Universitat Autònoma de Barcelona

Gestión y Visualización de museos virtuales 3d en línea

Memoria del proyecto de Ingeniería Técnica en Informática de Gestión

realizado por Jordi Nadal Gesto,

y dirigido per Daniel Riera Terrén,

Escola Universitària d'Informàtica

Sabadell, Septiembre de 2010

El/la abajo firmante, Daniel Riera Terrén
professor/a de la Escola Universitària d'Informàtica de la UAB,
CERTIFICA:
Que el trabajo al que corresponde la presente memoria
ha sido realizado bajo su dirección
por Jordi Nadal Gesto
I para que conste firma la presente.
Sabadell, Agosto de 2010

Firmado:

Índice de contenidos

1 Introducción

1.1 Descripción general

El mundo del desarrollo web es un mundo en cambio constante. Uno de los cambios que esta sufriendo actualmente es uno que lleva años intentándose llevar a cabo: traer el web a un nuevo nivel, páginas web en 3d. Este proyecto se propone tomar la iniciativa, y ser el pionero dando los primeros pasos por un camino a oscuras en busca de tal objetivo.

Determinar qué interfaces harán falta para crear páginas web con entornos tridimensionales fáciles de navegar para cualquier usuario con independencia de su condición y experiencia con entornos en tres dimensiones. Buscar las limitaciones que un entorno web impone a un entorno 3d y viceversa. Lidiar con las limitaciones que el *hardware* y *software* del usuario impone. Todo ello juntado en un aplicación web con una aplicación práctica: la gestión y visualización de un museo en 3d.

La aplicación constará de una herramienta para administradores que permitirá ver una sala de exposiciones virtual vacía, y una biblioteca de obras de arte; se podrán añadir y quitar obras de la sala de exposiciones arrastrándolas de/a la biblioteca, creando así galerías de arte virtuales. Éstas galerías se podrán visualizar por cualquier persona con un explorador web.

1.2 Objetivos

> "Es el objetivo lo que nos ha creado...el objetivo nos vincula, el objetivo nos motiva, nos guía, nos mueve, es el objetivo lo que nos define, el objetivo nos mantiene unidos."
>
> --agente Smith en Matrix

A continuación se presentan los objetivos principales del proyecto:

1. Creación de una web de gestión de galerías de arte en 3d. Ha de permitir colocar en un entorno tridimensional una obra de arte determinada de forma fácil e intuitiva. Desde colocar cuadros en las paredes o estatuas en el suelo, hasta jarrones en pedestales.
2. Creación de una biblioteca de obras de arte.
3. Poder guardar galerías de arte.
4. Creación de un módulo de visualización que permita visitar las galerías de arte con una cámara en tercera persona. La exploración de las galerías ha de ser sencilla y ha de tratar de dar la máxima sensación de realismo.

1.3 Acerca de este documento

El presente documento es la memoria del proyecto. La memoria esta dividida en varias secciones: introducción, estudio de viabilidad, análisis de requerimientos, fundamentos teóricos, tecnologías y herramientas usadas, fases de diseño, implementación y *tests*, y conclusiones.

Las secciones con la introducción y el estudio de viabilidad se han redactado con un lenguaje menos técnico de forma deliberada para facilitar su comprensión. Se recomienda pues empezar la lectura por estas secciones.

Las secciones fundamentos teóricos, tecnologías y herramientas usadas nos darán un bagaje de conocimientos suficiente para entender el resto de secciones, por lo tanto son las que se recomienda leer a continuación. Estarán redactadas en un lenguaje de carácter más técnico.

En las secciones análisis de requerimientos, diseño, implementación y *tests*, se describe el proyecto propiamente dicho, dando por hecho que el lector dispone de los conocimientos necesarios.

La sección conclusiones resume la consecución o no de los objetivos expuestos en la sección introducción y expone algunas de las posibles ampliaciones.

1.4 Motivación personal y agradecimientos

> "La habilidad es lo que eres capaz de hacer. La motivación determina lo que harás. La actitud determina lo bien que lo harás"
>
> -- Lou Holtz

Allá por 1996, a través de un anuncio publicitario de un curso de programación IBM, a la temprana edad de 14 años descubría yo, Jordi Nadal, la programación. Cambió mi vida. Descubrí un mundo nuevo, que a la postre sería mi trabajo, mi mayor *hobby*, y mi forma de vida. Actualmente, soy programador web profesional. Trato de estar siempre al tanto de cualquier tendencia nueva en el mundo del desarrollo web.

Por otra parte siempre he mostrado interés por el mundo del 3d, mundos virtuales, video-juegos, etc. A modo de proyectos personales, he desarrollado motores gráficos 3d, juegos, etc.

Así pues, la posibilidad que se me presenta en este proyecto de tratar de combinar sendos mundos es irrepetible. Tengo la oportunidad de experimentar con entornos 3d destinados a un uso en un entorno web y descubrir sus limitaciones. Tengo la oportunidad de analizar la posibilidad de explotar económicamente una tecnología radicalmente innovadora. Todo ello unido, despertó mi interés hasta el punto que se ha acabado materializando en mi Proyecto Final de Carrera.

Proyecto que sin la ayuda de mi familia por su apoyo, de mi tutor Daniel Riera por sus consejos, de Douglas Crockford por sus aportaciones al desarrollo en JavaScript y en especial del conjunto de desarrolladores de Google por los incontables productos que han traído al web como son: *O3D, translate api, ajax api, gears, closure tools, search api, maps api, visualization api*, y un largo etcétera!

A todos ellos, mil gracias.

2 Estudio de viabilidad

> "Es intentando lo imposible como se realiza lo posible."
>
> -- henry barbusse

Se ha realizado un análisis de viabilidad exhaustivo , los resultados del cual se resumen en los siguientes puntos.

2.1 Planificación

> "El mayor de los peligros para la mayoría de nosotros, no es que nuestro objetivo sea demasiado alto y no lo alcancemos, sino que sea demasiado bajo y lo logremos"
>
> – Michelangelo

A continuación se detalla la planificación prevista, la cual como puede verse en el diagrama de Gantt de la figura 1, se desglosa en un proyecto a realizar para 3 personas, un analista y gestor de proyectos, un diseñador, y un programador y beta-tester, que suman un total de 253 horas, o lo que es lo mismo, 63 días laborables y una hora según la planificación, que asume una jornada laboral de 4 horas al día, 2 días a la semana (todos los fines de semana) empezando el 10 de octubre y finalizando el proyecto el 18 de abril.

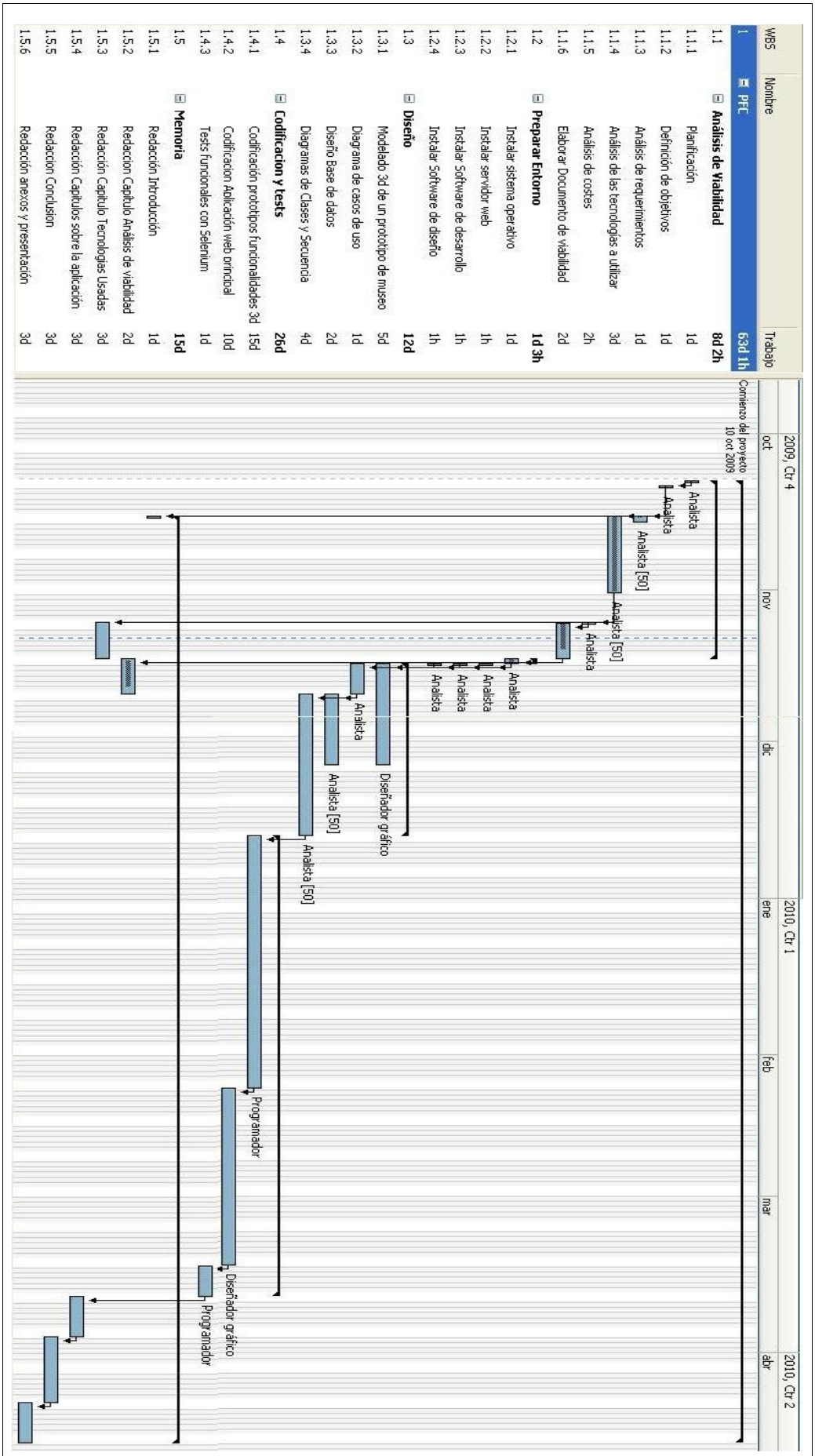
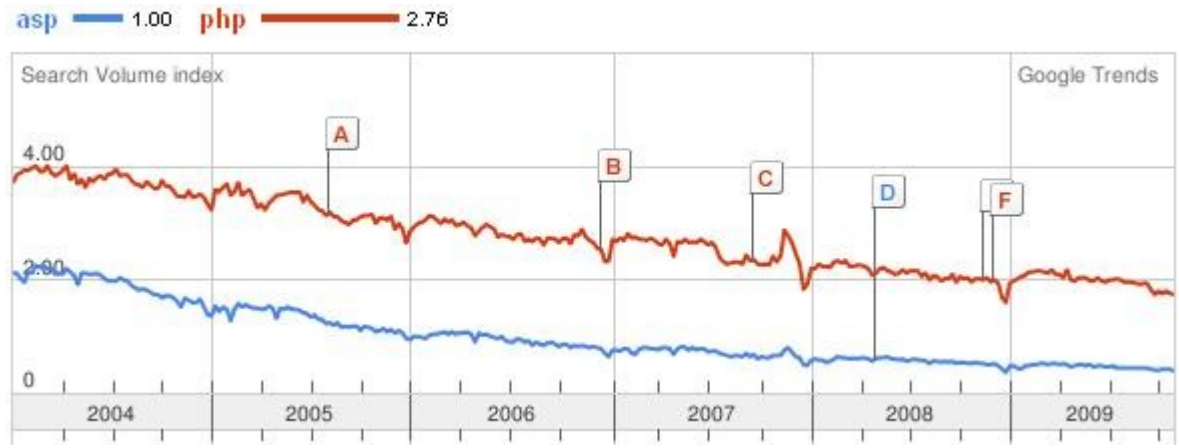


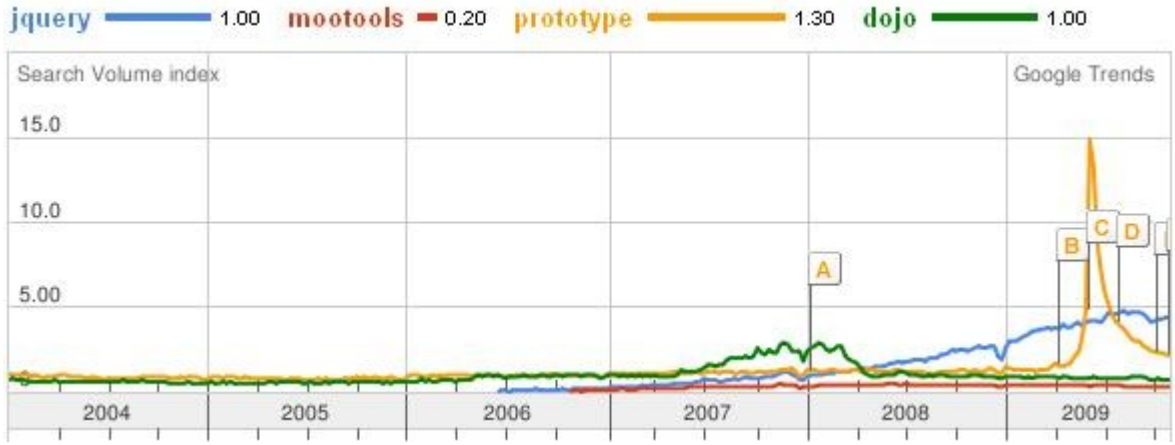
Figura 1: Planificación prevista

2.2 Viabilidad técnica

> "Los ordenadores son inútiles. Sólo pueden darte respuestas"
> -- Pablo Picasso

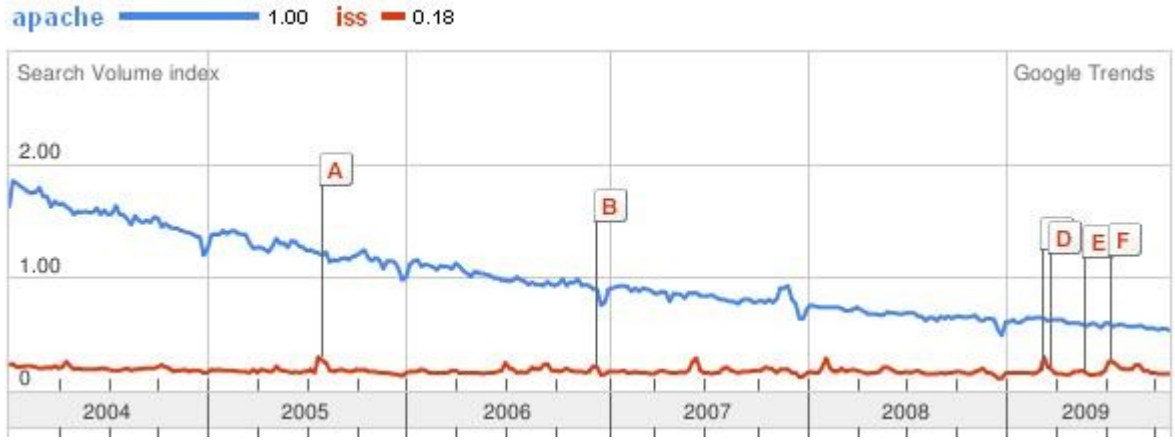
En este apartado se discute la elección de las tecnologías más adecuadas para la consecución exitosa a nivel técnico de los objetivos marcados. Se muestran en. forma de tabla las elecciones y el razonamiento que lleva a elegir estas tecnología en detrimento de otras que se han tenido en cuenta. Las tecnologías que se han descartado desde un principio ya sea por precio o por desconocimiento no se incluyen en la tabla. Para cada Grupo de tecnologías posibles se incluye un gráfico que muestra el volumen de búsquedas en Google dando un indicativo del uso de las mismas. Aunque este indicador no pueda validarse científicamente, sí que nos servirá para hacernos una idea de dicho uso en Internet. La mayoría de elecciones son típicas del desarrollo web y no requieren de un estudio en profundidad. La decisión más crítica en este caso es la elección de la *API* 3d. Para la elección de esta se ha hecho un estudio más detallado que se muestra en una segunda tabla tras las que se muestran a continuación.

