ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA

HERRAMIENTA PARA LA REPRESENTACIÓN 3D, USANDO WEBGL, DE CLUSTERS DE SUPERCOMPUTACIÓN

TOOL FOR 3D VISUALIZATION, BASED ON WEBGL, OF HIGH PERFORMANCE COMPUTING CLUSTERS

REALIZADO POR:
FRANCISCO LÓPEZ LIÑÁN

TUTORIZADO POR:
PABLO PÉREZ TRABADO

DEPARTAMENTO:
ARQUITECTURA DE COMPUTADORES

UNIVERSIDAD DE MÁLAGA
MÁLAGA, NOVIEMBRE DE 2014

Fecha defensa:
El Secretario del Tribunal.

Resumen:

Este Trabajo de Fin de Grado constituye el primero en una línea de Trabajos con un objetivo común: la creación de una aplicación o conjunto de aplicaciones que apoye a la administración de un cluster de supercomputación mediante una representación en tres dimensiones del mismo accesible desde un navegador. Esta aplicación deberá ser de fácil manejo para el personal que haga uso de ella, que recibirá información procedente de distintas fuentes sobre el estado de cada uno de los dispositivos del cluster.

Concretamente, este primer Trabajo se centra en la representación gráfica del cluster mediante WebGL, el estándar para renderizado 3D en navegadores basado en OpenGL, tomando como modelo de desarrollo el SCBI (Centro de Supercomputación y Bioinnovación) de la Universidad de Málaga. Para ello, se apoyará en la creación de una herramienta con la que describir texualmente de forma intuitiva los elementos de una sala de supercomputadores y los datos asociados a los mismos. Esta descripción será modificable para adaptarse a las necesidades del administrador de los datos.

Palabras clave:

Supercomputación, Servidores, Monitorización, Supevisión, WebGL, OpenGL, 3D, Navegador, Interfaz.

Abstract:

This Degree's Final Project constitutes the first in a series of Projects with a common goal: the creation of an application —or set of applications— that helps supporting the management of a supercomputing cluster by representing it in a three dimensional environment, accesible from a modern web browser. This application should enjoy a design with an intuitive functioning for the people to use it, who will receive a certain amount of organized information from different sources about the state of each and every device in the cluster.

Specifically, this first Project focuses on the cluster's graphic representation using WebGL, the 3D web browser rendering standard based on OpenGL with the SCBI (SuperComputing and BioInnovation Centre) of the University of Málaga taken as development model. In order to achieve that, it will rely on a tool developed to easily describe the elements in a supercomputing room and the metadata that can be associated to them. This description will be editable to adapt it as the administrator considers. It also takes the firsts steps in

Keywords:

Supercomputing, Servers, Monitoring, Supervision, WebGL, OpenGL, 3D, Web browser, Interface.

ÍNDICE

Capítulo 1 - Introducción

1.1 El problema tratado

En un centro de supercomputación, como el SCBI (Supercomputing and Bioinnovation Center) de la Universidad de Málaga, los computadores que realizan el cálculo intensivo para resolver los distintos problemas que se le plantean se encuentran en una localización concreta del edificio, acondicionada para el mantenimiento de los equipos. Estas salas de gran envergadura se diseñan específicamente para alojar los computadores en unas condiciones que favorezcan su funcionamiento de forma continuada. El tratamiento de la información aportada y la obtenida de los nodos de computación se considera crítico y, por tanto, se toman numerosas medidas para garantizar una correcta ejecución de los programas sin alteraciones externas.

Para ello, la infraestructura de la sala requiere un alto coste que asegure que no existan problemas de alimentación eléctrica, de altas temperaturas debido al calor disipado por cada computador o de saturación en el sistema de comunicaciones entre nodos, además de las potenciales averías de los mismos.

La inversión en un centro de cálculo de la magnitud del SCBI convierte en algo fundamental para su mantenimiento la supervisión frecuente por parte de personal especializado, que utiliza distintas herramientas disponibles (comerciales o de código abierto) para monitorizar el estado de los equipos y la eficiencia de éstos en su ejecución.

El objetivo de este trabajo es el de desarrollar, dentro de una línea de Trabajos de Fin de Grado, una aplicación que facilite la labor del personal supervisor de un centro de cómputo, tomando como ejemplo el SCBI de la Universidad. Esta aplicación (o conjunto de aplicaciones) se encargará de proveer de una representación tridimensional de la sala, de fácil acceso, uso y modificación, que asista a la hora de comprobar la información referente a cada máquina disponible, haciéndolo más intuitivo y ágil que la solución en uso. Se utilizará la tecnología WebGL para navegadores web y, concretamente, la librería de JavaScript de alto nivel three.js, dada la escalabilidad y la sencillez de su instalación (de cara al usuario, sólo es necesario el navegador, sin complementos adicionales).

El funcionamiento de la aplicación, en términos generales, será el siguiente. El usuario que tenga acceso a los datos, visualizará la representación 3D en un navegador web apto conectándose al servidor. Podrá desplazarse por el entorno libremente o mediante puntos de anclaje especificados en la aplicación. También podrá centrarse en cada equipo para obtener los datos disponibles.

Por otra parte, un administrador de la aplicación tendrá acceso a la base de datos que almacena el contenido a representar. Mediante otro interfaz web, podrá acceder a los datos de cada máquina almacenados en las tablas de esta base de datos desde un enfoque asociado a la

propia información. Podrá modificar cada dato o añadir nuevos campos o elementos (en forma de tablas) a la base y, cuando haya terminado, generar los ficheros necesarios para la actualización y que serán enviados desde el servidor de la aplicación de representación.

Desde el punto de vista personal, la realización de este trabajo me resultaba muy atractiva dada mi fascinación por la creación de entornos virtuales interactivos, y la posibilidad de llevar mi aprendizaje básico de la asignatura optativa de Programación de Videojuegos para resolver una problemática real más allá de aportar un entretenimiento pasajero no hizo más que confirmar mi decisión.

1.2 Estado de la técnica

A la hora de monitorizar un cluster de supercomputación, se puede obtener información útil mediante dos métodos: el uso de sensores colocados físicamente en los dispositivos y sondas software que capturen el estado del sistema.

Por un lado, los sensores incluidos por los fabricantes en los equipos de cada rack proporcionan información real de magnitudes físicas en distintos puntos del nodo de computación como temperatura, tensión eléctrica de la alimentación, consumo de potencia o el porcentaje de utilización de los recursos del equipo. Estos datos resultan cruciales para dirigir la atención a unidades defectuosas o programas mal diseñados que puedan consumir una mayor cantidad de recursos de los esperados (o incluso que infrautilicen los asignados) y se pueden obtener remotamente.

Por otra parte, el uso de aplicaciones de profiling, aunque afecta con su observación al programa monitorizado, resulta útil si se pretende analizar el rendimiento de una aplicación a lo largo de su ejecución y la eficiencia en el uso del sistema operativo o de los recursos proporcionados.

Se puede encontrar en el mercado un cierto número de aplicaciones tanto open-source como propietarias de uso frecuente por administradores de sistemas que resuelven el problema de la lectura de los datos de estas dos fuentes. Es el caso de la interfaz estandarizada de monitorización IPMI (Intelligent Platform Management Interface) y las distintas versiones diseñadas por los fabricantes de computadores como iLO (integrated Lights Out) de Hewlett-Packard, en uso en equipos del fabricante en el SCBI. Estas herramientas suponen un volcado de una gran cantidad de información tabulada o en gráficas difícil de asociar entre sí, y más aún de asociar espacialmente a dispositivos concretos dentro de una sala real. Tampoco se cuenta con medios de observar simultáneamente de una forma intuitiva varios orígenes de los parámetros recibidos para comprobar cuál es el impacto de cada uno en el funcionamiento del nodo de computación o representar un histórico de ese funcionamiento adecuadamente. Es por esto que el uso de una representación 3D que poder utilizar para monitorizar remotamente un cluster resulta tan interesante como pionera.

Capítulo 2 - Tecnologías utilizadas

En este capítulo se presentarán las tecnologías utilizadas en el trabajo. No es propósito de esta memoria el entrar en profundidad en el estudio de todas ellas ya que, mientras que algunas de las tecnologías son de uso cotidiano (como las relativas al desarrollo web), otras representan una vanguardia tecnológica que merece un mayor grado de atención.

2.1 JavaScript

2.1.1 Un lenguaje para el desarrollo web

JavaScript es un lenguaje de programación interpretado derivado del estándar ECMAScript. Aportando una orientación a objetos y prototipado de los mismos, este lenguaje imperativo de tipos débiles es de uso frecuente en la ejecución de programas en el lado del cliente de servicios web. A pesar de adoptar algunos nombres y convenciones del lenguaje compilado Java, no tiene relación directa con el mismo ni comparte los propósitos con los que está ideado. Se trata de un lenguaje multiplataforma, por lo que cualquier navegador web moderno incluye un intérprete de código JavaScript, al que provee de una implementación del DOM (Document Object Model), la API de representación de documentos HTML.

Aunque existen implementaciones de JavaScript como Node.js que ejecutan código en el lado del servidor, el uso mayoritario del lenguaje es en el lado del cliente. El código se plantea como una secuencia de funciones y llamadas a las mismas encerradas en etiquetas <script> en el documento HTML, pudiendo encontrarse éstas escritas en plano o en un fichero .js incluido como fuente.

2.1.2 Origen y evolución

La necesidad de la creación de JavaScript se plantea a principios de la década de los 90, en el nacimiento de las primeras aplicaciones web. El uso de formularios web complejos requería el envío de una cantidad de información incremental según las dimensiones de los mismos, cosa que ralentizaba los procesos dado el limitado ancho de banda de los módems, con una velocidad máxima de 28.8kbps. Para paliar el impacto de las peticiones y envíos de formularios, Brendan Eich, trabajador de NetScape, desarrolló un antecesor de JavaScript llamado LiveScript. Este lenguaje, ejecutable en la versión 2.0 del NetScape Navigator de 1995, contendría código ejecutable en el lado del cliente antes del envío de las respuestas de los formularios. Se comprobaría entonces si los datos introducidos en los campos del mismo contenían errores que normalmente se percibirían en el servidor, aligerando la comunicación entre el cliente y el servidor y reduciéndola a los mensajes necesarios.

NetScape anunció una alianza comercial con Sun Microsystems, que ese mismo año había publicado su primera versión de Java. Se decidió que NetScape soportaría este nuevo

lenguaje en sus navegadores, y se cambió el nombre de LiveScript por el de JavaScript. Este movimiento comercial no dejó de resultar confuso, ya que la similitud entre ambos lenguajes, como se ha comentado, es prácticamente anecdótica.

En 1997, el organismo internacional ECMA (European Computer Manufacturers Association) lo admitió como estándar, llamándolo ECMAScript y, posteriormente, pasó a ser un estándar ISO/IEC.

Al igual que sucedió con Java y otras patentes de Sun Microsystems, JavaScript pasó a ser una marca registrada de Oracle Corporation tras su adquisición de la compañía.

2.1.3 Tipado débil y ámbito de variables

El intérprete de JavaScript realiza una asignación dinámica de tipos, como es habitual en los lenguajes de script. La función typeof se encarga de proporcionar apoyo al programador que quiera hacer uso de esta capacidad, a modo de polimorfismo.

El código mantiene una amplia similitud con la sintaxis del lenguaje C, pero el ámbito de las variables tiene un tratamiento considerablemente distinto: mientras que en C el ámbito comprende el bloque en el que se define cada elemento, en JavaScript sólo lo es en la función en la que se declara. Esta aproximación, junto con la descripción de objetos, produjo en el desarrollo del trabajo algunos inconvenientes, ya que el concepto de variable global no se aplica de igual forma. Por ejemplo:

```
var valor = "a";
function imprime() {
     console.log(valor);
     valor = "b";
     console.log(valor);
}
imprime();
```

La función *imprime* toma la variable *valor* y la presenta en la salida estándar de la consola, incorporada en el navegador. Mediante un planteamiento basado en C, cabría esperar que se imprimiese la letra *a*, seguida de la letra *b*. Sin embargo, el resultado obtenido es un valor *undefined* seguido de *b*. El ámbito de la variable *valor* definida inicialmente no comprende la función *imprime*, por lo que ninguna transformación o impresión se realiza sobre ella. Sin embargo, a pesar de no haber declarado *valor* dentro de la función explícitamente con la palabra reservada *var*, esta variable se mantiene con un valor indefinido esperando la asignación que aparece en la segunda instrucción.

2.1.4 Orientación a objetos y prototipado

La mayor parte de la funcionalidad de JavaScript se basa en objetos. En este lenguaje, un objeto es un array asociativo que contiene una serie de propiedades en forma de pares clave-valor, pudiendo ser el valor otro objeto anidado. Para acceder al valor de cada propiedad, se utiliza la cadena de caracteres que hace de nombre de la misma. Esto se puede realizar escribiéndola tras un punto: obj.nombre o entre corchetes: obj['nombre'].

Los prototipos son la aproximación de JavaScript a la herencia de clases. Se trata de objetos que añaden esta funcionalidad a cada objeto desde Object.prototype, la implementación nativa. Un ejemplo de creación de un prototipo sería el siguiente:

```
function persona(nombre, altura){
     this.nombre = nombre;
     this.altura = altura;
}
```

Con esta función, se puede utilizar la palabra reservada "new" para crear objetos desde ese prototipo. La función actúa como un constructor, similar al de otros lenguajes compilados como Java.

2.2 WebGL

2.2.1 Representación 3D en navegador

WebGL es una tecnología emergente que permite la descripción, renderización e interacción con un entorno 3D complejo mediante navegadores web sin instalación de software adicional. Los propios navegadores de uso mayoritario (Internet Explorer, Firefox, Chrome, Safari…) ya incluyen una implementación de WebGL en sus versiones de escritorio y algunos están en el proceso de optimización de la misma para dispositivos móviles compatibles en el momento de la redacción de esta memoria.

