Eficiencia comparativa en animaciones en javascript (nota técnica)

Efficiency comparison between javascript animations (technical note)

Franklin Hernández-Castro[1], Jorge Monge-Fallas[2]

Fecha de recepción: 12 de mayo de 2017
Fecha de aprobación: 6 de agosto de 2017

Hernández-Castro, F; Monge-Fallas, J. Eficiencia comparativa en animaciones en javascript (nota técnica). *Tecnología en Marcha*. Vol. 31-3. Julio-Setiembre 2018. Pág 143-150.

DOI: 10.18845/tm.v31i3.3909

1 Escuela de Diseño Industrial. Instituto Tecnológico de Costa Rica. Costa Rica. Correo electrónico: franhernandez@itcr.ac.cr
2 Escuela de Matemática. Instituto Tecnológico de Costa Rica. Costa Rica. Correo electrónico: jomonge@itcr.ac.cr

Palabras clave

Simulación in javascript; browsers comparison; Safari; Firefox; Chrome; requestAnimationFrame (), window.setTimeout( ) y window.setInterval ().

Resumen

Este artículo expone los resultados de pruebas de eficiencia en el modo de realizar animaciones/simulaciones en javascript por el grupo de investigación iReal. iReal es un grupo de investigación del Instituto Tecnológico de Costa Rica(TEC), pertenece al programa de eScience del TEC y su área de investigación es la visualización científica. Desde el 2005 han venido desarrollando proyectos de forma ininterrumpida que involucran la visualización científica en distintas áreas de aplicación.

En el grupo de investigación iReal se desarrollan muchas animaciones y simulaciones que llevan al límite las capacidades de las computadoras personales, por esta razón se hace necesario que se evalúe en cuál plataforma y con qué herramientas se producen los resultados más fluidos.

En este primer artículo se trata de exponer los diferentes tipos de animaciones en javascript y cómo se comportan en cada browser y sistema operativo. Más adelante se tiene planeado realizar un estudio similar sobre algunos lenguajes de programación más usados en el proyecto.

Keywords

Javascript simulation; comparación de browsers; Safari; Firefox; Chrome; requestAnimationFrame (), window.setTimeout( ) y window.setInterval ().

Abstract

In this work we show results of an performance analysis for animation/simulation in javascript in three different browsers. In iReal research group, we work a lot with simulations that go beyond the common limits for personal computers, because of this we had to evaluate this, cases in order to take the right choice in our future work.

In this case, we analyze several ways to make simulations with javascript and them performances in different browsers and operative systems.

Introducción

En el grupo de investigación iReal (parte del programa eScience del Tecnológico de Costa Rica), se trabaja constantemente en visualización de datos científicos. iReal es un grupo de investigación que ha desarrollado proyectos desde el 2005.

La visualización científica es una temática ampliamente trabajada por los investigadores Franklin Hernández-Castro y Jorge Monge-Fallas desde el año 2005. A partir de ahí en forma ininterrumpida se han realizado proyectos de investigación en esta línea con proyectos tales como: Biovisualizador (software ya patentado), Refinamiento de los algoritmos de dimensionado y posicionamiento en Árboles de Conos, Sistema de Interfaces Intangibles (proyecto declarado confidencial), iReal: interfaces en ambientes de realidad virtual, Visualización de placas tectónicas, (en colaboración con Ovsicori), Simulación del terremoto del 5 de septiembre del 2012. (en colaboración con Ovsicori).

Actualmente se trabaja en dos proyectos en esta misma línea de investigación: Geotráfico: visualización de las cizallas en Costa Rica (iReal 4.0), Skygraph: visualización de vientos en Costa Rica (iReal 4.0)

A este grupo de investigación se le ha unido el ingeniero David Segura Solís y desde hace algunos años se cuenta con un laboratorio para la visualización de datos científicos con una tecnología 3D sin lentes (tecnología Alioscopy) y con un sistema de visualización tipo TDW (tile display wall). Todo esto para la visualización y análisis de información a través de ambientes tridimensionales inmersivos y no inmersivos.

Algunos de estos datos son de interés del público en general y por eso se pretende poner a disposición a través de la internet.