nombre	PROs	CONTRAs	Conclusiones
 *Figura 2: Código en el servidor*			
Php	Mayor compatibilidad menor precio independencia del servidor web menor curva de aprendizaje Mejor documentación		Se usará php
asp		Dependencia de IIS,	descartado

nombre	PROs	CONTRAs	Conclusiones
 *Figura 3: Librerías y frameworks Javascript* nota: los resultados de prototype y dojo están inflados puesto que tienen nombres con otros significados			
jQuery + jQueryUI	Mejor documentación elegancia y expresividad menor curva de aprendizaje mayor productividad soporte para widgets más usada	no dispone de sistema de herencia	Se usará jQuery y jQueryUI por su facilidad de uso, y productividad,
Moo Tools	Soporte para herencia	Mayor curva de aprendizaje	opcional, JavaScript 2.0 promete clases, por lo que su mayor virtud desaparecerá
prototype	Mejor rendimiento dispone herencia y clases	no dispone de widgets	opcional, JavaScript 2.0 promete clases, por lo que su mayor virtud desaparecerá
dojo	Soporte para widgets documentación	Mayor curva de aprendizaje	Opcional, Tras jQuery es la más interesante

nombre	PROs	CONTRAs	Conclusiones
 *Figura 4: Bases de datos*			
MySQL	Mayor rendimiento soporte para transacciones coste nulo menor curva de aprendizaje mejor documentación		Se usará mysql
PostgreSQL	Mayor soporte de funcionalidades como soporte para transacciones y subconsultas	Mayor curva de aprendizaje	descartado

nombre	PROs	CONTRAs	Conclusiones
 *Figura 5: Motores 3D* nota: para vrml97 y five3d no aparecen suficientes resultados			
O3D	Mayor potencia gráfica uso de la tarjeta gráfica mayor variedad de *hardware* y *software* soportado de todos las *API*s estudiadas soporte para shaders a través de JavaScript	Actualmente requiere el *plugin* en el explorador, se espera que cambie la instalación del *plugin* en Linux/Unix/Bsd no es sencilla, o simplemente no funciona	Se usará O3D
WebGL	promete potencia al mismo nivel que O3D promete soporte nativo del explorador que evita la necesidad de *plugins*.	Código inmaduro y en desarrollo	Descartado, tecnología demasiado nueva e inmadura, a largo plazo es la tecnología más interesante tras O3D
VRML97		Desfasado, menor potencia gráfica	descartado
Away3d	de las opciones estudiadas basadas en flash es la que tiene mayor potencia	requiere flash	descartado
Five3d	sencillo	requiere flash	descartado
papervision	de las opciones estudiadas basadas en flash es la que tiene mayor seguimiento	requiere flash	descartado

nombre	PROs	CONTRAs	Conclusiones
 *Figura 6: Servidores web*			
Apache	Mejor documentación y comunidad más activa independencia del sistema operativo		Se usará apache
ISS		Dependencia del sistema operativo costes	descartado

para la elección del *API* 3d, se han eliminado de la terna, todas las opciones basadas en flash para mantener la aplicación basada en tecnologías de licencia libre, y mantener así la viabilidad económica y legal del proyecto bajo control. Vrml97 queda descartada inmediatamente por desfasada. Así pues, todo lo que queda es decidir entre WebGL y Google O3D. Ambos pretenden transformase en el estándar de facto para gráficos 3d en el web.

WebGL es un proyecto conjunto del grupo de desarrollo WebGL del grupo Khronos (consorcio empresarial que mantiene el desarrollo de la especificación de OpenGL) y la fundación Mozilla (desarrolladores del explorador web Firefox). Su objetivo es exponer OpenGL Es 2.0 a javascript.

Google O3D es un proyecto de Google. Permite actualmente, exponer una versión reducida de OpenGL 2.1 a Javascript.

	O3D	WebGL
autor(es)	Google	Especificación: WebGL working group soporte en Firefox y WebKit: Mozilla Foundation
Objetivo	Exponer partes de OpenGL 2.1 a Javascript, se han eliminado las funciones estáticas, siguiendo un	Exponer OpenGL Es 2.0 a JavaScript. (basado en *shaders*, no usa funciones estáticas)

	modelo basado en *shaders*	
Shaders	Formato propio basado en una mezcla de Cg de Nvidia, y HLSL de Microsoft	Se espera que esté basado en *shaders* GLSL (versión por confirmar) de OpenGL
Sistemas Operativos Soportados	Windows XP/Vista/7, MacOs X, Linux, BSD, Unix	Windows XP/Vista/7, MacOs X, Linux, BSD, Unix
Exploradores Web Soportados	Firefox, Internet Explorer, Chrome, Safari, WebKit	Firefox, WebKit. Chrome
Hardware Gráfico Soportado	Todo el que tenga implementado a nivel de *drivers* soporte total para OpenGL 2.1 o Direct3d 9.0c	Todo el que tenga implementado a nivel de *drivers* soporte total para OpenGL Es 2.0

Tests realizados					
Descripción del *hardware/software* usado			O3D	WebGL	Notas
	procesador	Intel coreDuo 32bit	Fallo	Fallo	WebGL solo funciona con versiones superiores a 3.7prealpha. En el caso de O3D, se debe compilar manualmente, al ejecutarlo da un "fallo de segmentación"
	Tarjeta gráfica	AMD/Ati mobility radeon X1400			
	Drivers de la tarjeta gráfica	xf86-video-ati 6.12.4			
	Sistema operativo	Linux Ubuntu 9.10, Sabayon Linux 5.o			
	Explorador	Firefox 3.5, Konqueror			
	otros	OpenGL mesa 7.6 kernel linux 2.6.31.14 con KMS activado			
	procesador	Intel coreDuo 32bit	Fallo	Fallo	WebGL falla devolviendo el error "Canvas 3D: GLX_SGIX_pbuffer not supported ", mientras que glxinfo | grep pbuffer muestra: GLX_SGIS_multisample, GLX_SGIX_fbconfig, GLX_SGIX_pbuffer es un error de WebGL al comprobar el soporte de GLX_SGIX_pbuffer(lo busca
	Tarjeta gráfica	AMD/Ati mobility radeon X1400			
	Drivers de la tarjeta gráfica	xf86-video-ati 6.12.4			
	Sistema operativo	Linux Ubuntu 9.10, Sabayon Linux 5.o			
	Explorador	Firefox 3.7prealpha			
	otros	OpenGL mesa 7.6,			

		kernel linux 2.6.31.14 con KMS activado			en los módulos cliente y servidor: solamente las tarjetas de Nvidia lo tienen en ambos módulos y en el *hardware* usado solamente en el de cliente) . En el caso de O3D, se debe compilar manualmente, al ejecutarlo da un "fallo de segmentación"
	procesador	Intel core2 Duo 64bit	Fallo	Fallo	WebGL falla devolviendo el error "Canvas 3D: GLX_SGIX_pbuffer not supported ", mientras que glxinfo | grep pbuffer muestra: GLX_SGIS_multisample, GLX_SGIX_fbconfig, GLX_SGIX_pbuffer es un error de WebGL al comprobar el soporte de GLX_SGIX_pbuffer(lo busca en los módulos cliente y servidor: solamente las tarjetas de Nvidia lo tienen en ambos módulos y en el *hardware* usado solamente en el de cliente) . En el caso de O3D, se debe compilar manualmente, al ejecutarlo el área 3d se queda en blanco
	Tarjeta gráfica	intel express serie 4			
	Drivers de la tarjeta gráfica	xf86-video-intel 2.9.1			
	Sistema operativo	Linux Ubuntu 9.10			
	Explorador	Firefox 3.7prealpha			
	otros	Opengl mesa 7.6, kernel linux 2.6.31			
	procesador	Intel core2 Duo 64bit	Fallo	Fallo	WebGL solo funciona con versiones superiores a 3.7prealpha. En el caso de O3D, se debe compilar manualmente, al ejecutarlo el área 3d se queda en blanco
	Tarjeta gráfica	intel express serie 4			
	Drivers de la tarjeta gráfica	xf86-video-intel 2.9.1			
	Sistema operativo	Linux Ubuntu 9.10			
	Explorador	Firefox 3.5			
	otros	OpenGL mesa 7.6, kernel linux 2.6.31			
	procesador	Intel core2 Duo 64bit	Ok	Fallo	WebGL solo funciona con versiones superiores a 3.7prealpha.
	Tarjeta gráfica	intel express serie 4			
	Drivers de la tarjeta gráfica	intel v15.13.6.64.1908			

	Sistema operativo	Windows Vista/7			
	Explorador	Firefox 3.5, Internet Explorer 7,8, Chrome			
	procesador	Intel core2 Duo 64bit	Ok	Funciona parcialmente	Para WebGL los resultados son dispares, algunas demos funcionan, otras no
	Tarjeta gráfica	intel express serie 4			
	Drivers de la tarjeta gráfica	intel v15.13.6.64.1908			
	Sistema operativo	Windows Vista/7			
	Explorador	Firefox 3.7prealpha			
	procesador	Intel coreDuo 32bit	Ok	Falla	WebGL solo esta soportado para la versión 3.7prealpha o superior de Firefox
	Tarjeta gráfica	AMD/Ati mobility radeon X1400			
	Drivers de la tarjeta gráfica	AMD/Ati Catalyst v9.3			
	Sistema operativo	Windows XP			
	Explorador	Firefox 3.5, Internet Explorer 7,8, Chrome			
	procesador	Intel coreDuo 32bit	Ok	Funciona parcialmente	Para WebGL los resultados son dispares, algunas demos funcionan, otras no
	Tarjeta gráfica	AMD/Ati mobility radeon X1400			
	Drivers de la tarjeta gráfica	AMD/Ati Catalyst v9.3			
	Sistema operativo	Windows XP			
	Explorador	Firefox 3.7prealpha			

En resumen, para MacOs X no se han podido realizar pruebas por falta de disponibilidad del *hardware/software* necesario, para Linux no se ha podido lograr hacer ir ninguna librería en ninguna de las configuraciones de *hardware* probadas por problemas con los *drivers* de las mismas. Para Windows O3D funciona en todas las combinaciones de *hardware* testeados, y WebGL solo ha dado resultados parciales.

La conclusión es clara, usando Google O3D, MySql, PHP, Apache, jQuery, jQueryUI el proyecto es técnicamente viable.

2.3 Viabilidad económica

> "No se trata de bits, bytes y protocolos, sino de beneficios, pérdidas y márgenes"
> -- Lou Gerstner

En este caso, se ha podido usar una terna de tecnologías , todas ellas de licencia libre, y por tanto coste nulo.

Se han seleccionado, herramientas que reduzcan al mínimo indispensable el número de horas de trabajo, maximizando productividad, minimizando costes en forma de horas/hombre, los cuales tenemos ya previstos en la planificación.

En cuanto a costes materiales, podemos contabilizar la amortización del *hardware* con el que se realizarán los *tests* y el desarrollo. Se podría contabilizar también los gastos proveniente del consumo eléctrico y conexión a Internet, pero los consideraré nulos.

El proyecto no incluye el *hosting* de la web que se va a crear, luego podemos suponer este coste nulo.

En la siguiente tabla se detallan los costes previstos,

Concepto	**Valor**
Total horas/hombre (Analista,programador y diseñador)	1.850,00 €
Amortización equipamiento (2 portátiles valorados en 600€, con una vida útil de 4 años, amortizados uniformemente al 25%, 1r año)	300,00 €
Impuesto sobre beneficios al 30%	645,00 €
Total	2.795,00 €

Vendiendo el producto final a un precio superior a 2.795,00 € el proyecto se puede considerar económicamente viable. Teniendo en cuenta que es un producto nuevo y distintivo, que crearía valor añadido para un hipotético cliente, podemos esperar que el cliente esté dispuesto a pagar esta cantidad. Así pues el proyecto es económicamente viable.

2.4 Viabilidad legal

> "En 2031, los abogados serán componentes habituales de la mayoría de los equipos de desarrollo"
> -- Grady Booch

El marco legal actual, que puede afectar al proyecto, se puede ceñir a la LSSI[1] y a la LOPD[2].

El primero de sendos documentos, establece las obligaciones para toda web que realice una actividad económica. No es el caso.

El Segundo, establece las obligaciones para toda web que manipule datos personales. En el caso de del proyecto solamente se requiere un nombre de usuario y por lo tanto tampoco afecta al proyecto.

No hay riesgo de problemas por patentes, ni se violan licencias.

Por todo ello el proyecto es legalmente viable.

2.5 Conclusiones

Dadas las razones expuestas en las secciones anteriores se considera que el proyecto es viable, técnica, legal, y económicamente en los plazos especificados en la planificación del proyecto.

1 Ley de Servicios de la Sociedad de la Información ver http://www.lssi.es

2 Ley Orgánica de Protección de Datos ver https://www.agpd.es

3 Análisis de requerimientos

> "Un programador es la persona considerada experta en ser capaz de sacar, después de innumerables tecleos, una serie infinita de respuestas incomprensibles calculadas con precisión micrométrica a partir de vagas asunciones basadas en discutibles cifras tomadas de documentos inconcluyentes y llevados a cabo con instrumentos de escasa precisión, por personas de fiabilidad dudosa y cuestionable mentalidad con el propósito declarado de molestar y confundir al desesperado e indefenso departamento que tuvo la mala fortuna de pedir la información en primer lugar"
> -- IEEE Grid newsmagazine

Una vez hemos concluido el análisis de viabilidad técnica del proyecto, podemos profundizar en los requerimientos del mismo extendiendo los objetivos marcados en el apartado de introducción.

A continuación se detallan los tipos de usuarios que deberán interactuar con el sistema, los requerimientos funcionales y los requerimientos no funcionales identificados.

3.1 Roles

> "Desde el punto de vista de un programador, el usuario no es más que un periférico que teclea cuando se le envía una petición de lectura"
> -- P. Williams

La aplicación web propuesta será un punto de encuentro donde el administrador/gerente de un museo real se registre como usuario. Cada usuario dispondrá de un o más museos virtuales, en los que celebrar una o más exposiciones. A tal efecto cada usuario dispondrá de una cuenta de acceso.

Sin embargo para la visualización de dichas obras los visitantes de dichos museos no requerirán de cuenta de acceso particular para visualizar las exposiciones.

Para la administración de la web en general, se requiere de un rol de administrador.

A tal efecto se han identificado los siguientes roles de usuario y tareas que podrán interactuar con el sistema:

1. Cliente/Usuario - administrar sus museos, sus obras de arte, y sus exposiciones.
2. Administrador - gestión (administración de usuarios, ...)
3. Visitante - visualizar exposiciones

3.2 Requerimientos funcionales

> "En software, muy raramente partimos de requisitos con sentido. Incluso teniéndolos, la única medida del éxito que importa es si nuestra solución resuelve la cambiante idea que el cliente tiene de lo que es su problema"
> -- Jeff Atwood

A continuación se citan los requerimientos funcionales previstos. Nótese que no es una lista cerrada, pues se pretende seguir una metodología de trabajo ágil, la cual permite a cada iteración de la misma re-enfocar los objetivos pendientes, añadir nuevos o eliminar antiguos.