Es altamente preferible el uso de aceleración hardware para conseguir una experiencia óptima con WebGL, al igual que con cualquier sistema de renderizado 3D. Esto no resulta un inconveniente reseñable, ya que incluso los ordenadores portátiles de gama baja cuentan ya con tarjetas gráficas integradas que pueden cumplir habitualmente con estos requerimientos. El hecho de que WebGL acceda a los recursos de la tarjeta gráfica permite, con el código adecuado, unos gráficos visualmente realistas con resoluciones superiores a 1080p y una animación fluida a 60 fotogramas por segundo estables, considerados en la industria del videojuego un estándar técnico por alcanzar.

De cara al programador, WebGL se plantea como una API en JavaScript con la que enviar, desde la CPU, ciertas órdenes a la tarjeta gráfica disponible en el sistema. Estas instrucciones están contenidas en unos programas específicos, los shaders, los cuales son compilados en tiempo de ejecución por la propia API y enviados a la GPU como sucedería con cualquier herramienta de visualización 3D o videojuego tradicional.

La salida gráfica de las GPUs es dirigida sobre un elemento etiquetado como <canvas> en HTML5, que representará la escena en la página HTML que lo contenga según se haya especificado mediante el objeto WebGLRenderer asociado. Estos elementos son nativos en HTML5, por lo que no es necesaria instalación de plugins ni otros complementos para plantear la escena.

2.2.2 OpenGL y OpenGL ES, base del estándar

La tecnología WebGL toma su nombre del estándar OpenGL (Open Graphics Library), con cuya versión OpenGL ES está estrechamente relacionado, dada su especialización en sistemas embebidos como videoconsolas, smartphones y tabletas.

En directa competencia con Direct3D de Microsoft, OpenGL es una especificación estandarizada que presenta la descripción de una API abstracta manejable en múltiples plataformas para la representación de gráficos en dos y tres dimensiones. La interfaz plantea una interacción entre el usuario y el hardware para renderizar escenas complejas mediante una serie de primitivas geométricas básicas como puntos, líneas y triángulos, base tradicional de creación de polígonos y superficies en el campo de los gráficos por computador.

La primera versión de OpenGL fue desarrollada en 1992 por Mark Segal y Kurt Akeley a modo de librería open-source que hiciera de alternativa a IRIS GL (Integrated Raster Imaging System Graphics Library), en uso en las costosas estaciones de trabajo de Silicon Graphics Incorporated. La segunda versión del estándar introdujo por primera vez el concepto de shaders programables en el GLSL (GL Shading Language).

El grupo Khronos, un consorcio industrial sin ánimo de lucro especializado en APIs y otras herramientas asociadas a la creación y reproducción multimedia, tomó el control del proyecto de especificación OpenGL en 2006. Se eliminaba así la barrera entre el desarrollo de OpenGL y OpenGL ES, que ya estaba en desarrollo por parte del grupo, y se impulsaron notablemente ambos proyectos. En el momento de redacción de esta memoria, OpenGL se encuentra en la versión 4.5, publicada el mes de agosto de 2014.

Cuando Khronos comenzó a trabajar a principios de los 2000 en OpenGL ES (Embedded Systems), pretendía crear un estándar a partir de un subconjunto de la especificación OpenGL orientado a una ejecución en ordenadores empotrados. En un período anterior a la explosión en el mercado de smartphones y tabletas, el consorcio dio uno de los primeros pasos en los gráficos por computador de alto rendimiento como aquellos en las videoconsolas de sobremesa de la época. A lo largo de los años, y gracias a la especificación más ligera y sencilla de implementar que la de OpenGL, sucesivas actualizaciones de

OpenGL ES se han incluido en sistemas de juego uso mayoritario. La videoconsola Playstation 3, la portátil Nintendo 3DS (que emplea 3D estereoscópico sin necesidad de las habituales gafas) y los dispositivos Android, iOS y Symbian, entre otros, siempre de los principales fabricantes. A fecha de publicación de este texto, se encuentra en la versión 3.1, con fecha de marzo de 2014.

WebGL se basa en la versión 2.0 de OpenGL ES como evolución estandarizada a la implementación del prototipo de Canvas 3D que los ingenieros de Mozilla y Opera habían desarrollado por su cuenta para el elemento <canvas> de HTML5. En 2009, Khronos dio el paso de creación del grupo de trabajo que se encargaría de compatibilizar la implementación de cada navegador. Dos años más tarde, se publicó la especificación WebGL 1.0 y, en marzo de 2013, la versión 1.0.2, en la que se encuentra hasta este momento en espera de la versión 2.0 basada en OpenGL ES 3.0.

2.2.3 Shaders y pipeline gráfico

Como ya se ha mencionado, WebGL proporciona una interacción con el hardware de aceleración 3D desde código JavaScript estandarizado, incrustable en una página HTML. Esto sucede por medio de la ejecución de unos programas, los shaders, en las tarjetas gráficas u otro hardware integrado destinado a la representación de gráficos en tres dimensiones. Para comprender el funcionamiento de los shaders, es necesario introducir otro concepto: el de pipeline gráfico.

Se llama pipeline gráfico o pipeline de renderizado al proceso por el cual se rasteriza una imagen 2D desde su definición 3D, esto es: se crea una representación en píxeles visualizable en un monitor a partir de los vectores y formas que están siendo objeto de las distintas transformaciones en la máquina. Tanto OpenGL como DirectX (Direct3D) ofrecen pipelines gráficos y permiten la programación de shaders adicionales para alterar el resultado con distintos efectos.

Originalmente, el pipeline gráfico de OpenGL estaba formado por una serie de etapas con una funcionalidad fijada atendiendo al hardware de los aceleradores gráficos. Era posible habilitar o no algunas de estas etapas, que podían añadir, por ejemplo, iluminación a la escena; pero las propias operaciones no eran alterables. Fue a partir de la segunda versión de OpenGL cuando se permitió la programación de los pipelines gráficos. Los componentes físicos que implementan estos procesos se corresponden con las GPU (Graphics Processing Unit), que sí permiten la carga de shaders dinámicamente al crear el pipeline. Los pasos que sigue el pipeline son, resumidamente, los siguientes:

- Preparar el array de vértices y renderizarlo.
- Aplicación de un Vertex Shader (Shader de vértice) a cada uno.
- Opcionalmente, aplicación de herramientas de teselación (añaden detalle a superficies 3D, haciéndolas más realistas).

◆ Opcionalmente, uso de shaders de geometría (se obtiene una serie de primitivas).

➢ Post-procesado de vértices y ajustes necesarios.

➢ Se prepara la rasterización y se obtiene una serie de fragmentos a partir de interpolación de primitivas.

➢ Aplicación de un Fragment Shader (Shader de fragmento) que genera varias salidas.

◆ Opcionalmente, aplicación de una serie de tests que, por ejemplo, determinarán si debe mostrarse cada fragmento según su posición o fusionará los mismos en caso de transparencia.

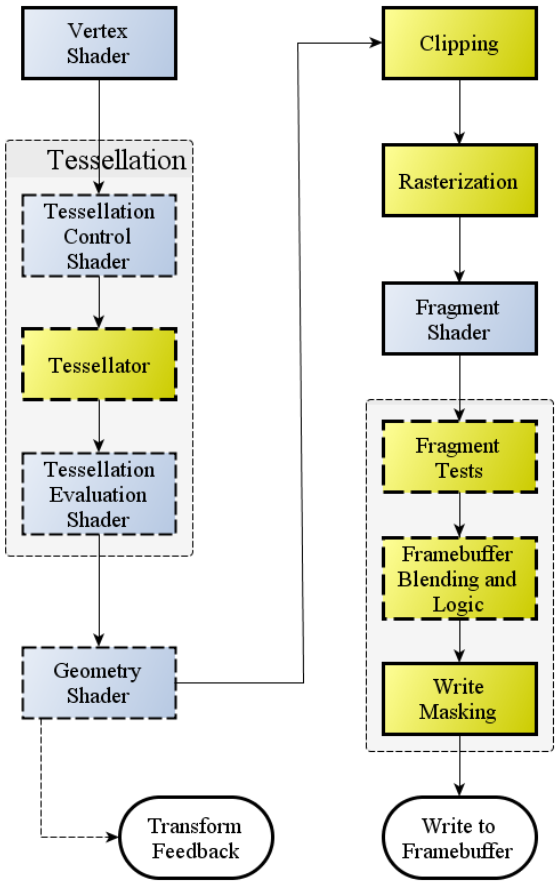


Figura 2.1: Pipeline gráfico de OpenGL
Fuente: www.opengl.org

La figura 2.1 muestra el diagrama del pipeline de renderizado. Los elementos con el borde discontinuo son opcionales, mientras que los que contienen shaders constituyen las etapas programables del proceso.

En el caso de WebGL, los Vertex Shader y Fragment Shader se pueden incluir en elementos <script> que contienen código GLSL. Este código será compilado al comenzar la ejecución en el navegador y, una vez enlazado, se cargará para su uso en el dispositivo de aceleración gráfica. Los datos que se hayan especificado en el código WebGL se utilizarán en la representación de la imagen para cada fotograma, que habrá seguido el proceso de pipeline descrito.

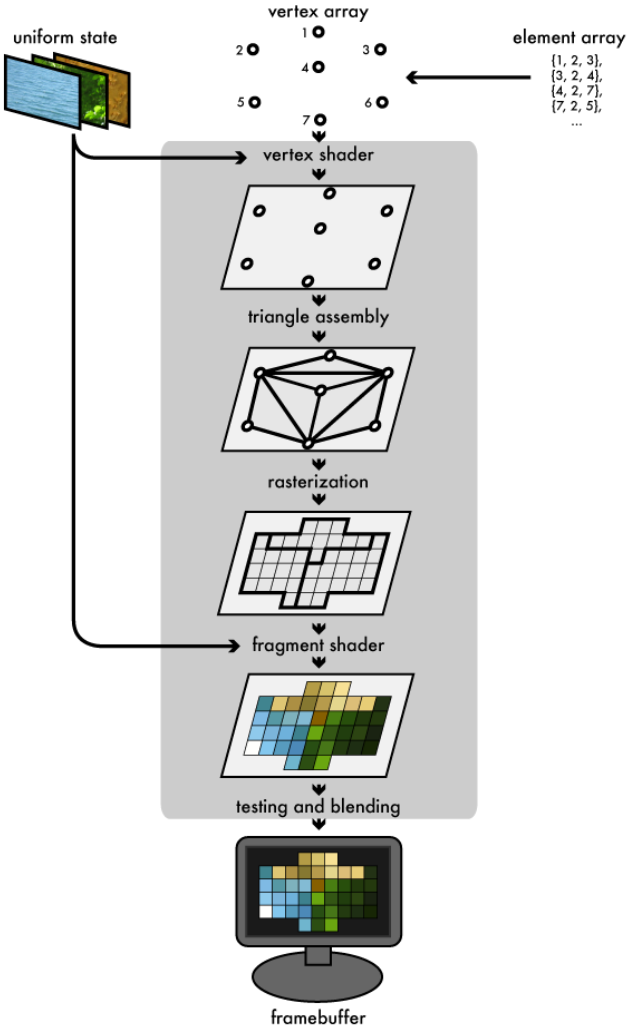


Figura 2.2: Ejemplo de pipeline gráfico de OpenGL
Fuente: http://duriansoftware.com/joe/

La figura 2.2 muestra de una forma más intuitiva el proceso. El Vertex Shader se encarga de calcular transformaciones en los vértices, mientras que el Fragment Shader se ocupa de otras opciones de presentación como iluminación, texturas, color o canal alpha en caso de transparencia. A efectos de compilación, ambos shaders son enlazados para su ejecución en la GPU. La funcionalidad de los shaders se define mediante código en lenguaje GLSL (GL Shading Language), que para el caso que nos atañe será en su versión ES (de OpenGL ES, para dispositivos embebidos), que está basado en C++.

2.3 three.js

2.3.1 Características básicas de la librería

Un programador web sin conocimientos avanzados de programación gráfica encontraría inabarcable el uso de WebGL sin ningún apoyo adicional. El estándar no ofrece una orientación a objetos de alto nivel y requiere una explicitud de la descripción de cada transformación y operación matricial que se encargue de los cálculos de los vértices y el resto de elementos de la pipeline. También debería programar cada shader necesario en la página, necesitándose conocimientos avanzados de GLSL. Ante esta problemática aparece three.js, una librería JavaScript que actúa como framework y la soluciona generosamente.

three.js fue creada en 2010 y liberada en GitHub con licencia MIT por Ricardo Cabello, conocido por el seudónimo "Mr.Doob" y sitúa a WebGL a una altura alcanzable por usuarios no experimentados en programación gráfica. La escritura del código se vuelve semejante a la de otros motores gráficos mayoritarios gracias a, por ejemplo:

- Estructuración de la representación 3D mediante un grafo de escena.
- Funciones de alto nivel en una API propia con sistemas de cámaras, luces, geometrías básicas y materiales con numerosas opciones.
- Posibilidad de cargar modelos 3D desde distintas fuentes como CAD, 3DSMAX o Blender, entre otros, en distintos formatos.
- La opción de la activación del renderizador nativo mediante software en caso de no encontrarse disponible hardware de aceleración gráfica.

2.3.2 Grafo de escena

Habitualmente, el software de representación de escenas tridimensionales utiliza el concepto de grafo de escena para describirlas. WebGL no incluye tal apoyo conceptual, por lo que se dificulta notablemente la programación. three.js se encarga de definir el modelo de este árbol de nodos con relaciones padre-hijo mediante los objetos THREE.Object3D.

Este objeto JavaScript puede utilizarse para definir un elemento individual de las geometrías básicas disponibles, un modelo 3D cargado desde fichero, una cámara o, en esencia, cualquier entidad de la escena. Más aún: un THREE.Object3D puede tener varios objetos hijo asociados, por lo que son frecuentes agrupaciones de elementos bajo nodos concretos.

