Web-based Graphics Library

Carlos Samuel Cantero González.
samucg89@gmail.com

Universidad Católica Nuestra Señora de la Asunción
Facultad de Ciencias y Tecnología
Ingeniería informática.

Resumen Éste artículo trata sobre uno de los nuevos estándares de la Web3D: WebGL. Aquí se dará a conocer qué es WebGL, cómo trabajar con ella, algunas aplicaciones desarrolladas y todo lo relacionado con su crecimiento en la industria.

1. Introducción

Antes de empezar a dar una definición formal de WebGL, se dará a conocer otros términos que se utilizarán a lo largo del texto. Es imprescindible entrar en estos detalles para una mejor comprensión del tema.
El primer término importante a mencionar es HTML5. **HTML5** (HyperText Markup Language, versión 5), es la quinta versión del lenguaje para estructurar y presentar contenido en la web. Actualmente se encuentra en modo experimental aunque ya es utilizado por muchos desarrolladores. Su principal objetivo es manejar la mayoría de los contenidos multimedias actuales sin la necesidad de plugins.[11] Ofrece muchas características significativas, pero la que nos interesa ahora es el elemento canvas.
Canvas es un elemento de HTML5 que permite la generación de gráficos dinámicamente por medio del scripting. Permite generar gráficos estáticos y animaciones. Este objeto puede ser accedido a través de javascript, permitiendo generar gráficos 2D, animaciones, juegos y composición de imágenes.[10] Actualmente está soportado por la mayoría de los navegadores, incluyendo Internet explorer 9. Con canvas podemos crear rectángulos, líneas, arcos, curvas, dibujar imágenes, añadir colores y estilos, además de transformaciones y composiciones, y lo más importante, animaciones. Nos permite hacer imágenes dinámicas pero sin plugins externos.
OpenGL(Open Graphics Library) es una especificación estándar que define una API (application programming interface) multiplataforma para escribir aplicaciones que contengan gráficos 2D y 3D. [12]. OpenGL está manejado por el grupo tecnológico **Khronos Group**, el cual es un consorcio industrial sin fines de lucro encargado de crear estándares abiertos para permitir la creación y aceleración de la computación paralela, gráficos y medios dinámicos en una variedad de plataformas y dispositivos.[23]
OpenGL ES 2.0 (OpenGL for Embedded Systems) es una variante simplificada de OpenGL para dispositivos integrados, tales como smartphones, PDAs,

consolas, entre otros. [13] Consiste de un subconjunto bien definido de OpenGL. Permite la programación completa de gráficos 3D. Al igual que OpenGL, está manejado por **Khronos Group**.
Toda la familia de OpenGL tiene una característica muy importante: la aceleración por hardware.
Hardware-Acceleration (Aceleración por hardware) es el uso del hardware para desempeñar algunas funciones mucho más rápido de los que es posible en software corriendo en la CPU de propósito general. De esta manera se utiliza la GPU de la tarjeta gráfica para procesar grandes cargas de gráficos. [6]
La **GPU** (Graphics Processing unit) o unidad de procesamiento de gráficos es un procesador dedicado al procesamiento de gráficos u operaciones de coma flotante.[15]
JavaScript es un lenguaje de scripting multiparadigma que soporta los estilos de programación orientada a objetos, imperativa y funcional. Es un lenguaje del lado del cliente (client-side), implementado como parte del navegador web, permitiendo mejoras en la interfaz de usuario y lo más importante, páginas web dinámicas. [7] Todos los navegadores web modernos interpretan el código javascript integrado en las páginas web. Javascript permite a los navegadores ser capaces de reconocer objetos en una página HTML a través del DOM.
Por último, **DOM** (Document Object Model) o modelado de objetos del documento es una API multiplataforma para representar e interactuar con objetos en documentos HTML, XHTML y XML. [5] Esto permitirá a los programas y scripts acceder dinámicamente y modificar el contenido, estructura y estilo de los documentos HTML y XHTML. DOM es requerido por JavaScript para inspeccionar y modificar páginas web dinámicamente.