- Creación de una aplicación web multiusuario, con soporte para almacenamiento de datos sobre base de datos.
- Necesidad de mecanismos para mantener bibliotecas de obres de arte, incluye la posibilidad de crear y eliminar bibliotecas, añadir y quitar obras y mecanismos para determinar la "orientación" de las obras, es decir, la posición en que una vez colocadas en una superficie determinada, se tengan que orientar para dar la sensación de estar sentadas sobre dicha superficie. Para aclarar el concepto nada mejor un ejemplo: para un cuadro que irá en una pared, se orientará horizontalmente, y nunca verticalmente mientras que para una estatua que se coloque en el suelo se ha de orientar verticalmente.
- Necesidad de poder cargar una "sala de exposiciones virtual" sin ninguna obra. Poder añadir y quitar obras de arte tomadas de las bibliotecas (citadas en el punto anterior) a la sala de exposiciones. Se debe poder seleccionar la posición y orientación de las obras en dicha sala mediante interfaces tridimensionales. Y en última instancia, soporte para guardar y eliminar salas ya creadas.
- Necesidad de herramientas para la gestión de "*widgets*" html, entiéndase por *widgets*, la posibilidad de exportar de algún modo, las galerías creadas, para ser incrustadas en páginas web externas como blogs. El *widget* debe permitir explorar las galerías virtualmente, en un principio mediante una cámara en primera persona.
- Necesidad de algún sistema para la gestión de usuarios. Se espera que se puedan crear usuarios con diferentes permisos para administrar cada biblioteca. Si un usuario quiere que "su" biblioteca sea de uso público, o solamente de uso personal se ha de proveer de los mecanismos necesarios para que así sea.

3.3 Requerimientos no funcionales

> "Es más fácil cambiar las especificaciones para que encajen con el software que hacerlo al revés"
> -- Alan Perlis

En cuanto a requerimientos no funcionales, una web en 3d implica gran variedad de limitaciones; en la siguiente lista se describen estas limitaciones:

- ***Extensibilidad, Calidad y Reusabilidad***. La *aplication programming interface*-API de ahora en adelante- O3D es una librería de bajo nivel en muchos aspectos. Para usarla es necesario crear algún tipo de extensión de mayor nivel de abstracción, ya seo externa o creada internamente, que aumente su reusabilidad, puesto que es una tecnología en auge que probablemente será usada en proyectos futuros, y disponer de parte del trabajo hecho y probado en el futuro es un requerimiento deseable que reducirá costes y aumentará la productividad a largo plazo.

- ***Aspectos legales y de licencias.*** Ceñirse en la medida de lo posible a herramientas y librerías con licencias libres.

- ***Eficiencia***. La eficiencia será importante, en una aplicación 3d que depende del *hardware* gráfico de alto rendimiento ya es muy importante que el código sea eficiente, pero en una aplicación *web* 3d, el problema es aun más importante pues aparecen varias restricciones más, entre las más importantes encontramos el ancho de banda del cliente, el del servidor y la posibilidad del cliente de abrir multitud de pestañas en su explorador con multitud de aplicaciones web tratando de acceder paralelamente al *hardware* gráfico. Todas estas restricciones se deben tener en cuenta, tomando medidas tales como reducir el tamaño y resolución de las imágenes tanto como sea necesario.

- ***Plataforma y compatibilidad:*** Un objetivo deseable es maximizar la compatibilidad con la máxima variedad de combinaciones de *hardware* gráfico, sistema operativo y explorador web.

- ***Usabilidad***: Este aspecto tiene especial importancia en este proyecto. La usabilidad se entiende como la facilidad con que las personas pueden utilizar una herramienta particular. En este caso se debe ofrecer una interfaz tridimensional a la que muchos usuarios no están acostumbrados. Así pues es deseable buscar estructuras, ideas, y diseños propios de herramientas que conozcan aplicables en este proyecto. A modo de ejemplo paradigmático, la aplicación de modelado 3d Google SketchUp dispone de una herramienta de medición de ángulos con aspecto e icono de transportador que facilita mucho su uso.

4 Fundamentos teóricos

> "La diferencia entre la teoría y la práctica es que, en teoría, no hay diferencia entre la teoría y la práctica"
> - Richard Moore

Todo proyecto que incorpora 3d requiere de una serie de conocimientos básicos de álgebra, que incluyen vectores, matrices, cuaterniones, productos escalares, productos vectoriales, y sus aplicaciones en los cálculos necesarios para el renderizado 3d. Así mismo , los procesos físicos que se tratan de simular en un entorno virtual, como por ejemplo el comportamiento de la luz al incidir sobre diferentes formas provocando efectos como la reflexión y la refracción son también requeridos. Algunos de estos conocimientos se detallan a continuación con un enfoque orientado a su uso en el proyecto:

4.1 Álgebra y 3D

> "El álgebra es generosa: a menudo da más de lo que se le pide."
> --D'Alembert

Para representar una posición o una dirección en un espacio tridimensional, se usan vectores.

Se define un **vector** de n dimensiones como una tupla de n números reales llamados componentes del vector:

$\bar{v} = (c_1,c_2,..,c_n)$.

En un espacio tridimensional pues, se usan vectores con 3 componentes, que representan el desplazamiento a lo largo de los ejes X, Y y Z respectivamente para desplazamientos, o las coordenadas de un punto en el espacio (en la figura se muestra un ejemplo):

$\bar{v} = (d_x,d_y,d_z)$

Tradicionalmente, se usa el eje Z para representar la profundidad desde el punto de vista de la pantalla: para valores positivos

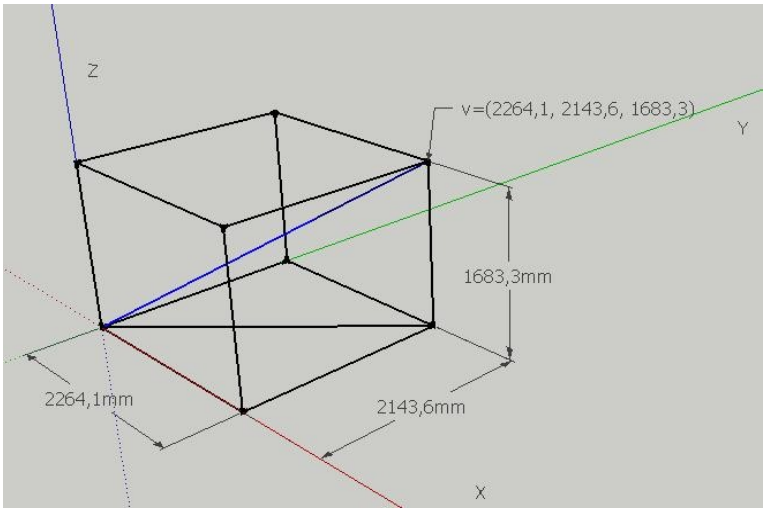


Figura 7: Vector en un espacio tridimensional

más al fondo, valores negativos al frente de la pantalla; el eje X se usa para el desplazamiento lateral, izquierda para valores negativos, derecha para positivos; por último el eje vertical se representa con el eje Y, valores positivos para arriba y valores negativos para abajo.

Se define como **módulo de un vector** $\bar{v}$ la longitud del mismo; dados 2 puntos $A=(a_x,a_y,a_z)$, $B=(b_x,b_y,b_z)$, la distancia que los separa se expresa como

$$|\bar{v}| = \sqrt{[(b_x-a_x)^2 + (b_y-a_y)^2+ (b_z-a_z)^2]}$$

Varias operaciones se pueden aplicar sobre vectores, todas ellas necesarias en los cálculos básicos con vectores.

La **suma de vectores,** permite componer vectores y se define como:

$$\bar{v'}=\overline{v1}+\overline{v2}=(v1_x,v1_y,v1_z)+(v2_x,v2_y,v2_z)=(v1_x+v2_x,v1_y+v2_y,v1_z+v2_z)$$

La **multiplicación de un vector por un real**, permite obtener vectores con mismo origen, sentido y dirección pero diferente módulo; se obtiene de la expresión:

$$\bar{v'}=\bar{v}*k=(v_x*k,v_y*k,v_z*k)$$

Se conoce como la **normalización de un vector**, el hecho de multiplicar un vector por la inversa de su módulo para obtener un nuevo vector de módulo unitario. Esta operación se expresa como : $\bar{u} = \bar{v}* (1/|\bar{v}|)$

Cobra especial importancia en computación pues como veremos a continuación permite reducir notablemente cálculos más complejos.

El **producto escalar** de 2 vectores se define como

$$\bar{a}\cdot\bar{b}= |\bar{a}|*|\bar{b}|*\cos(\alpha) =(a_x*b_x)+(a_y*b_y)+(a_z*b_z)$$

donde α representa el menor ángulo que forman sendos vectores. Tomando el hilo del proyecto, para colocar un cuadro en una pared vamos a necesitar un vector indicando la orientación de la superficie de dicha pared, respecto de una dirección arbitraria. Aislando α de esta fórmula se obtiene el angulo que forman. Repetiremos el cálculo sobre la orientación del cuadro, y obtendremos los ángulos que forman respecto de la misma dirección tomada arbitrariamente anteriormente. Restará pues solamente restar dichos ángulos y rotar el cuadro según el valor obtenido para alinear el cuadro con la pared. Este cálculo implica obtener el módulo de 2 vectores, lo cual a su vez implica operaciones con raíces cuadradas y exponentes; todo ello es muy costoso computacionalmente, y recordemos que es un calculo que se repetirá miles de veces por segundo. Pero retomando la idea de normalizar vectores, si antes de realizar ningún cálculo tomamos los vectores y los normalizamos, obtenemos 2 vectores nuevos cuya orientación se mantiene, por lo tanto forman el mismo angulo, pero sus módulos son unitarios; si observamos la fórmula descubriremos que $\alpha=\arccos([(a_x*b_x)+(a_y*b_y)+(a_z*b_z)]/|\bar{a}|*|\bar{b}|)$ y puesto que $|\bar{a}|*|\bar{b}| = 1*1= 1$ el cálculo se reduce a

$\alpha = \arccos((a_x*b_x)+(a_y*b_y)+(a_z*b_z))$

El último obstáculo pues es determinar alrededor de que eje debemos rotar el cuadro; la respuesta la encontramos en el producto vectorial de 2 vectores, dados 2 vectores v1 y v2, el **producto vectorial** de v1 y v2 es un nuevo vector ortogonal a v1 y v2, es decir perpendicular al plano que forman v1 y v2 y se define como:

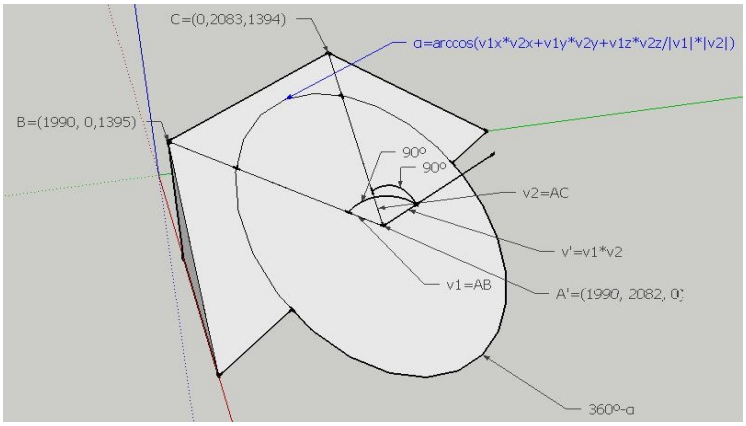


Figura 8: Producto vectorial

$\overline{v'}=\overline{v1} \times \overline{v2} = (u_y*v_z-u_z*v_y,\ u_z*v_x-u_x*v_z,\ u_x*v_y-u_y*v_x)$

Una condición necesaria para su calculo es que v1 y v2 deben formar un plano, es decir deben, obligatoriamente, ser linealmente independientes: no pueden ser paralelos. Sin embargo eso no es un problema para nuestro objetivo pues si son linealmente dependientes significa que la orientación de nuestro cuadro ya es la misma que la de la pared, con la salvedad de que aun no sabemos si el cuadro esta puesto con el lienzo encarado hacia la pared o al revés que es lo que pretendemos, solamente debemos mirar si un vector es el inverso del otro, si lo son debemos rotar la figura 180 grados.

Los cálculos expuestos anteriormente son herramientas muy potentes y sorprendentemente sencillas para obtener ángulos, posiciones, intersecciones o proyecciones. Pero no son la solución para realizar transformaciones tales como traslaciones, rotaciones o escalas sobre figuras complejas. De esto último se encargan las matrices.

Una matriz se define como un conjunto de números dispuestos en filas y columnas y definimos una matriz de orden m,n a una matriz con m filas y n columnas.

En general una transformación se aplica a un vector dado, conteniendo las coordenadas de un punto en el espacio, pre-multiplicando este vector por la matriz de transformación requerida en cada caso. Estas matrices pueden ser multiplicadas a su vez por otras, para componer transformaciones mas complejas. Las transformaciones básicas son:

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

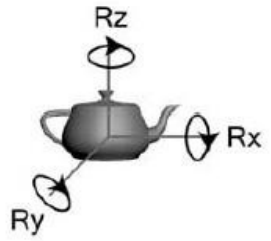
Figura 9: Traslación

Matriz de traslación: permite trasladar figuras, las cuales se ven desplazadas tx unidades de distancia a lo largo del eje x, ty a lo largo del eje y y tz a lo largo del eje z. (ver figura 9).

Matriz de escala: permite escalar, es decir, cambiar el tamaño de una figura alejando todos sus vértices del origen de coordenadas en una proporción indicada por los parámetros S_x,S_y,S_z respectivamente (ver figura 10).

$$\begin{pmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Figura 10: Escala



$$Rx = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$Ry = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad Rz = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figura 11: Rotación alrededor de los ejes X,Y y Z

Matrices de rotación X, Y y Z : permiten rotar una figura θº alrededor de los ejes X, Y y Z (ver figura 11).

Matriz de rotaciones para ejes arbitrarios: permiten rotar figuras a partir de un eje arbitrario $\underline{u}=(u_x,u_y,u_z)$. En el proyecto este tipo de matrices se usan para orientar un cuadro respecto de una pared (ver figura 12).

