щадь по сути дела вычисляется по приведённым формулам, а рисунок по этим же данным можно сделать по стандартной программе для отображения радарных диаграмм [3], один из вариантов которой можно бесплатно получить через Интернет, например, в системе EXSEL.

Заключение. Использование метода радарных диаграмм относительно сложно, так как при привлечении экспертов возникает проблема их выбора. Однако при решении задач соблюдается единый подход, независящий от личных симпатий ЛПР при оценке качества изделий и вмешательстве в вопросы подбора кадров. Предлагаемый вариант метода имеет преимущество перед другими, так как позволяет избежать поиска образцового изделия или процесса. Вместо этого в скрытой форме фактически используется идеальный объект (в природе его может не существовать), так как его представляют лучшие значения признаков из множества объектов, исследуемой совокупности процессов, изделий и т. п. Ключевым моментом является однородность объектов (их оценка всегда ведётся по полному набору всех признаков). Исследование радарных диаграмм показало, что во многих случаях они дают результаты, близкие к другим моделям (геометрических средних и др.) [2].

Часто при решении только задачи определения рейтинга объекта можно ограничиться более простой моделью радарной диаграммы, когда все углы между радиусами принимаются равными ($a=360^0/n$ или $2\pi/n$). Рейтинг объекта при построении комплексной системы можно получить и с помощью нейронной сети [4].

Качественные показатели рекомендуется оценивать в баллах, что позволяет сохранить единую методику расчётов

Применение облачных технологий, рекомендованных в [1], может привести к экономии средств на малых предприятиях при совершенствовании их информационной инфраструктуры. Однако на этом пути пока много трудностей по взаимодействию с фирмами владельцами средств, требуемых для реализации этой технологии.

СПИСОК ЦИТИРОВАННЫХ ИСТОЧНИКОВ

1. Провалов, В. С. Информационные технологии в малом бизнесе: особенности использования / В. С. Провалов // Естественные и математические науки в современном мире : сб. ст. по материалам XIX Междунар. науч.-практ. конф. – Новосибирск : СибАК. – 2014. – № 6 (18). – С. 43–48.
2. Матюшков, Л. П. Диаграммный метод оценки сложных однородных объектов / Л. П. Матюшков, М. Н. Григорович // Вестник БрГУ. – 2009. – № 1 (36). – С. 136–142.
3. Программа вычерчивания радарных и лепестковых диаграмм (бесплатно разрешается использовать в EXEL).
4. Головко, В. А. Сетецентрические системы и их адаптация / В. А. Головко, А. Л. Матюшков, Г. Л. Матюшкова, В. С. Рубанов // Вестник БрГТУ. – № 5. – 2018.

Материал поступил в редакцию 26.03.2019

MATYUSHKOV A. L., MATYUSHKOVA G. L., VAITSEKHOVICH O. U., RUBANOV V. S. IT of equipment selection and quality assessment of products and processes for small enterprises

An algorithm is developed to assess the quality products (services) and management processes of a small enterprise, as well as the purchased equipment, software and information support.

УДК 004.2

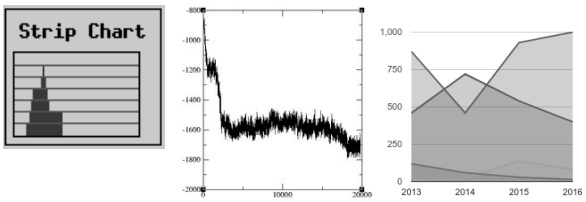
Латий О. О.

РЕАЛИЗАЦИЯ УЛУЧШЕННОЙ ОТРИСОВКИ ГРАФИКОВ В РЕАЛЬНОМ МАСШТАБЕ ВРЕМЕНИ ДЛЯ QT-ПРИЛОЖЕНИЙ

1. Введение. В ряде задач, связанных, в частности, с передачей в компьютер потока данных от измерительных средств, возникает необходимость оперативной визуализации получаемых численных значений в виде обновляемых в реальном времени кривых. Очевидно, что жесткие временные ограничения накладывают дополнительные условия на программные компоненты, выполняющие отрисовку графика по поступающим данным [1–4].