El potencial de este tipo de estructura queda claro cuando se pretende aplicar una transformación espacial de posición, rotación o escala a un elemento (o grupo de elementos),

utilizándose propiedades incluidas en los objetos THREE.Object3D. Como ejemplo, para desplazar un automóvil por una carretera, se cambiaría el parámetro position (representado mediante un vector de tres dimensiones) de un nodo que agrupase como hijos al propio vehículo, al conductor y a los pasajeros. Por otra parte, el objeto asociado a cada rueda podría incrementar cada fotograma el parámetro rotation en algunos de sus ejes para representar un movimiento realista mediante su giro.

Se permite, de esta forma, la elaboración de escenas complejas mediante un código intuitivo y ligero de fácil mantenimiento.

2.3.3 Geometrías, materiales y mallas

En la mayoría de casos, para la descripción de una escena, se requiere el uso de geometrías sencillas como cubos, esferas y figuras de revolución, no sólo objetos 3D cargados desde un fichero generado por un software de modelado como Blender o Maya. Es por esto que three.js incluye en su API un gran número de funciones parametrizadas que generan geometrías simples, como las mencionadas, o más complejas, como toroides o nudos.

Los objetos que retornan estas funciones son del tipo TREE.Geometry y no contienen información de texturas por defecto, por lo que su superficie no representará más que el color de la iluminación que esté recibiendo. El uso habitual de las geometrías se realiza mediante la declaración de un objeto THREE.Mesh (malla), que tomará la figura geométrica y un objeto THREE.Material que contenga la textura.

Los materiales de las geometrías en three.js se implementan según tres aproximaciones con distinto impacto en el rendimiento y apariencia en la escena:

- MeshBasicMaterial: Se toman los valores de color, textura y opacidad para representar la superficie, mientras que la iluminación (o falta de ella) no afecta a la misma.

- MeshPhongMaterial: Implementa una técnica de iluminación similar al Phong Shading, que reacciona ante la iluminación de la escena de forma realista y con buen rendimiento. La reacción a la luz es resultado de una interpolación a partir de componentes ambientales, difusa y especular de la iluminación a lo largo de la superficie.

- MeshLambertMaterial: Su aproximación se basa en la Lambertian Reflectance, que calcula el valor del brillo en cada píxel mediante una interpolación que no tiene en cuenta la dirección desde la que se observa.

2.3.4 Iluminación

Al igual que en otros motores gráficos, los elementos lumínicos de alto nivel ofrecidos por three.js son elementos adicionales de la escena, contenibles por objetos del grafo y posicionables en cualquier punto del espacio. Los parámetros que se aporten pueden modificar la intensidad, color o, dado el caso, radio de efecto hasta que comienzan a

desvanecerse. Se encuentran en la librería, entre otras, tres tipos de luces de uso frecuente:

- AmbientLight: Ilumina toda la escena por igual con el color e intensidad fijado.
- DirectionalLight: Dirige un haz de luz en una dirección marcada por un vector desde un punto dado.
- PointLight: Ilumina en todas las direcciones desde un punto concreto dentro de un radio fijado.

A la hora de representar las sombras de los objetos en la escena producidas por la iluminación, se debe activar la propiedad booleana castShadow del objeto Mesh deseado y especificar en el renderizador una serie de parámetros. Esto se debe a que la generación de sombras es muy costosa computacionalmente, por lo que debe limitarse a objetos concretos.

2.4 TWEENJS

2.4.1 Animaciones simples en HTML5

Desarrollado por el equipo CreateJS como parte de su paquete de código libre para diseño web, TWEENJS es una pequeña API JavaScript diseñada para aplicar “tweening” (del inglés “inbetweening”, intermediado). Utiliza técnicas de interpolación mediante distintos modelos matemáticos para realizar cambios graduales en valores de propiedades de HTML5 y JavaScript.

Mediante sucesivas llamadas al procedimiento TWEEN.Update, aquellas transformaciones que hayan sido creadas y comenzadas explícitamente se aplicarán. Por ejemplo, asignando una variable numérica en la propiedad de un objeto JavaScript a un objeto tween y especificando el valor final, usar el método start del objeto resultará en una transición de valores a lo largo del tiempo según, por ejemplo, una función cuadrática. Los tweens se pueden secuenciar, dando la posibilidad de bucles de animación para elementos HTML, cambios graduales en el color de los mismos y todo tipo de transiciones.

TWEENJS se plantea, entonces, como un añadido sencillo de implementar mediante su API que puede proporcionar cierto grado de animación a las escenas 3D de WebGL sin necesidad de especificar el cambio para cada fotograma literalmente en el código. En lo que respecta a este Trabajo, la librería ha servido para que los cambios en la posición de la cámara (y el lugar al que ésta observa) durante la ejecución del programa sean graduales y suavizados, además de añadir elementos decorativos de iluminación

2.4.2 Uso de la librería

El siguiente es un ejemplo de declaración de tween en JavaScript para un objeto con una

posición en dos dimensiones dada:

```
var object = new TWEEN.Tween(object.position)
             .to({x: 10, y: 5}, 1000)
             .easing(TWEEN.easing.Quadratic.InOut)
             .delay(500)
             .start();
```

El proceso de animación del objeto comenzaría tras un retraso de 500 milisegundos después de ejecutarse la instrucción start. Durante el siguiente segundo, la posición del objeto pasaría a ser la especificada (10, 5), siguiendo un desplazamiento cuadrático. La especificación InOut del easing (suavizado) situará el punto de inflexión de la función utilizada en la mitad del desplazamiento, siguiendo el objeto un movimiento en zigzag en los dos ejes.

2.5 JSON

2.5.1 Descripción textual de objetos

JSON (JavaScript Object Notation) es un estándar ligero de marcado diseñado para el intercambio de datos, planteado como un subconjunto de la notación de objetos JavaScript que lo hace intuitivo de generar y leer por humanos y analizadores sintácticos. Fue especificado por Douglas Crockford y descrito en el RFC 4627 como estándar abierto muy ligado a JavaScript, aunque mantiene independencia con el lenguaje.

Una descripción JSON se construye mediante dos estructuras, que se corresponden con los arrays típicos en cualquier lenguaje y aquellos asociativos de clave y valor. Los tipos base de JSON son numéricos, cadenas de caracteres y valores booleanos.

2.5.2 Sintaxis

La sencilla sintaxis de JSON resulta uno de sus principales atractivos. Para crear un objeto, basta con delimitar entre llaves las propiedades a almacenar. Éstas se separan entre ellas con comas, y unos dos puntos realizan la asignación propiedad-valor. Las cadenas de caracteres, que representan tanto a los tipos base como a los nombres de los elementos, se deben escribir entre comillas dobles. Por último, los arrays se representan delimitando con corchetes una lista de elementos, también separados por comas.

Así, un ejemplo de objeto JSON sería el siguiente:

```
[{"rol" : "usuario", "permisos" : true},  {"rol" : "administrador"}]
```

La función JSON.Parse, disponible de forma nativa en JavaScript, se encarga muy eficientemente de analizar sintácticamente el objeto JSON y generar uno en la variable que se especifique. En el caso del ejemplo, se obtendría un objeto contenedor de un array, listando dos objetos con las propiedades descritas.

2.6 PHP

2.6.1 Un lenguaje del lado del servidor

PHP (del acrónimo recursivo PHP Hypertext Pre-Processor) es un lenguaje de programación de propósito general diseñado originalmente para ser ejecutado en el lado del servidor en desarrollo web de páginas HTML dinámicas. Uno de los primeros lenguajes del lado del servidor en ser implementado como código incrustable en documentos HTML, esta contrapartida a JavaScript debe ser interpretada por un procesador de PHP en el servidor, que genera en su salida estándar el código de la propia página resultante. El hecho de ser uno de los lenguajes más flexibles y de alto rendimiento, además del uso gratuito de su intérprete mediante la licencia open-source con la que se distribuye ha hecho que gane notoriedad en los últimos años entre las grandes plataformas de Internet, como es el caso de Wordpress.

2.6.2 Origen y evolución

Al igual que JavaScript, PHP tiene su razón de ser en una problemática básica de principios de los años 90 que hoy se consideraría trivial. El programador danés-canadiense Rasmus Lerdorf, en 1995, publicó Personal Home Page Tools, una versión inicial del lenguaje que había evolucionado desde un primer diseño en el lenguaje Perl creado el año anterior en conjunción con Form Interpreter. Este grupo de CGI binarios en C que antecede a PHP tenía como objetivo dinamizar la presentación de su página web personal, pudiendo usarse para comunicarse con distintas bases de datos, trabajar con formularios web o, simplemente, obtener información de las visitas recibidas.

La primera versión de PHP Tools ya contaba con la funcionalidad básica del lenguaje a día de hoy en cuanto a variables semejantes a las de Perl, manejo de formularios y la capacidad de incrustarse en código HTML. A finales de 1997, un equipo de desarrollo publicó una segunda versión con numerosos cambios, y en 1998 apareció la tercera, tomando ya el conocido acrónimo recursivo. Al ser un lenguaje sin un diseño planteado, sino desarrollado gradualmente por un equipo reducido y para situaciones específicas en un principio, muchas críticas se referían a PHP tildándolo de inconsistente en su descripción.

En el año 2000 tuvo salida la cuarta versión de PHP, con un motor interno reescrito llamado Zend Engine, desarrollado por Zeev Suraski y Andi Gutmans, dos programadores israelíes que fundaron la empresa Zend Technologies, "la compañía PHP", que actualmente tiene sede en Cupertino, California. Cuatro años después aparecería PHP5, la versión más popular del lenguaje, que aportó orientación a objetos y una interfaz consistente de acceso a bases de datos llamada PDO (PHP Data Objects). Esta última versión ha contado con numerosas actualizaciones e incrementos en su funcionalidad desde entonces, incluyendo la abandonada sexta iteración del lenguaje, que pasó a ser PHP 5.3.

2.6.3 Código embebido en HTML

Aunque no es necesario que el código PHP esté incluido en un documento HTML, la capacidad de generar elementos para la página dinámicamente en el lado del servidor hace que sea decisión habitual incluirlo en el desarrollo web. El código se debe encerrar entre los caracteres *<?php* y *?>*, generando un elemento etiquetado más del documento que será procesado por el servidor antes de enviar la página HTML. Un ejemplo de código PHP funcional sería el siguiente:

```
<!DOCTYPE HTML>
<HTML>
    <head>
        <title>Prueba PHP</title>
    </head>
    <body>
        <?php echo '<p>Texto generado</p>'; ?>
    </body>
</HTML>
```

Donde el elemento párrafo que contiene "Texto generado" se visualizaría en el navegador como si hubiera formado parte del fichero .HTML desde el principio, sin ninguna traza de uso de PHP. Esto facilita, por ejemplo, el acceso seguro a bases de datos sin tener que enviar ningún tipo de credencial al cliente que recibe la página. Un usuario de la base de datos especificado para el servidor sería lo único necesario, siendo toda lectura de consultas la misma transparente al cliente mientras tenga denegado el acceso al fichero con las

instrucciones PHP.

2.6.4 Servidor XAMPP

En el caso de este Trabajo, el desarrollo de la aplicación se ha realizado mediante un servidor XAMPP (Apache, MySQL, PHP, Perl en X sistema operativo), un paquete multiplataforma que incluye lo necesario para crear rápidamente y gestionar el servidor web, el intérprete y la base de datos a utilizar. XAMPP está liberado bajo licencia GNU y es instalable en sistemas Windows, GNU/Linux, Solaris y MacOS X, por lo que resulta una decisión apropiada para comenzar.

Capítulo 3 - Diseño e Implementación del Trabajo

3.1 Objetivos

Este Trabajo de Fin de Grado tiene los siguientes objetivos:

1. Profundizar en el suficiente aprendizaje sobre la especificación WebGL y, en particular, la librería de alto nivel three.js.

2. Familiarizarse con los lenguajes JavaScript y PHP orientados a la programación AJAX de páginas web dinámicas.

3. Conocer la arquitectura y distribución del cluster Intel E5-2670 del SCBI de la Universidad de Málaga, para orientar la base del trabajo a las necesidades de su administración.

4. Desarrollar la descripción WebGL de una sala de servidores o nodos de supercomputación genérica a partir de información textual aportada al programa.

5. Desarrollar una serie de programas del lado del servidor que atiendan a las modificaciones de los datos de la descripción, así como de los propios metadatos asociados.

La primera meta del trabajo resultó la más extensa, pero instructiva y enriquecedora, además de la más satisfactoria de completar. Este objetivo estaba muy ligado al segundo, ya que la librería three.js sigue siendo un complemento al lenguaje JavaScript, con el que no había tenido experiencia académica previa. Sí resultaron muy valiosos los conocimientos adquiridos en la asignatura optativa del Grado en Ingeniería Informática de Programación de Videojuegos: la estructura del grafo que representa la escena tridimensional en la librería sobre WebGL tiene gran paralelismo con la del motor gráfico Java3D utilizado en esa asignatura. También resultaron similares las resoluciones de algunos de los problemas encontrados durante la programación de la escena: por ejemplo, la inclusión de una fuente de luz en tiempo de ejecución en el escenario no produce resultados si no se ordena la actualización de cada uno de los materiales de los objetos en el mismo. El proceso de aprendizaje, a partir de un cierto grado de comprensión en las particularidades del lenguaje y librerías, estuvo estrechamente relacionado con la finalización de la cuarta meta: la descripción de los objetos 3D que a representar visualmente.

El segundo objetivo requirió una cantidad de tiempo considerable en comparación con el primero, ya que, aunque a lo largo del Grado se han planteado asignaturas troncales y optativas con contenido referente a desarrollos web, no habían alcanzado una utilidad más que anecdótica ante una problemática real. El temario de estas asignaturas se encontraba más centralizado en tecnologías de Java como los servlets, que no suelen ser la decisión de diseño tomada en este tipo de proyectos. Por el lado positivo, el conocimiento previo adquirido

apoyó el inicio del aprendizaje de las tecnologías AJAX. En el caso de PHP, había contado con una experiencia básica en las Prácticas Externas del Grado, por lo que la quinta meta, relacionada con programación en el servidor, acabó siendo en cierto grado menos compleja de abordar de lo que podría haber resultado.

