МЕТОДЫ ОПТИМИЗАЦИИ ПРОИЗВОДИТЕЛЬНОСТИ ДВУХМЕРНОЙ ГРАФИКИ В WEB-ПРИЛОЖЕНИЯХ ПРИ РАБОТЕ С 2D-КОНТЕКСТОМ ЭЛЕМЕНТА HTML5 <CANVAS>

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

Ясюкевич П. П., Лычковский Е.В.

Скудняков Ю.А. – канд. техн. наук, доцент

HTML5 предоставил достаточно большое количество возможностей web-разработчикам, что позволило расширить функциональные возможности web-приложений в области компьютерной графики. Данный элемент можно использовать для создания статической и интерактивной графики, включая диаграммы, анимированные рисунки, игры, презентации. Поэтому важным остаётся вопрос производительности решений, использующих для своей работы элемент <canvas>.

Самым простым способом улучшить производительность web-приложения, использующего элемент <canvas>, является включение аппаратного ускорения в браузере (если поддержка такового имеется), в результате чего браузер переключается на использование низкоуровневых операций OpenGL, что положительно сказывается на производительности приложений. Однако следует не забывать, что, во-первых, пользователь должен сделать самостоятельно (т.е. это невозможно сделать программно в рамках web-приложения), а во-вторых, не все браузеры имеют встроенную поддержку данной технологии. Поэтому рассмотрим способы улучшения производительности, которые можно использовать непосредственно в приложении.

Первым методом оптимизации производительности графики в web-приложении является использование техники пререндеринга с помощью списков отображения (display list) [1]. Данный метод предполагает наличие, помимо основного, дополнительного элемента <canvas>, который находится вне области видимости страницы, т.е. вне DOM-модели документа, который используется для отрисовки сложных элементов. Как только элементы прорисованы, они передаются на основной (видимый) <canvas>. Такой подход напоминает технологию двойной буферизации в графических приложениях OpenGL и DirectX. Выигрыш в производительности достигается за счёт отсутствия вывода сцены на экран до момента завершения её прорисовки. Однако, для данной техники существуют некоторые ограничения: размер дополнительного элемента <canvas> должен быть относительно небольшим, так как в противном случае выигрыш в производительности может быть нивелирован долгим переносом данных на основной холст.

Вторым методом оптимизации является повторное использование данных ранее отрисованной сцены. Суть в кэшировании на постоянной основе тех отрисованных частей сцены, которые являются весьма трудоёмкими или часто повторяются при анимации сцены. Т.е. кадр или часть кадра, которая уже была отрисована, не рисуется повторно, а выводится из кэша. Для данных целей хорошо подходит интерфейс ImageData, который позволяет хранить в себе массив пикселей изображения в формате RGBA, что, во-первых, позволяет не создавать новый элемент <canvas> для кэширования очередного объекта, а во-вторых, позволяет выполнять операции над пикселями уже закэшированных фрагментов.

Простейшим примером, к которому можно применить обе вышеперечисленные оптимизации, является отрисовка текста, которая, к сожалению, является весьма трудоёмкой операцией: можно производить кэшировние растра однажды нарисованных отдельных символов, или, если это позволяет семантика текста, целых слов или выражений. В это же время к таким элементам можно применить эффекты, например, субпиксельного рендеринга. Затем вместо повторной прорисовки выполняется извлечение растра символа или слова из кэша и прорисовка этих данных внутри сцены. Стоит не забывать, что данный подход плохо себя проявит в случае, если визульный стиль текста неоднороден, т.е. размер, наклон, толщина или размер шрифта отличаются, так как понадобится кэширование одних и тех же символов с различными стилями, что влечёт за собой расход памяти.

Третьим методом оптимизации является декомпозиция сцены на отдельные подсцены по признаку их динамичности и прорисовка каждой их подсцен на отдельном элемента <canvas>. Выигрыш от данного подхода тем более существенен, чем разительнее уровень динамичности подсцен. Например, двухмерные игры в наиболее частых случаях можно разбить на 3 слоя: задний план, основной план (где прорисовываются главные динамичные элементы сцены) и план графического интерфейса игры (меню, панели показателей и т.п.). Меньше всего из них в большинстве случаев меняется слой графического интерфейса, поэтому продолжительное время он не требует перерисовок, и, соответственно, вычислительных расходов при формировании всей сцены. Задний план перетекает медленнее, чем основной, и часто представлен монолитным изображением, что заставляет его перерисовывать только при движении игрока. Таким образом, наименее динамичные и не требующие постоянной перерисовки элементы отделяются от основной сцены и обновляются только по мере необходимости, что также увеличивает общую производительность.