С момента появления виджет-тулкитов (библиотек с наборами виджетов – элементов управления для построения графического интерфейса пользователя) в их составе появлялись специализированные элементы для вывода графиков. Так, один из первых виджет-тулкитов, собственно введший в обиход слово «виджет» – «Проект Афина», разработанный в 1983 году МТИ при участии DEC и IBM для создания распределенной вычислительной среды университетского кампуса – содержал в своем составе компонент StripChart, предназначенный для вывода графической диаграммы, динамически отображающей изменяющееся значение «приблизительно в реальном времени» [5].

Зачастую программные инструменты и компоненты для отрисовки графиков изначально создавались для ОС Unix (в силу популярности в научной и инженерной среде) и ориентировались либо на простое и нетребовательное к ресурсам представление динамических графиков (рис. 1 а), либо на их качественную статическую отрисовку в стиле, востребованном печатными изданиями (рис. 1 б).



а) б) в)

Рисунок 1 – Компонент StripChart из состава Athena Widgets (а), отображение графика в Grace plotting tool (б), и Google Charts (в)

Изначально сформировавшийся набор характерных особенностей подобных компонент (кроссплатформенность, нетребовательность к ресурсам, аскетизм и индустриальный стиль визуального оформления) сохранялся в течение длительного времени. При этом использование более выразительных визуальных средств (например, для диаграмм презентационного характера) оставалось характерным для программных продуктов соответствующего профиля и вовсе не распространялось на виджеты, выполняющие динамическую отрисовку графиков в реальном времени, за исключением отдельных сторонних компонентов [6].

Толчком к переосмыслению этой концепции послужили средства отрисовки графиков для веб. Изначально ориентированные на дизайнерское оформление и вёрстку веб-сайта, последние получили дополнительный импульс в своем развитии с ростом популярности

Латий Олег Олегович, *ассистент кафедры ЭВМиС Брестского государственного технического университета. Беларусь, БрГТУ, 224017, г. Брест, ул. Московская, 267.*

асинхронного JavaScript и HTML5, позволяющих выполнять красочную динамическую отрисовку (рис. 1-в). В результате более производительные настольные системы оказались в арьергарде медленных и хуже приспособленных к динамике онлайн-систем, и лишь сравнительно недавно в виджет-тулкитах настольных ОС стали появляться средства динамической отрисовки графиков, реализующие сравнимый набор выразительных средств.

2. Архитектура компонентов отрисовки графиков для Qt-приложений. При необходимости реализовать функционал построения графиков C++/Qt-разработчик сталкивается с выбором из нескольких вариантов, причем это одна из немногочисленных задач, где на сегодняшний день еще сохраняется конкурентоспособность проприетарных Qt-компонентов. Так, к актуальным построителям графиков для Qt можно отнести следующие [1]: QCustomPlot, QChart, Qwt, ChartDirector. К этой же категории относится компонент Plotter – построитель диаграмм, разработанный и опубликованный под свободной лицензией автором настоящей статьи (изначальной причиной разработки собственного инструмента послужила недостаточная производительность одних бесплатных фреймворков и сильно ограниченные визуальные возможности других; код проекта доступен по адресу https://github.com/lattoo/plotter под лицензией GPL v.2.).

При разработке кода на основе библиотеки Qt, разработчик сталкивается с выбором между двумя наиболее популярными вариантами: написанием проекта на языке C++, либо на декларативном языке разметки QML, в свою очередь основанном на JavaScript. При разработке как компонента Plotter, так и остальных перечисленных компонентов, использован язык C++, что сохраняет разработку кросс-платформенной, но при этом отвечает также требованиям максимальной производительности кода.

Программная реализация рассматриваемых компонентов во многом сходна и представляет собой многоуровневую архитектуру, в которой в отдельные классы вынесен функционал, связанный с самостоятельными элементами графика (кривые, оси, надписи и т. д.). Также к особенностям реализации стоит отнести следование паттернам проектирования, разделяющим модель и представление, что отделяет код, выполняющий вычисления, необходимые при построении графиков, от непосредственно визуализации [7].