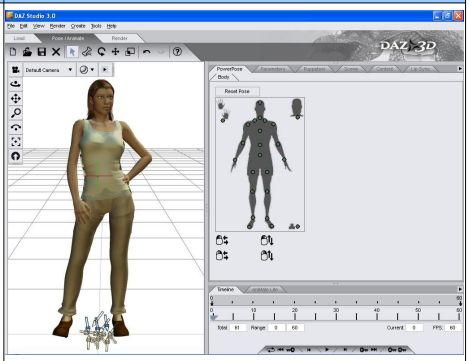
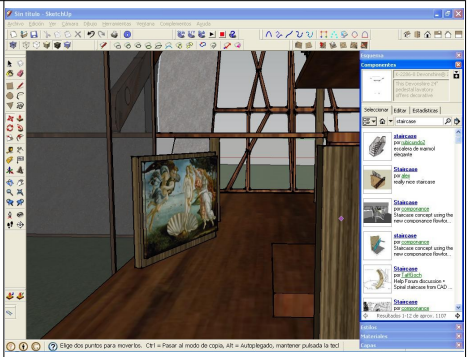
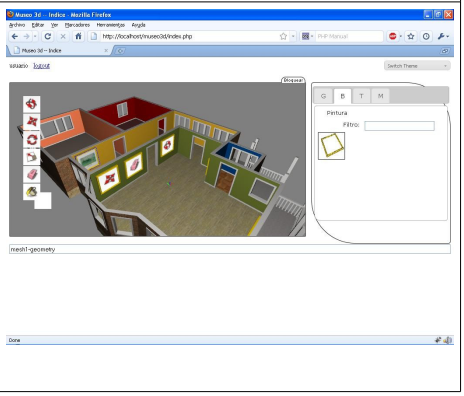
$$\begin{bmatrix} 1+(1-\cos\theta)(u_x^2-1) & (1-\cos\theta)u_xu_y-u_z\sin\theta & (1-\cos\theta)u_xu_z+u_y\sin\theta & 0 \\ (1-\cos\theta)u_xu_y+u_z\sin\theta & 1+(1-\cos\theta)(u_y^2-1) & (1-\cos\theta)u_yu_z-u_x\sin\theta & 0 \\ (1-\cos\theta)u_xu_z-u_y\sin\theta & (1-\cos\theta)u_yu_z+u_x\sin\theta & 1+(1-\cos\theta)(u_z^2-1) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

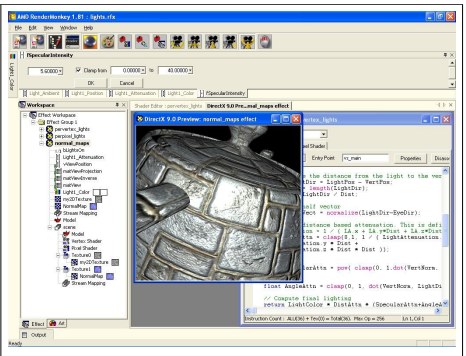
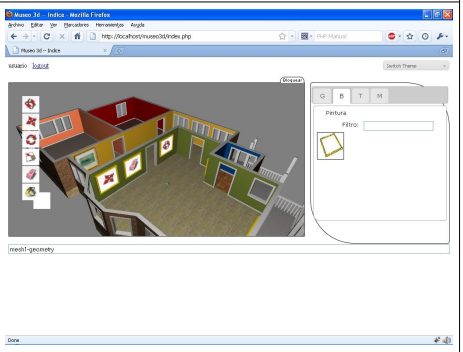
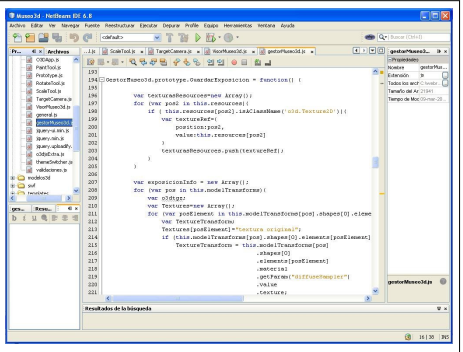
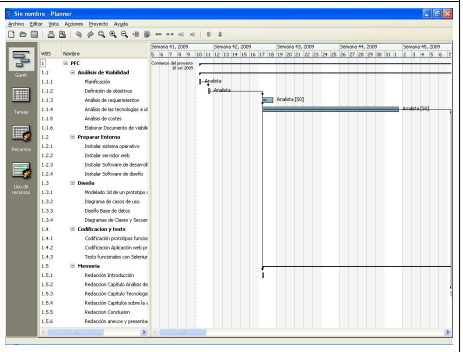
Figura 12: Rotación alrededor de un eje arbitrario

5 Tecnologías y herramientas usadas

> "El verdadero progreso es el que pone la tecnología al alcance de todos."
> – Henry Ford

Este apartado contiene una relación de herramientas de diferente índole que han sido usadas en algún momento durante la elaboración del proyecto. Para cada una de ellas, hay una captura de pantalla, una descripción y una valoración personal.

tarea		nombre	Descripción / valoración	captura
3D	animación	DAZ Studio	herramienta orientada a la creación y publicación de contenidos de animaciones de uso muy sencillo, con soporte para animación facial, composición de animaciones, importar, exportar, y *morphs*. Entre otras opciones	
	modelado	SketchUp	Herramienta diseñada para facilitar el modelado. No soporta animaciones, y las herramientas para tratar texturas son muy pobres. Capaz de acceder a librerías de modelos en Internet. Curva de aprendizaje muy pequeña.	
	visualizado	Google O3D + Firefox	API para gráficos 3d web.	

	shaders	Render Monkey	*Framework* orientado al desarrollo específico de *shaders* creado por AMD/ATI muy completo e intuitivo. Soporta HLSL, Cg y GLSL.	
web	Servidor http	Apache	Servidor web de uso muy extendido, ofrece rendimiento y fiabilidad.	
	Lenguaje scripts en servidor	PHP	Lenguaje de programación web para los scripts en el servidor.	
	Bases de datos	MySql	Base de datos eficiente y funcional.	
IDE	Desarrollo	Netbeans	Entorno de desarrollo integrado, con soporte para múltiples lenguajes y plataformas. En constante evolución. Dispone de *plugins* para aumentar su funcionalidad.	
	Diseño	Netbeans + UML plugin	*Plugin* integrado con Netbeans para diseño UML para lenguaje java.	
	Tests	Netbeans + phpUnit + Selenium	Herramienta para *tests* funcionales basados en phpUnit integrado en Netbeans.	
Planificación		Planner	Herramienta de planificación con soporte para diagramas de Gantt básicos.	

6 Fases de diseño, implementación y tests

> "Programar sin una arquitectura o diseño en mente es como explorar una gruta sólo con una linterna: no sabes dónde estás, dónde has estado ni hacia dónde vas"
> -- Danny Thorpe

Para el desarrollo de la aplicación se ha optado por seguir una metodología basada en el método ágil. Este, a grandes rasgos, consiste en seleccionar un requerimiento funcional de la lista , diseñar poco y rápido para cubrir las necesidades del mismo, e implementar código, a poder ser con calidad de producto acabado. Tras esto seleccionar un nuevo requerimiento y repetir el proceso. Entre ciclo y ciclo los requerimientos pueden cambiar. Y los ciclos deben ser cortos. Se prefiere la programación en pareja, o cuando menos el diseño en pareja o grupo.

Así pues, las fases de diseño e implementación del proyecto se han alternado durante varios ciclos para cubrir los requerimientos expuestos en el apartado con el análisis de requerimientos. Estos requerimientos durante el desarrollo han sufrido una serie de cambios comentados a continuación:

La creación de una web, multiusuario se mantiene. Las herramientas de administración de bibliotecas, obras de arte, salas de exposiciones y exposiciones también se mantienen.

La creación de una herramienta de gestión de *widgets*, finalmente, se ha descartado; los widgets, pasan a ser simplemente aplicaciones web enmarcadas en *iframes* html. Se ha creado no obstante un *widget* a modo de demostración que permite realizar una visita virtual a un museo en 3ra persona.

La gestión de usuarios se ha implementado parcialmente para evitar alargar el proyecto, es posible añadir nuevos usuarios pero no gestionar las cuentas/permisos existentes.

En total, se ha encarado el proyecto en 3 fases, una primera fase para crear la web y toda la infraestructura que toda web requiere; una segunda fase para implementar las herramientas 3d, y una tercera fase para el *widget* con la visita virtual. Este apartado ha sido organizado del mismo modo. A continuación se muestra el diagrama de casos de uso previsto inicialmente. Posteriormente se muestran los diagramas de secuencia y/o clases diseñados al inicio de cada fase seguidos de una descripción textual de los mismo y de un resumen de los problemas que se encontraron durante la implementación junto con la solución aportada para solventarlos en cada caso.

Nótese que en el método ágil se pretende un uso mínimo de diagramas UML-*Unified Modelling Language*-, con una función extremadamente orientada a la productividad: no se desean diagramas perfectos, integrados con el código fuente, se desea tener una idea general lo antes posible del diseño. Normalmente con un diagrama de clases y uno de secuencia escritos en paralelo suele considerarse suficiente. Todo esto implica que no necesariamente los nombres de las clases en los diagramas se acaban correspondiendo con los nombres en el código fuente final. Y no solamente eso, sino que , incluso, no se considera una mala praxis el hacer los diseños en papel o en una pizarra sin “pasarlos a

limpio". En este caso se ha optado por modelar los diseños en papel y posteriormente "pasarlos a limpio" por motivos de legibilidad en la documentación, no obstante, los nombres de las clases difieren de los finales.

6.1 Diagrama de Casos de Uso

> "Hay dos maneras de diseñar software: una es hacerlo tan simple que sea obvia su falta de deficiencias, y la otra es hacerlo tan complejo que no haya deficiencias obvias"
> -- C.A.R. Hoare

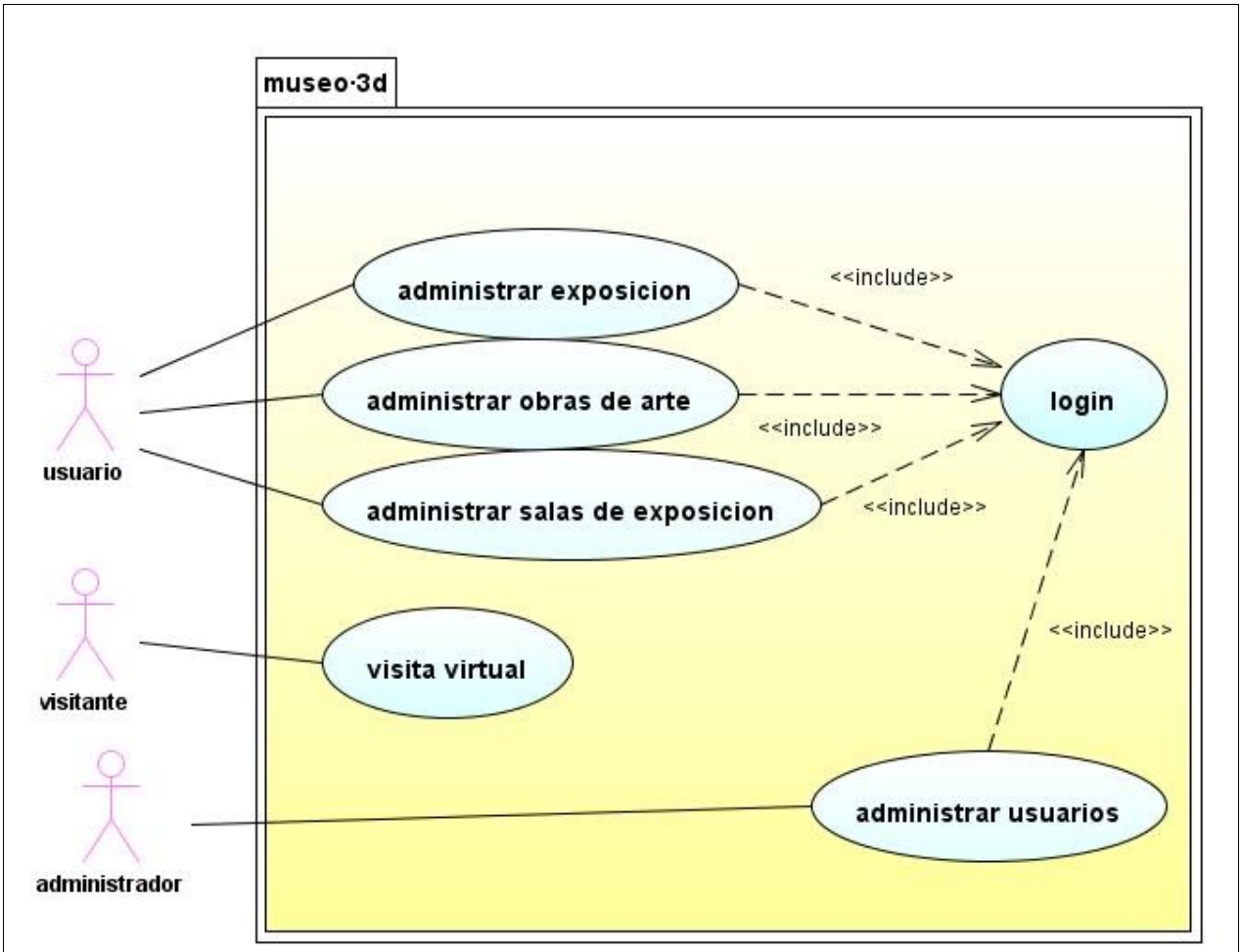


Figura 13: Diagrama de casos de uso

Como puede observarse en la figura 13, la aplicación prevee tres tipos de usuario, un usuario de administración global, uno para los propietarios de las galerías los cuales deberán acceder al sistema mediante *login*, y por último los visitantes, quienes solamente podrán realizar visitas virtuales a los museos.

El sistema consiste en una aplicación web, el diseño de la cual, se analiza en la fase 1, descrita a continuación.

6.2 Fase 1

> "El primer 90% del código corresponde al primer 90% del tiempo de desarrollo. El 10% restante corresponde al otro 90% del desarrollo"
> -- Tom Cargill

Para el diseño de la web que ha de sostener el sistema se ha optado por un patrón *Model-View-Controller* -de ahora en adelante MVC- como eje principal de diseño. El patrón MVC consiste en separar en tres capas el código fuente: la vista, que encapsula el modo en que se muestran los datos; el modelo que encapsula los datos y la capa con la lógica de la aplicación; y el controlador que es el encargado de organizar el flujo de ejecución de la aplicación.

Este patrón se ha ampliado usando los patrones *Front Controller* y *Two Step View*. *Front Controller* propone poner un único Controlador principal sirviendo todas las peticiones y enrutándolas al controlador correspondiente en cada caso.

Two Step View prefiere generar la vista en 2 fases una para preparar los datos y una para insertarlos en una plantilla.

En este entorno, cuando un cliente realiza una petición un controlador la toma (1), instancía un objeto de la capa del modelo (2), este a su vez prepara y procesa los datos necesarios tomándolos, cuando sea necesario, de la base de datos (3-4), y una vez estos datos están preparados, el controlador toma el mando nuevamente (5) y prepara una instancia de un objeto de la capa vista (6), normalmente una plantilla , la rellena con datos y esta plantilla es, en última instancia devuelta por el controlador al cliente en forma de página web(7), como puede observarse en la figura 14.

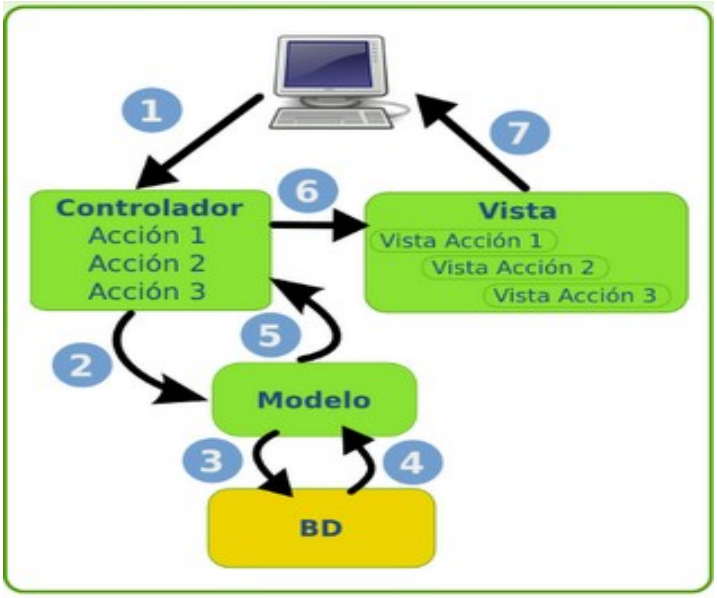


Figura 14: Arquitectura Model-View-Controller (MVC)

Para implementar este diseño se han creado tres clases básicas MVCController, MVCView y MVCModel que encapsular el código genérico común a un controlador, una vista y un modelo respectivamente. Se han creado varias clases que extienden estas para encapsular funcionalidades más específicas. Así, por ejemplo, Tenemos una clase llamada GaleriaModel que extiende la clase MVCModel y contiene datos relacionados con una galería de arte virtual; o tenemos 2 clases llamadas LoginController y IndexController que se encargan de las pantallas de login e índice respectivamente.

