УДК 004.412
DOI 10.17513/snt.40363

УНИВЕРСАЛЬНЫЙ ПОДХОД К МОНИТОРИНГУ ПРОИЗВОДИТЕЛЬНОСТИ ВЕБ-ГРАФИКИ В ИНФОРМАЦИОННО-ТЕХНОЛОГИЧЕСКОЙ ИНФРАСТРУКТУРЕ

[1]Готская И.Б., [2]Абрамов В.С.

[1]Российский государственный педагогический университет им. А. Герцена, Санкт-Петербург, e-mail: iringot@mail.ru;
[2]Университет ИТМО, Санкт-Петербург, e-mail: vadim-abramov-00@mail.ru

Статья посвящена разработке инструмента для наблюдения за производительностью веб-графики, направленного на рациональное использование ресурсов информационно-технологической инфраструктуры организаций. Актуальность исследования обусловлена стремительным ростом использования графически насыщенных веб-приложений в корпоративной среде, что приводит к увеличению нагрузки на вычислительные ресурсы и требует более точного контроля за их использованием. Отсутствие универсальных и доступных средств мониторинга затрудняет своевременное выявление узких мест сценариев использования графических компонентов, что влечет за собой рост эксплуатационных затрат и снижение устойчивости систем. Цель работы – создание программного обеспечения и метода, позволяющего анализировать нагрузку на ресурсы, выявлять слабые места в работе приложений. Рассмотрены ключевые аспекты внедрения системы наблюдения в контексте корпоративных информационно-технологических систем. Основное внимание уделено проектированию архитектуры расширения для браузера, способного отслеживать выполнение команд, собирать данные о производительности (время обработки, использование памяти, загрузку устройств). Описан механизм взаимодействия между изолированными частями в браузере, включая настройку конфигурации и организацию обмена данными между процессами. Создана система наблюдения, предоставляющая инструменты для анализа работы приложений, изучения кода и оценки нагрузки на устройства в реальном времени. Разработанное решение позволяет организациям рационально использовать ресурсы за счет детального анализа производительности приложений. Система помогает сократить затраты на инфраструктуру, выявляя избыточное потребление ресурсов, и обеспечивает основу для прогнозной аналитики. Внедрение инструмента способствует устойчивому развитию веб-графических сервисов в корпоративной среде, сочетая глубокую техническую диагностику с практическими задачами оптимизации информационных технологий.

Ключевые слова: инструмент разработчика, веб-графика, браузер, графика, управление ресурсами, мониторинг

A UNIVERSAL APPROACH TO MONITORING THE PERFORMANCE OF WEB GRAPHICS IN THE INFORMATION TECHNOLOGY INFRASTRUCTURE

[1]Gotskaya I.B., [2]Abramov V.S.

[1]Herzen Russian State Pedagogical University, St. Petersburg, e-mail: iringot@mail.ru;
[2]ITMO University, St. Petersburg, e-mail: vadim-abramov-00@mail.ru

The article is devoted to the development of a tool for monitoring the performance of web graphics, aimed at the rational use of resources of information technology infrastructure of organizations. The relevance of the research is due to the rapid growth of the use of graphically rich web applications in the corporate environment, which leads to an increase in the load on computing resources and requires more accurate control over their use. The lack of universal and accessible monitoring tools makes it difficult to timely identify bottlenecks of scenarios of graphical components utilization, which entails an increase in operating costs and a decrease in the stability of systems. The aim of the work is to create software and a method that allows analyzing the load on resources, identifying weak points in the work of applications. Key aspects of surveillance system implementation in the context of corporate information technology systems are considered. The main attention is paid to the design of browser extension architecture capable of tracking command execution, collecting performance data (processing time, memory usage, device load). The mechanism of interaction between isolated parts in the browser is described, including configuration setup and organization of data exchange between processes. A surveillance system is created that provides tools for analyzing application performance, examining code, and estimating device load in real time. The solution enables organizations to use resources efficiently through detailed analysis of application performance. The system helps reduce infrastructure costs by identifying excessive resource consumption and provides a basis for predictive analytics. Implementation of the tool promotes sustainable development of web-graphic services in the corporate environment, combining deep technical diagnostics with practical tasks of information technology optimization.