2. Comenzando con WebGL

WebGL fue creado inicialmente por Mozilla, y más tarde estandarizado por el grupo tecnológico *Khronos Group*, el mismo grupo responsable de OpenGL y OpenGL ES. El primer prototipo fue diseñado por Mozilla en el año 2006 y a principios del 2009, Mozilla y Khronos Group comenzaron el WebGL Working Group. Además de los ya mencionados, actualmente los principales fabricantes de navegadores, Apple (Safari), Google (Chrome) y Opera (Opera), así como algunos proveedores de hardware son miembros del grupo de trabajo WebGL o WebGL Working Group.
Todos ellos están interesados en verificar que el contenido WebGL pueda correr tanto en desktop y hardware de dispositivos móviles.
Pero, ¿Qué es WebGL?. **WebGL** (Web-based Graphics Library) es un estándar web multiplataforma para una API de gráficos 3D de bajo nivel basado en OpenGL ES 2.0 y expuesto a través del elemento canvas de HTML5 como interfaces DOM (Document Object Model).[24] Esta API provee enlaces de JavaScript a funciones OpenGL haciendo posible proveer contenido 3D acelerado en hardware a las páginas web.[1] Esto hace posible la creación de gráficos 3D que se actualizan en tiempo real, corriendo en el navegador.

A partir de la definición anterior, podemos decir también que WebGL, es una librería de software que extiende al lenguaje de programación javaScript para permitir generar gráficos interactivos 3D dentro de cualquier navegador web compatible. WebGL es un *contexto del elemento canvas* que provee un API de gráficos 3D sin la necesidad de plugins.[8] Decir que WebGL es un contexto del elemento canvas, podría no entenderse hasta que se muestre como trabajan juntos en la siguiente sección.
WebGL trae el API de OpenGL ES 2.0 al elemento canvas. El contenido 3D está limitado a canvas. Se hicieron cambios en la API de WebGL en relación con la API de OpenGL ES 2.0 para mejorar la portabilidad a través de varios sistemas operativos y dispositivos.
Podemos observar un modelo conceptual de la arquitectura de WebGL en la figura 1. En este modelo se utiliza JavaScript para obtener a través del DOM

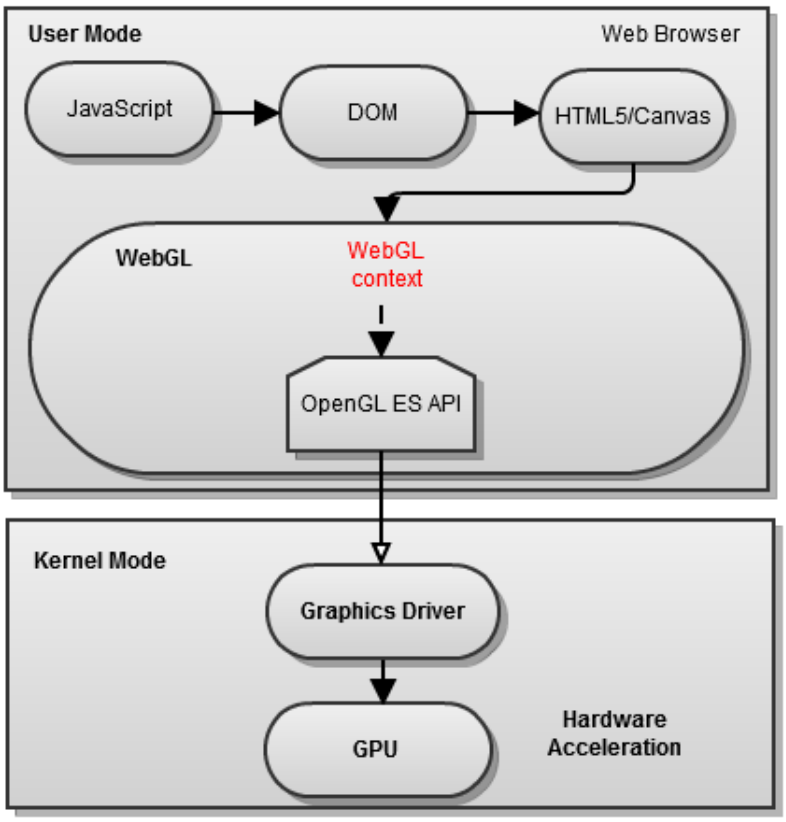

Figura 1. Modelo conceptual de la arquitectura WebGL.