Para implementar el patrón *Front Controller* en el mismo directorio en que se encuentra el índice se ha preparado un archivo .htacces encargado de redirigir toda petición del cliente al archivo índex.php que a su vez crea una instancia de un objeto de tipo FrontController encargado de enrutar las peticiones. Éste último, implementa un patrón Factory para crear instancias de controladores según convenga.

Para la vista se barajó la opción de incluir algún motor de plantillas html conocido como smarty, pero finalmente se optó por implementar un soporte basado en lenguaje php.

Para el acceso a base de datos se ha previsto un diseño basado en el patrón *Data Acces Object* -DAO de ahora en adelante- el cual consiste en separar la capa con la lógica de la aplicación (el modelo) del acceso a base de datos, de forma que desde el punto de vista del modelo, el acceso a los datos es una API de la que no ha de conocer su implementación interna. Esto permite a largo plazo sustituir la base de datos por una diferente.

Para obtener conexiones a base de datos se ha optado por disponer de una clase que gestione el acceso a base de datos que se ha llamado ConnectionFactory. Esta clase implementa los patrones Factory y Singleton, los cuales permiten abstraer la creación de instancias de una determinada clase y asegurar la existencia de una única instancia de una determinada clase en un momento dado respectivamente. ConnectionFactory, contiene una única conexión a base de datos que sirve a través de su método getConnection(). El objetivo de este diseño es el de abstraer la obtención de la conexión. Esto permite que si más adelante, el número de usuarios crece, y por extensión, el número de conexiones crece, se puede muy fácilmente aplicar un patrón diferente como podrá ser el patrón *Thread Pool*.

La clase DAO ofrece una interfaz a la que realizar consultas sql mediante sus métodos *retrieve* o *update*, que retornan objetos de tipo DataAccesResult.

En la figura 15 se muestra un diagrama de clases con un resumen de las clases que intervienen en el acceso a datos.

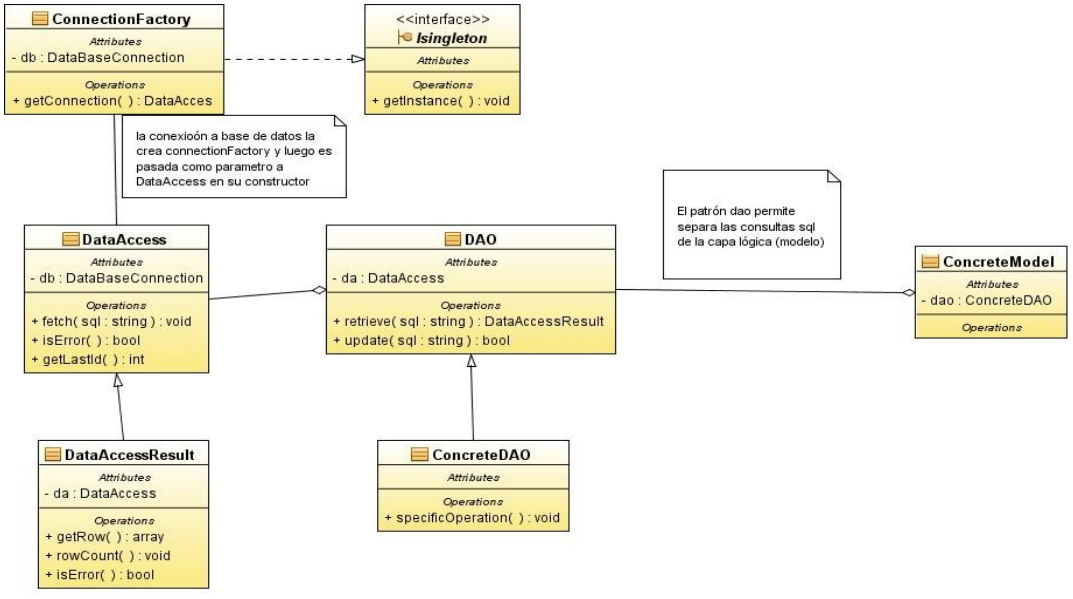


Figura 15: Diagrama de clases para el patrón Data Access Object (DAO)

En cuanto al resto de funcionalidades básicas para la *web* , podemos observar la creación de una clase de nombre SessionManager encargada de almacenar datos de sesión que permitiría a largo plazo albergar variables relacionadas con la sesión de un usuario, el lenguaje en que prefiere ver la interfaz de la *web*, etc, que en la versión presentada, no es usada. Implementa un patrón Singleton.

Otra clase menor, que también implementa Singleton, es la clase de configuración, encargada de almacenar valores de configuración como el usuario y *login* de la base de datos, los *paths* a las diferentes carpetas, etc.

La figura 16 muestra un diagrama de clases que resume el conjunto de clases implicadas en la fase uno del proyecto, dando una visión de conjunto del mismo.

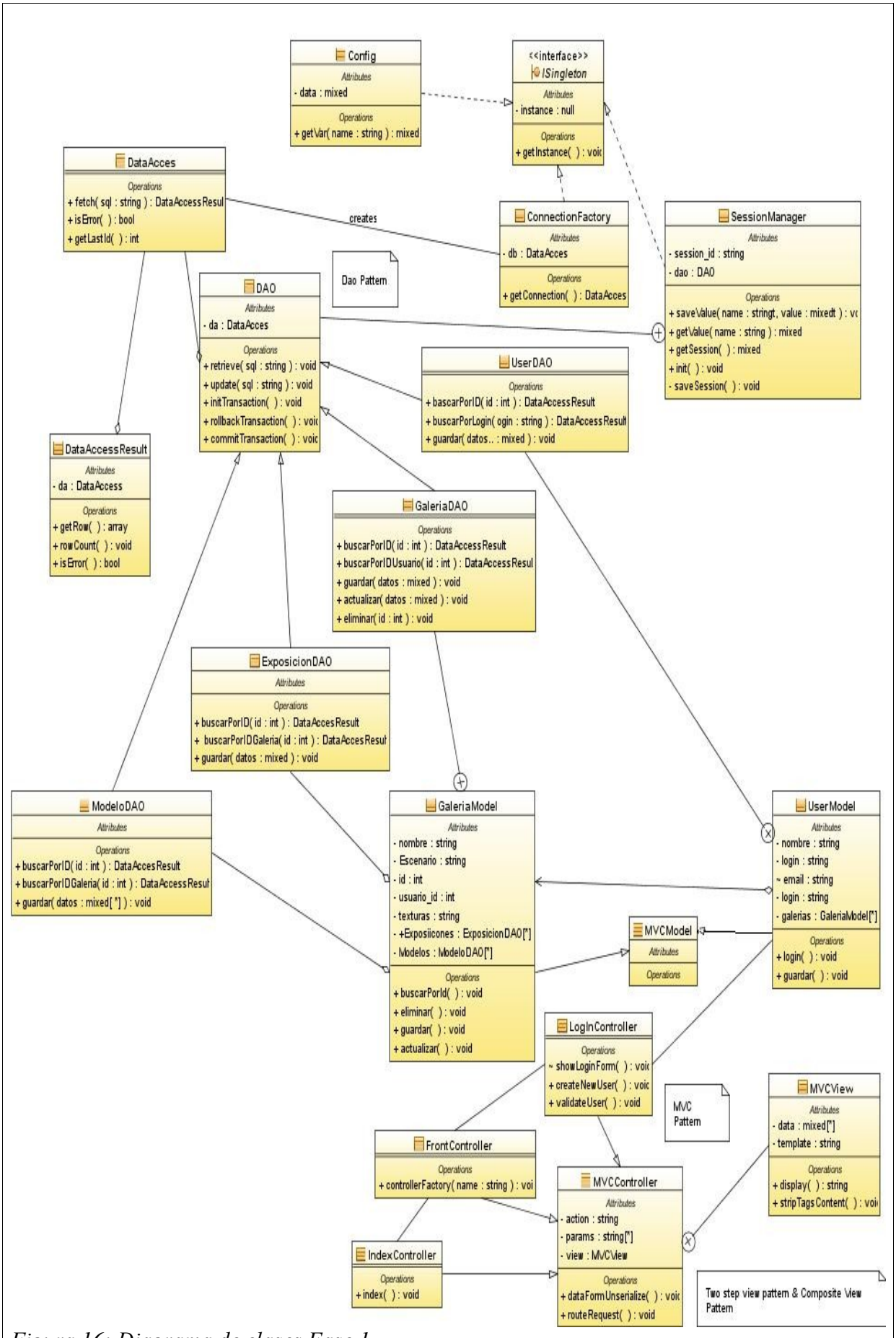


Figura 16: Diagrama de clases Fase 1

La *web*, pues, muestra al acceder a ella, un formulario de acceso, para ello el index cede el control al FrontController, éste enruta la petición al LoginController que prepara una vista con una plantilla con un formulario de *login* y esta es devuelta al cliente por el controlador. Todo el proceso se puede ver reflejado en el diagrama de secuencia de la figura 17.

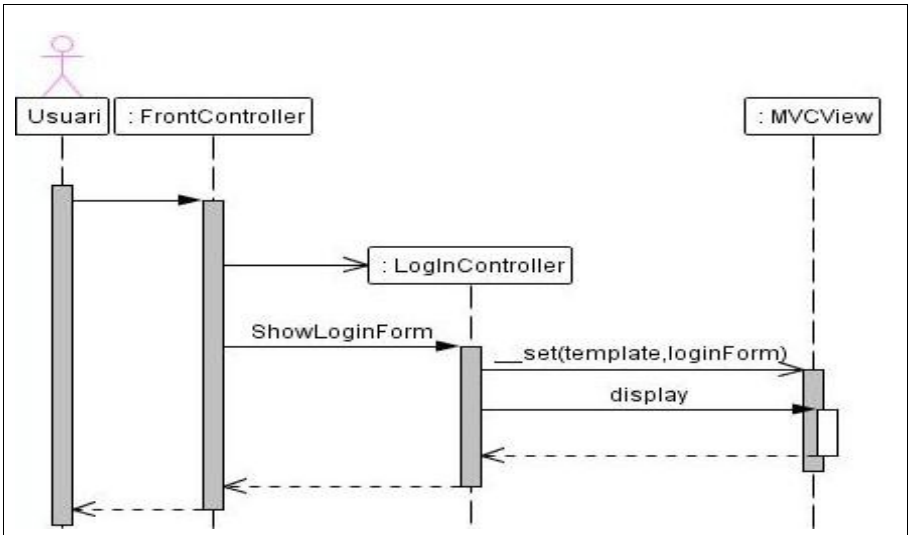


Figura 17: Diagrama de secuencia, mostrar el formulario de login

Cuando el usuario introduce su *login* y *password* una petición de login es enviada al servidor. El *.htacces* la redirige al index, este se la pasa al FrontController que a su vez la enruta al LoginController. El LoginController crea un objeto UserModel con los datos provistos; El objeto UserModel se conecta a la base de datos con una conexión creada por el ConnectionFactory. A través de la implementación del patrón DAO se obtiene un objeto de tipo DataAccesResult que contiene La información del usuario. Si el *login* es correcto el LoginController pasa el testigo al IndexControler quien se encarga de preparar una vista con la pantalla principal, que es finalmente devuelta al cliente. Este proceso se ve reflejado en el diagrama de secuencia de la figura 18.

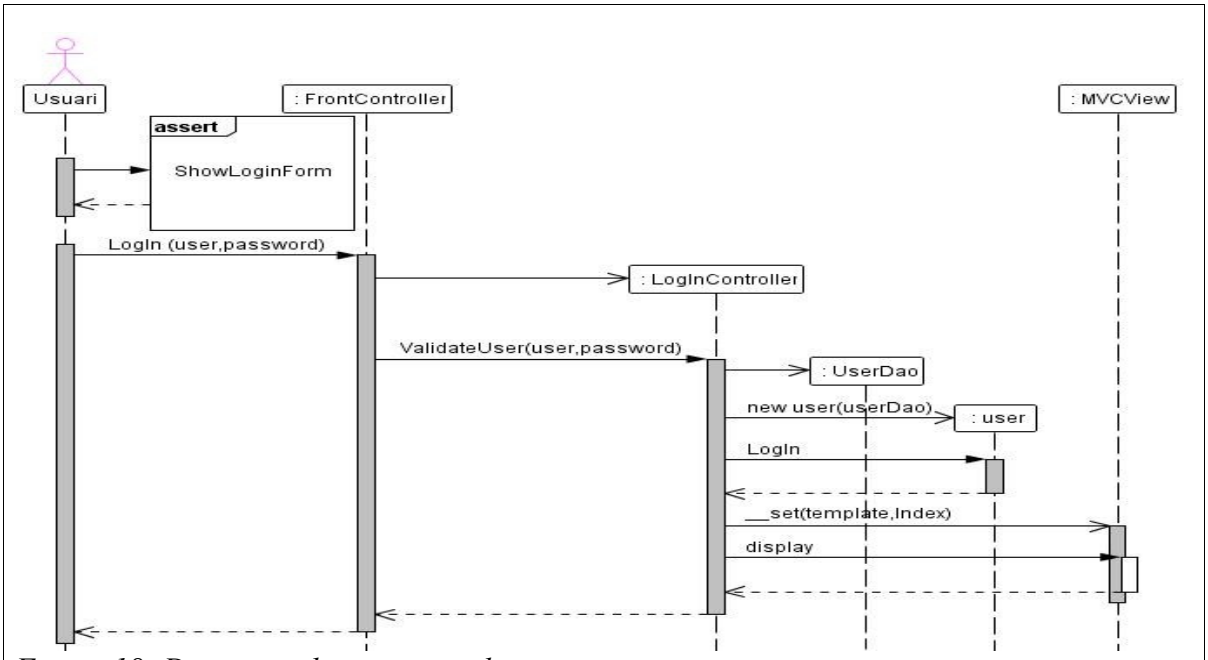


Figura 18: Diagrama de secuencia, login correcto

6.3 Fase 2

> . "Antes de que un software sea reutilizable debería ser utilizable"
> -- Ralph Johnson

Para la fase 2, se pretendía implementar el gestor 3d. Este estaría basado en el motor gráfico Google O3D. Pero esta API no esconde el hecho de ser de muy bajo nivel. Para realizar tareas abstractas se requieren muchas lineas de código repetitivo. Así pues era obligatorio crear un objeto que encapsulara dicho código y permitiera trabajar a mayor nivel de abstracción.

El diseño requiere de un paradigma orientado a objeto, que permita estructurar el código de forma reusable. El lenguaje de programación a tal efecto es Javascript, luego la primera decisión a tomar es que sistema de herencia usar; JavaScript es un lenguaje muy libre y no tiene una sintaxis predefinida para la herencia, no obstante se puede simular de diversos modos: Se puede usar una API externa como Moo Tools, implementar uno nuevo basado en el operador new, o se puede hacer uso del modelo basado en prototipos; éste último es el modelo escogido en el proyecto.

