COMPUTACION GRAFICA Y VISUAL

ALUMNO: EDWAR MEJIA VASQUEZ

TEORIA

1. Describir WebGL y sus principales características.

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). Esta API provee enlaces de JavaScript a funciones OpenGL haciendo posible proveer contenido 3D acelerado en hardware a las páginas web. Esto hace posible la creación de gráficos 3D que se actualizan en tiempo real, corriendo en el navegador. (Cantero Gonzales, 2012)

WebGL creció desde los experimentos del canvas 3D comenzados por Mozilla. Primero demostró un prototipo de Canvas 3D en 2006. A finales de 2007, tanto Mozilla como Opera habían hecho sus propias implementaciones separadas. A principio de 2009 Mozilla y Khronos comenzaron el WebGL Working Group (Grupo de Trabajo del WebGL).
Algunas bibliotecas en desarrollo que se están incorporando WebGL incluyen el C3DL y el WebGLU. Utiliza el elemento canvas del HTML 5. (Antuña Díez, Díaz Solares, & González Losada, 2012)

WebGL proporciona la capacidad de usar características del **hardware de la tarjeta gráfica**, permitiendo **mostrar gráficos 3D de alta calidad y hacerlos interactivos**, dentro del elemento Canvas de HTML5. WebGL es manejado por el consorcio de tecnología Khronos Group sin fines de lucro.

En este modelo se utiliza JavaScript para obtener a través del DOM 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 WebG.

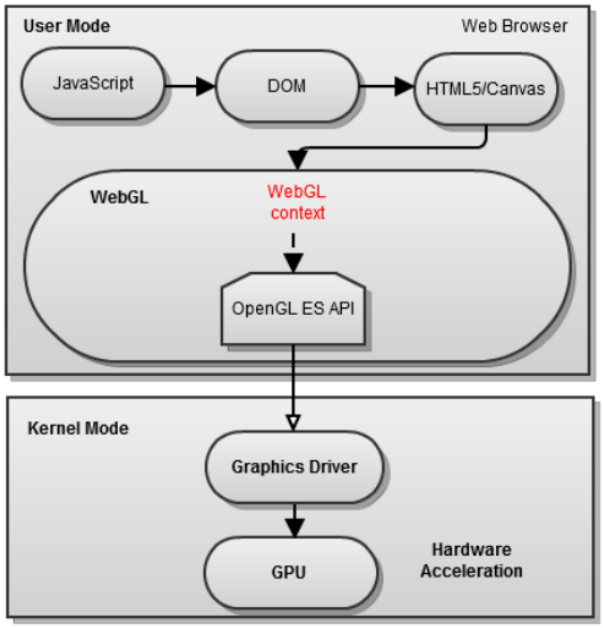

Figura 1. Modelo conceptual de la arquitectura WebGL.

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. Actualmente está soportado por la mayoría de los navegadores. Con canvas podemos crear rectángulos, líneas, arcos, curvas, dibujar imágenes, añadir colores y estilos, además de transformaciones, composiciones y animaciones. Nos permite hacer imágenes dinámicas pero sin plugins externos.

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.
La GPU (Graphics Processing unit) o unidad de procesamiento de gráficos es un procesador dedicado al procesamiento de gráficos u operaciones de coma flotante.

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, implementado como parte del navegador web, permitiendo mejoras en la interfaz de usuario y lo más importante, páginas web dinámicas
Javascript permite a los navegadores ser capaces de reconocer objetos en una página HTML a través del 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.

Three.js Es una biblioteca escrita en JavaScript para crear y mostrar gráficos animados por ordenador en 3D en un navegador Web y puede ser utilizada en conjunción con el elemento canvas de HTML5, SVG ó WebGL. Permiten crear animaciones 3D que se muestran en el navegador sin el esfuerzo que se requiere para una aplicación independiente tradicional con un plugin.

Como utilizarlo

```
<script src="js/three.js"></script>
```

El render es básicamente el objeto WebGL donde la tarjeta gráfica pintará todos los gráficos. Podríamos tomarlo como lo que llamábamos “canvas” y “contexto” en la serie WebGL. Un mundo 3D puede ser enorme, pero en pantalla sólo se pintará aquello que quede dentro del encuadre de la “cámara”.

Para crear un render, necesitamos:

- El ancho en píxeles que tendrá el render
- El alto en píxeles
- El color con el que “limpiar” el render

Cada vez que se dibuja una escena en el canvas, lo primero que hace WebGL es pintar todo el canvas de un color determinado, y luego dibujar todos los objetos de la escena sobre ese color. Ese color es lo que se conoce como color de “limpieza” del render. Podríamos verlo como el color del “espacio” que rodea la escena que queremos dibujar en el canvas.

Veamos cómo definirla en JavaScript usando Three.js:

```
var render;
var canvasWidth = 500;
var canvasHeight = 500;

render = new THREE.WebGLRenderer();
render.setClearColorHex(0x000000, 1);
render.setSize(canvasWidth, canvasHeight);
```

Lo que estamos haciendo es crear un render, de 500×500 píxeles usando el método **setSize**, y estableciéndole un color de limpieza negro mediante **setClearColorHex**, con un alfa de 1, es decir, totalmente opaco.