Keywords: developer's tool, web- graphic, browser, graphics, resource management, monitoring

Введение

Современные ИТ-инфраструктуры организаций сталкиваются с растущей нагрузкой, вызванной миграцией приложений в веб-среду, а также активной разработкой нового программного обеспечения для нее. Веб-графика, включая такие технологии, как WebGPU (Web Graphics Processing Unit – технология для рендеринга графики в браузерах и выполнения вычислений на GPU

(Graphics Processing Unit – графический процессор)), становится ключевым элементом для высокопроизводительных решений – от интерактивных аналитических панелей до сложных приложений и виртуальных рабочих пространств [1]. Однако интенсивное использование графических ресурсов в браузерах создает риски перегрузки GPU, неэффективного распределения вычислительных мощностей и роста эксплуатационных затрат. В таких условиях мониторинг производительности веб-графики превращается в критический компонент управления ИТ-ресурсами, обеспечивающий прозрачность использования оборудования, прогнозирование нагрузок и оптимизацию инфраструктуры.

Несмотря на активное развитие WebGPU как преемника WebGL, инструменты для анализа его работы остаются ограниченными [2]. Существующие решения браузеров фокусируются на отладке кода, но игнорируют задачи мониторинга в масштабах организации: сбор метрик потребления GPU, анализ распределения ресурсов между приложениями, интеграцию с системами управления инфраструктурой. На сегодняшний день наиболее известными open source решениями для отслеживания работы WebGPU являются WebGPU DevTools [3] и WebGPU recorder [4]. Однако новых версий первой утилиты не выходило с 2023 года, что означает отсутствие метрик для новых версий API. Другой же аналог более современный, но не имеет функционала для работы с внешними сервисами и не способен масштабироваться в случае большого количества событий. Упомянутые инструменты предоставляют базовую функциональность для перехвата вызовов интерфейса и последующего анализа узких мест, но не способны работать в контексте корпоративных систем. Это создает пробел в средах, где неконтролируемый анализ и отсутствие возможности прогнозирования нагрузки на графические процессоры могут привести к простоям, увеличению затрат на оборудование и снижению производительности критически важных сервисов.

В данной работе представлен усовершенствованный метод мониторинга веб-графики, который был реализован при разработке приложения для системы анализа и отслеживания видеопамяти. Помимо функционала просмотра кода шейдеров и ошибок, отслеживания процесса рендеринга, утилита предоставляет возможности для учета общей задействованной видеопамяти, отслеживания API на протяжении всего процесса работы, отправки собранной информации во внешние сервисы для последующего анализа; также предложен более универсальный метод отслеживания событий. Разработанное решение направлено на устранение разрыва между техническими возможностями WebGPU и операционными требованиями организаций и обеспечивает баланс между инновациями в веб-графике и устойчивостью ИТ-инфраструктуры.

Современные инструменты отладки браузеров предоставляют широкий спектр возможностей для анализа и оптимизации веб-приложений. Эти инструменты позволяют инспектировать DOM (Document Object Model), отлаживать JavaScript код, анализировать сетевые запросы, профилировать производительность и визуализировать время выполнения кода [5]. Однако они имеют ряд ограничений в контексте мониторинга графики WebGPU, что делает их недостаточными для полноценного анализа производительности и оптимизации ресурсов в корпоративных ИТ-средах. WebGPU – это относительно новая технология, и существующие инструменты отладки не предоставляют встроенной поддержки для анализа ее работы. DevTools Chrome и Firefox Developer Tools ориентированы на WebGL [6]. Существующие open source решения, такие как WebGPU DevTools от Hoodgail Benjamin и Takahiro, способны закрыть ряд требований, однако их функционала не хватает для использования в корпоративных системах, так как они не способны проводить анализ собранных данных и отправлять итоговую статистику во внешние сервисы. Также недостатком разработанного инструмента является отслеживание работы интерфейса лишь первые 50 кадров, что делает невозможным сбор данных на протяжении долгого времени.