Para el tercer punto, visité con mi tutor el SCBI de la Universidad, en el Parque Tecnológico de Andalucía. Nos recibieron dos trabajadores de la administración del cluster de supercomputación, en una visita realmente instructiva en la que se plantearon numerosas ideas para éste y futuros Trabajos de Fin de Grado que complementen el desarrollo. Se tomaron numerosas fotografías y un breve clip de vídeo para tener como referencia a la hora de la descripción básica de la escena. Algunas de estas imágenes fueron necesarias para, una vez procesadas, utilizarlas como texturas en los modelos 3D. También se intercambiaron algunos correos con los mencionados trabajadores para conseguir añadir más funcionalidad a esta etapa preliminar de la aplicación según sus necesidades.

3.2 Arquitectura de la aplicación

La arquitectura planteada para el programa consta de un servidor web como núcleo, que se encargará de representar la escena 3D de forma que sea accesible incluso por parte de usuarios no experimentados. El servidor deberá tomar una descripción textual del sistema de ficheros al que tiene acceso e interpretar la estructura definida para recrear el cluster de supercomputación de la sala descrita.

Algunos de los ficheros relativos a los metadatos útiles de cada estante serán modificables manualmente, de cara a aumentar o reducir la información representable al usuario que acceda al entorno 3D. Esta decisión se toma debido a comentarios de los responsables de la administración y soporte del SCBI, que insistieron en la utilidad de un sistema en el que pudieran fácilmente modificar dichos parámetros mediante un procesador de texto básico.

Otra interfaz web se encargará de comunicar al administrador de los metadatos de que sus cambios han tenido efecto y generará nuevos ficheros textuales en base a dicha información, que se almacenará en una base de datos actualizada.

El servidor web a utilizar estará contenido en el paquete distribuido gratuitamente XAMPP, dada la facilidad de su instalación y condición multiplataforma con validez en Windows, Linux y Mac OS X. Se hará uso de tres de los módulos la distribución:

- Apache, para mantener el servidor web con el sistema de ficheros.
- MySQL, que contendrá la base de datos desde la que se generarán los ficheros descriptivos de la sala.
- PHP, que accederá a la base de datos y manipulará su contenido a las órdenes del administrador, además de gestionar el sistema de ficheros a representar.

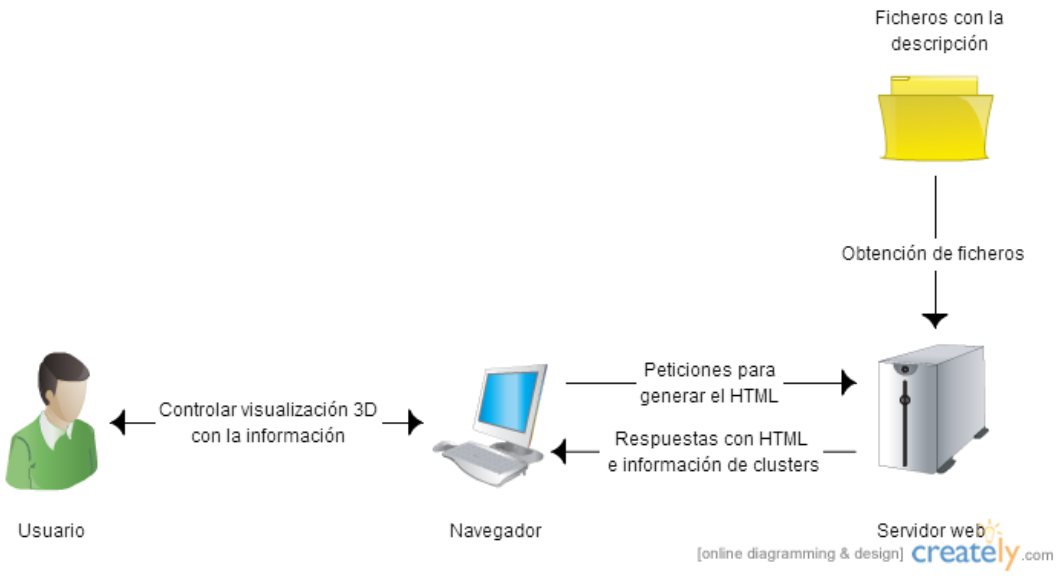


Figura 3.1: Un usuario accede a la visualización 3D

El proceso típico desde la perspectiva del usuario, tal como queda descrito en la figura 3.1, sería el siguiente:

- El usuario entra en la página web con la representación.
- El servidor Apache acepta la conexión, y el código JavaScript de las librerías incluidas en el HTML comienza a realizar las peticiones HTTP pertinentes al código PHP en el lado del servidor.
- El código PHP accede a los ficheros almacenados con la descripción textual que cada función JavaScript le ha pedido.
- Se genera completamente la página HTML en el lado del cliente con la información recibida.
- El usuario visualiza la sala y controla la cámara en la escena. La información de cada rack de computadores está disponible en todo momento en la representación.

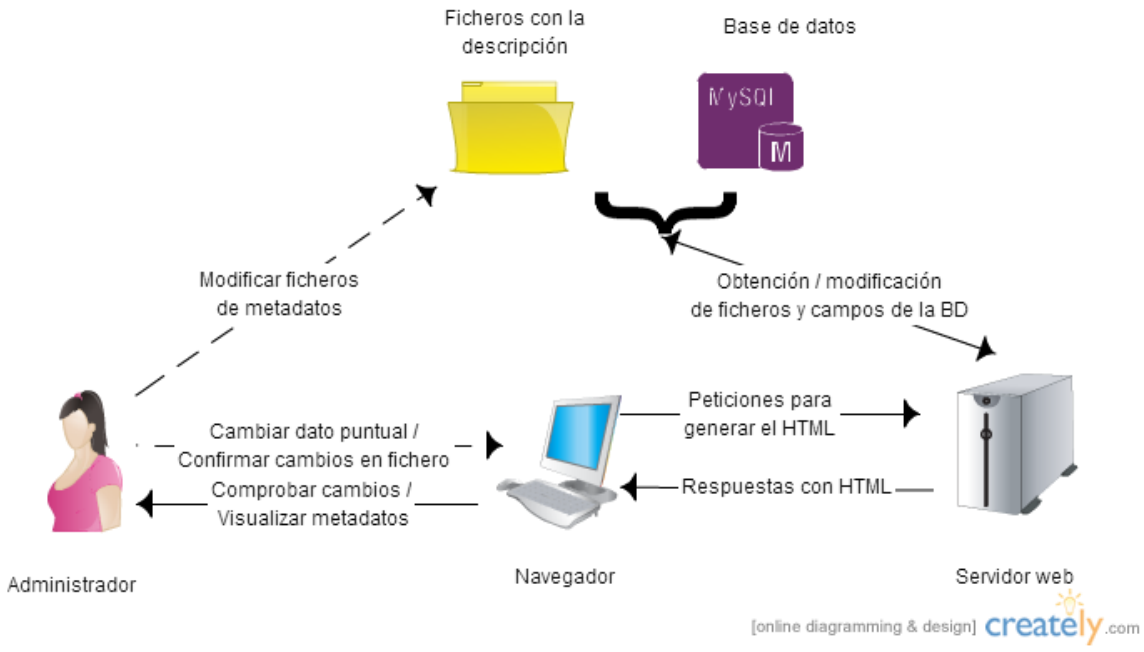


Figura 3.2: El administrador manipula los ficheros

Por otra parte, el rol del administrador se desarrolla de la forma descrita en la figura 3.2:

- El administrador puede editar, añadir o eliminar los ficheros de texto almacenados en el directorio de los metadatos.
- Al acceder a la interfaz web, puede decidir confirmar esos cambios. Al realizar esta acción, producirá:
  - Una petición HTTP al servidor, que leerá los archivos y analizará los cambios en los mismos a partir del estado de los anteriores.
  - La manipulación adecuada de la base de datos desde código PHP (y los propios archivos, de ser necesario).
  - Una respuesta al usuario con los cambios realizados.
- También puede visualizar el listado disponible:
  - Se enviará una o varias peticiones HTTP al servidor con los nombres de las tablas de la base de datos y los identificadores necesarios.
  - El código PHP lee la base de datos y envía lo necesario en un formulario.
  - El administrador puede decidir modificar los campos del formulario y enviar la confirmación mediante una petición POST. El código PHP que la reciba se encargará de las modificaciones y enviará una respuesta HTML orientativa.

3.3 Metodología del desarrollo

Al estar escrita desde cero y dependiente de mis decisiones, la arquitectura de la aplicación no contaba con ninguna restricción o procedimiento concreto más que el del propio desarrollo de gráficos por computador habitual y el desarrollo web a pequeña escala. No era necesario plantear ningún tipo de framework adicional a three.js ni convención para el código más allá de mi propio criterio, por lo que fui añadiendo distintas funcionalidades a medida que iba conociendo las librerías disponibles o planteando las mejoras que consideraba más interesantes.

El principal componente de la arquitectura, la representación WebGL, se estableció al comienzo del Trabajo en una serie de reuniones con el tutor y la visita al SCBI a lo largo del último cuatrimestre del Grado. Al resultar un producto basado en la visualización, resultaba más evidente que en otras situaciones lo necesario para concluir este apartado.

Mientras se concluían el resto de las asignaturas del Grado y preparaba los estudios de mi certificación de idioma necesaria para la obtención del Título, comencé el estudio de la conjunción WebGL-three.js, además de las otras tecnologías necesarias para el desarrollo del Trabajo. Este hecho supuso que tanto la creación de las aplicaciones como la elaboración de esta memoria sucedieran bajo una mayor presión relativa al plazo de defensa del Trabajo en el mes de diciembre.

La recreación de la escena resultó un reto durante las siguientes semanas de desarrollo, ya que se presentaron algunos problemas imprevistos que ralentizaron el avance, propios de mi inexperiencia con el lenguaje JavaScript. Una vez hube alcanzado un estado de la representación que podía considerar cercano a la solución final, volví a contactar con el tutor, que dio el visto bueno a la aplicación y me orientó en el desarrollo del programa para la modificación de los metadatos.

Este segundo componente tomó mucho más tiempo del esperado y se prolongó durante prácticamente un mes adicional, ya que el paradigma de la programación web en las perspectivas del cliente y el servidor no han sido objeto de estudio académico exhaustivo a lo largo del Grado. Fue necesario tomar distintas aproximaciones para plantear los cambios en la información de forma que se adaptase a las orientaciones del tutor. Finalmente, se remodelaron las estructuras de datos planteadas inicialmente para adaptarlas a los cambios definitivos.

Cabe destacar que, si bien la información presentada a los usuarios (incluyendo los metadatos de los objetos de la escena) en las interfaces ha sido redactada en castellano de cara al usuario final, la nomenclatura de variables, funciones y los comentarios pertinentes del código se ha planteado en inglés.

3.4 La descripción textual

Al comienzo del Trabajo, planteamos aproximadamente lo que sería la información relativa al cluster de supercomputación que esperábamos poder aportar visualmente. Estos parámetros deberían ser de fácil edición y podrían resumirse en:

- Posición de cada estante.
- Lugar preciso ocupado por los dispositivos en los racks.
- Información adicional de estos elementos, accesible mediante la interfaz gráfica.

La librería three.js proporciona una especificación funcional de más alto nivel sobre OpenGL con la orientación a objetos propia del lenguaje JavaScript. Es por esto que la decisión a tomar fue la de mantener un listado de objetos JavaScript con la información de cada elemento de la escena, asociada a los propios objetos 3D de three.js.

3.4.1 Primera aproximación a la descripción: XML

En una primera instancia, y dado el carácter orientado a objetos de three.js para representar los nodos del grafo de la escena 3D, se planteó la descripción de la sala y los elementos de la misma como una serie de ficheros relacionados entre sí. En estos archivos se incluirían los parámetros fácilmente editables por parte del personal administrativo y serían enviados a la aplicación por medio de las habituales peticiones XMLHTTP que JavaScript facilita.

La decisión, así, fue la de almacenar los datos en una estructura de ficheros XML, que sería leída en el lado del cliente a consecuencia de las llamadas AJAX necesarias.

La claridad de la notación etiquetada de este lenguaje de marcado permite una cercanía al usuario evidente, por lo que parecía la opción idónea. Sin embargo, se encontraron algunos problemas a la hora de realizar la lectura de los ficheros.

No encontré ninguna librería que facilitara el uso de XML en JavaScript más allá de la propia implementación nativa del lenguaje, algo tosca y que requiere varias líneas de código para acceder, por ejemplo, al valor de un nodo XML. Tras un cierto tiempo, no llegué a averiguar métodos que no requiriesen una nueva curva de aprendizaje que añadir a las ya enfrentadas. También era mi interés mantener la aplicación lo más ligera y eficiente posible, ya que la elección del propio renderizador gráfico orientó el proyecto a mantener un rendimiento aceptable sin necesidad de añadidos ni complementos instalables.

Así, preparé una función que realizaría el análisis sintáctico de cada fichero XML y lo convertiría en un objeto JavaScript: las etiquetas contenidas serían propiedades del objeto, y su contenido, objetos hijos en la estructura, formando un árbol fácilmente accesible. Esta no

fue una labor sencilla ni breve, dada la complejidad mencionada de la implementación de las funciones para el acceso a cada valor almacenado. Resultaba muy complicado, por ejemplo, obtener nodos anidados o con iguales etiquetas, pero distintos identificadores únicos. Esto podía llegar a inutilizar el sistema, ya que la jerarquía descrita

Lo complicado del desarrollo de la función llevó a un segundo problema: una vez puesto en marcha, la aplicación funcionaría sin problemas, pero se había ido adecuando gradualmente a la descripción del XML creado inicialmente. Cuando se hubo completado la fase de desarrollo en que existía una base de datos desde la que se generarían los ficheros XML, eran necesarios aún más cambios y no se estaba asegurando que pudieran aparecer más fallos, por lo que decidí abandonar el planteamiento.