Структура основных классов Plotter является наглядным примером (фрагмент иерархии классов проекта изображен на рис. 2). Ключевыми классами являются Plotter, QImage, QTextEngine, QPainterPath, Marker. Класс Plotter унаследован от QWidget и содержит объекты подклассов, унаследованных от интерфейсов IGraphicItem, IFilledItem, ITextItem, ITitleItem. Подклассы, унаследованные от интерфейса IGraphicItem, соответствуют различным видам линий: выполняют отрисовку главной и второстепенной размерных сеток и их штрихов, а также осей (нулевых линий). Подклассы интерфейса IFilledItem отвечают за отрисовку графических элементов с заливкой: внешней области виджета, области построения, области под графиком. Наконец, подклассы интерфейса ITextItem отвечают за отрисовку текстовых полей: подписей и названий осей, легенды, заголовка.

Класс QPainterPath предоставляет контейнер для операций рисования, позволяющий создавать и повторно использовать графические фигуры, класс Marker используется для отрисовки маркеров и точек на графиках. Также предусмотрена возможность экспорта графиков в формат масштабируемой векторной графики SVG, и опциональное хранение пакета настроек графика в конфигурационном файле [8].

QCustomPlot унаследован от QWidget и использует для отрисовки стандартный класс QImage – аппаратно-независимое представление растровых изображений с прямым доступом к пикселям; за отдельные элементы графика отвечают экземпляры классов QCPAxis (отображение координатной оси), QTextEngine (отрисовка текстовых данных), и QCPGraph (отрисовка графика в области построения) [9].

В отличие от двух предыдущих проектов, класс QChart унаследован не от класса QWidget, а от QGraphicsWidget, который официальная документация Qt рекомендует использовать для создания нестандартных графических элементов интерфейса. С точки зрения вопросов производительности важно, что QGraphicsWidget отличается использованием координат на основе арифметики с плавающей точкой, в то время как QWidget использует целочисленные координаты. Класс QChart связан с унаследованными от QObject классами ChartDataSet (набор значений для визуализации), ChartPresenter (представление и геометрические особенности графика/диаграммы), ChartThemeManager (применение «тем визуализации», т. е. предварительно сохраненных настроек оформления) [10].

Заметим, что многоуровневая архитектура (присутствующая во всех рассмотренных проектах и имеющая наибольшую глубину в Plotter) существенно упрощает адаптацию построителя под требования конкретного проекта, а также позволяет более эффективно отделять вычисления, необходимые для отрисовки графических элементов, от собственно отрисовки. Особенностью класса Plotter является также то, что он реализован не в виде динамически подключаемого библиотечного модуля, а в качестве кода, встраиваемого непосредственно в пользовательский проект. Такой подход продиктован практическими соображениями, связанными с бóльшим удобством распространения прикладного проекта, не имеющего дополнительных нестандартных зависимостей, и техническими сложностями, сопровождающими статическую линковку приложений, которая на текущий момент не поддерживается официально библиотекой Qt.