Для устранения ограничений существующих инструментов отладки и обеспечения полноценного мониторинга графики требуется разработка специализированного программного обеспечения, которое сможет интегрироваться с браузером на уровне его архитектуры. Одним из таких подходов является использование content и background скриптов, которые позволяют обойти ограничения встроенных утилит и обеспечить сбор данных о производительности WebGPU. Эти скрипты работают в изолированных контекстах браузера, что дает им возможность взаимодействовать с WebGPU API, собирать метрики и передавать их в системы мониторинга, недоступные для стандартных инструментов отладки.

Content-скрипт исполняется в контексте веб-страниц, предоставляет доступ к DOM и контексту веб-приложения [7]. Это позво-

ляет отслеживать параметры рендеринга, потребление видеопамяти и выполнение шейдеров. В рамках мониторинга он выступает как агент сбора данных: фиксирует события GPU, слушает вызовы API и передает сырые метрики через защищенные каналы связи в background-скрипт для последующей агрегации.

Благодаря работе в изолированном окружении [8] background-скрипт обеспечивает безопасность данных и выполняет функцию модуля обработки. Он анализирует полученные метрики, добавляет метаданные (идентификатор устройства, время выполнения) и форматирует статистику для интеграции с внешними системами мониторинга. Также он управляет передачей информации в корпоративные хранилища.

Взаимодействие скриптов через API chrome.runtime обеспечивает безопасный обмен данными, соответствующий требованиям корпоративной безопасности [9]. Например, content-скрипт передает только структурированные метрики, исключая доступ к чувствительным данным пользователя, а background-скрипт шифрует информацию перед отправкой в облачные сервисы. Работа скриптов в отдельном изолированном потоке позволила обеспечить защиту полученных данных от внешнего вмешательства и не нагружать основной поток, что могло быть повлиять на итоговые метрики.

Для отслеживания событий API в упомянутом ранее решении используется метод декорирования, что дает возможность реализовать дополнительную логику поверх исходного кода без его изменения. Однако данный подход имеет ряд существенных минусов:

• создание оберток для каждой функции требует больших усилий, так как API содержит множество методов. Это усложняет разработку и повышает риск ошибок [10];

• изменения в API требуют ручного обновления оберток [11], что усложняет поддержку кода, особенно учитывая его активное развитие;

• обертки добавляют дополнительный слой вызовов, что может снижать производительность, особенно в критичных для рендеринга операциях [12].

Цель исследования – разработать усовершенствованный и универсальный метод мониторинга для WebGPU, позволяющий оценивать использование графических ресурсов в реальном времени.

Материалы и методы исследования

В исследовании применен комбинированный подход, включающий анализ ограничений существующих средств отладки и open source решений, а также разработку специализированного расширения для мониторинга WebGPU. На основе анализа встроенных браузерных инструментов предложен процесс обмена сообщениями, использующий связку content- и background-скриптов для сбора и передачи метрик производительности. В качестве основного метода отслеживания использовано проксирование, позволяющее автоматически прослушивать вызовы API без ручной обработки и с учетом будущих изменений. Для передачи данных между контекстами реализована интервальная группировка событий, снижающая нагрузку на сеть.

Результаты исследования и их обсуждение

С учетом выявленных недостатков был реализован метод проксирования, который отслеживает все свойства переданного объекта, способен динамически мониторировать новые функции и удалять обработчики для уже не актуальных. Это позволило отслеживать взаимодействия с API без ручного создания дополнительных оберток, уменьшило количество кода, а также дало возможность учитывать любые изменения в будущих версиях технологии.

Преимущества:

• один обработчик заменяет множество оберток;

• новый метод API автоматически перехватывается без изменений в коде;

• минимизируется риск пропущенных вызовов и ошибок;

• прокси отслеживают все обращения к объектам и их свойствам.