3.4.2 Segunda aproximación a la descripción: JSON

Siendo una tecnología que no había usado previamente, no llegué a plantearme el uso de lo que resultó ser una decisión muy adecuada para la descripción: el lenguaje de descripción de objetos JSON. Con su funcionalidad nativa en JavaScript, permite almacenar y obtener objetos de este lenguaje dentro de ficheros de texto de fácil legibilidad, de forma mucho más eficiente que el sistema de etiquetado de XML. Estos ficheros de texto podían ser generados desde la aplicación PHP con igual o mayor facilidad que los que estaban siendo planteados como XML, estando la posterior lectura totalmente cubierta por la función JSON.parse disponible en JavaScript. La sustitución fue prácticamente inmediata y, tras realizar los cambios oportunos en el código, la aplicación funciona sin ningún tipo de traba y con una escalabilidad asegurada.

3.5 La representación WebGL

Como se ha explicado en el anterior capítulo, uno de los aspectos llamativos de WebGL es que permite la inclusión de programación gráfica avanzada en elementos <script>. El uso de la librería three.js facilita esta labor, no requiriéndose la programación de shaders ni otros elementos de bajo nivel para representar un entorno virtual. En esta sección se entrará en detalle en la implementación del código de la visualización, núcleo del proyecto.

3.5.1 Inicialización de renderizador, escena y cámara

Antes de comenzar el dibujado en el elemento asignado, el motor gráfico de WebGL precisa de varios elementos antes de poder recrear la escena:

- Un renderizador.
- Un elemento cámara.

- Un grafo de escena definido.

Una vez creado el renderizador THREE.WebGLRenderer, se le deben proporcionar parámetros relativos al tamaño del lienzo sobre el que dibujará la escena. En el caso de la aplicación desarrollada, era de nuestro interés que ocupase la totalidad de la ventana disponible, por lo que son los valores window.innerWidth y window.innerHeight del DOM los aportados. La estructura vacía del grafo de escena se crea mediante la construcción de un objeto THREE.Scene, que posteriormente se asociará con el renderizador.

La creación de la cámara requiere una primera decisión de diseño: el uso de una proyección ortográfica mediante THREE.OrthographicCamera o una proyección con perspectiva usando THREE.PerspectiveCamera.

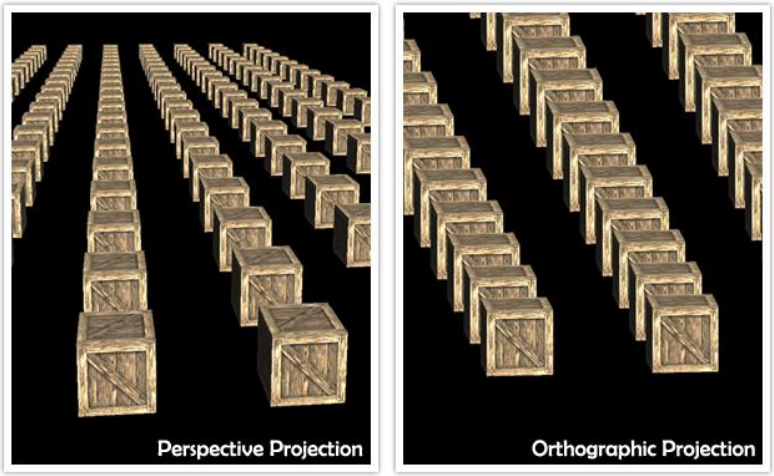


Figura 3.3: Tipos de proyección en three.js
Fuente: https://github.com/MasDennis/Rajawali/wiki

La figura 3.3 representa ambos casos. Como se puede observar, el resultado usando la variante PerspectiveCamera produce un resultado más natural que la proyección ortográfica. Esto se debe a la implementación del motor gráfico del frustum de renderizado.

El frustum de cámara es una pirámide truncada situada en la escena, que toma como bases dos planos: uno delimita los elementos más alejados de la posición de la cámara que se van a renderizar, mientras que el otro realiza una separación similar con aquellos más cerca espacialmente. Todo elemento fuera de este volumen no es renderizado en la escena, aunque se encuentre en la estructura y su estado en el grafo de escena pueda estar cambiando. En el caso de la proyección ortográfica, ambas bases de la pirámide son idénticas, por lo que el frustum pasa a ser un paralelepípedo recto que no realiza ninguna transformación a la imagen. Son los planos laterales del frustum en perspectiva los que añaden la sensación más natural del punto de vista humano a la escena.

Una vez decidido el uso de la cámara de perspectiva, se le asignan los parámetros de

FOV (Field of View, ángulo de visión de la cámara), el Aspect Ratio (la relación de aspecto del lienzo: cociente entre la anchura y la altura de la ventana en que se dibuja) y las distancias a la que situar cada base del frustum.

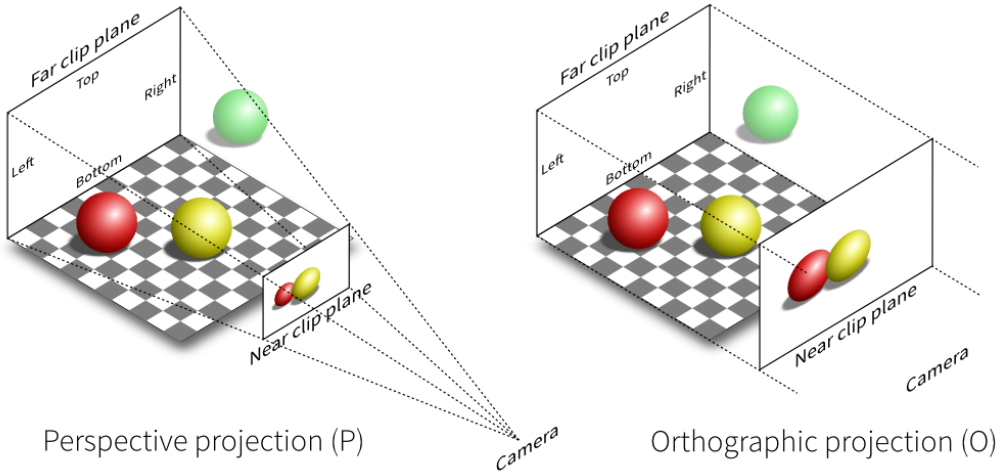


Figura 3.4: Diferencias en los frustums de proyección
Fuente: http://www.labri.fr/perso/nrougier/teaching/opengl/

La función initializeSceneParameters se encarga de crear los elementos necesarios para la creación de la escena, además de añadir una iluminación ambiental que permita ver los objetos incluidos. El elemento del DOM generado por el renderizador se asocia con el correspondiente en la página HTML, cuya variable se ha llamado *container*. Se incluirán posteriormente otros elementos relacionados con la interacción del usuario, que serán descritos más adelante.

```
function initializeSceneParameters() {
   camera = new THREE.PerspectiveCamera(
     70, window.innerWidth / window.innerHeight, 0.01, 10000);
   scene = new THREE.Scene();
   ambientlight = new THREE.AmbientLight(0xbbbbbb);
   scene.add(ambientlight);
   renderer = new THREE.WebGLRenderer();
   renderer.setSize(window.innerWidth, window.innerHeight);
}
```

3.5.2 El bucle de animación

El conjunto de funciones que define la escena inicial se ha contenido en el trabajo en el fichero JavaScript mainscene.js. La función mainScene, contenida en tal fichero, establece los parámetros y objetos que preparan el lienzo, además del control por ratón (o táctil en caso de dispositivos compatibles) y, en última instancia, puebla de objetos el grafo de escena.

Tras la ejecución de esta función, tiene lugar una llamada a la instrucción animate, embebida en el documento HTML. La creación de esta función tiene su origen en el desarrollo de motores gráficos: habitualmente, el dibujado de escenas virtuales se representa como una función recursiva o una iteración ininterrumpida. Los pasos que sigue este bucle se pueden simplificar en la escena que se pretende mostrar como los especificados en la Figura 3.5.

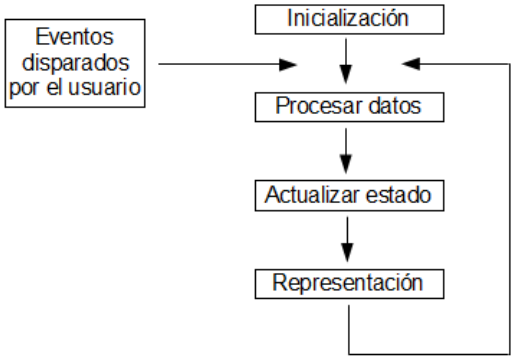


Figura 3.5: Bucle de animación

Una vez inicializado el proceso, los datos de las posibles entradas al programa (en nuestro caso, eventos de control disparados por el usuario de la aplicación), además de los contenidos en la escena, siguen el proceso de three.js. Se obtiene la posición de cada elemento en la escena, se efectúan las transformaciones de la imagen resultante según la cámara y los shaders incluidos y se *rasteriza* la imagen en pantalla. Este procedimiento se facilita gracias a la función requestAnimationFrame, que se asegura de mantener una ejecución estable y con un tope de 60 fotogramas renderizados por segundo para evitar una sobreutilización del hardware de aceleración gráfica.

```
function animate() {
    controls.update();
    requestAnimationFrame(animate);
    renderer.render(scene, camera);
}
```

Una primera aproximación a la función animate actualiza los valores del sistema de control, ejecuta la petición del fotograma y pide al renderizador WebGL que utilice la escena y cámara determinada. Cabe destacar que el párámetro necesario para la ejecución de requestAnimationFrame es la propia función animate, que se ejecutará de nuevo al concluir, manteniendo el bucle de animación.

3.5.3 Recreación de la escena

Una vez dispuesta la escena, comienza el procedimiento de creación de los objetos

que la poblarán. Para tal fin entran en juego tres funciones: getRoomInformation, generateRacks y generateScenario. A modo de convención, la creación de variables que almacenen objetos 3D o punteros a los mismos inclurián *object* al final de su nombre, para distinguirlos de aquellos que contengan los metadatos u otra información. Con esto en mente, en la recreación de la escena intervendrán tres arrays llamados *rackObjects*, *deviceObjects* y *scenarioObjects*, además de dos objetos comunes de JavaScript: *room* y *racks*.

La función getRoomInformation realiza la petición XMLHttpRequest al servidor, que le proporcionará el fichero .JSON especificado con la información descrita textualmente de la sala. En esta función se asegura que el documento recibido sea *parseado* desde el analizador de JSON nativo a JavaScript y se almacenen todos los datos contenidos como pares clave-valor en la variable room.

```
function getRoomInformation() {
   file = "/json/sala/1.json";
   xmlhttp = new XMLHttpRequest();
   xmlhttp.onreadystatechange = function () {
      if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
         room = JSON.parse(xmlhttp.responseText);
         room = room.sala[1];
      }
   };
   xmlhttp.open("GET", file, false);
   xmlhttp.send();
}
```

Una vez obtenida la variable *room*, comienzan su ejecución las funciones generateRacks y generateScenario, que añadirán los objetos de cada estante y los elementos que conforman la representación de la sala, respectivamente.

La función generateScenario crea una serie de geometrías planas utilizando las dimensiones especificadas de la sala en la descripción, además de añadir las texturas descritas. Cada elemento es añadido al array scenarioObjects para un acceso más sencillo que el de la búsqueda en el grafo de escena.

En el caso de generateRacks, se itera sobre el conjunto de las claves bajo la propiedad rack de la variable room, obtenida en la descripción JSON. Una nueva variable racks almacena punteros a estos estantes para una mayor facilidad de acceso a los datos y algunos de los nombres de los metadatos son modificados para mantener una consistencia idiomática en el código. Posteriormente, se crea una serie de geometrías THREE.BoxGeometry (paralelepípedo rectángulo) adecuadas a las dimensiones del rack, descritas según el tamaño de las unidades rack (puntos de anclaje disponibles en el estante de tamaño estándar) y el número de las mismas. Los objetos se disponen en el espacio en el punto descrito desde la función createRack y el estante completo se gira sobre su eje vertical

en la dirección indicada en la descripción textual.

```
      var geomH = new THREE.BoxGeometry(rack_width, rack_thickness,
rack_depth);
      var geomV = new THREE.BoxGeometry(rack_thickness, rack_height +
                  rack_thickness, rack_depth);
      var geomD = new THREE.BoxGeometry(rack.unit_size.w / 2 +
rack_thickness,         rack.units * rack.unit_size.h, rack_thickness / 4);
      var materialRack = new THREE.MeshLambertMaterial(
            {map: new THREE.ImageUtils.loadTexture("/textures/rack.png")});
      var materialDoor = new THREE.MeshLambertMaterial(
            {map: new THREE.ImageUtils.loadTexture("/textures/mesh.png")});
      createRack(rack, scene, {h: geomH, v: geomV, d: geomD},
             {r: materialRack, d: materialDoor});
```

La generación del rack incluye la llamada a la función rackInterior, que itera sobre los valores almacenados de los dispositivos del estante sobre el que actúa. Se crea así una BoxGeometry para cada elemento del cluster de computación, utilizando un material que cuenta con las texturas necesarias para representar eficazmente la máquina como es en la realidad. En el caso de resultar huecos de unidades no utilizadas en el rack, se añaden elementos decorativos que hacen que se mantenga un aspecto consistente. Estos elementos no añaden información, por lo que no serán almacenados en las variables de apoyo.

Figura 3.6: Representación de los elementos descritos

3.5.4 El modelo de control

La API de three.js cuenta con el apoyo de una serie de pequeñas librerías adicionales a incluir en los proyectos web para resolver distintas problemáticas comunes en la representación 3D. Una de ellas es TrackballControls.js, que permite una parametrización del sistema de cámara de three.js para su uso mediante ratón (e incluso pantallas táctiles en dispositivos móviles, con ciertas limitaciones).

La librería se encarga de ajustar el movimiento de la cámara en términos como la velocidad del mismo o de la rotación, el vector de dirección al que apunta, el coeficiente de zoom al usar la rueda del ratón o incluso métodos para evitar que la cámara se descontrole.