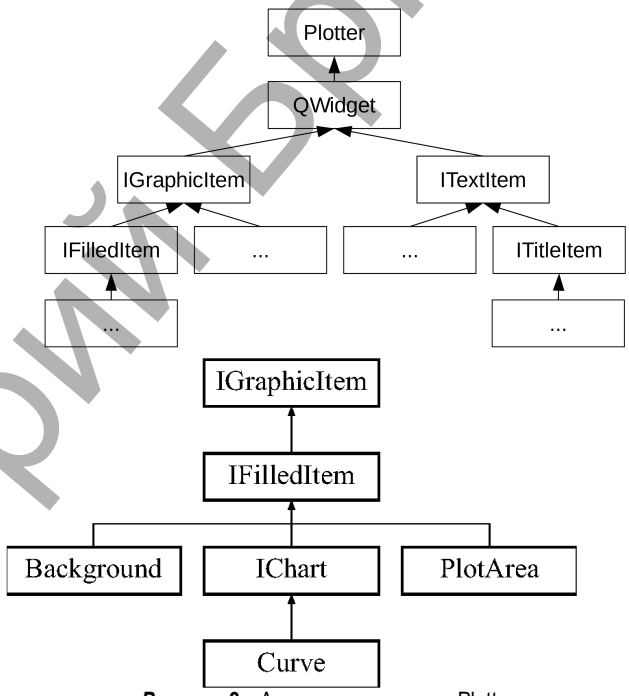


Рисунок 2 – Архитектура проекта Plotter

На рисунке 3 приведен минимальный код для подключения, инициализации, добавления данных и отрисовки графика с помощью Plotter. Строка 3 отвечает за добавление графика на область построения, строка 5 – за инициализацию пера отрисовки. В строке 6 приведен пример добавления данных. Строка 7 выполняет прокрутку области построения графика, а строка 8 инициирует обновление области построения. Примеры работы Plotter приведены на рис. 4 а и 4 б.

```
1 Plotter* plotter = new Plotter;
2 const QString chart_name = "Chart 1";
3 plotter->add_chart(chart);
4 const QPen pen = QPen(QColor(220,170,170),
  1, Qt::SolidLine);
5 plotter->chart(chart)->set_pen(pen);
6 plotter->chart(chart)->add_data(x, y);
7 plotter->scroll_graph();
8 plotter->replot();
```

Рисунок 3 – Пример инициализации и использования Plotter

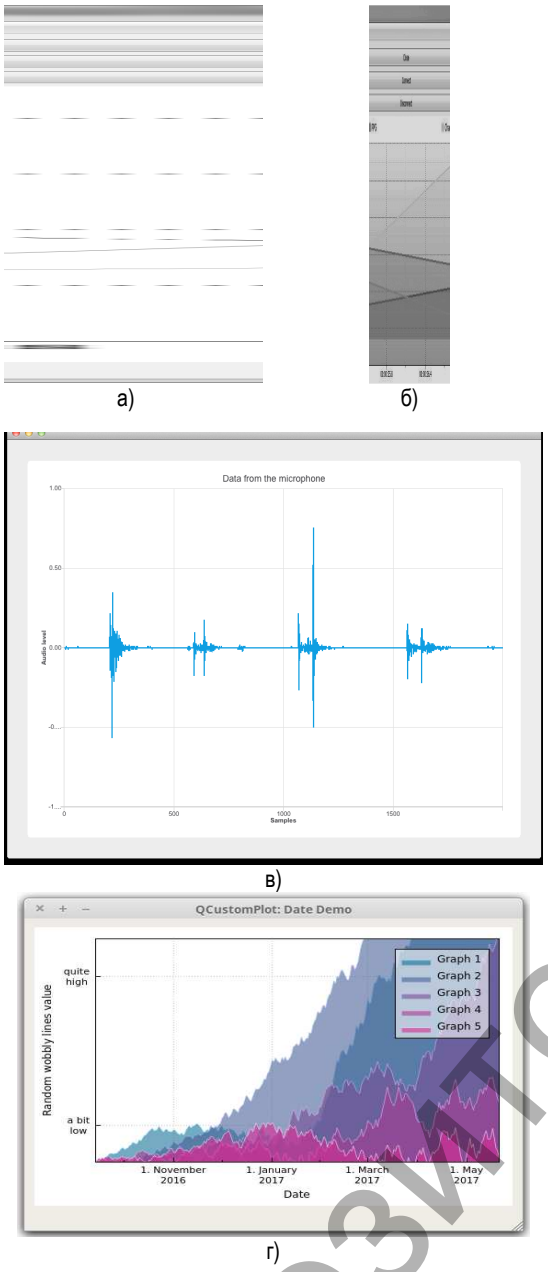


Рисунок 4 – Сравниваемые средства построения графиков: Plotter в базовый режиме (а) и в режиме расширенного визуального оформления (б), QChart (в), QCustomPlot в режиме расширенного визуального оформления (г)