el elemento Canvas de HTML5. Una vez obtenido el elemento Canvas, se define el *contexto WebGL*, por medio del cual accedemos a la API de WebGL, la cual está basada en la API de OpenGL ES. Por eso se dice que, *técnicamente es un enlace (binding) para JavaScript para usar la implementación nativa de OpenGL ES 2.0.*[8] Este último, se encarga de comunicarse con el driver de la tarjeta gráfica y así poder realizar la aceleración por hardware en la GPU. Como vemos, se diferencian los eventos que ocurren en el espacio de usuario y en el kernel.
La API de WebGL interactúa bien con el resto de las plataformas web; específicamente se proporciona apoyo para la carga de texturas 3D a partir de imágenes HTML o video, y la entrada del teclado y mouse se manejan mediante los conocidos eventos DOM.
Este API poderoso es de muy bajo nivel y su uso requiere buenos conocimientos acerca de la programación 3D y matemática 3D. Otra desventaja de WebGL es que consume mucha CPU.
Una característica importante de WebGL es que, brinda la posibilidad de contenidos 3D a la web sin la necesidad de utilizar plugins, ya que se encuentra implementado en el navegador.
Otra característica relevante: WebGL es un estándar web, pero ¿Porqué es importante un estándar?. Un estándar web, es un conjunto de especificaciones técnicas en *constante evolución* y de mejores prácticas para construir sitios web. Con ello se facilita el mantenimiento, la usabilidad, la interoperabilidad y la calidad de los trabajos.[9]
Utilizar un estándar asegura una larga vida a los proyectos, ya que provee por lo menos una pequeña estructura de mantenimiento. También asegura de que la mayoría de las personas puedan visitar el sitio web, sin importar que navegador se esté utilizando. La compatibilidad hacia adelante y hacia atrás (forward and backward compatibility) es posible.[4]
En el caso particular de WebGL, su API está basado en un estándar de gráficos 3D familiar y ampliamente aceptado.
No utilizar un estándar tiene varias desventajas, como poco soporte, mayor exposición a problemas de seguridad y problemas de performance. Además la curva de aprendizaje suele ser muy alta, con un costo-beneficio menor. Otro problema que se puede dar es el del *vendor lock in*, en donde se tiene una dependencia absoluta del proveedor, el cual puede tener un costo elevado.[26]
El lanzamiento (release) de la especificación de la versión final de WebGL 1.0 se dió el 3 de marzo del 2011 en la conferencia de desarrolladores de juego (Game Developers Conference - GDC), en San Francisco, Estados Unidos.

3. Utilizando WebGL

A partir de esta sección se busca entender más a fondo la definición anterior de WebGL, y su funcionamiento.
En general, para empezar a trabajar con WebGL, tenemos que entender cómo funciona canvas. En la introducción se había dado una definición de la misma.

Como sabemos, canvas es un elemento de HTML5 que nos permite la generación de gráficos de forma dinámica, el cual posee dos atributos, width (ancho) y height (alto), y cuyo tamaño por defecto es 150. Este objeto puede ser accedido a través de javaScript. A canvas hay que indicarle en que contexto queremos trabajar, ya sea 2D o 3D (webGL). Podemos decir que el contexto es el acceso a una API particular, para dibujar ya sea en 2D o WebGL.[18]

3.1. Ejemplo en contexto 2D

A partir de ahora se irá creando un ejemplo básico del uso del elementos canvas utilizando un contexto 2D:

```
<canvas id="micanvas" width="200" height="150">
El navegador no soporta el elemento canvas de HTML5.
</canvas>
```

Con esto agregamos al código HTML, la zona correspondiente al canvas, con un tamaño de 200 x 150 píxeles. El contenido insertando entre los tags *canvas* se mostrará en los navegadores que no soporten este elemento.
Para poder hacer referencia al elemento canvas y trabajar sobre él necesitamos javaScript. Lo primero que debemos realizar es obtener el elemento canvas o la referencia hacia el elemento mediante el DOM, con el siguiente código:

```
var canvas = document.getElementById("micanvas");
```

Una vez obtenido el elemento, debemos establecer el contexto en el cual vamos a trabajar, utilizando el método *getContext* y pasándole como argumento el contexto deseado. En este caso en particular vamos a utilizar "2d":

```
var contexto= canvas.getContext("2d");
```

Una vez obtenido el contexto podemos empezar a dibujar en él, con los varios métodos existentes que provee canvas.
Una práctica buena es comprobar el contexto canvas, de manera a declarar código en un navegador que no soporte canvas y mostrar algo aceptable en lugar de un error. Podemos realizar esto, uniendo las lineas anteriores en el siguiente fragmento de código.

```
var canvas = document.getElementById("micanvas");
if (canvas.getContext){
   var context = canvas.getContext("2d");
   //Nuestro codigo para trabajar sobre el contexto canvas.
}else{
   //Nuestro codigo para navegadores que no soportan canvas.
}
```