De esta inquietud se desprende la idea de usar el ambiente HTML-javascript-CSS para programar las animaciones o simulaciones que se puedan con esas capacidades, de modo que estén preparadas para una divulgación expedita.

Ya algunos como [2], habían realizado trabajos para medir la velocidad y rendimiento de algunos navegadores. Las pruebas fueron realizadas con Windows 7 de 32 bits recién instalado en una máquina virtual y con las últimas versiones de los navegadores: Internet Explorer 9.0.8, Firefox 4, Safari 5.0.4, Opera 11.10 Beta y Google Chrome 11.0.696.

Su trabajo se enfatizó en varios aspectos tales como: inicia más rápido, qué navegadores consumen más memoria, cuáles son los mejores interpretando JavaScript y cuáles soportan mejor los estándares. El tercer aspecto es de interés, por cuanto está relacionado al trabajo presentado en esta investigación.

Tanto el arranque en frío como en caliente el navegador más rápido de todos fue Opera, seguido de Safari, Chrome, Firefox 4 y de último Internet Explorer.

En cuanto al consumo de recursos, para la evaluación se abrieron cinco pestañas: un vídeo de YouTube en HD, Twitter, Genbeta, El País y Google; y midieron el consumo de RAM y de CPU medio.

En cuanto CPU, los que menos consumieron fueron Internet Explorer y Chrome, luego les sigue Firefox, Opera y por último Safari. En RAM, Firefox fue el que menos consume y Chrome el peor.

Por último, aplicaron el test de JavaScript, utilizaron un test (relativamente) independiente, el de SunSpider. Los mejores fueron: Chrome, Internet Explorer y Opera y en este caso Safari y Firefox no salieron muy bien.

Como conclusiones finales no se declaró ningún ganador global, dado que todos tienen sus puntos fuertes y sus puntos débiles. Uno de los peores fue Safari, sin embargo, como dice el mismo autor, esto debido posiblemente a que estaba haciendo las pruebas en Windows y no en un Mac.

Otros trabajos como en [4], se enfocan en la validación de directrices de JavaScript en distintos navegadores web, estas directrices permitían escribir código JavaScript en forma eficiente. La investigación trataba principalmente si los programadores deberían aún seguir utilizando estas directrices a pesar de la incorporación de JIT technology por algunos navegadores. Dentro de sus conclusiones resalta el hecho de que el efecto de las directrices es dependiente del modelo de ejecución del motor de JavasScript del navegador. Por ejemplo en el caso de Safari, Chrome y Firefox que utilizan JIT technology el efecto de las directrices fue inmenso, sin embargo en el caso de intérpretes como Opera y Internet Explorer el efecto no fue significativo.

Hay muchas maneras de programar una aplicación en este ambiente, en nuestro caso se tomaron algunas consideraciones básicas:

- Minimizar el uso de *css* y *html* para la animación.
- Usar solo *javascript* para controlar tanto el proceso de simulación como la visualización misma.
- Mantenerse en ambientes multi-plataforma.
- Minimizar el uso de bibliotecas.

Conservando estas condiciones se procedió a diseñar una comparación práctica de las diferentes posibilidades que se tenían a disposición.

Desarrollo

Se diseñaron y programaron dos simulaciones físicas en las que se interactúa con partículas. Se escogió interacción de partículas pues es una simulación muy recurrente en nuestro trabajo y nos permite aumentar el número de las mismas hasta llegar a la capacidad máxima del ambiente.

Una de ellas se programó desde cero y otra se usó la biblioteca Box2D para javascript, específicamente boxDjs[2].

Usando Box2D

En el caso de la biblioteca Box2D se diseñó una animación (figura 1) en forma de embudo en el que caen en forma aleatoria 500 objetos, la mitad de ellos son cuadrados y la mitad círculos y en cuanto uno sale de la pantalla uno nuevo nace de modo de mantener la cantidad constante.