3. Оценка производительности отрисовки. Конкурентоспособность компонентов динамического построения графиков и диаграмм в режиме реального времени характеризуется как минимум тремя параметрами: потреблением ресурсов (в первую очередь – загрузкой процессора), качеством визуализации графиков (включая дополнительные визуальные эффекты) и удобством использования в программном коде. Заметим, что по последнему критерию рассматриваемые в настоящей работе средства построения графиков предоставляют сравнимый уровень удобства программирования, во многом благодаря используемому объектно-ориентированному подходу и рассмотренным паттернам проектирования. В свою очередь, оценка производительности средств отрисовки важна в плане энергоэффективности, актуальной для мобильных компьютеров, для обеспечения динамики визуализации при большом потоке входящих данных, а также для сохранения свободных ресурсов на целевую обработку поступающих данных и другие задачи, выполняемые на той же системе.

Для анализа производительности Plotter, QcustomPlot и QChart были написаны одинаковые с точки зрения верхнего уровня программирования тестовые программные проекты, однотипно задействующие функционал каждой библиотеки.

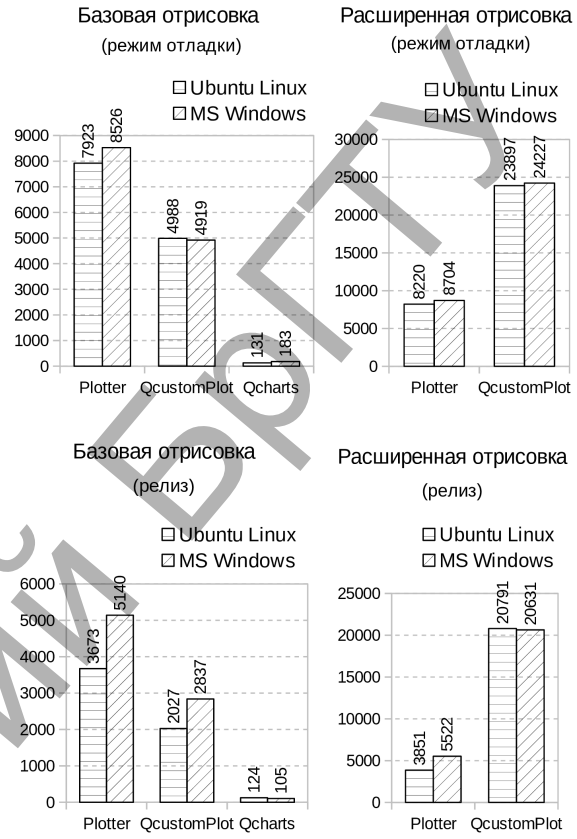


Рисунок 5 – Время выполнения теста в 2000 итераций отрисовки графика

Тестирование осуществлялось в двух режимах: базовом (basic) и расширенного визуального оформления (pro). В базовом режиме выполнялась одновременная отрисовка трех кривых, отображаемых сплошной линией толщиной в 1 пиксель, с «легендой», содержащей их подписи, заголовком графика, сеткой, двухуровневой градуировкой осей. В режиме расширенного визуального оформления были дополнительно включены градиентная заливка области построения, а также отображение зонных графиков с эффектами градиента и прозрачности вместо базовых кривых (рис. 4-б). При тестировании отдельно исследовалось функционирование отладочной и релизной сборок проекта. Интервал обновления данных при отрисовке графиков для обоих режимов был задан равным 50 мс. Тестирование выполнялось в течение циклов 1000, 2000 и 5000 повторений построения графика, в сборках для ОС Windows и Ubuntu Linux.

В случае отрисовки графиков в базовом режиме заметно меньшее время для обработки требуется встроенному построителю QChart (который, однако, не имеет визуальных возможностей расширенного режима вообще). Следующим по скорости работы в базовом режиме выступает построитель QCustomPlot: при 2000 циклов отрисовки графика в режиме отладки он на 59% быстрее чем Plotter, а в режиме релиза – на 81%. В случае режима дополнительного визуального оформления на всех этапах тестирования по скорости работы наилучшие показатели дает Plotter – разница составляет 190% в режиме отладки и 440% в режиме релиза (рис. 5).