En una aplicación 3D, el control de la cámara resulta crucial en el resultado de una experiencia satisfactoria: debe adaptarse a movimientos que puedan poner en una situación comprometida a la cámara e inutilice la aplicación. Con esto en mente, se estableció en el trabajo un sistema de cambios graduales de la posición de la cámara y del objeto *target* que contiene el punto en el espacio al que la cámara se dirige. Como se cuenta en la escena con una serie de elementos identificables únicamente (racks y dispositivos), el curso a seguir resultó el de establecer un modelo de control basado en clicks en los mismos que centraran la cámara para recibir la información pertinente.

Para ello, la librería TWEENJS proporciona una solución muy satisfactoria. Al crearse la escena y la cámara asociada a la misma, también lo hacen dos variables TWEEN.Tween: cameraPositionTween y targetTween. Estas variables harán las veces de intermediario cada vez que se precise un movimiento de cámara fijado hacia un elemento concreto. En el fichero mousecontrol.js se establece la reacción del sistema a los eventos de ratón que hará uso de los dos elementos Tween.

Un click de puntero en la aplicación se recoge en un evento como un vector de dos dimensiones ya que, evidentemente, el navegador comprende la página como un elemento plano. Es la labor del programador en este caso el generar el vector de tres dimensiones que resulta realmente al señalar la escena 3D.

```
        var mouse3D = new THREE.Vector3(
                        (event.clientX / container.offsetWidth) * 2 - 1,
//x
                        -(event.clientY / container.offsetHeight) * 2 + 1,
//y
                        0.5); //z
```

La variable event contiene la posición en *X* e *Y* del ratón en el momento de dispararse el evento con el presionado del botón. Las componentes del nuevo vector se toman en base al elemento en que se dibuja la escena, añadiendo el valor *z* de 0.5 por convención para evitar un vector con una longitud excesiva que causaría problemas en el proceso.

```
projector.unprojectVector(mouse3D, camera);
mouse3D.sub(camera.position);
mouse3D.normalize();
var raycaster = new THREE.Raycaster(camera.position, mouse3D);
intersects = raycaster.intersectObjects(rackObjects, true);
```

El objeto THREE.Projector realiza la *desproyección* del vector: el proceso inverso a la proyección de un punto 3D en 2D. Se obtiene así el vector, que tiene en cuenta la posición de la cámara y es normalizado para mayor facilidad de los cálculos. Después entra en juego el objeto THREE.Raycaster, que proyecta un rayo invisible en la escena desde la posición de la cámara en la dirección dada por el vector. Este rayo pasa por todos y cada uno de los objetos de la escena en su dirección y almacena información sobre el lugar en que ha sucedido el contacto. Gracias al método intersectObjets, es posible obtener un array que contenga los objetos 3D que nos interesan que han sido señalados; se evita así, por ejemplo, la detección de clicks en elementos decorativos o del propio escenario que no contienen información relevante para el control a implementar.

A partir de este punto, la función clickHandler, encargada de manejar la información de cada click, determina su curso de acción dependiendo de la clase del objeto que ha recibido el rayo en primer lugar: si se trata de un rack o una unidad vacía del mismo, se colocará una luz indicadora parpadeante bajo el mismo y la cámara se centrará en él. Si, por contra, es un dispositivo interno, la cámara realizará un desplazamiento para situarse en un punto de anclaje definido frente a él.

Los puntos de anclaje dependen de la situación de la cámara en el momento del click:

- Si se señala un rack y se encuentra a una altura considerable, se mantendrá por encima del suelo a la altura definida de la sala. En caso de suceder un segundo click en el mismo rack, se acercará para obtener una vista frontal si la cámara se situaba delante del rack, o trasera si se encontraba al otro lado.

- Por otra parte, entendiendo que el acceso a dispositivos individuales requiere de mayor velocidad en el control, si se hace click en un componente, automáticamente se realiza el traslado de la cámara al punto de anclaje.

El apuntado a dispositivos internos de los racks está apoyado por la función mouseIsOver, que realiza un procedimiento similar al de clickHandler para marcar la máquina sobre la que se encuentra el puntero en ese momento. Se añade a la escena un objeto de igual geometría, pero con una escala ligeramente mayor y un material coloreado que usa el parámetro THREE.BackSide. Esto produce un efecto de rotulado de bordes de el dispositivo que lo marca de forma explícita. La selección de dispositivos está facilitada ignorando las paredes de los estantes al capturar la intersección del rayo.

```
var customMaterial = new THREE.MeshBasicMaterial(
      {color: 0xFF9933, side: THREE.BackSide});
var mouseOverMesh = mouseOver.children[0];
markedGlow = new THREE.Mesh(mouseOverMesh.geometry,
```

```
customMaterial);
            markedGlow.name = "markedGlow";
            markedGlow.position = {
                  x: mouseOver.parent.position.x + mouseOver.position.x,
                  y: mouseOver.parent.position.y + mouseOver.position.y,
                  z: mouseOver.parent.position.z + mouseOver.position.z};
            markedGlow.scale = mouseOverMesh.scale.clone();
            markedGlow.scale.multiplyScalar(1.06);
            scene.add(markedGlow);
```

3.5.5 Mostrando la información disponible

Cuando se realiza la selección de un dispositivo, la función selectClicked llamada por clickHandler ordena a su vez la ejecución de la instrucción displayObject en el fichero displayingInformation.js. Previamente, en la función generateMainScene, se ha utilizado otra de este fichero llamada setInfoDiv, que prepara un contenedor en el documento HTML pero se encuentra vacío hasta que sea necesario.

La función displayObject toma el objeto JavaScript que contiene los datos del dispositivo seleccionado (cuyo puntero se encuentra en el objeto racks) y los representa como un listado en un elemento ajeno al renderizado WebGL. Esto permite el uso de CSS para situarlo en la ventana, sobre la representación 3D, fijando su posición. Se permite así seleccionar el texto en caso de necesitar extraer valores y se abre paso a un gran número de funcionalidades potenciales.

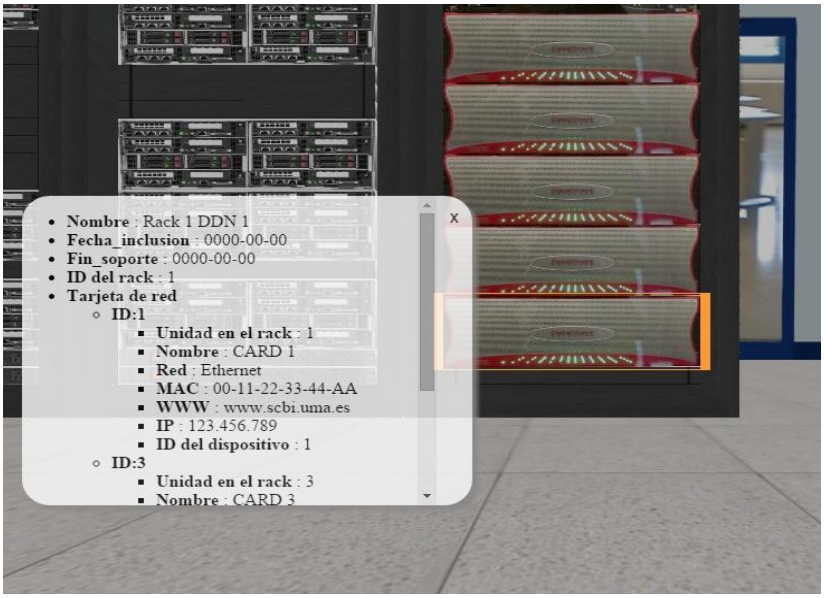


Figura 3.7: representación de información de ejemplo

3.5.6 Interfaz gráfica sencilla

A la hora de ejecutar pequeñas funciones como el mostrado del estadístico de fotogramas por segundo, paralizar el renderizado o reiniciar la cámara, resulta útil disponer de una interfaz gráfica que evite el uso de la consola de comandos del navegador utilizado. Para ello se mantiene en la aplicación un objeto dat.GUI, una pequeña interfaz capaz de ejecutar código JavaScript enviándole parámetros mediante botones, sliders, cuadros de texto y casillas. Dat.GUI es una librería JavaScript disponible en Google Code (anteriormente en GitHub) como uno de sus *Chrome Experiments*. El fichero guibuttons.js contiene las funciones necesarias para, una vez declarado el objeto dat.GUI, representar un panel extensible con los botones que se le asignen mediante el método gui.add().

3.6 Gestión de los datos

3.6.1 Enfoque inicial: XML

El almacenamiento de la información asociada a la sala y al cluster de supercomputación se planteó inicialmente como un sistema de ficheros XML enlazados entre sí. La solución fue descartada ante sucesivos problemas con el analizador sintáctico programado para la obtención de los datos por la aplicación, además del planteamiento una decisión de diseño que mejoraría la arquitectura: la inclusión de una base de datos.

Antes de abandonar este lenguaje, se estudió el estándar XSLT (EXtensible Stylesheet Language Transformations) como posible solución a la hora de introducir la información de los XML en la base de datos. Es posible utilizar una descripción en XSL que plantee en un documento HTML los contenidos de un archivo XML como un formulario. Esto serviría para modificar puntualmente los datos del XML y, en última instancia, enviar cada campo a la base de datos para su actualización. Esto permitiría almacenar la información en la base de datos en caso de necesitar grandes modificaciones del sistema de ficheros, además de mantener una copia de respaldo de cada valor. Los XML a utilizar se generarían desde una interfaz web que utilizase conexiones con esta base de datos.

Tras experimentar una gran complejidad en el estándar de XSLT, se decidió no continuar con esta línea de desarrollo para la gestión de la información.

3.6.2 Enfoque final: JSON, MySQL y sistema de ficheros

La aproximación tomada finalmente fue la de mantener los datos que anteriormente se encontraban en ficheros XML en nuevos ficheros JSON, mucho más sencillos de manipular (son, esencialmente, descripciones de objetos JavaScript) y eficientes de procesar por el analizador sintáctico. Se mantuvo la idea de la gestión de la base de datos como respaldo de la información, por lo que era necesario el desarrollo de una aplicación con PHP y JavaScript.

Los administradores del cluster de supercomputación del SCBI insistieron en que era

necesario un método para añadir fácilmente metadatos adicionales al sistema, preferiblemente desde ficheros de texto editables. El fichero JSON permite la edición de estos datos sin mayor impacto en el programa que el añadido o retirada de propiedades a los objetos decodificados por el motor de JavaScript, siempre y cuando no se modifiquen los valores de los que se hace uso en el código. Aún así, resulta engorroso añadir un mismo campo a cada elemento, ya que la ramificación de los ficheros en presencia de una mayor estructuración de la información lo va convirtiendo gradualmente en algo menos legible a simple vista.

Es por esto que se decidió mantener el uso de la base de datos, desde la cual se generarían los ficheros JSON necesarios para la ejecución correcta de la aplicación WebGL. Sin embargo, pretende facilitarse su uso mediante una aplicación en el lado del servidor con distintas funciones:

- Acceder a la base de datos para generar los ficheros JSON.
- Mostrar un formulario con los campos disponibles en la base de datos para modificarlos puntualmente o añadir elementos nuevos.
- Crear nuevas tablas o columnas a la base de datos, relacionadas entre sí.

Para mantener en el servidor un listado de los metadatos, se utiliza un conjunto de ficheros almacenado en el directorio *metadata*. En esta carpeta se encuentra también el directorio *old*, que mantiene los archivos con los metadatos de la anterior edición, en caso de necesitarse como respaldo y para realizar la comparación con los archivos a manipular.

Cada fichero txt en la carpeta *metadata* supone el añadido de una tabla adicional a la base de datos en caso de no encontrarse, tomando como nombre de la tabla el del propio fichero. Dentro de estos ficheros se incluye una lista de elementos que se corresponderá con las distintas columnas a mantener en la base de datos, ateniéndose a una descripción según el primer carácter de la línea:

- Un signo “>” supondrá una columna con el nombre especificado.
- Un signo “+” creará una relación *uno-a-muchos* con la tabla descrita.
- Un signo “-” planteará una relación *uno-a-uno* con otra tabla con el nombre indicado. Se podrá utilizar como una relación inversa a la descrita con el signo "+" (*muchos-a-uno)* para mayor facilidad de definición de campos.

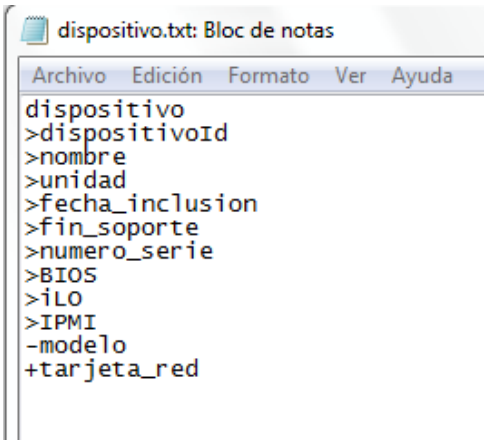


Figura 3.8: Uno de los ficheros de metadatos

Este diseño implica el uso de identificadores enteros autoincrementales para cada fila de las tablas, que serán los que se utilicen en los ficheros JSON y, definitivamente, en la aplicación WebGL. Los campos son declarados en MySQL como de tipo VARCHAR de 500 caracteres, lo que especifica este número como longitud máxima, mientras que sólo se ocupará en disco el espacio necesario para almacenar los caracteres escritos.