En el paradigma prototipado todos los objetos son funciones y todas las funciones son objetos. Toda función tiene un prototipo que se puede entender como un objeto indicando su morfología. Toda función es una variable y como tal puede ser modificada. El prototipo de la función puede por tanto ser modificado o sustituido completamente por el de otra, permitiendo que propiedades definidas (anterior o posteriormente) en la segunda sean accesibles inmediatamente en la primera: esta es la base en la que se sustenta el paradigma prototipado.

En el proyecto se ha creado un objeto llamado O3DApp del que hereda toda aplicación 3d. En la fase 2 el gestor hereda de O3DApp, y en la fase 3 el visor para visitas también deberá heredar de O3DApp. Así pues tanto GestorMuseO3D como VisorMuseO3D serán funciones vacías a las cuales les hemos sustituido su prototipo por una nueva instancia de O3DApp creada a través del operador *new*. Las propiedades de las mismas se añaden al prototipo para que todas sus instancias automáticamente puedan acceder a las mismas. Desde las herramientas que dispondrán, hasta la información de toda la geometría a renderizar pasando por referencias a objetos de la página como paneles, botones, etiquetas, etc. quedan incluidos.

Las diferentes herramientas se desea que sean reutilizables y requieren de un diseño similar, luego se deben mover a clases separadas. A tal efecto se ha creado un objeto llamado DinamicObject, y un objeto Init que son céntricos en el diseño de la aplicación. DinamicObject será todo objeto no estático, desde una cámara que debe poder moverse por un entorno tridimensional hasta un botón presentado como un ente dentro del área tridimensional y que deba responder a eventos de ratón pasando por un actor capaz de moverse según una animación predeterminada. Todo DinamicObject dispondrá de una serie de parámetros de configuración diferentes en cada caso. Para volver el código más consistente se deseaba reducir el número de parámetros a dos en todos los casos, uno sería una referencia a la aplicación que va a usar el objeto, y el otro sería un objeto de tipo Init

encapsulando todos los parámetros de configuración para los demás objetos. Luego todo objeto deberá disponer de un método de inicialización que arbitrariamente se ha decidido que se llame startup*NombredelObjeto*(template:O3DApp,initObject:Init).

Los DinamicObject, en muchos casos, deben responder a eventos. Para poder registrar un método como *callback* para un determinado evento es necesaria una técnica muy común en JavaScript llamada *closure*.

```
var quo = function (status)
{
      return
      {
            get_status: function (  )
            {
                  return status;
            }
      };
};

var myQuo = quo("amazed");
```

Texto 1: Ejemplo de closure

Veamos un ejemplo:

En el ejemplo quo es un objeto con un atributo privado status y un método público get_status que devuelve status. Es importante comprender que get_status tiene acceso a status directamente por el mero hecho de estar declarada dentro de quo. Cuando quo llega a su return y es eliminada, get_status continúa gozando de acceso a status, y no a una copia de status como cabe suponer, sino directamente a status. Al hecho de devolver un método con acceso a los atributos de un objeto determinado incluso cuando este se ha destruido se le llama *closure*.

Para definir *callbacks*, es decir métodos a los que invocar en respuesta a determinados eventos el uso de *closures* es ideal. Se puede declarar todos los eventos de ratón -onMoseDown, onMouseUp, onMouseWheel-, los de teclado -onClick-, los de renderizado -onRender- de esta forma. Y registrar todos estos métodos como *callbacks* tanto del sistema de gestión de eventos de Google O3D como del sistema de gestión de eventos del explorador al inicio de la aplicación.

Para el proyecto se ha llevado la idea un paso más allá creando una función para crear *closures* a petición llamada makeClosure que se usa siempre que es necesario un *closure*. Esta función permite declarar *callbacks* para los eventos directamente en el prototipo en vez de necesitar que sea fuera del mismo como exige la sintaxis usada en el ejemplo del texto 1. Esto permite mantener la reusabilidad y la consistencia con un entorno orientado a objeto con soporte de herencia basada en prototipos.

La figura 19 muestra un diagrama de clases que resume el diseño, (del cual se ha excluido O3DApp, por su tamaño principalmente, y por aportar luz extra al diseño).

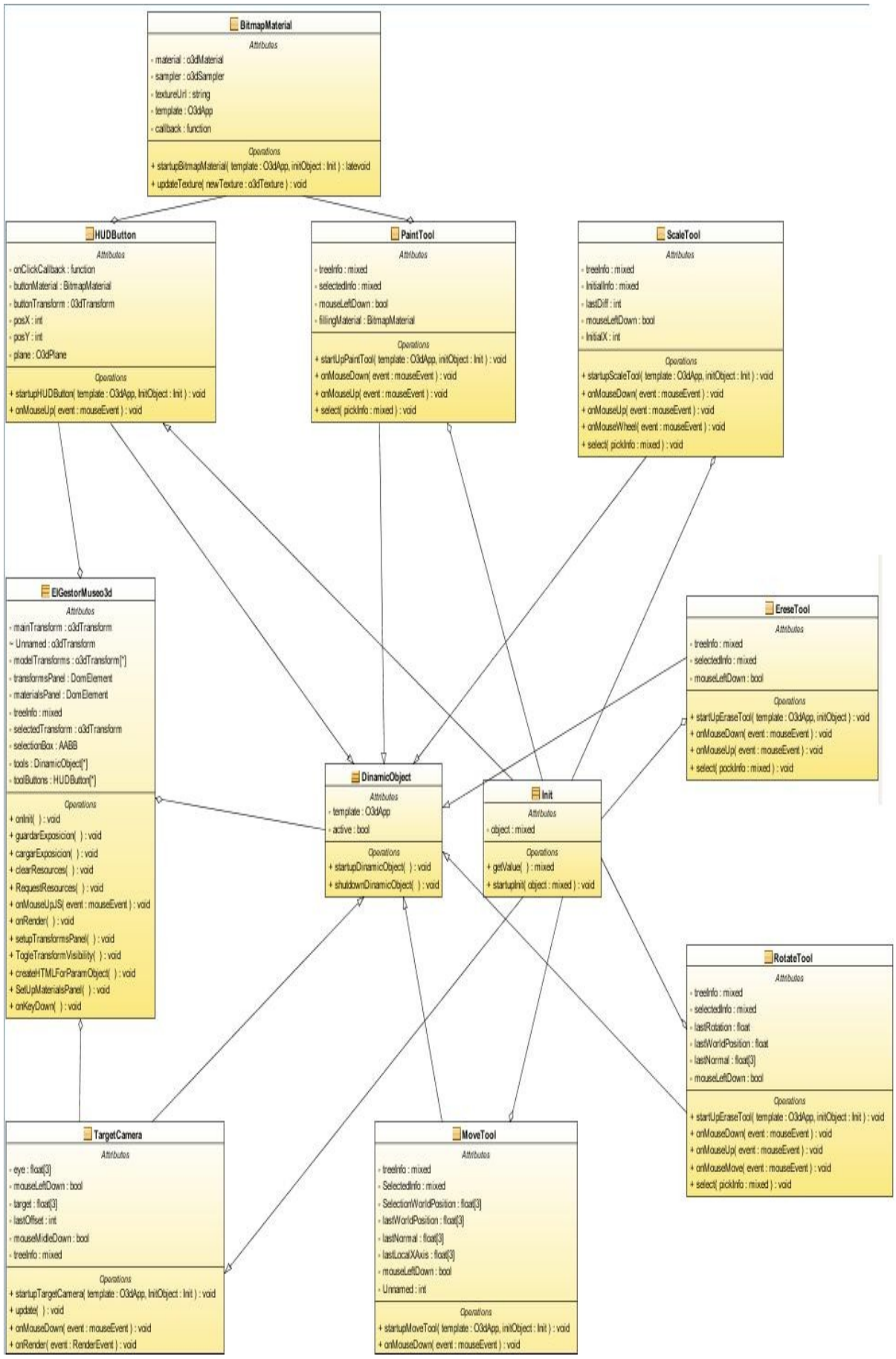


Figura 19: Diagrama de clases, Fase 2

Una vez se ha tenido un diseño general en mente la dificultad de su implementación, se enmarca en solventar las limitaciones técnicas de la tecnología a disposición y en buscar ideas que mejoren la usabilidad de la aplicación.

Google O3D es la tecnología principal de la que depende el proyecto. Veamos en que consiste, sus limitaciones, qué uso se hace de sus posibilidades en el proyecto y que opciones se han dejado para hipotéticas ampliaciones.

Google O3D es un *plugin* para el explorador que permite renderizar escenarios tridimensionales en un explorador web. Esta implementado mayormente en C++ y se encarga de exponer a JavaScript una API que permite hacer uso de las dos librerías de aceleración gráfica por *hardware* por excelencia: Microsoft Direct3D y OpenGL. La figura 20 muestra la arquitectura de Google O3D, se muestra en azul la aplicación cliente (El gestor en el caso de este proyecto), en verde el *plugin* en si mismo, en lila las librerías de aceleración y en amarillo el *hardware* gráfico del cliente .

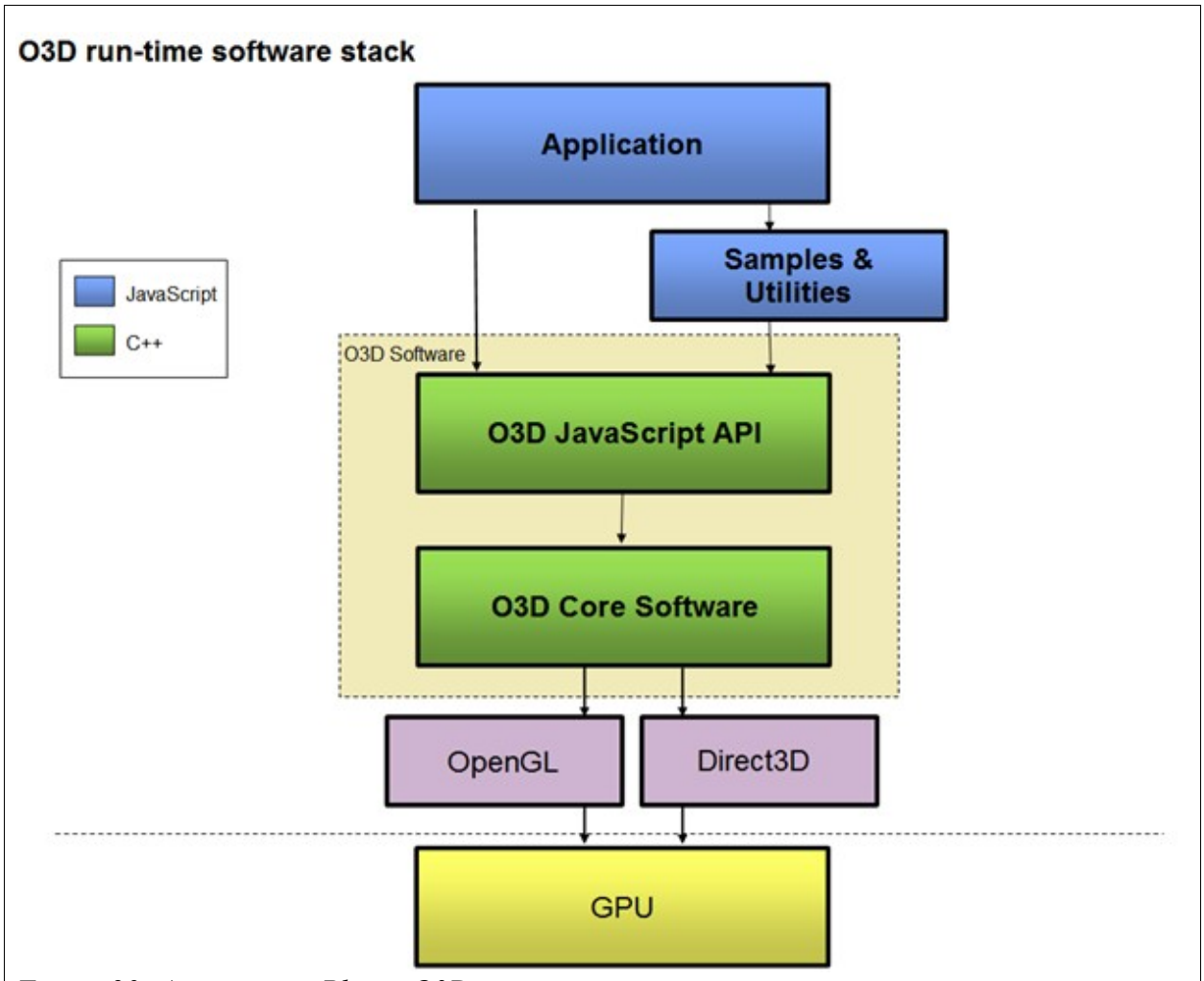


Figura 20: Arquitectura Plugin O3D

Históricamente el *hardware* gráfico disponía de un procesador de vértices capaz de realizar una determinada serie de operaciones básicas sobre un conjunto de datos (vértices) que se enviaban a la tarjeta gráfica constantemente. Esto implicaba usar los

llamados *fixed function pipelines*, que no eran más que conjuntos de funcionalidades que venían predefinidas en las librerías gráficas como OpenGL o Direct3D. El hecho de usar estas funciones fijas predefinidas daba muy poco margen a la creatividad: la tarjeta gráfica se encargaba de las transformaciones, de la proyección, del *clipping* (eliminar figuras que no entraban dentro del cono de proyección) y por último del rasterizado (transformar los polígonos en pídeles) . La única tarea del programador era pues , preparar los vértices y especificar que operaciones quería usar de entre las disponibles.

Pero con la llegada del *hardware* gráfico programable llegó una revolución: los *programmable pipelines*. Los procesadores gráficos , permitirían de ese momento en adelante escribir pequeños programas que la tarjeta gráfica interpretaría y ejecutaría en cada una de las fases del renderizado , permitiendo así una mayor libertad para el programador.

Lo primero en aparecer fueron los llamados *vertex shaders*, que permitían hacer transformaciones en las coordenadas de los vértices en la tarjeta gráfica, sus aplicaciones variaban desde animaciones (*skinning*) hasta automatizar el comportamiento de fluidos como superficies de agua “ondulante” o banderas que se mueven al son del viento.

La verdadera revolución tardó en aparecer un poco más. Fueron los *fragment shaders* también llamados *pixel shaders*, que permitían aplicar efectos y variar las propiedades de los *píxeles*. Permitiendo efectos aplicables a toda la pantalla como mascaras de color sepia, detección de contornos, efectos de iluminación , *bump mapping, normal mapping, cell shading* y un largo etcétera.

La tercera y ultima revolución son los *geometry shaders*, los cuales permiten generar geometría nueva en tiempo real, permitiendo efectos variados, por ejemplo a partir de una textura representando unos escalones y un único polígono para toda una escalera, generar en tiempo real toda la geometría necesaria con todos los escalones.

Los *geometry shaders* se aplican inmediatamente después de los vertex shaders. Tras estos se aplican las labores genéricas de renderizado hasta el rasterizado, tras este se aplican los *Fragment shaders* generando la imagen final.

En sus inicios, los *shaders* se escribían directamente en ensamblador (lenguaje de muy bajo nivel). Pero temprano aparecieron los primeros lenguajes de programación de *shaders*. Con el tiempo los lenguajes que se han vuelto estándar de facto para escribir *shaders* son HLSL, GLSL y Cg.

HLSL es el lenguaje para *shaders* de Direct3D, creado por Microsoft. Goza de soporte perfecto por todo el *hardware* gráfico y su única flaqueza es que solamente puede ser usado bajo los sistemas operativos de Microsoft: Windows.