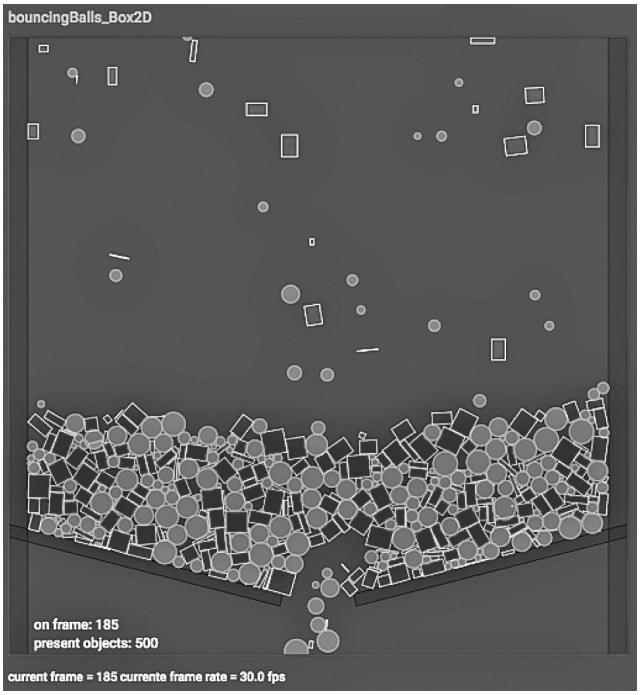

Figura 1. Simulación de 500 objetos cayendo usando la biblioteca Box2D, en el margen inferior se puede ver el número de “frames per seconds” (ftp) de 30.0 .

2 http://box2d-js.sourceforge.net

Animación usando solo javascript

En este caso se programó una animación que mantuviera una cantidad definida de partículas interactuando unas con otras en un ambiente de gravedad cero. Una vez más la idea era llevar el sistema al límite para poder comparar diferentes hipótesis de análisis (figura 2).

La velocidad máxima en cuadros por segundo (fps por sus siglas en inglés) sería de 60 fps, debido a la tecnología de 60 Herz que se usa en el hardware. Así que se afinaron las cantidad de objetos en ambas animaciones para requerir unos 30 fps, dejando espacio para el análisis hacia arriba o hacia abajo en cada caso.

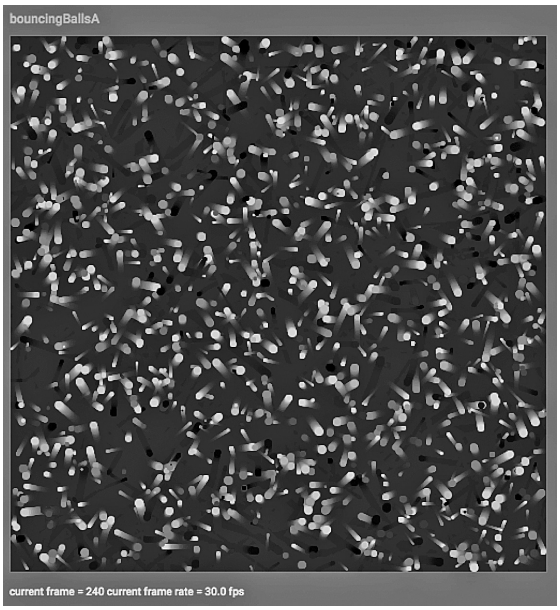

Figura 2. Simulación de 2500 objetos interactuando entre ellos, en el margen inferior se puede ver el número de "frames per seconds" (ftp) de 30.0 .

Tipos de simulación

Los navegadores no actualizan las ventanas a menos que una interacción lo haga, como tocar un botón o algo similar [3]. Por eso se hace necesario definir un método para lograr que una simulación se actualice lo mejor posible.

En javascript existen varios modos de ordenar la frecuencia del ciclo de actualización de la simulación. Los dos más usados según la literatura [5] son window.setTimeout() y window.setInterval().

Window.setTimeout() trata de repetir la función designada en intervalos regulares, mientras que window.setInterval() trata de hacerlo en un lapso definido después de que la última termina, es decir en este caso el tiempo que dura la función en realizarse se suma al lapso definido, mientras que en el caso anterior no sea así.