Ya teniendo todo lo necesario, se presenta un ejemplo completo donde se dibujan dos rectángulos superpuestos, que usted podría pegar en un editor de texto y probarlo.

```
<html>
<head>
<title>Elemento Canvas</title>
<script type="application/javascript">
   window.onload = function(){
      var canvas = document.getElementById("micanvas");
      if (canvas.getContext) {
         var contexto = canvas.getContext("2d");
         contexto.fillStyle = "rgb(200,0,0)";
         contexto.fillRect (0, 0, 100, 100);
         contexto.fillStyle = "rgba(0, 0, 200, 0.5)";
         contexto.fillRect (30, 30, 100, 100);
      }
   }
</script>
</head>
<body>
<canvas id="micanvas" width="200" height="150">
 El navegador no soporta el elemento canvas de HTML5.
</canvas>
</body>
</html>
```

3.2. Canvas y WebGL

Una vez entendido cómo funciona canvas utilizando un contexto 2D, podemos empezar a trabajar con Canvas y WebGL.
Para trabajar en canvas 3D existen dos métodos: uno de ellos implica desarrollar un plano 2D desde un objeto 3D, controlando el plano y la cámara que queremos mostrar. La otra alternativa, y el punto central de este artículo es utilizando **WebGL**.
Como vimos en el punto anterior, para trabajar en 2D, debíamos utilizar el contexto "2d" en el elemento canvas, y para trabajar en WebGL debemos utilizar el contexto "webgl" o en caso de que no funcione, el "experimental-webgl". Las cadenas son case sensitive. Una vez obtenido el contexto para WebGL utilizando el elemento canvas, podemos empezar a trabajar con WebGL.[19]
Se irán presentando una serie de funciones que nos ayudarán a entender mejor cómo funciona WebGL. Estas funciones deben ir dentro de la etiqueta `<script>`,ya que se encuentran escritas en javaScript.
Para empezar, se muestra una función que recibe como parámetro el elemento canvas obtenido a través del DOM, y que se encarga de inicializar el contexto WebGL.

```
function initWebGL(canvas){
    contexto = canvas.getContext("webgl");
    if (!contexto) {
        contexto = canvas.getContext("experimental-webgl");
    }
    if (!contexto) {
        alert("Tu navegador no soporta WebGL");
        return;
    }
}
```

Ya finalizada la función anterior, se procede a crear la función **start()** que será llamada en el *body*, de la siguiente manera:

```
<body onload="start()">
```

Esta función obtendrá el elemento canvas, y llamará a la función **initWebGL()** para obtener el contexto WebGL. Aquí se muestra:

```
function start(){
    var canvas = document.getElementById("micanvas");
    initWebGL(canvas);
    if (contexto){
       //Nuestro codigo WebGL
    }
}
```

Desde ahora podemos empezar a crear nuestros fragmentos de código para probar WebGL. Debemos sustituir cada uno de los siguientes códigos dentro del if anterior: *Nuestro código WebGL*.
En el siguiente código sencillo WebGL se muestra información acerca de la versión de WebGL, el navegador, y el renderizador usado. Como vemos, se utilizan algunos atributos y métodos del objeto contexto.

```
alert(
  "WebGL version=" + contexto.getParameter(contexto.VERSION) +"\n"+
  "WebGL vendor="  + contexto.getParameter(contexto.VENDOR)  +"\n"+
  "WebGL renderer="+ contexto.getParameter(contexto.RENDERER)+"\n"
);
```

Hasta ahora, sólo se realizo una inicialización del contexto WebGL, pero para trabajar con WebGL se requieren de muchas cosas más.
Una vez que se ha obtenido el contexto WebGL, se debe crear el buffer de dibujo (drawing buffer) en el cual son renderizadas todas las llamadas a la API. El tamaño del buffer, está determinado por los atributos *width* y *height* del elemento canvas. Se debe limpiar (clear) el buffer con un color y una profundidad (Depth) específica, cuyos valores por defecto son (0,0,0,0) y 1.0 respectivamente. Luego, se deben crear unos buffers con los cuales trabajar.
Podemos expandir nuestra función **start()**, agregando dos funciones, realizando el clear del buffer de dibujo mencionado, y una llamada a la función drawScene().

```
function start(){
    var canvas = document.getElementById("micanvas");
    initWebGL(canvas);
    initShaders();
    initBuffers();

    contexto.clearColor(0.0,0.0,0.0,1.0);
    contexto.enable(contexto.DEPTH_TEST);

    drawScene();

}
```