В целом время работы каждого из построителей в ОС Windows и Linux является сравнимым, но более быстрой является работа в Linux: в режиме отладки разница обычно незначительна и становится заметной только в случае базовой отрисовки в Plotter. В режиме

релиза версия Plotter под Linux оказывается заметно быстрее, а в случае QCustomPlot аналогичный эффект наблюдается при базовой отрисовке и практически отсутствует при расширенной. В случае QChart эффект незначителен в обоих вариантах.

Заметим, что на разницу в производительности сборок для разных ОС могут влиять такие факторы, как эффективность двумерной графики в каждой из ОС, оптимизация средств отрисовки библиотеки Qt (QWidget в случае Plotter и QCustomPlot, QGraphicsWidget в случае QChart) и оптимизация программного кода самого построителя.

Для прояснения того, какая доля в различиях производительности приходится на каждый из этих факторов, был выполнен динамический анализ программного кода с помощью средств профилирования Valgrind [11, 12] для оценки времени выполнения отдельных функций и методов классов в каждом из проектов.

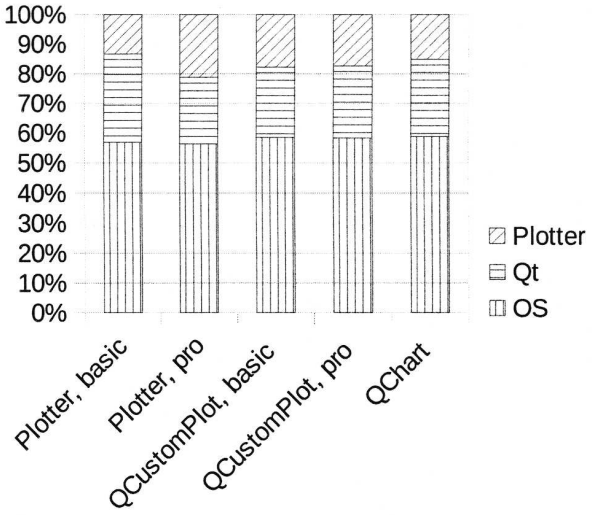


Рисунок 6 – Вычислительная нагрузка, приходящаяся на код построителя, библиотеку Qt и ОС

Результаты профилирования представлены на рис. 6 и в процентном соотношении одинаковы для ОС Windows и Linux. Как можно заметить, в Plotter больше нагрузка на отрисовку расширенных средств визуального оформления собственно построителем, в то время как одинаковая нагрузка на код построителя в случае QCustomPlot в базовом режиме и режиме расширенной отрисовки свидетельствует о преимущественном задействовании в визуальном оформлении кода Qt. Кроме того, немаловажным фактором с точки зрения производительности должно являться целевое использование QCustomPlot аппаратно-ускоренной двумерной графики для отрисовки [7], происходящее за счет средств Qt и ОС. Однако результаты, полученные для расширенной отрисовки на рис. 5, показывают, что в задаче динамического отображения графиков данный подход не показывает себя оптимальным, и код отрисовки, реализованный на стороне построителя, демонстрирует бóльшую производительность (очевидно, аппаратно-ускоренная графика оправдывает себя лишь при значительно более интенсивных нагрузках на обновление сцены).

По полученным результатам можно сделать следующие выводы. Несмотря на независимое развитие, исследованные средства отрисовки графиков имеют общие архитектурные особенности и сходный интерфейс для программирования. Также все три компонента не показали существенных расхождений в производительности своих сборок под Windows и Linux, что говорит о хорошей переносимости их кода. При отрисовке с использованием базовых средств графики оптимальную производительность показывает компонент QChart, отличающийся использованием встроенных в Qt средств поддержки графики с плавающей точкой. В режиме улучшенного визуального оформления, наилучшую производительность показывает Plotter, более активно использующий собственный код в ходе отрисовки, в то время как QCustomPlot, в большей степени полагающийся на средства библиотеки Qt, демонстрирует меньшую производительность. Также следует заметить, что ощутимое увеличение разрыва в производительности между компонентами, построенными на базе целочисленной пиксельной графики, наблюдается при переходе от режима отладки кода к режиму релиза, в дополнение к традиционно более медленному функционированию кода, скомпилированного в режиме отладки.