Sin embargo, el método requestAnimationFrame () deja al navegador decidir en qué momento se puede llevar a cabo la actualización. Según Dirksen [1] así se obtiene una simulación más fluida y la mejor eficiencia, además de este modo el navegador no hará ningún cálculo mientras la ventana (o la pestaña) no esté activa o visible, lo que no carga el procesador sin necesidad, de forma oculta al usuario.

Así que en nuestro análisis se tomaron las mismas simulaciones y se probaron con todas las posibilidades anotando los “cuadros por segundo” (fps por sus siglas en inglés) en cada caso.

Resultados

Los resultados se analizaron desde varios puntos de vista, con respecto a los sistemas operativos quedó claro (figura 3) que OS X, es notablemente más eficiente en esta prueba que windows 8 (cerca del 30%).

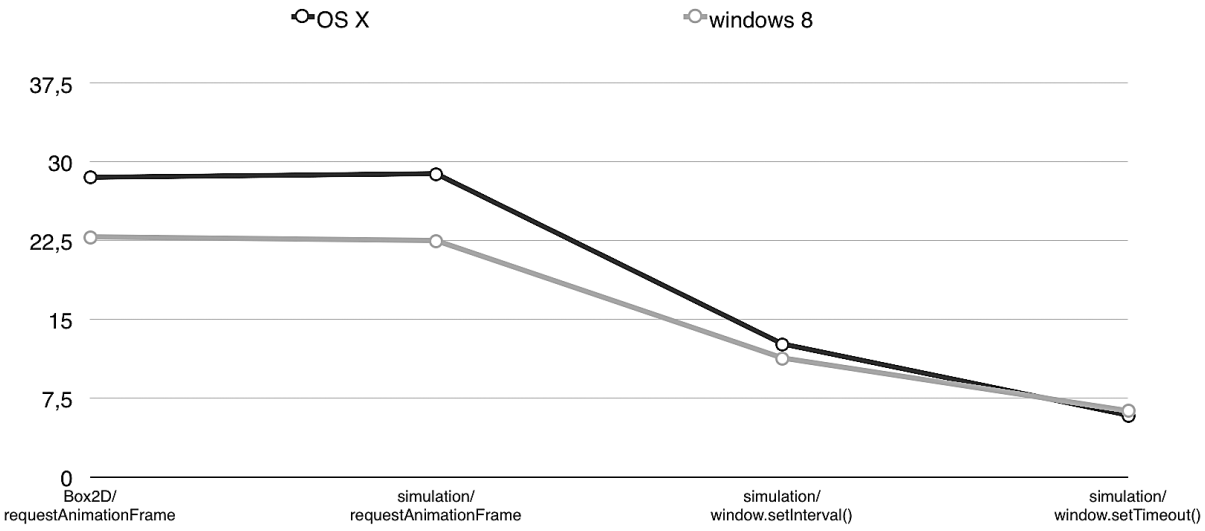

Figura 3. Gráfico mostrando los fps en cada sistema operativo, se tomo el promedio de todos los navegadors analizados para este gráfico.

Además se notó inmediatamente que las funciones para definir lapsos de tiempo en Firefox y Chrome se bloquean a menudo y definitivamente no dejan correr este tipo de simulaciones. El cuadro 1, muestra que en el caso de Firefox, el 100% de las veces las funciones mencionadas no funcionaron. Mientras que en el caso de Chrome el 80% de las veces tampoco funcionaron. En safari funcionaron siempre pero con muy poca cantidad de fps.

Cuadro 1. Resumen de las veces que los navegadoress fueron incapaces de hacer funcionar las animaciones según su función de origen, porcentaje conjunto en ambos sistemas operativos.

sin funcionamiento	safari	firefox	chrome
Box2D/ requestAnimationFrame	0%	0%	0%
simulation/ requestAnimationFrame	0%	0%	0%
simulation/ window.setInterval()	0%	100%	80%
simulation/ window.setTimeout()	0%	100%	80%

Finalmente se analizó el caso de las funciones según los navegadores (figura 4) y se notó diferencias importantes en su eficiencia.