En *initBuffers()*, se crean los buffers que almacenaran los vértices de los gráficos. Los vértices son los puntos en el espacio 3D que definen la figura que estamos dibujando. Este buffer se crea con el método del objeto contexto, **contexto.createBuffer()**. Este buffer, es en realidad un poco de memoria en la tarjeta gráfica.
Para manejar los buffers, existen dos métodos, **contexto.bindBuffer()** y **contexto.bufferData()**. El primero especifica el buffer actual, en el cual deben realizarse las operaciones y el segundo permite cargar el buffer actual.
La función *drawScene()*, es el lugar en donde utilizamos los buffers para dibujar la imagen que se ve en pantalla. En ella se utiliza el método **viewport()** el cual define el lugar de los resultados de renderización en el buffer de dibujo. Sin el viewport, no se manejará adecuadamente el caso donde canvas cambie de tamaño. Luego, se define la perspectiva con el cual queremos observar la escena. Por defecto, WebGL dibuja las cosas que se encuentran cerca con el mismo tamaño que el de las cosas que se encuentran alejadas. Para definir que las cosas alejadas se vean más pequeñas, hay que modificar la perspectiva.
Para empezar a dibujar, hay que moverse al centro de la escena 3D. Al dibujar una escena, se especifica la posición actual y la rotación actual, ambos mantenidos en una matriz. La matriz que se usa para representar el estado actual move/rotate es llamado model-view matrix. Esta matriz nos mueve al punto de origen desde el cual podemos empezar a dibujar.[16]
La función *initShaders()*, inicializa los Shaders, evidentemente. Pero, ¿Qué es un Shader?. Un **Shader** es un conjunto de instrucciones de software que es utilizado para realizar transformaciones y crear efectos especiales, como por ejemplo iluminación, fuego o niebla. Proporciona una interacción con la GPU hasta ahora imposible, por lo cual la renderización se realiza en hardware gráfico a gran velocidad.[14]
Se utilizan los shaders de manera a que los segmentos de programa gráficos sean ejecutados en la GPU, y no en javaScript, lo cual sería muy ineficiente y lento. Un lenguaje de sombreado muy utilizado es **GLSL**(OpenGL Shading Language), el cual es un lenguaje de alto nivel basado en C.
Por eso, algunos desarrolladores reconocen a WebGL como una API basado en el uso de shaders GLSL.

Ya no se entran en más detalles, pero con esos pasos mencionados, podemos empezar a escribir nuestro programa utilizando WebGL. Como vimos, utilizar WebGL tiene su complejidad inherente, debido a su API de bajo nivel.
OpenGL maneja muchos tipos de recursos como parte de su estado. WebGL representa estos recursos como objetos DOM. Los recursos que son soportados actualmente son: texturas, buffers, framebuffers, renderbuffers, shaders y programas. La interfaz del contexto tiene un método para crear un objeto para cada tipo.[21]
Es importante mencionar que podemos agregar contenido 2D al contexto WebGL, pero debemos tener en cuenta que se está dibujando sobre un espacio tridimensional.
Podemos encontrar una serie de lecciones y ejemplos muy útiles en los siguientes enlaces:

1. Learning WebGL
2. Developer Mozilla

4. Frameworks

Debido a que WebGL posee un API de muy bajo nivel, podría provocar el rechazo por parte de algunos programadores. Gracias a su creciente uso, muchos desarrolladores han trabajado duro para proveer una API de alto nivel, a través de la creación de varios frameworks. Estos frameworks son sólidos y garantizan que corren en cualquier dispositivo, haciendo la programación más fácil. Esto permite que los desarrolladores web no tengan que lidiar con las complejidades WebGL y la matemática 3D.
Algunos frameworks son: C3DL, CopperLicht, CubicVR, EnergizeGL, GammaJS, GLGE, GTW, Jax, O3D, Oak3D, PhiloGL, SceneJS, SpiderGL, TDL, Three.js y X3DOM. La mayoría son librerías JavaScript.[25]
A parte de los frameworks WebGL, existen otras librerías útiles para el manejo de matrices que también facilitan la programación, tales como: Sylvester, mjs, glMatrix.