El segundo es el lenguaje descrito por la especificación de OpenGL, dispone de diversas implementaciones y el soporte de *hardware* es dispar. En general Nvidia suele dar buen soporte, pero el resto de fabricantes priorizan HLSL claramente.

Cg, es un lenguaje creado por Nvidia y solamente esta soportado por su *hardware*.

Veamos un ejemplo de *shader* escrito en HLSL. Consiste en una figura a la que se le

aplican los reflejos de luz que provienen de un entorno simulado con una textura cubica aplicada a una semiesfera que envuelve la figura a la que se aplica el efecto con los reflejos. En este caso el *vertex shader* no ha de realizar ninguna tarea salvo pasar al fragment shader los datos que este necesitara.

Veamos el código del *vertex shader* en cuestión:

```
float4x4 matViewProjection;
float4 vViewPosition;
struct VS_OUTPUT {
      float4 Pos:POSITION;
      float3 Normal:TEXCOORD0;
      float3 View:TEXCOORD1;
};
VS_OUTPUT vs_main(float4 inPos:POSITION,float3 inNormal:NORMAL){
      VS_OUTPUT Out;
      Out.Pos=mul(inPos,matViewProjection);
      Out.Normal=normalize(inNormal);
      Out.View=normalize(vViewPosition-inPos);
      return (Out);
}
```

Texto 2: Ejemplo de vertex shader HLSL

Veamos ahora el código del *fragment shader*:

```
samplerCUBE Environment;
float4 ps_main(float3 inNormal:TEXCOORD0,float3 inView:TEXCOORD1){
      inNormal=normalize(inNormal);
      inView=normalize(inView);
      float3 refVect=reflect(-inView,inNormal);
      float4 color = texCUBE(Environment.refVect);
      return (color);
}
```

Texto 3: Ejemplo de Fragment Shader HLSL

En la figura 21 puede verse el resultado de dicho *shader*:

Figura 21: Resultado del shader de reflexión HLSL

O3D, dispone de un *pipeline* programable con soporte para *vertex shaders* y *fragment shaders*. Los *geometry shaders* no están disponibles. El lenguaje de *shaders* de Google O3D es una mezcla entre HLSL y Cg, permitiendo fácilmente importar *shaders* escritos para estos siempre que no usen algunas de sus características más complejas. Una de las limitaciones reside en el hecho de no poder importar *shaders* llamados multipase, por ser capaces de realizar en un pase operaciones de renderizado que normalmente se harían en varios pases, como por ejemplo cuando un shader necesita crear datos intermedios y guardarlos en un *buffer* en memoria para luego usar el color de los píxeles como referencia para hacer una determinada acción u otra. Estos *shaders* no son compatibles con O3D, pero siempre se puede simular el funcionamiento dividiéndolo en fases y haciendo uso de *buffers* como el FrameBuffer o el StencilBuffer.

Veamos un ejemplo de *shader* en el lenguaje de O3D:

```
float4x4 worldViewProjection : WORLDVIEWPROJECTION;
float4 color;
struct VertexShaderInput { float4 position : POSITION;};
struct PixelShaderInput { float4 position : POSITION;};
PixelShaderInput vertexShaderFunction(VertexShaderInput input) {
  PixelShaderInput output;
  output.position = mul(input.position, worldViewProjection);
  return output;
}
float4 pixelShaderFunction(PixelShaderInput input): COLOR {
  return color;
}
// #O3D VertexShaderEntryPoint vertexShaderFunction
// #O3D PixelShaderEntryPoint pixelShaderFunction
// #O3D MatrixLoadOrder RowMajor
```

Texto 4: Vertex Shader y Fragment Shader O3D, efecto de color sólido

Figura 22: Resultado del shader de color sóido O3D

Este *shader* renderiza una figura con un color sólido como puede verse en la figura 22.

Para el proyecto, se han usado *fragment shaders* para texturizar con y sin sombreado , *vertex shaders* genéricos y *vertex shaders* para efectos de *skinning* (animación de actores

en la fase 3).

Del renderizado final se encarga el *hardware* gráfico, pero antes la aplicación debe aportar los datos necesarios a tal efecto, incluyendo y no limitado a , texturas, geometría , animaciones, mapas de normales, *bump maps*, semillas generadoras de números aleatorios para generar texturas proceduralmente, colores, matrices para especificar filtros, etc. A continuación se detalla como O3D se encarga de proveer al *shader* con todos estos datos.

O3D ofrece una serie de clases con las que componer el escenario. Estos objetos están organizados en un grafo llamado TransformGraph. Toda aplicación 3d deberá tener un TransformGraph. Cada objeto tendrá una transformada y una serie do objetos que lo componen cada uno con su transformada respectiva y así sucesivamente.

Un nodo del árbol tendrá una o más *Shapes*. Cada *shape* se crea por separado de su transformada, se registra a esta y esta le da un espacio de coordenadas local. Cada *shape* dispone de una o más *Primitives*. Cada *Primitive* tendrá un único Material que contendrá la textura y el efecto a aplicar sobre la misma. Los efectos se crean por separado de su material y se le registran.

La figura 23 muestra como se compone un TransformGraph.

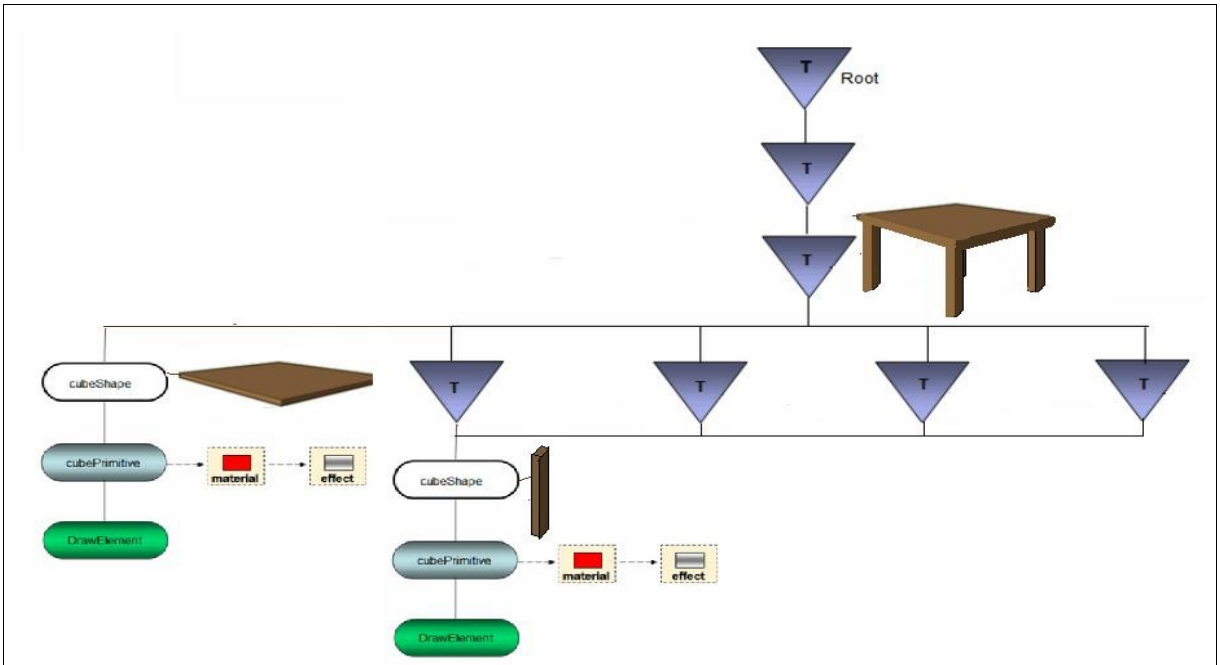


Figura 23: TransformGraph

.Por otro lado O3D exige la creación de al menos un segundo grafo llamado RenderGraph el cual contiene información de estado del motor usada en el renderizado, así como un nodo con información de la vista(cámara), un nodo que recorre el transformGraph para obtener la geometría a renderizar y una serie de nodos, que se pueden componer de diversas maneras, tantos como pases requiera el renderizado. En general se suele usar un pase para renderizar materiales opacos, y un pase para materiales transparentes que requieren de algoritmos de *z-sorting* y por consiguiente son mucho más lentos.

La figura 24 muestra como es un RenderGraph.

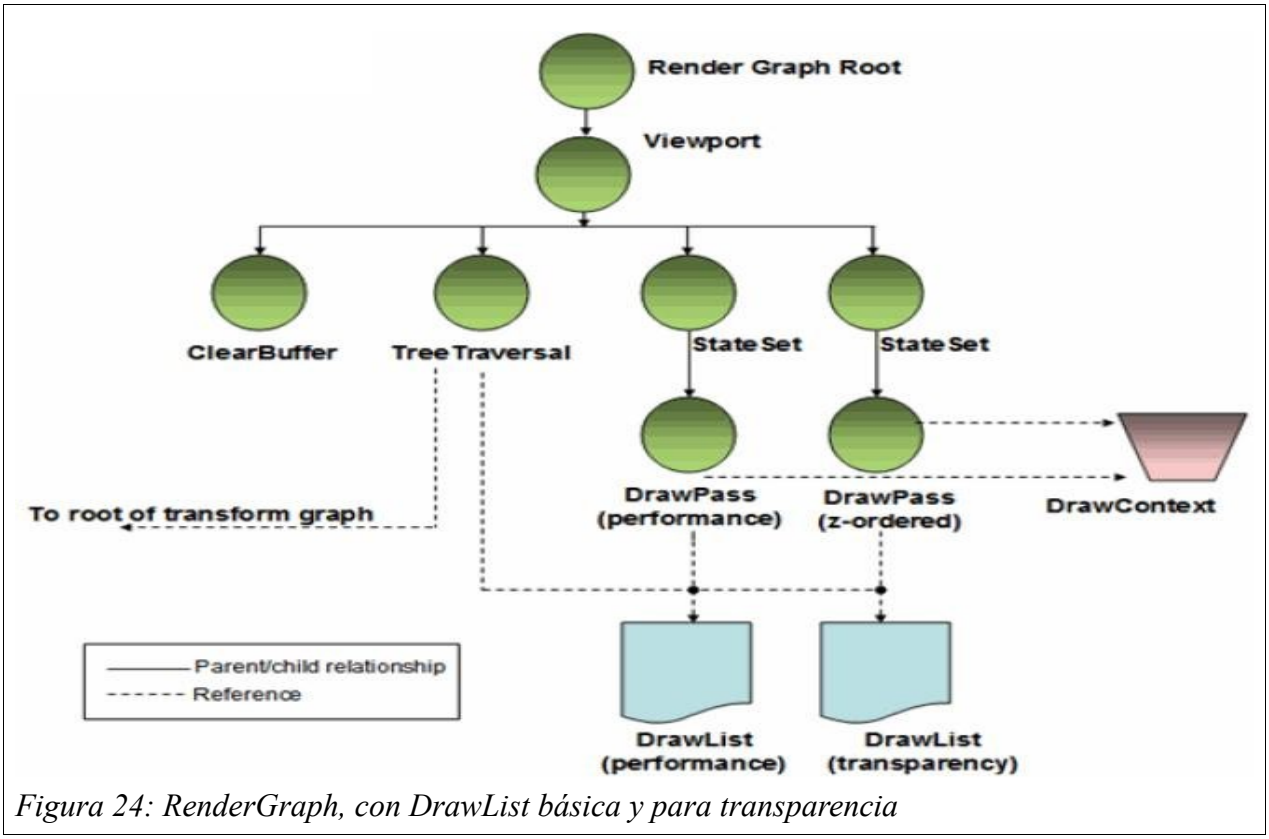


Figura 24: RenderGraph, con DrawList básica y para transparencia

Una vez se conoce en que forma se deben estructurar los datos para los *shaders* se puede empezar a pensar en la aplicación. Para presentar una serie de herramientas se ha decidido que a nivel de interfaz de usuario sea lo más intuitivo posible, por lo tanto es una buena idea imitar conceptos a los que el usuario esta acostumbrado a encontrar en otras aplicaciones como barras de herramientas, barras de estado, botones, etc. Para cambiar de herramienta solo hará falta hacer clic en el botón correspondiente.

Se han creado varias herramientas:

- MoveTool para trasladar figuras, estas se orientan automáticamente según la orientación del plano que contiene al polígono que se encuentra bajo el puntero del ratón en cada momento. Funciona Siguiendo el conocido sistema de *Drag & Drop* para facilitar al usuario su uso.

- RotateTool para rotar figuras alrededor de una vertical imaginaria tomando como referencia el vector director del plano que contiene al polígono que se encuentra bajo el puntero del ratón en cada momento.

- ScaleTool para escalar figuras de forma proporcional en los tres ejes. Tiene dos opciones una basada en *Drag & Drop* que toma el desplazamiento lateral del ratón como referencia para generar un factor de escala, y una segunda forma de trabajo basada en la rueda del ratón.

- PaintTool, el bote de pintura, herramienta para volcar texturas sobre el lienzo de un cuadro.

- EraseTool, la goma de borrar, herramienta para eliminar transformadas del TransformGraph, que permite pinchar con el puntero del ratón la transformada que se desea eliminar.
- CameraTool , herramienta de cámara. Toma un punto en el espacio como foco de atención. Permite hacer *zoom* usando la rueda del ratón, cambiar el foco haciendo clic sobre algún punto de la geometría. Al arrastrar el ratón permite rotar la cámara alrededor del foco. Simula el comportamiento de un *trackball*, limitado a solamente media esfera para que el movimiento del ratón sea lo más intuitivo posible.

O3D renderiza con máxima prioridad, y cualquier botón que se quiera mostrar al usuario debe colocarse, o bien fuera del área 3d, o bien crear una superficie en 3d y colocarla justo delante de la cámara, de modo que siempre sea visible y responda a los clics del usuario simulando así el comportamiento de un botón.

En el proyecto se ha optado por la segunda alternativa. Para su implementación, se crean 2 escenarios diferentes cada uno con un árbol de transformadas diferente, uno tiene una vista con proyección en perspectiva (realista, a mayor profundidad los objetos se ven más pequeños) y el escenario 3d; el otro con una proyección ortográfica (los objetos no se vuelven pequeños con la distancia,suele usarse en herramientas CAD 3d) con los botones a modo de *Heads Up Display* -HUD de ahora en adelante-, que contendrá un plano alineado con la cámara sobre el que renderizar los botones, y mayor prioridad de renderizado para asegurar que siempre se dibuje por encima del escenario.

La ilustración 1 muestra la diferencia entre las proyecciones en perspectiva y ortográfica.