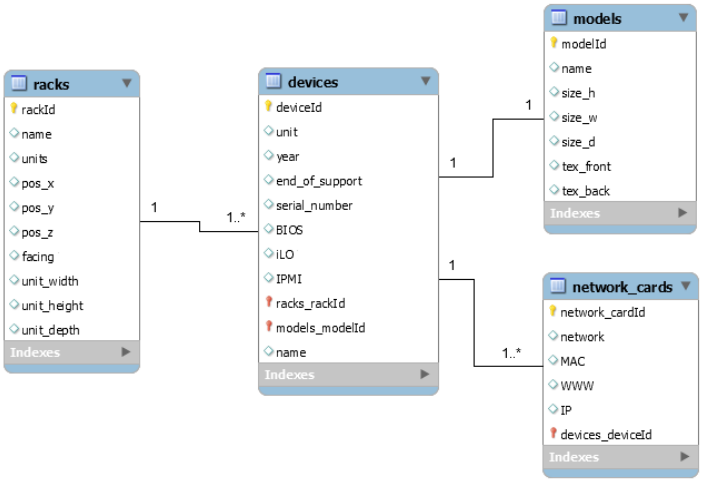


Figura 3.9: Ejemplo de descripción del contenido de la sala

En el ejemplo de la figura 3.8, se creará la tabla *dispositivo* con los campos indicados, además de una referencia mediante un identificador a otra tabla *modelo* (que será creada de no existir) en una referencia *uno-a-uno*. En una tabla *tarjeta_red* se incluirá una columna para contener identificadores de *dispositivo*, creando una relación *uno-a-muchos* desde la tabla *dispositivo*. La figura 3.9 representa de forma sencilla las tablas y relaciones de un posible ejemplo.

3.6.3 La aplicación de gestión

La aplicación consiste en una página web que centraliza, mediante botones y selectores, la gestión de los datos almacenados. Para ello se hace uso de una serie de funciones JavaScript incluidas en el fichero database.js que realizan las peticiones AJAX respectivas a código PHP.

Figura 3.10: Menú de gestión de los datos

El botón *Actualizar metadatos desde ficheros* realiza la secuencia de pasos descrita sobre cada fichero del directorio *metadata* desde la función updateMetaData.php, que se apoya en una serie de instrucciones contenidas en updateMetaFunctions.php para tal fin. Al finalizar las modificaciones necesarias en la base de datos y el sistema de ficheros, se muestra por pantalla el estado de los ambos antes y después de la ejecución.

El selector, codificado en el archivo getSelectorFromTable.php y asociado a las tablas en la base de datos, permite el añadido de filas con el botón *añadir*, a la vez que representa cada elemento por su identificador de tabla y su nombre en la lista. La selección de un elemento representa un formulario en la página con los campos almacenados, permitiendo la modificación de los mismos o la eliminación del cada elemento. En la figura 3.11 se puede comprobar que este formulario se estructura como un árbol en el que cada elemento añade un selector adicional para cada tabla con relación *uno-a-muchos* en los metadatos, mientras que se representa la relación *uno-a-uno* directamente. Tras la modificación, se informa al administrador de los datos de las instrucciones enviadas a la base de datos y se vuelve a la página principal. Como MySQL no permite comprobar la existencia de columnas antes de intentar crear una, se utiliza un procedimiento almacenado llamado AddColumnUnlessExists en lugar de la creación habitual.

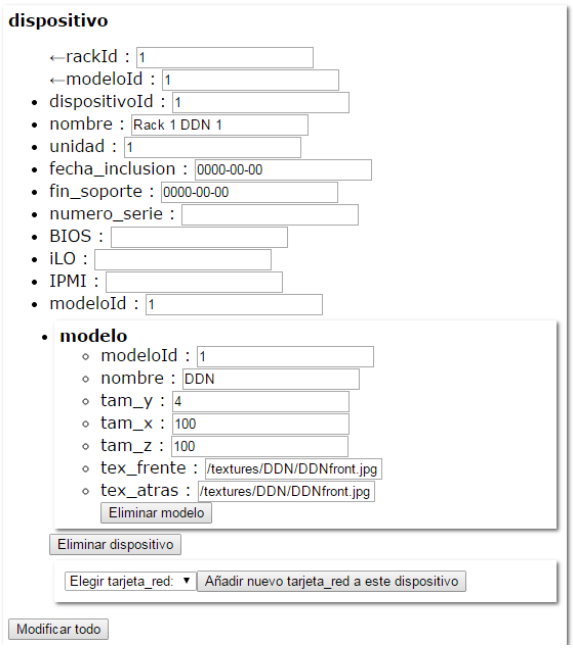


Figura 3.11: Ejemplo de formulario de modificación de datos

El botón *Generar JSON* llama al fichero getJSON.php, que toma de forma similar los datos, codificándolos en un grupo de ficheros en la carpeta *json*. Se da soporte, dada la estructura, a distintas salas posibles, por lo que la representación actual genera un directorio *sala* que cuenta con un fichero con la descripción de la misma y de los elementos en ella. Se utiliza un procedimiento semejante en el caso de las tablas de relación *uno-a-uno*, para evitar una repetición en los datos almacenados de, por ejemplo, numerosos dispositivos con el mismo modelo.

Capítulo 4 – Conclusiones y trabajos futuros

En este capítulo se exponen las conclusiones a las que se ha llegado con el proyecto, así como posibles expansiones en la funcionalidad del mismo.

4.1 Conclusiones

El desarrollo del trabajo ha supuesto un reto mucho mayor al esperado en un principio. Esencialmente, el motivo principal de las dificultades ha sido mi escasa experiencia previa con los lenguajes utilizados en particular y con el desarrollo web en general. Este hecho se sumó a las restricciones de tiempo para la defensa del trabajo, al que no pude dedicar todo mi tiempo hasta una vez concluidos el resto de estudios del Grado. Subestimé la complejidad del desarrollo de las estructuras de almacenamiento y gestión de los datos sobre los que se fundamente la aplicación, por lo que el resultado presenta un acabado más tosco del que se había planteado originalmente. En este aspecto, el trabajo ha supuesto una importante lección sobre la planificación de procesos propia de un ingeniero.

A nivel académico, tanto el desarrollo de las aplicaciones como la elaboración de la memoria han servido muy notablemente para afianzar y entrar en profundidad en unos conocimientos que, por limitaciones de tiempo, no son posibles de aportar eficazmente en las asignaturas del Grado. Así, he acabado familiarizándome con tecnologías que no había tenido la ocasión de utilizar y que están a la orden del día en el mundo laboral como JavaScript, PHP, JSON y, principalmente, WebGL.

Este uso de nuevos paradigmas de programación y el aprendizaje de tecnologías desconocidas no dejó de ser un obstáculo para el proyecto: el desarrollo fue mucho más lento del estimado y, en ocasiones, se dedicó una cantidad de tiempo desproporcionada a la resolución de problemas “menores” que podría haberse destinado a presentar una interfaz más pulida para los usuarios de la aplicación.

El increíble avance tecnológico en el desarrollo web de la última década ha traído consigo una serie de tecnologías que aún están ocultas al público general, bien porque la curva de aprendizaje de las mismas es muy acentuada, o bien porque se trata de sistemas emergentes que aún no han alcanzado cierto grado de popularidad. WebGL, en conjunción con la librería three.js, ambos en constante actualización, se encuentran en el segundo grupo y, en mi opinión, no tardarán en postularse como elementos de uso frecuente en las páginas web que apuesten por una visualización elegante de gráficos 3D.

A nivel personal, siempre me he sentido fascinado por la dirección que ha tomado el desarrollo de sistemas multimedia. Hace poco más de diez años, habría resultado imposible plantearse la explosión tecnológica y comercial de los actuales smartphones y tabletas. El hecho de poder ejecutar la aplicación de este trabajo (con las trabas propias del dispositivo a utilizar) en un dispositivo de bolsillo, no hace más que aumentar mis expectativas de cara al

futuro.

La capacidad de representar tridimensionalmente un problema real como es el de la administración de sistemas complejos, aportando una valiosa información al personal encargado del mantenimiento, resulta un concepto mucho más ambicioso y atractivo que el que podría haber esperado de mis estudios académicos. Si bien las distintas restricciones de tiempo y conocimientos han limitado el resultado final, mi valoración general del trabajo es positiva y una buena base para el desarrollo de un proyecto aún mayor. Debo agradecer personalmente a mi tutor el hecho de haberme ofrecido este primer paso en un sistema pionero con una utilidad tan reseñable y que demuestra, una vez más, que "3D" no tiene que ser siempre sinónimo de entretenimiento pasajero.

4.2 Trabajos futuros

Como se mencionó al principio de esta memoria, este proyecto supone un punto de inicio para el desarrollo de una línea de Trabajos Fin de Grado que complete la aplicación para la monitorización en tres dimensiones de centros de cómputo como el SCBI de la Universidad de Málaga. Para ello, se plantean ciertas funcionalidades adicionales.

El principal añadido visual sería la implementación de un sistema de capas en el lienzo de la representación que permita el monitorizado en tiempo real o en un histórico de distintos parámetros tales como la temperatura en los dispositivos, estado de la red de interconexiones del cluster o porcentaje de acceso a los recursos del sistema en cada máquina. Se permitiría así comprobar no sólo el estado de salud o rendimiento general del cluster, sino el impacto que suponen los distintos programas que se ejecutan en los mismos, lo que podría orientar en caso de errores o de modificaciones necesarias en las instrucciones del código. A modo de prueba de los posibles resultados de este sistema, se ha incluido en el trabajo un método de generación aleatoria de temperaturas medias a cada dispositivo incluido en la escena. Estos valores son representados visualmente por medio de bloques semi-transparentes que rodean a los dispositivos y que muestran distintos colores atendiendo a lo elevada que esté la temperatura. El resultado llama la atención a las zonas que alcancen valores críticos mediante tonos rojizos, haciéndola un buen sistema de control del rendimiento de la refrigeración de la sala con un simple vistazo.

Para la creación de este sistema de capas, es necesario desarrollar un canal de información y un sistema de almacenamiento del histórico preparado para la cantidad masiva de datos que las sondas software y los sensores físicos proporcionan. El rumbo claro a seguir implica el desarrollo de una base de datos segura y capaz de atender las peticiones de una aplicación que genere la información relevante en intervalos a especificar. También serán necesarias las modificaciones pertinentes a la visualización que permita el acceso a este histórico o a la información en tiempo real.

Bibliografía

En este apartado se enumeran las fuentes consultadas para el desarrollo del trabajo, según la parte del mismo en la que se necesitaron, además de los recursos multimedia utilizados. Las imágenes utilizadas como figuras en esta memoria, si no se especifica, son de elaboración propia.

WebGL y three.js

Sitio oficial de OpenGL, con información sobre la implementación:

`https://www.opengl.org/`

Publicaciones consultadas:

WebGL: Beginner's guide. Diego Cantor, Brandon Jones. Packt Publishing, 2012.

Beginning WebGL for HTML5. Brian Danchilla. Apress, 2012

Web oficial del proyecto three.js, con descarga y documentación:

`http://threejs.org/`

Tutoriales de uso y ejemplos de escenas:

`http://www.html5rocks.com/en/tutorials/three/intro/`

`http://mandemeskel.wordpress.com/2013/08/19/`
`mouse-events-raycasting-with-three-js/`

`http://stemkoski.github.io/three.js/`

`http://solutiondesign.com/WebGL-and-three-js-texture-mapping/`

Librerías externas:

`http://www.createjs.com/#!/TWEENJS`

`http://workshop.chromeexperiments.com/examples/gui/#1--Basic-Usage`

`https://threejsdoc.appspot.com/doc/three.js/`
`src.source/extras/controls/TrackballControls.js.HTML`

`https://github.com/mrdoob/stats.js/`

Desarrollo web

Descarga y documentación de XAMPP:
https://www.apachefriends.org/es/index.HTML

Información sobre los lenguajes y documentación:
http://librosweb.es/JavaScript/capitulo_1/breve_historia.HTML
http://www.w3schools.com/js/
http://www.w3schools.com/ajax/
http://www.w3schools.com/PHP
http://PHP.net/
http://www.w3schools.com/JSON
http://www.w3schools.com/xsl/
http://www.w3schools.com/XML/

Consultas para problemas concretos:
http://cryer.co.uk/brian/mysql/howto_add_column_unless_exists.htm
http://stackoverflow.com/questions/
5111856/generating-form-fields-from-XML-tags
http://www.cryer.co.uk/brian/MySQL/
howto_add_column_unless_exists.htm
http://stackoverflow.com/questions/9085839/surprised-
that-global-variable-has-undefined-value-in-JavaScript

Recursos adicionales

Sitio web del SCBI:
http://www.SCBI.uma.es/

Imágenes de dispositivos usadas como textura:
http://www8.hp.com/es/es/home.HTML
http://tf3dm.com/3d-model/server-rack-76768.html

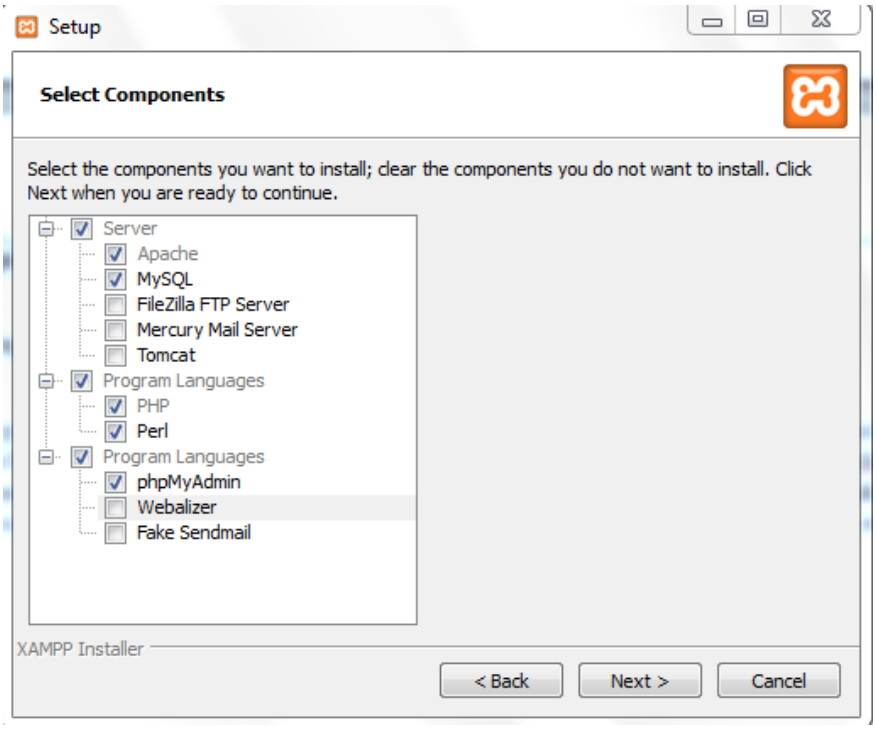
Consultas y algoritmo para representar temperatura en color:
https://gist.github.com/paulkaplan/5184275
http://en.wikipedia.org/wiki/Color_temperature

Creación de los diagramas de uso:
https://creately.com/app/

Apéndice A – Preparación del servidor

La ventaja del uso de lenguaje de scripts, en este caso, radica en que no es necesario compilarlos ni realizar ninguna preparación adicional más que disponerlos en el sistema de ficheros de forma que sean accesibles por el servidor web. Así, la puesta en marcha de este trabajo no requiere más que el uso de un servidor con integración de PHP y una base de datos MySQL. Para ello, es posible utilizar el paquete open-source XAMPP, multiplataforma, que contiene lo necesario para iniciar un servidor Apache y la base de datos. Se ha utilizado esta distribución en un entorno Windows, aunque no habría ningún inconveniente de utilizar, por ejemplo, cualquier paquete LAMP disponible en un entorno Linux.