El objeto render nos proporciona el canvas directamente. Sólo tenemos que meterlo en su sito correcto; por ejemplo, para meterlo en un DIV con id=”canvas” haremos:

```
document.getElementById("canvas").appendChild(render.domElement);
```

Y cada vez que queremos dibujar el render en el canvas, esté donde esté, tenemos que llamar a su método **render**:

```
render.render(escena, camara);
```

Dicho método necesita dos objetos: La escena que queremos renderizar, y la cámara desde donde se renderizará la escena.

Escena es el contenedor de todos los gráficos que quieras dibujar. Todo proyecto Three.js debe contener al menos una escena, y dentro de dicha escena irán los objetos que pertenezcan a la misma “dimensión” de tu mundo, es decir, que puedan ser visibles entre sí dependiendo de la posición de la cámara. Los objetos pueden ser de tres tipos: Figuras, luces y cámaras; hablaremos de ellos después. A veces, según lo que quieras conseguir, puedes necesitar de varias escenas, cada una con diferentes objetos, para ir alternándolas siguiendo algún tipo de lógica, o para dibujar una escena como textura de una figura de otra escena, etc. Pero de momento, vamos a lo simple, sólo usaremos una escena.

```
var escena;
escena = new THREE.Scene();
```

Y para añadir objetos a dicha escena:

```
escena.add(objeto);
```

Se pueden añadir objetos a la escena del tipo Scene dinámicamente, pero como es natural sólo se dibujarán en llamadas posteriores al método render.

Cámara son más que un tipo de objetos que se pueden a añadir a la escena, pero como tienen una importancia fundamental.

Una escena puede contener tantas cámaras como deseemos, pero sólo se podrá utilizar una de ellas para renderizar la escena en un momento dado, usando el método render del objeto con el mismo nombre que vimos anteriormente. Una cámara puede ser posicionada y rotada tantas veces como queramos, pero el resultado de dichos cambios sólo se dibujará en el canvas en posteriores llamadas al método render, como debería serte evidente.

Para definir una cámara hay que tener en cuenta qué tipo de proyección queremos tener. Three.js nos proporciona tres tipos de proyecciones:

- **Proyección paralela**, cámara del tipo **OrthographicCamera**
- **Proyección cónica**, cámara del tipo **PerspectiveCamera**
- **Proyección combinada**, cámara del tipo **CombinedCamera**, y que permite cambiar entre la proyección cónica y paralela en tiempo de ejecución.

OrthographicCamera es una perspectiva que respeta el tamaño de los objetos, independientemente de la distancia a la que se encuentren de la cámara.

PerspectiveCamera es la que deforma los objetos según la distancia y posición que se encuentren con respecto a la cámara, tal y como ocurre en el mundo real.

Para crear una PerspectiveCamera necesitaremos indicarle cuatro valores básicos:

- Ángulo del campo de visión en grados
- Ratio de aspecto, que normalmente es la relación entre el WIDTH y el HEIGHT del canvas donde se va a renderizar la imagen
- Distancia mínima de dibujado
- Distancia máxima de dibujado

```
var camara;
camara = new THREE.PerspectiveCamera(45, canvasWidth / canvasHeight, 0.1, 100);
escena.add(camara);
```

2. Explicar la estructura de un archivo WebGL.

    1. En primer lugar vamos a ver como iniciar nuestro código de webGL en cualquier página html.
    2. Definir en la etiqueta <body> la función encargada gestionar el interface gráfico.
    3. Definir un canvas, asignarle una etiqueta y su tamaño.
    4. Cargar los scripts externos que vayamos a necesitar (en este caso los propios de la API webGL y otro necesario para trabajar con matrices).
    5. Una vez definidos estos aspectos ya podemos comenzar a programar nuestro propio script en webGL.

Hacer una comparación entre OpenGL y WebGL, similitudes y diferencias.

	similitudes	diferencias
$OpenGL$ Vs $WebGL$	✓ nombres de las funciones y los parámetros son muy similares y prácticamente todas las funciones admitidas en WebGL son compatibles con OpenGL. ✓ Ambas necesitan de librería adicional es para su funcionamiento	✓ WebGL está basado en OpenGL ES ✓ WebGL utiliza JavaScript ✓ WebGL es multiplataforma ✓ OpenGL es desarrollado en C ++. ✓ WebGL es un estándar que está creciendo rápidamente y cuyo uso se ha ampliado considerablemente. ✓ WebGL sigue creciendo e incluso se han desarrollado plugins para que pueda correr en los navegadores ✓ Estas aplicaciones no necesitan de ninguna Instalación. ✓ WebGL es una gran tecnología que puede revolucionar la Web3D.

PRACTICA:

1. Desarrollar un programa en WebGL que muestre las figuras básicas: Cuadrado, circunferencia, triángulo, y rectángulo.

COMPUTACION GRAFICA Y VISUAL

EDWAR MEJIA VASQUEZ