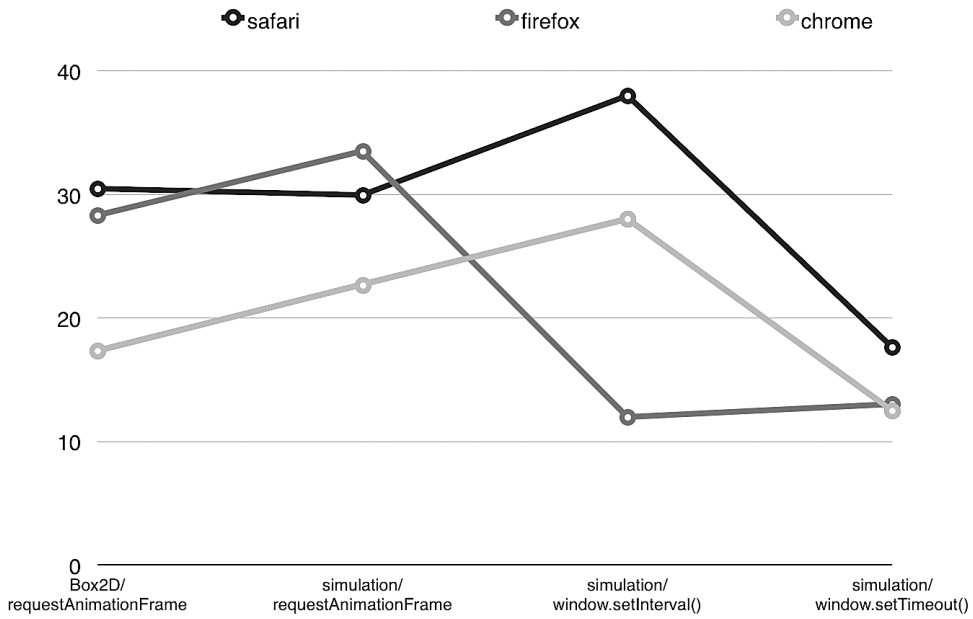

Figura 4. Gráfico mostrando los fps en cada navegador, se tomo el promedio de todos los sistemas operativos.

Como se ve en la mayoría de los casos, con una sola excepción Safari supera por mucho a los otros dos navegadores.

Conclusiones

La primera y más importante conclusión de este análisis es que no se debe usar las funciones de lapso de tiempo, window.setTimeout( ) y window.setInterval (), para este tipo de animaciones en javascript.

A pesar de que la literatura así lo recomienda (3), los resultados fueron muy por debajo de lo esperado. Es decir, el único método recomendado a partir de los resultados es requestAnimationFrame (), que dio resultados fluido en todos los casos, aunque con diferencias de eficiencia según navegadores y sistemas operativos.

En cuanto a navegadores, en este análisis Safari supera por mucho a Firefox y Chrome a veces hasta en el doble de la velocidad y en sistemas operativos OS X El Capitán, supera al menos en 30% en eficiencia a Windows 8.

Trabajo futuro

En cuanto al proyecto iReal se refiere, esto nos da alguna idea de donde seguir trabajando, sin embargo, queda por analizar al menos dos cosas más, si el lenguaje javascript es más eficiente en el uso de recursos que otros similares en sus capacidades multi-plataforma y si el uso de ambientes en 3 dimensiones cambia en algo los resultados de este primer análisis, ambas condiciones muy importantes para nuestro trabajo.

Referencias

[1] Dirksen. J. Three.js Essentials. Packt Publishing Birmingham, UK.

[2] Julían, G. Comparamos velocidad, rendimiento y más de Internet Explorer, Safari, Chrome, Firefox y Opera. Marzo del 2011. [Online]. Disponible en: https://www.genbeta.com/comparativa/comparamos-velocidad-rendimiento-y-mas-de-internet-explorer-safari-chrome-firefox-y-opera

[3] Haverbeke, M. 2015. Eloquent Javascript. No Starch Press. San Francisco. CA. USA

[4] Herczeg, Zoltán, et al. "Validating JavaScript guidelines across multiple web browsers." *Nordic Journal of Computing* 15.12 (2012): 18-31.

[5] Stefanov, S. Chetan-Sharma, K. 2014. Object-Oriented JavaScript. Packt Publishing Birmingham, UK.