Детализация метода прослушивания с использованием метода проксирования:

1) осуществляется связка прокси обработчика и отслеживаемого объекта (WebGPU);

2) прокси отслеживает вызовы всех методов, фиксируя начальное время выполнения и переданные аргументы;

3) в процессе работы регистрируются ключевые параметры, такие как использование видеопамяти, время, затраченное на рендер одного кадра, текущая частота кадров и количество вызовов, что помогает выявлять узкие места в рендеринге;

4) после выполнения метода прокси отправляет собранные данные в background-скрипт для агрегирования и интеграции с системами мониторинга. Также на уровне прокси происходит перехват ошибок.

Для обеспечения передачи данных о вызовах API между изолированными средами (веб-приложение и расширение) был разработан подход обмена сообщениями на осно-

ве пользовательских событий, который позволил преодолеть ограничения изоляции контекстов, обеспечивая надежную передачу метрик производительности в систему мониторинга. Этапы обмена следующие.

1. Генерация событий на стороне веб-приложения

При вызове проксированного метода override-скрипт создает пользовательское событие, содержащее всю необходимую информацию: аргументы вызова, временные метки, идентификаторы контекста и другие параметры. Оно регистрируется в DOM веб-страницы, что позволяет content-скрипту получить доступ к данным, не нарушая изоляцию контекстов.

2. Перехват событий в content-скрипте

Content-скрипт, работающий в контексте веб-страницы, прослушивает пользовательские события и извлекает из них данные о вызовах API. Так как интерфейс способен генерировать большое количество событий, то передача информации о каждом может не только существенно понизить работу самой утилиты, но и загрузить сеть устройства. Для решения этой проблемы разработан метод интервальной группировки, при которой события накапливаются 1 раз в 16 миллисекунд (примерно 1 кадр при 60 FPS), и после окончания таймера накопленные данные передаются в background-скрипт.

3. Обработка данных в background-скрипте

Background-скрипт, выполняющийся в изолированной среде расширения, получает сообщения от content-скрипта и агрегирует их. На этом этапе данные обогащаются дополнительной информацией (например, идентификатором сессии или метаданными устройства).

4. Ожидание сообщений в коде расширения

Код расширения ожидает сообщений от background-скрипта, что позволяет синхронизировать данные между средами и обеспечивает непрерывный поток метрик для анализа изменения использования памяти во времени. Схема процесса сбора данных представлена на рисунке.

Для передачи дополнительных данных в объект события используется интерфейс CustomEvent, который позволяет включать пользовательскую информацию через свойство detail [13]. Это дает возможность обеспечить гибкость в передаче метрик производительности WebGPU без необходимости изменения структуры события. Поскольку механизм передачи пользовательских событий реализован во всех современных браузерах, разработанное решение поддерживается на различных операционных системах (Windows, macOS, Linux) и устройствах (десктопах, мобильных устройствах) [14]. Это особенно важно для организаций с разнородной ИТ-инфраструктурой.

Так как требуется непрерывный процесс мониторинга в режиме реального времени, возникла необходимость в реализации механизма динамического управления памятью. Для решения этой задачи в рамках content-скрипта был создан специализированный класс-наблюдатель. Этот класс использует API MutationObserver для отслеживания следующих событий: закрытие страницы, удаление экземпляров WebGPU и появление новых. Класс-наблюдатель использует слабые ссылки для хранения, что позволило избежать утечек памяти, а также указывать сборщику мусора на неактуальные данные.

