

## АЛГОРИТМИЧЕСКИЕ ОСНОВЫ ВЕБ-АНИМИРОВАНИЯ НА ЯЗЫКЕ JAVASCRIPT

Жук Н.Е.

Белорусский государственный университет информатики и радиоэлектроники,  
г. Минск, Республика Беларусь

Научный руководитель: Карпович Е.Б. – ст. преподаватель кафедры ИПиЭ

**Аннотация.** Рассмотрены современные алгоритмические подходы к оптимизации веб-анимирования с использованием языка JavaScript. Анализируются ключевые техники, такие как использование метода *requestAnimationFrame*, ленивый (*lazy*) рендеринг, принудительное применение 3D-трансформаций для аппаратного ускорения, оптимизация числовых вычислений, кеширование, объединение и оптимизация изменений в *DOM*. Представленные подходы направлены на снижение нагрузки на центральный процессор, минимизацию количества перерисовок и улучшение общей отзывчивости веб-интерфейсов.

**Ключевые слова:** алгоритмы, веб-анимирование, оптимизация, отзывчивость интерфейса, *JavaScript*, *requestAnimationFrame*.

**Введение.** В современной веб-разработке анимация играет важную роль в улучшении пользовательского опыта, делая интерфейсы более интуитивными и динамичными. Однако при реализации анимаций необходимо учитывать производительность устройства, так как неэффективная работа с *DOM* и избыточные вычисления могут привести к задержкам, заиканиям и даже к блокировке основного потока исполнения.

В данной статье рассматриваются алгоритмические основы оптимизации анимаций на *JavaScript*, а именно такие техники как использование функции *requestAnimationFrame*, использование ленивого рендеринга, принудительное использование 3D для аппаратного ускорения, использование свойства *transform* вместо абсолютного позиционирования элемента, а также кеширование.

**Основная часть.** Функция *requestAnimationFrame* является основным инструментом для создания анимаций в веб-разработке, поскольку она обеспечивает синхронизацию пересчета анимации с частотой перерисовки экрана, обычно около 60 кадров в секунду, что придает визуальным эффектам плавность. Используя этот метод, браузер планирует вызов функции-рендерера непосредственно перед следующим циклом отрисовки, что позволяет максимально точно и экономно использовать ресурсы, поскольку обновление происходит только тогда, когда это действительно необходимо. Это означает, что в отличие от традиционных таймеров, таких как *setTimeout* или *setInterval*, *requestAnimationFrame* позволяет избежать накопления задержек между кадрами и автоматически приостанавливает обновление анимации, когда вкладка не активна, тем самым снижая нагрузку на. Кроме того, функция предоставляет метку времени, которая помогает точно рассчитать интервалы между кадрами и адаптировать анимационные эффекты к возможным задержкам, что способствует более естественному отображению движения. Возможность отмены запланированного вызова через *cancelAnimationFrame* позволяет разработчикам эффективно управлять ресурсами, прекращая анимацию, когда она больше не нужна, что помогает предотвратить утечки памяти.

Ленивый рендеринг представляет собой метод оптимизации веб-анимирования, основанный на отложенной отрисовке элементов, не находящихся в области видимости пользователя, что позволяет существенно снизить вычислительную нагрузку и повысить общую производительность интерфейса. В научном контексте данный подход реализуется посредством динамического контроля времени и условий рендеринга, что достигается, в

частности, через использование *API*, такого как *IntersectionObserver*, позволяющего отслеживать момент появления элементов в видимой части окна браузера и инициировать их отрисовку только в тот момент, когда они становятся актуальными для пользователя. Также важным аспектом является применение условного рендеринга на основе событий прокрутки или других пользовательских взаимодействий, что позволяет динамически подгружать компоненты и анимационные эффекты, минимизируя тем самым затраты на перерасчёт стилей и перерисовку *DOM*. Практическая польза ленивого рендеринга заключается в повышении отзывчивости веб-приложений, уменьшении времени загрузки и снижении энергопотребления, поскольку ресурсы распределяются и задействуются только в момент необходимости.

Принудительное использование *3D* в веб-анимировании представляет собой метод оптимизации, при котором элементы страницы переводятся в контекст *3D* с целью задействования графического процессора для рендеринга. Данный подход осуществляется посредством применения *CSS*-свойств, таких как *translateZ(0)* или *translate3d(0, 0, 0)*, что приводит к созданию нового композиционного слоя, обрабатываемого независимо от основного потока выполнения, в результате чего браузер выделяет эти элементы для аппаратного ускорения. Механизмы перехода от *2D* к *3D* включают преобразование плоского слоя в трехмерное пространство, что заставляет браузер использовать дополнительные оптимизации, связанные с композитингом, а именно распределение задач между графическим и центральным процессорами. Помимо этого, применение *3D*-трансформаций позволяет эффективнее управлять визуальными эффектами, например, тенями и перспективой, поскольку графический процессор более оптимизирован для выполнения параллельных графических операций.