La instalación de XAMPP es sencilla y guiada mediante el ejecutable que puede encontarse en su sitio web. Tan sólo será necesario indicarle la ubicación de la instalación y los complementos necesarios. En el caso de este trabajo, sólo se precisan los marcados en la siguiente imagen.



En el fichero *httpd.conf*, que almacena la configuración del servidor Apache y es accesible desde el botón *Config* del panel de XAMPP debe ser modificado para servir la aplicación. Se añade el parámetros DocumentRoot (que utilizará el directorio raíz para servir los documentos), un Directory con la misma ruta y un VirtualHost. En este caso, el directorio se corresponde con la carpeta *Proyecto*, dentro de *Código desarrollado* incluida en el CD.

```
DocumentRoot "Ruta de la raíz de la aplicación"
<Directory "Ruta de la raíz de la aplicación">
```

```
    Options Indexes FollowSymLinks Includes ExecCGI
    AllowOverride All
    Require all granted
</Directory>

# Virtual hosts
<VirtualHost *:80>
    ServerName nombre_de_la_aplicacion
    DocumentRoot "Ruta de la raíz de la aplicación"
</VirtualHost>
```

Una vez están dispuestos los parámetos y se ha reiniciado el servidor, se puede comprobar que la aplicación funciona correctamente a través del navegador en el enlace *localhost/sala.html*. Para permitir el acceso desde el exterior, será necesario configurar el servidor mediante el parámetro Listen seguido del puerto a utilizar, que deberá estar abierto por el sistema debidamente.

En la dirección *http://localhost/phpmyadmin/* se permite gestionar las bases de datos asociadas al servidor. El código PHP que accede a esta base utiliza unos parámetros genéricos como ejemplo. El nombre de la base de datos a la que pretende acceder es *proyecto*, se accede con el usuario *root* y sin contraseña. Estos parámetros pueden cambiarse fácilmente en los ficheros .php según la base de datos creada sin mayor repercusión.

Los ficheros de metadatos incluidos proporcionan la base de la representación WebGL. No es recomendable eliminar ni los ficheros *dispositivo*, *modelo*, *rack* y *sala* ni sus campos idetificativos o de posición que se usan en el código JavaScript. Por supuesto, tampoco lo es retirarlos de la base de datos.

Para mantener consistencia con lo desarrollado, se puede utilizar el administrador de phpmyadmin para, pulsando el botón *Importar,* crear la base de datos seleccionando el fichero *descripcion_proyecto.sql* incluido en la carpeta *mysql* dentro de *Código desarrollado* y pulsando el botón *Continuar*. Esto creará el esquema de la base de datos, las tablas con el contenido de ejemplo y el procedimiento almacenado que se utiliza para la modificación de las columnas.

Apéndice B - Manual de usuario de la aplicación

La aplicación web se divide en dos páginas distintas accesibles para el usuario: *sala.html* y *datos.html*. Se puede acceder a ambas desde el navegador, una vez el servidor se encuentre en funcionamiento y tenga dirigido *localhost* a la carpeta *Proyecto*.

La representación 3D

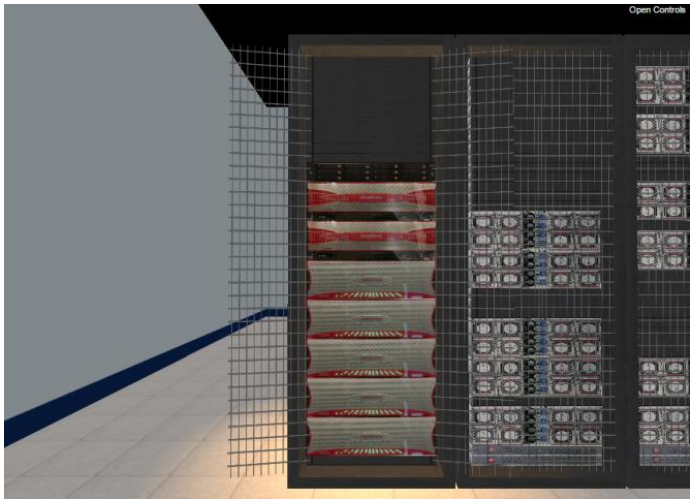
Al introducir en el navegador compatible con WebGL la dirección *localhost/sala.html*, se cargan los elementos de la escena a la vez que la cámara se sitúa lentamente en un extremo de la sala. Los controles de la interfaz gráfica ligera con comandos de prueba se encuentran en la esquina superior derecha en un panel extensible cerrado hasta que se pulse sobre él.

A partir de este punto, el usuario puede utilizar el ratón para, haciendo click con ambos botones y arrastrando, girar la cámara o trasladarla en los ejes X e Y, respectivamente. Por otra parte, la rueda del ratón sirve para aumentar o reducir el zoom de la cámara. Este movimiento tiene sus limitaciones, por lo que, para realizar la selección de un elemento, basta con pulsar sobre él.

- En caso de no estar marcándose ningún dispositivo interno al pulsar en un estante, la cámara se desplazará en la escena dirigiendo su mirada hacia el objeto.
    - Si estaba situada a una altura por encima de la máxima de la sala, se mantendrá a dicha altura, manteniéndose una perspectiva que permita ver el resto de la misma. Una luz azul parpadeante se situará en el suelo bajo el estante, indicando su selección
    - Si, por el contrario, estaba a una altura cercana a los estantes, se entiende que el usuario pretende observar de cerca ese estante en concreto y se dirige directamente hacia él, mientras una luz naranja parpadeante se situa bajo el rack. Esto sucede también en caso de realizar un segundo click en un estante ya marcado.
- Por otra parte, si una máquina dentro del estante muestra un bordeado de color naranja a su alrededor, el objeto se considera selecionado. La cámara se dirige hacia él y representa los valores almacenados con su información relevante.

Sea cual sea el objeto seleccionado, en caso de fijarse en él, la cámara se desplazará a una zona distinta según la posición e la que se encuentre al haber pulsado:

- Si la cámara estaba situada delante de la parte frontal del objeto, se colocará delante de esa zona.
- Si, por el contrario, se encontraba detrás del objeto, la cámara se situará en un punto de anclaje tras el estante pulsado (o el correspondiente al dispositivo pulsado) y una animación representará la apertura de las puertas traseras del mismo para una mejor observación.



Los botones de prueba del panel extensible realizan distintos funciones al pulsarlos:

- *dorender* manipula la variable booleana con este nombre, paralizando o continuando el renderizado. Esto resulta útil si se tienen errores en el desarrollo que se repiten con cada iteración del bucle de animación.
- *reset* vuelve a colocar la cámara como se encontraba al comienzo de la escena, en caso de resultar más cómodo esta acción que el control por ratón habitual.
- *viaje* realiza un desplazamiento de la cámara por la parte trasera de cada estante de la escena.
- *temperatura* añade coloreado a los dispositivos para informar, usando valores aleatorios, sobre este parámetro. Se plantea como una posible idea de representación para implementaciones futuras de esta funcionalidad.
- *stats* carga y coloca en pantalla el generador del estadístico de rendimiento homónimo.

La gestión de los metadatos

El directorio *metadata* (y una copia de respaldo con los valores de inicio en el directorio *metadata_init*), situado dentro del raíz *Proyecto*, almacena los metadatos asociados, como se explicó en la sección 3.6.2 de esta memoria.

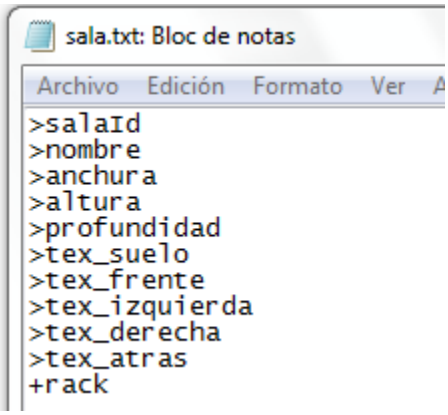
Las acciones a tomar en el directorio son las siguientes:

- Crear un fichero .txt que genere una nueva tabla en la base de datos.
- Modificar los campos de un fichero, produciendo los cambios de las columnas de su tabla.
- Eliminar un fichero, retirando de la base de datos la tabla (Para evitar pérdida accidental de información, la tabla no se elimina hasta hacerlo explícitamente en la base de datos, pero no se representa).

La creación y modificación del fichero atiende al uso de caracteres especiales al comienzo de cada línea:

- Un signo “>” supondrá una columna con el nombre especificado.
- Un signo “-” planteará una relación uno-a-uno con otra tabla con el nombre indicado.
- Un signo “+” creará una relación uno-a-muchos con la tabla descrita.

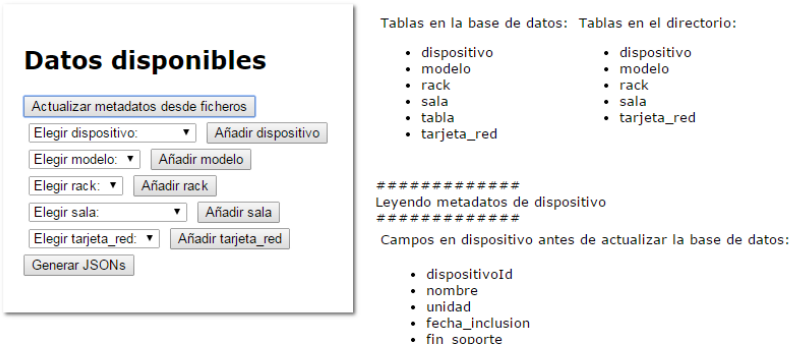
Es recomendable que el fichero comience con un identificador basado en el nombre de la tabla seguido de "Id" en un campo antecedido por un signo ">", por ejemplo: ">tablaId". Este será el valor autoincremental creado en la tabla, se especifique o no en el fichero. Si el fichero ha sido creado por la aplicación para completar el diseño de otra tabla (con una relación definida por un signo "-" o "+"), esta línea se añade automáticamente.



En este caso, se crearía la tabla *sala*, con el valor autoincremental de identificador *salaId* y los campos que le siguen. Posteriormente, se crearía una tabla *rack* con su respectivo identificador, además de otro llamado *salaId* que haga las veces de clave foránea.

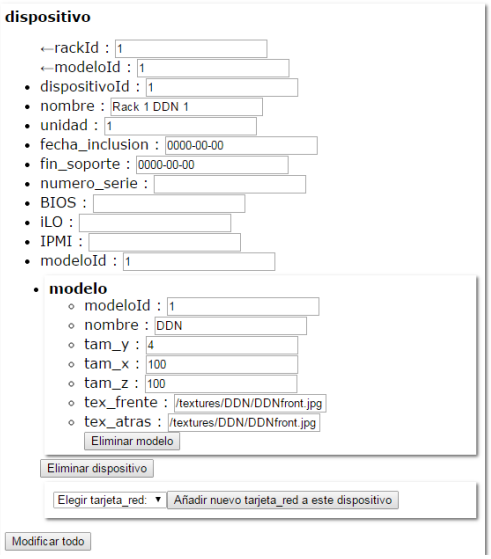
La aplicación web para la gestión

Por otra parte, si se introduce en el navegador la dirección *localhost/datos.html*, se obtiene el selector de la aplicación de gestión. El botón *Actualizar metadatos desde ficheros* toma los archivos modificados anteriormente y realiza las funciones para actualizar la base de datos. Posteriormente, se representa en pantalla el estado antes y después de la ejecución.



El grupo de selectores situado bajo ese botón representa al pulsar alguno de ellos, en forma de árbol, la información almacenada de la tabla y aquellas asociadas directamente en relación *uno-a-uno*. Las que lo estén mediante una relación *uno-a-muchos* aparecen en un nuevo selector, que generará también la información necesaria al pulsarlo.

Los datos se presentan dentro de un formulario, para poder enviar los cambios que sean necesarios. Cada elemento del formulario cuenta con un botón *Eliminar* que retira la columna referenciada de la tabla que la contiene. El botón *Añadir* de cada selector inserta en la tabla una nueva fila vacía.



El botón generar JSONs crea dentro de la carpeta *json*, situada en el raíz *Proyecto*, la estructura necesaria para comenzar la aplicación de visualización. Se genera un directorio para cada elemento único, por lo que las tablas definidas con un signo "-" en relación *muchos-a-uno*, en el caso son almacenadas por separado para evitar el envío de información repetida (en este caso, la tabla almacena información con el modelo de un dispositivo y muchos dispositivos pueden ser del mismo modelo). Una vez generados correctamente, se puede comenzar la visualización en *localhost/datos.html*.