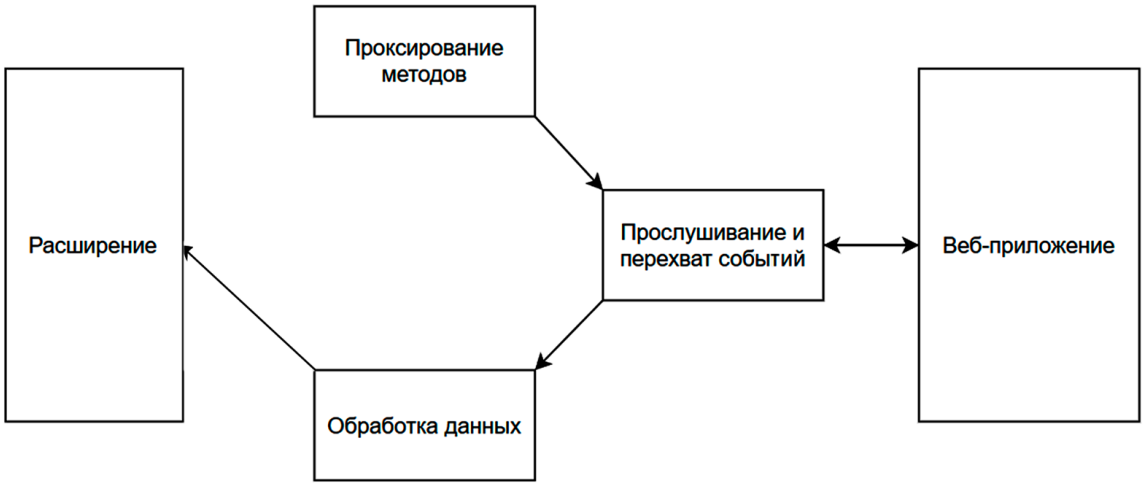

Схема процесса сбора данных
Источник: составлено авторами

Сравнение разработанного приложения с аналогами

Возможности приложения	Приложение WebGPU recorder	Приложение WebGPU devtools	Разработанное приложение
Сбор основных метрик (память, частота кадров, скорость выполнения)	Реализовано	Реализовано	Реализовано
Способ отслеживания событий	Декарирование	Декарирование	Проксирование
Режим мониторинга	Весь процесс работы приложения	Первые 50 секунд работы приложения	Весь процесс работы приложения
Совместимость с версиями WebGPU	Версия выпуска 2024 года	Версия выпуска 2023 года	Любые версии
Выгрузка данных в другие сервисы	Отсутствует	Отсутствует	Реализовано
Обработка большого количества событий	Отсутствует	Отсутствует	Реализовано
Предотвращение утечек памяти	Отсутствует	Отсутствует	Реализовано

Примечание: составлено авторами

Интерфейс расширения создан с помощью библиотеки React. Она представляет собой открытый программный инструмент, разработанный для создания пользовательских интерфейсов веб-приложений. React базируется на принципе компонентного подхода, что позволяет разбивать интерфейс на небольшие независимые компоненты, управляемые данными, что предоставляет эффективный и декларативный способ создания интерфейсов путем использования JavaScript и JSX [15]. Компоненты похожи на функции JavaScript, они выполняют ту же задачу, но в другой среде и с другим подходом. Компоненты принимают входные данные, известные как props, и возвращают их в виде элементов React, которые можно увидеть на экране пользователя [16].

Одним из ключевых преимуществ является виртуальный DOM, который позволил эффективно обновлять только измененные части интерфейса вместо полного перерисовывания всего дерева DOM [17]. Это повышает производительность приложений, улучшает пользовательский опыт и сокращает потребляемые ресурсы при обновлении экрана. Сравнение разработанного приложения с аналогами представлено в таблице.

Выводы

Созданное программное обеспечение имеет следующие возможности:

• динамический метод непрерывного мониторинга событий WebGPU;

• механизм интеграции для отправки данных в системы аналитики;

• интервальная группировка событий;

• отслеживание ошибок и учет общей использованной видеопамяти.

Метод проксирования, используемый для отслеживания вызовов API, упростил процесс мониторинга, позволяя автоматически перехватывать все новые функции API без необходимости в ручной настройке. Это снизило вероятность ошибок и сделало систему более гибкой в условиях изменений в версиях WebGPU. Применение интервальной группировки событий дало возможность оптимизировать использование видеопамяти и сократить эксплуатационные затраты.

Таким образом, предложенное решение является шагом вперед в области мониторинга веб-графики и может быть эффективно использовано в корпоративных ИТ-системах для повышения стабильности и производительности сервисов, а также для оптимизации расходования ресурсов.

Список литературы

1. Usher W., Pascucci V. Interactive visualization of terascale data in the browser: Fact or fiction? // IEEE 10th Symposium on Large Data Analysis and Visualization (LDAV). IEEE, 2020. P. 27-36. DOI: 10.1109/LDAV51489.2020.00010.