СПИСОК ЦИТИРОВАННЫХ ИСТОЧНИКОВ

1. Латий, О. О. Программное обеспечение для визуализации потоки данных, поступающих в ПЭВМ от микроконтроллерной системы // Информационные технологии и системы 2018 (ИТС 2018): материалы Международной научной конференции, Минск, БГУиР, 25 октября 2018 г. – Минск, 2018. – С. 168–169.
2. Fakory, R. Real-time distributed expert system for automated monitoring of key monitors in Hubble space telescope / R. Fakory, M. Jahangiri // Measuring the Performance and Intelligence of Systems: Proceedings of the 2000 PerMIS Workshop, August 14-16, 2000. – P. 218–224.
3. Маркина, А. А. Система параллельного тестирования эффективности человеко-машинного взаимодействия // Тринадцатая конференция разработчиков свободных программ: Тезисы докладов / Калуга, 01-02 октября 2016 г. – М.: Базальт СПО, 2016. – С. 32–37.
4. Костюк, Д. А. Подход к биометрической оценке эргономики графического интерфейса пользователя / Д. А. Костюк, О. О. Латий, А. А. Маркина // Вестник БрГТУ. – № 5: Физика, математика, информатика. – 2016. – С. 46–49.
5. Quercia, V. The Definitive Guides to the X Window System / V. Quercia, T. O'Reilly. – Vol 3: X Window System User's Guide for X11, Release 4. – O'Reilly Media Inc., 1993. – 835 p.
6. Gladkov, L. The development of hybrid algorithms and program solutions of placement and routing problems / L. Gladkov, S. Leyba, N. Gladkova // Software engineering trends and techniques in intelligent systems: Proc. of the 6th Computer Science On-line Conference 2017 (CSOC2017). – Vol. 3. – P. 406–415.
7. Krajewski, M. Hands-On High Performance Programming with Qt 5: Build cross-platform applications using concurrency, parallel programming, and memory management. – Birmingham: Pakt Publ., 2019. – 384 p.
8. Латий, О. О. Высокопроизводительный модуль отрисовки графиков на базе использования библиотеки Qt // XV конференция разработчиков свободных программ: тезисы докладов. – Калуга, 28-30 сентября 2018 г. – М.: Альт Линукс, 2018. – С. 27–32.
9. QcustomPlot – Introduction. [Электронный ресурс] – Режим доступа: https://www.qcustomplot.com. – Дата доступа: 22.01.2018.
10. Qt Charts Overview [Электронный ресурс] – Режим доступа: https://doc.qt.io/qt-5.10/qchart.htm. – Дата доступа: 14.09.2018.
11. Nethercote, N. A Framework for Heavyweight Dynamic Binary Instrumentation / N. Nethercote, J. Seward Valgrind // Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation. – San Diego, USA, June 10–13. – 2007. – P. 89–100.
12. Peña, A. J. A Framework for Tracking Memory Accesses in Scientific Applications / A. J. Peña, P. Balaji // Proceedings of the 2014 43rd International Conference on Parallel Processing Workshops, September 09-12, Minneapolis, USA. – 2014. – P. 235–244.

Материал поступил в редакцию 26.03.2019

LATIY O. O. Implementing improved real-time graph plotting for Qt applications

The article presents an analysis of architectural features and implementation of the cross-platform widgets for dynamic plotting of graphs in real time by applications with a Qt library based graphical user interface. The architecture and features of existing components are considered, and the author's implementation is presented for the widget carrying out the dynamic plotting of graphs being embedded in the application code. A performance comparison of the widgets is presented as far as the results of the dynamic analysis of their code.