead (computing) (дата обращения: 04.07.2018).

Вечканов Андрей Викторович
АО «НПП «Рубин»,
г. Пенза, Россия

Vechkanov A.V.
SC «SIE «Rubin»,
Penza, Russia

Куц Леонид Валентинович
АО «НПП «Рубин»,
г. Пенза, Россия
E-mail: kuts_leonid@mail.ru

Качалин Сергей Викторович
АО «НПП «Рубин»,
г. Пенза, Россия
E-mail: s.kachalin@gmail.com

Kuts L.V.
SC «SIE «Rubin»,
Penza, Russia

Kachalin S.V.
SC «SIE «Rubin»,
Penza, Russia