2. Scarpino M. The WebGPU Sourcebook: High-Performance Graphics and Machine Learning in the Browser. CRC Press, 2024. DOI: 10.1201/9781003422839.

3. WebGPU devtools // GitHub [Электронный ресурс]. URL: https://github.com/takahirox/webgpu-devtools (дата обращения: 15.02.2025).

4. WebGPU recorder // GitHub [Электронный ресурс]. URL: https://github.com/brendan-duncan/webgpu_recorder (дата обращения: 19.02.2025).

5. García B., Ricca F., del Alamo J.M., Leotta M. Enhancing web applications observability through instrumented automated browsers // Journal of Systems and Software. 2023. T. 203. P. 111723. DOI: 10.1016/j.jss.2023.111723.

6. Верхотурова Г.Н., Киселев А.В. Технологии 3D-графики в web-приложениях // Системная инженерия и информационные технологии. 2021. Т. 3, № 1 (5). С. 96-103. URL: https://elibrary.ru/item.asp?id=45771136 (дата обращения: 01.02.2025).

7. Solomos K., Llia P., Karami S., Nikiforaks N., Polakis J. The dangers of human touch: fingerprinting browser extensions through user actions //31st USENIX Security Symposium (USENIX Security 22). 2022. P. 717-733. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/solomos (дата обращения: 20.02.2025).

8. Yu J., Li S., Zhu J., Cao Y. CoCo: Efficient Browser Extension Vulnerability Detection via Coverage-guided, Concurrent Abstract Interpretation // Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. 2023. P. 2441-2455. DOI: 10.1145/3576915.3616584.

9. Moreno J.M., Vallina-Rodriguez N., Tapiador J. Did I Vet You Before? Assessing the Chrome Web Store Vetting Process through Browser Extension Similarity. 2024. DOI: 10.48550/arXiv.2406.00374.

10. Шмаков С.Э., Щенникова Е.В., Определенцева А.Е. Взаимодействие паттернов проектирования для эффективной web-разработки // Ученые записки УлГУ. Серия: Математика и информационные технологии. 2021. № 2. С. 90-96. URL: https://www.mathnet.ru/php/getFT.phtml?jrnid=ulsu&paperid=56&what=fullt (дата обращения: 01.02.2025).

11. Alkhaeir T., Walter B. The effect of code smells on the relationship between design patterns and defects // IEEE Access. 2020. T. 9. P. 3360-3373. DOI: 10.1109/ACCESS.2020.3047870.

12. Ngaogate W. Memory Usage Comparison in an Android Application: Basic Object-Oriented Programming vs Decorator Design Pattern: Coding styles for keeping low memory usage in a mobile application // Proceedings of the 4th International Conference on Information Technology and Computer Communications. 2022. P. 125-131. DOI: 10.1145/3548636.3548655.

13. Seo D., Yoo B. Interoperable information model for geovisualization and interaction in XR environments //International Journal of Geographical Information Science. 2020. T. 34, №. 7. P. 1323-1352. DOI: 10.1080/13658816.2019.1706739.

14. Lin K., Cao H., Fard A.M. Browser Fingerprint Detection and Anti-Tracking. 2025. DOI: 10.48550/arXiv.2502.14326.

15. Ибрагимов Т.Ф., Ференец А.А. Автоматическое аннотирование html-документов по стандарту Microdata // Электронные библиотеки. 2024. Т. 27, № 5. С. 730-744. DOI: 10.26907/1562-5419-2024-27-5-730-744.

16. Jartarghar H.A., Salanke G., Kumar A. Sharvani G.S, Dalali S. React apps with Server-Side rendering: Next. js // Journal of Telecommunication, Electronic and Computer Engineering (JTEC). 2022. T. 14, №. 4. P. 25-29. DOI: 10.54554/jtec.2022.14.04.005.

17. Шогенов А.М. Исследование архитектуры микрофронтенда: инструменты и рекомендации для использования // Актуальные исследования. 2023. С. 24. DOI: 10.5281/zenodo.8394227.