5. Aplicaciones

La habilidad para poner contenido 3D acelerado en hardware en el navegador proveerá un medio para la creación de nuevas aplicaciones basadas en web que fueron previamente del dominio exclusivo del entorno de computadores de escritorio o desktop.[1]
Aplicaciones que han sido posible solo en desktop o con plugins se vuelven posible en cualquier navegador moderno que soporte WebGL: juegos 3D, demostraciones interactivas de productos, visualización científica y médica, entornos virtuales compartidos, entre otros.
Para los usuarios, esto significa una web más interactiva e interesante visualmente. Significa tener acceso a una gama más amplia de aplicaciones y experiencias

sobre cualquier dispositivo que soporte totalmente la web, en lugar de limitarse a determinados dispositivos y plataformas.[3]
Para ejecutar estas aplicaciones es necesario que el usuario cuente con una tarjeta gráfica que soporte OpenGL.
Una muestra de lo que podemos realizar con WebGL, se observa en la Figura 2. Podemos verlo directamente clickeando la leyenda de la figura. Otro ejemplo

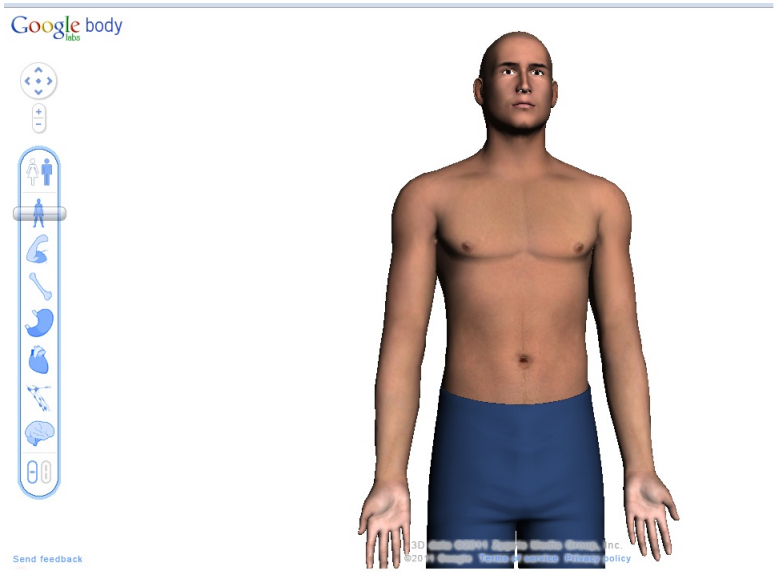

Figura 2. El cuerpo humano de Google.

se puede observar en la figura 3. Se han realizado una serie de experimentos con WebGL, generando un montón de muestras y códigos. Algunos enlaces que contienen ejemplos interesantes son los siguientes:

1. Or so they say.
2. Experimentos de Google.
3. WebGL Samples

Podemos encontrar un montón de ejemplos en la web. El crecimiento es cada vez mayor, y su única limitante es Internet Explorer, quien se opone a utilizar WebGL, del cual hablaremos más adelante.

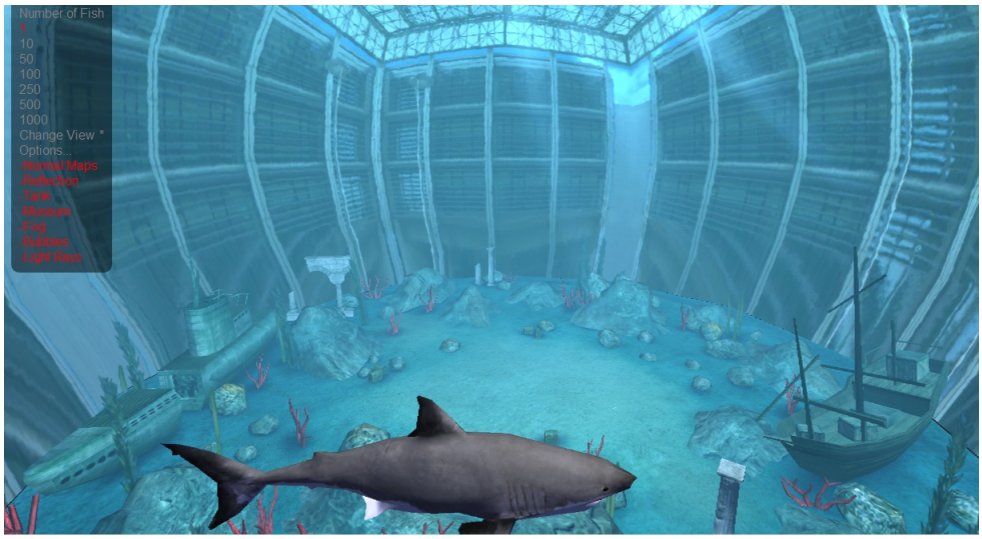

Figura 3. El acuario.