Использование свойства *transform* для перемещения элемента вместо изменения его абсолютного позиционирования оказывает положительное влияние на производительность, поскольку операции, основанные на свойстве *transform*, осуществляются на композитном слое и, как правило, задействуют аппаратное ускорение, что позволяет избежать повторных перерасчетов компоновки и изменения потока документа. При изменении свойств *top* или *left* браузеру необходимо пересчитывать расположение элемента в структуре страницы, что влечёт за собой перерасчёт стилей и возможные перерисовки, существенно увеличивая нагрузку на центральный процессор, особенно при динамических изменениях, характерных для анимаций. В отличие от этого, применение *transform* изменяет лишь матрицу преобразования элемента, не затрагивая поток документа, что позволяет браузеру эффективно обновлять только визуальное представление без необходимости полного пересчёта верстки.

Кеширование в *JavaScript* представляет собой метод оптимизации, заключающийся в сохранении результатов вычислений или доступа к данным для последующего их быстрого извлечения, что позволяет существенно снизить вычислительные затраты при повторном выполнении одних и тех же операций. Этот подход может быть реализован различными способами, включая использование ин-мемори кеша, где результаты дорогостоящих вычислений сохраняются в объектах или замыканиях, а также посредством мемоизации, позволяющей сохранять результаты функций для избежания повторных вычислений в циклах анимаций. Кроме того, данные могут кешироваться во встроенных веб-хранилищах, таких как *localStorage* или *sessionStorage*, что полезно для сохранения состояния между сессиями пользователя или для предварительной загрузки данных, необходимых для анимационных эффектов. В контексте веб-анимирования кеширование играет важную роль в оптимизации производительности, так как позволяет минимизировать количество обращений к *DOM*, снизить число перерасчетов стилей и перерисовок, а также уменьшить нагрузку на центральный процессор за счет быстрого доступа к уже вычисленным значениям.

Помимо всех вышеперечисленных техник, существуют хорошие решения по оптимизации кода анимационных библиотек или же модулей, такие как объединение и

применение всех изменений единой с целью минимизации количества обращений к *DOM*. Чаще всего функция для применения всех изменений вызывается вместе с *callback*-функцией, переданной в *requestAnimationFrame*. Таким образом можно избежать так называемого *layout thrashing*, когда постоянное переключение между чтением и записью в *DOM* приводит к значительным задержкам. Помимо этого, для оптимизации интенсивных математических операций можно использовать специализированные структуры данных, такие как *TypedArray*. В совокупности данные методики способствуют повышению производительности веб-приложений за счет более эффективного распределения вычислительных ресурсов

**Заключение.** Оптимизация анимаций в веб-приложениях является важным аспектом повышения производительности и улучшения пользовательского опыта. Применение таких техник, как использование функции *requestAnimationFrame*, имплементация ленивого рендеринга, принудительное использование *3D* для аппаратного ускорения, использование свойства *transform* и кеширование позволяет существенно снизить нагрузку на систему оставляя достаточно ресурсов на плавную отрисовку анимаций. Внедрение данных подходов в практику разработки требует глубокого понимания механизмов работы браузера и постоянного анализа производительности, однако реализация данных техник существенно улучшает производительность и, как следствие, пользовательский опыт.

### Список литературы

1. Web Animation Techniques: User Engagement with Optimized Animations/ Chakradhar Avinash Devarapalli; // Journal of Scientific and Engineering Research, 2021, 8(4):228-234.
2. JSAV: The JavaScript Algorithm Visualization Library/ Ville Karavirta Dept. of Computer Science and Engineering, Clifford A. Shaffer Dept. of Computer Science Virginia Tech. – 2022. – Vol. 1, N 14. – Pp. 1–16.
3. Brekalo S., Pap K., Trstenjak B. Enhancing Rendering Performance in Complex Visualizations by using Optimization Techniques and Algorithms in Browser Environments // Engineering, Technology & Applied Science Research. – 2024. – Vol. 14. – №. 3. – P. 14049-14055..

UDC [004.65:004.928]:331.101.1

## ALGORITHMIC FOUNDATIONS OF WEB ANIMATION IN JAVASCRIPT

Zhuk M.E.

Belarusian State University of Informatics and Radioelectronics, Minsk, Republic of Belarus

Kharpovich E.B. – senior lecturer of the department of ICSD

**Annotation:** The paper examines modern algorithmic approaches to optimizing web animation using JavaScript. Key techniques are analyzed, including the use of the *requestAnimationFrame* method, lazy rendering, forced application of *3D* transformations for hardware acceleration, numerical computation optimization, caching, and the batching and optimization of *DOM* updates. The presented approaches aim to reduce CPU load, minimize the number of re-renders, and improve the overall responsiveness of web interfaces..

**Keywords:** algorithms, web animation, optimization, interface responsiveness, JavaScript, *requestAnimationFrame*.