Четвёртым методом оптимизации является отказ от затратных в плане производительности операций, в т.ч. от использования координат элементов с плавающей точкой (использование целочисленной арифметики), от использования дорогостоящей операции встроенной размытия тени объекта (shadowBlur), от частого изменения состояния элемента <canvas>, от частой прорисовки сцены (FPS), а также от прорисовки объектов, не попадающих в область видимости сцены, на уровне приложения [2]. Тут по первому и третьему пунктам необходимо сделать пояснение. Прорисовка элемента с нецелочисленными координатами сказывается на визуальном качестве объекта (объект становится размытым), а во-вторых, на данное размытие неэффективно тратится процессорное время. Частое же изменение состояния контекста элемента <canvas>, например,

изменение цвета заливки, приводит к потере возможных оптимизаций внутри самого графического движка. Чтобы не потерять внутренние оптимизации, рекомендуется по мере возможности формировать последовательность прорисовки таким образом, чтобы операции, не требующие изменения состояния контекста, располагались последовательно. Основные подсцены внутри общей сцены на примере игры-платформера представлены на рисунке 1.

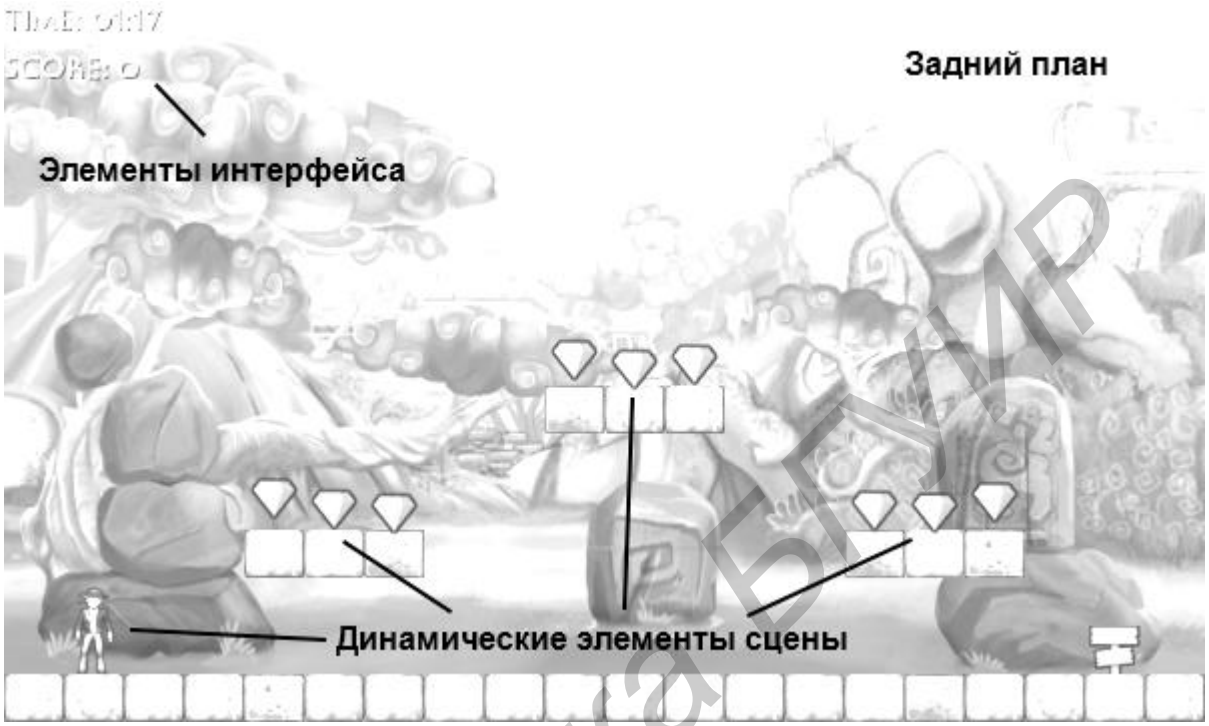

Рисунок 1 – Основные подсцены внутри общей сцены на примере игры-платформера

Таким образом, выработанные методы оптимизации позволяют улучшить производительность двухмерной графики в web-приложениях, использующих элемент <canvas>, однако стоит помнить, что некоторые из перечисленных методов могут быть использованы только в определённых ситуациях.

Список использованных источников:

1. Оптимизация элемента canvas / Mozilla Developer Network – Режим доступа: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Optimizing_canvas . – Дата доступа: 22.03.2015.

2. Смас, Б. Как улучшить производительность элемента canvas [Электронный ресурс] // Б. Смас / HTML5 Rocks – Режим доступа: http://www.html5rocks.com/ru/tutorials/canvas/performance/. – Дата доступа: 23.03.2015.