6. Seguridad

Al decir que WebGL da a las páginas Web acceso al hardware gráfico, varias personas se preocupan de que WebGL sea una grave amenaza de seguridad.
En mayo del año 2011 la firma de seguridad *Context Information Security* descubrió una vulnerabilidad en WebGL que podría permitir a los atacantes ejecutar código malicioso de forma remota y tomar el control de los equipos. [27] Se abre una línea directa desde Internet a la GPU del sistema, abriendo un gran hueco de seguridad. Entonces, con las versiones actuales de WebGL, un hacker podría escribir una página WebGL que haga que la tarjeta gráfica deje de responder a otras aplicaciones.
Con respecto al problema, Google dijo que muchos procesos WebGL, incluyendo el de la GPU, se ejecutan por separado y dentro del Sandbox de Chrome (filtro) como medida para prevenir posibles ataques.
El grupo Mozilla afirma que han tomado sus precauciones para evitar al máximo que esto no suceda, por lo que cuentan con una lista de drivers admitidos.
También lo hizo Khronos Group, diciendo que se *están evaluando estas advertencias* y que los fabricantes de la GPU están añadiendo soporte para un mecanismo que ayudaría a resolver el problema.

El mecanismo del ataque [27] se puede observar en la figura 4. Los siguientes pasos del ataque se indican en el gráfico:

1. Un usuario visita un sitio web donde un código WebGL malicioso se encuentra alojado.
2. El componente WebGL carga las geometrías 3D específicas y el código a la tarjeta gráfica.
3. La geometría o el código malicioso explota los errores en los drivers de la tarjeta gráfica.
4. El hardware gráfico puede ser atacado causando que todo el sistema se congele o caiga.

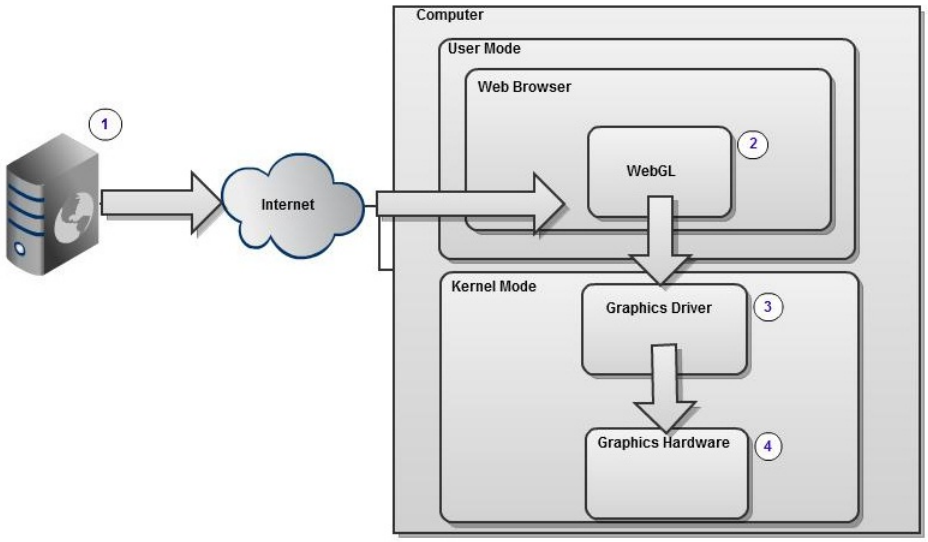

Figura 4. El mecanismo de un ataque WebGL.

7. Microsoft y WebGL

Microsoft ha anunciado que no soportará el estándar WebGL por considerar que es una fuente continua de vulnerabilidades difíciles de corregir.
En un blog reciente, Microsoft dijo: "En su forma actual, WebGL no es una tecnología que Microsoft pueda apoyar desde una perspectiva de seguridad".
Microsoft llega a la conclusión de que WebGL "tendría dificultades pasando los requerimientos del ciclo de vida para el desarrollo de la seguridad de Microsoft".

Según Microsoft, además del problema de seguridad mencionado en la sección anterior, WebGL puede ser vulnerable a ataques DoS (Denial of Service) y señalan que su seguridad dependen de los niveles más bajos del sistema, incluyendo los drivers de OEM.[17]
Esta postura de Microsoft, impide que WebGL se convierta en un estándar soportado por todos los principales navegadores. El soporte universal significa que todos los desarrolladores web podrían contar con WebGL estando disponible y por lo tanto usarlo. Su ausencia significa que algunos sitios y aplicaciones web requerirán chequeos de compatibilidad.
Para algunos, los problemas de seguridad de WebGL, es una excusa para Microsoft, ya que si se expande WebGL, tanto DirectX como Silverlight saldrían perjudicadas.

8. Competencia

En la actualidad se está empezando a crear una guerra entre WebGL, Molehill y Silverlight. Todos ellos forman parte de lo que se denomina Web 3D, el cual es una tecnología que permite que se despleguen gráficos 3D en el navegador.
Silverlight es un framework propiedad de Microsoft y con su última versión lanzada soportaría gráficos 3D y la aceleración de video por GPU. Para utilizar Silverlight es necesario la instalación de un plugin.[22][2]
Molehill, oficialmente conocido como **Stage3D API**, es una API que permite la renderización de gráficos 3D y Shaders utilizando la GPU. Molehill es propiedad de Adobe y será publicado en la nueva versión de flash player. Como el plugin para flash está implementado para casi todos los navegadores, se considera que eso no será un problema. Molehill utiliza OpenGL en OSX y Linux, y DirectX en Windows. El enfoque de trabajo es muy similar a WebGL.[2][20]
Se considera a Molehill como la principal competencia de WebGL. Ambos acceden directamente a las capacidades nativas de las tarjetas gráficas y se espera que no existan diferencias significativas en cuanto al perfomance de los dos.
Es difícil decidir quien ganará la batalla, y aunque algunos piensen que WebGL será el ganador debido a que es un estándar abierto y libre de plugins, Molehill es una propuesta muy interesante y prometedora de un grande como Adobe.
La batalla recién comienza y aún no se puede predecir quién será el líder en la Web 3D.

9. Conclusión

WebGL es un estándar que está creciendo rápidamente y cuyo uso se ha ampliado considerablemente.
Es una tecnología 3D muy prometedora, ya que se encuentra apoyada por los grandes empresas como Google, Apple, Mozilla y otros proveedores de hardware. Y aunque Microsoft no lo apoye por diversas razones, WebGL sigue creciendo e incluso se han desarrollado plugins para que pueda correr en Internet Explorer, lo que indica un gran apoyo hacia la evolución de WebGL por parte de los desarrolladores web.
La posibilidad que brinda WebGL a los usuarios es una revolución total, ya que aplicaciones que antes sólo fueron del dominio de las PCs, ahora se hacen posible en la Web. Y lo más interesante, estas aplicaciones no necesitan de ninguna instalación.
Sin dudas, WebGL es una gran tecnología que puede revolucionar la Web3D.

Referencias

1. Catherine Leung and Andor Salga. Enabling webgl. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 1369–1370, New York, NY, USA, 2010. ACM.
2. http://blog.finalpromise.com/2011/01/why-webgl/. finalpromise blog.
3. http://blog.vlad1.com/. Vladimir vukicevic.
4. http://dustinbrewer.com/why-web-standards-are-important-in-web-design/. Standard.
5. http://en.wikipedia.org/wiki/Document_Object_Model. Dom.
6. http://en.wikipedia.org/wiki/Hardware_acceleration. Hardware acceleration.
7. http://en.wikipedia.org/wiki/JavaScript. Javascript.
8. http://en.wikipedia.org/wiki/WebGL. Webgl.
9. http://en.wikipedia.org/wiki/Web_standards. Web standard.
10. http://es.wikipedia.org/wiki/Canvas. Canvas.
11. http://es.wikipedia.org/wiki/HTML_5. Html5.
12. http://es.wikipedia.org/wiki/OpenGL. Opengl.
13. http://es.wikipedia.org/wiki/OpenGL_ES. Opengl es 2.0.
14. http://es.wikipedia.org/wiki/Shader. Shader.
15. http://es.wikipedia.org/wiki/Unidad_de_procesamiento_grafico. Gpu.
16. http://learningwebgl.com/blog/?p=28. Learning webgl.
17. http://news.cnet.com/8301-30685_3-20071726-264/. Cnet news.
18. http://sabia.tic.udc.es/gc/Contenidos%20adicionales/trabajos/Tutoriales/Canvas/. Canvas.
19. https://developer.mozilla.org/en/WebGL/Getting_started_with_WebGL. Getting started mozilla.
20. http://sebleedelisle.com/2011/06/. Sebleedelisle.
21. https://www.khronos.org/registry/webgl/specs/1.0/. Especificacion webgl.
22. http://www.domusinc.com/blog/2011/04/. Do musinc.
23. http://www.khronos.org/. Khronos group.
24. http://www.khronos.org/webgl/. Webgl.
25. http://www.khronos.org/webgl/wiki/User_Contributions. Frameworks.
26. http://www.masio.com.mx/la-importancia-de-los-estandares. Standard.
27. http://www.theinquirer.es/2011/05/11/. The inquirer.