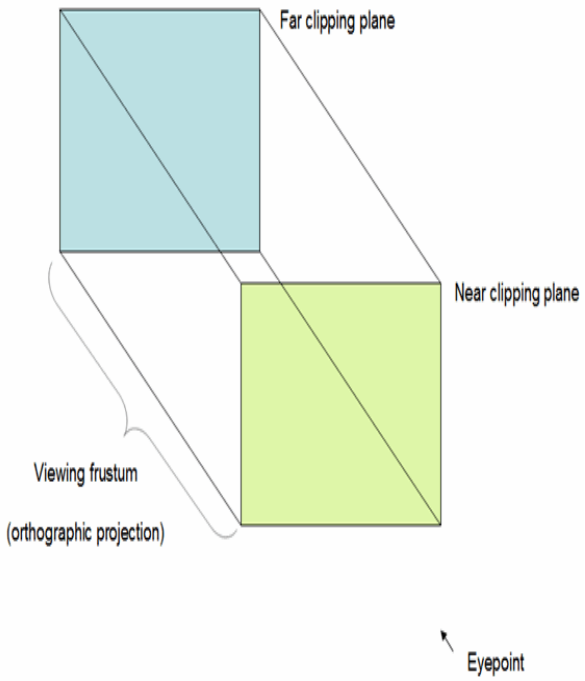


Figura 25: Proyección ortográfica

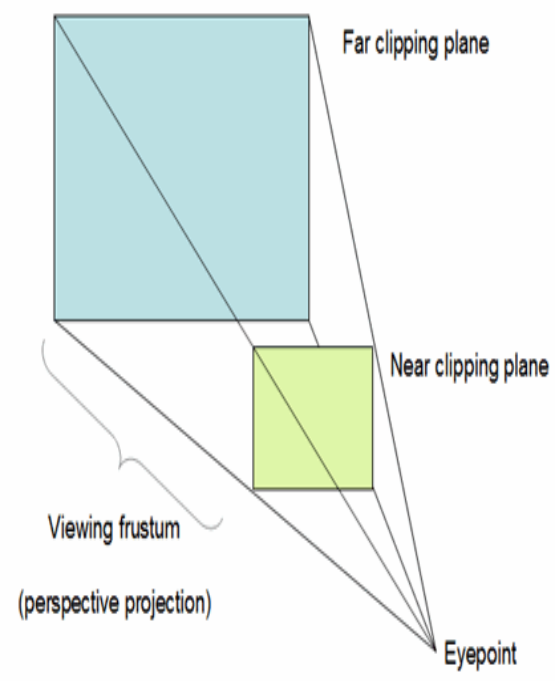


Figura 26: Proyección en perspectiva

Ilustración 1: Comparativa entre proyecciones ortográfica y en perspectiva

Otro ejemplo donde, se ha optado por buscar mecanismos reconocibles para el usuario ha sido la herramienta de PaintTool. Se pretendía poder cargar un único marco de cuadro y luego con un bote de pintura como el usado en las herramienta de diseño 2d "rellenar" el lienzo con una imagen preseleccionada por el usuario. Esta herramienta requiere de un "selector de relleno" al igual que en de diseño herramientas 2d tales como Photoshop.

Para seleccionar las imágenes no obstante se ha preferido subir todas las imágenes de una tacada al servidor, comprimir-las en el servidor y descargar el conjunto comprimido con todas las imágenes al inicio de la aplicación. Esta decisión obedece a una restricción impuesta por el protocolo *Hyper Text Transfer Protocol* -HTTP- que no permite más de 2 peticiones asíncronas al mismo tiempo; esto a su vez provoca que si se intentan pedir muchas imágenes separadas se producen errores en el *plugin* O3D sobre los que no se tiene control y que han sido el mayor problema en el desarrollo de la aplicación.

El otro gran problema es el hecho de que para diferentes aplicaciones 3d se suelen escoger arbitrariamente los ejes que representan profundidad, altura y anchura respectivamente, con lo que muchas veces se modela una figura en una aplicación externa, y al cargarla en O3D la figura aparece volteada varios grados alrededor de algún eje, y se debe volver a la aplicación de diseño (o herramienta de exportación) y rotar la figura varias veces hasta dar con la orientación que requiere O3D, ralentizando el desarrollo de la aplicación.

Otro factor que ralentiza notablemente el desarrollo 3d es la complejidad de las estructuras de datos con las que se trabaja.

Todo ello fuerza a dedicar mucho tiempo a tareas de *debug*. Por ello para el proyecto se han añadido algunos paneles al lateral de la web para dar información de *debug* sobre materiales y transformadas. Estos se podrían ampliar para modificar dichos materiales y transformadas pero se optó por no realizar dichas ampliaciones. El acceso a estos paneles es público si bién en un futuro sería deseable que fuera restringido.

La figura 27 muestra una captura de pantalla del gestor.

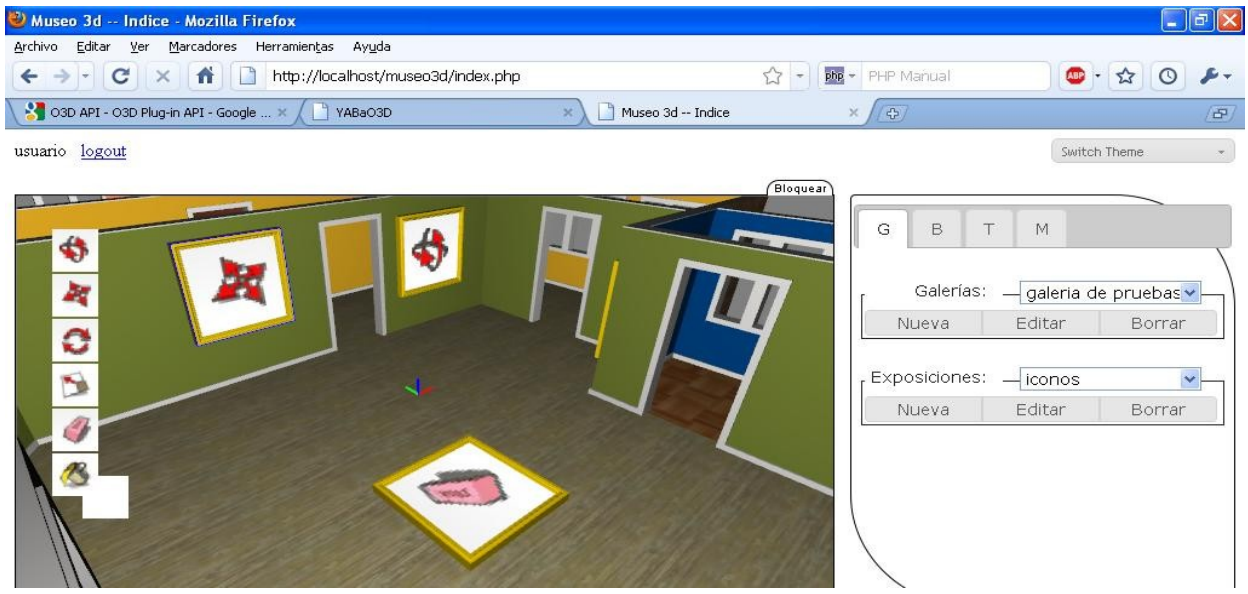


Figura 27: Captura de pantalla del gestor de museos

6.4 Fase 3

> "Es importante destacar que ningún ingeniero software con ética consentiría escribir un procedimiento llamado DestruirBaghdad. Su ética le obligaría a escribir un procedimiento DestruirCiudad, al que se pasaría el parámetro Baghdad"
> -- Nathaniel S. Borenstein

La tercera fase de implementación ha sido la más rápida de las tres, en gran parte gracias al esfuerzo hecho en la segunda para diseñar código reutilizable. En esta fase solamente se han creado dos objetos nuevos VisorMuseo3D y Actor.

Gran parte de la funcionalidad de cargar museos y exposiciones se implementó ya en la segunda fase. La respuesta a eventos también estaba hecha. Por todo esto el Visor solamente es un contenedor que contiene una serie diferente de atributos respecto del gestorMuseo3D. Las mayores diferencia residen en el hecho de no disponer de las herramientas del gestor y de disponer de un Actor.

Un actor es un avatar tridimensional capaz de caminar por el museo, controlado por el usuario con las teclas de dirección del teclado, capaz de detectar las paredes y que se detiene ante una. La cámara sigue al avatar colocándose tras su espalda. El usuario puede hacer girar la rueda del ratón para que la cámara se acerque hasta el avatar lo suficiente como para que este no bloquee la vista, de forma que el usuario pueda admirar las obras de arte expuestas en la exposición. El avatar es capaz de caminar adelante y atrás usando la misma animación marcha atrás y adelante respectivamente.

Dada la alta cantidad de datos que implica toda la geometría, para el actor junto con todas sus texturas y animaciones , se ha tenido que tener mucho cuidado con el tamaño de las texturas pues O3D tiene un limite máximo, y no conforme con eso , el hecho de cargar un modelo para un actor que solamente el actor supera las 15Mb significa que la descarga de dicho modelo puede ralentizar la carga de la página sensiblemente. Para minimizar este efecto, que se aprecia más sensiblemente cuanto más lenta es la conexión del cliente, se optó por una carga asíncrona que permite cargar la página antes y luego mostrar algún mensaje al usuario con el porcentaje de carga del modelo.

La figura 28 muestra un diagrama de clases que lo resume.

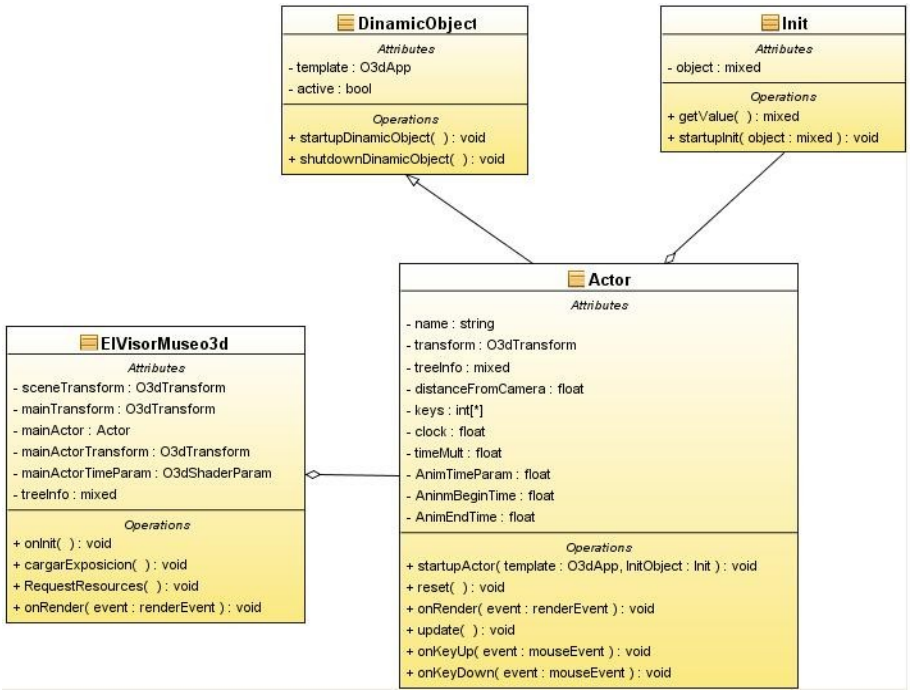


Figura 28: Diagrama de clases Fase 3

La figura 29 muestra una captura de pantalla con el visor,

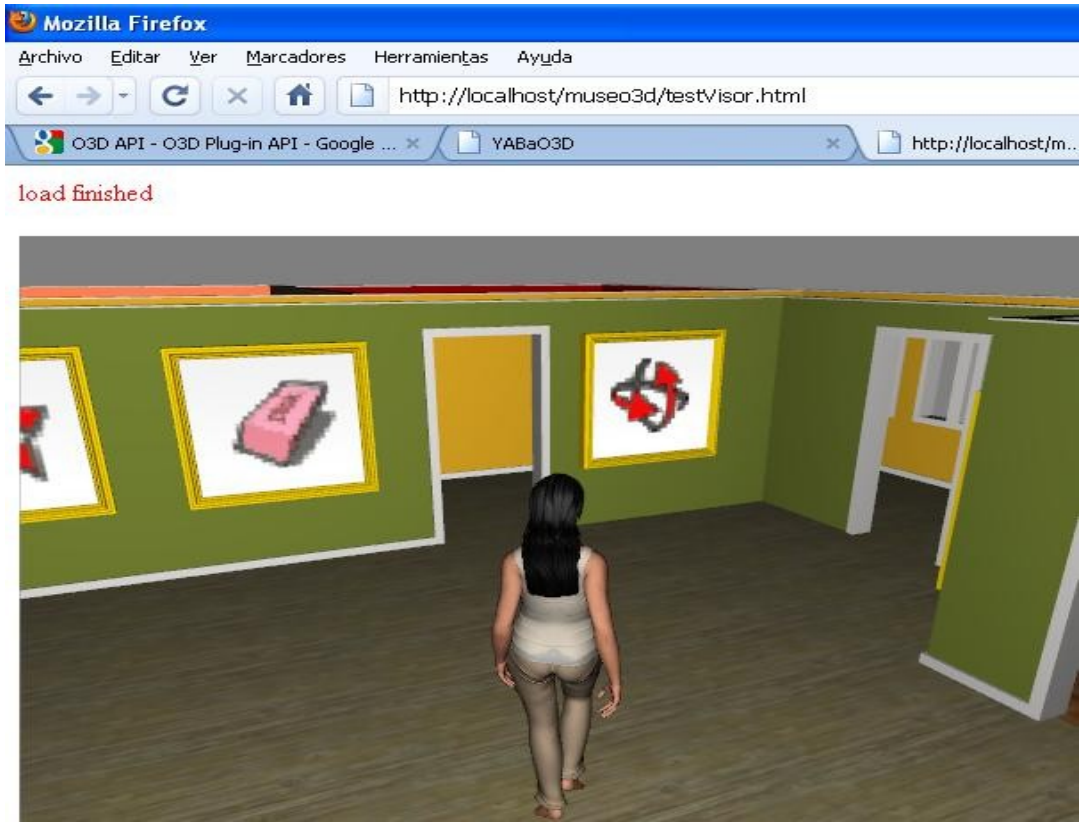


Figura 29: Captura de pantalla del visor de museos

6.5 Tests

> "No te preocupes si no funciona bien. Si todo estuviera correcto, serías despedido de tu trabajo"
> -- Ley de Mosher de la Ingeniería del Software

Testear una aplicación sin acabar no es la mejor de las ideas. No obstante, a modo de muestra se ha incorporado un único test funcional basado en Selenium que es un *Framework* para automatizar *tests* en el desarrollo web. El test en cuestión testea el correcto *login*. No se han creado más *tests* por falta de material a testear por un lado y falta de tiempo para escribir más tests por el otro.

Para ello se ha usado un *plugin* para el explorador Firefox llamado SeleniumIDE que permite grabar las actividades de un usuario en una determinada página web y generar un archivo que se toma como base para un test de Selenium. Luego con un *plugin* que permite integrar Selenium, y PHPUnit en NetBeans se pueden crear tests que realizan automáticamente las mismas acciones que se han grabado con SeleniumIDE y al mismo tiempo se realizan una serie de aserciones para determinar el correcto funcionamiento de la página, constituyendo así un excelente sistema de *tests* automatizado, capaz de ejecutar la misma tarea un número indeterminado de veces sin necesidad de presencia o acción humana y generando un informe con los errores detectados, es decir las aserciones que no se han cumplido. Todo ello se invoca desde el NetBeans.

6.6 Bugs conocidos y funcionalidades incompletas.

> "Depurar es al menos dos veces más duro que escribir el código por primera vez. Por tanto, si tu escribes el código de la forma más inteligente posible no serás, por definición, lo suficientemente inteligente para depurarlo"
> -- Brian Kernighan

Para la consecución del proyecto, a lo largo del desarrollo del mismo, se han dejado pendientes varias tareas descritas a continuación:

- No hay ningún perfil para administrador.
- No se ha `probado la aplicación lo suficiente. Más notoriamente no se han desarrollado *tests* unitarios para las clases implementadas.
- No se ha comprobado el correcto funcionamiento de la página web en ningún otro explorador a parte de Mozilla Firefox, esto incluye a Chrome, Internet Explorer, Opera, Konqueror o Safari entre otros.
- No se aplica la detección de colisiones en el visor cuando el actor camina hacia atrás, y por lo tanto atraviesa las paredes.

- No se ha encontrado ningún sistema de documentación automático para el código JavaScript y por ende no se ha generado tal documentación.

6.7 Documentación

> "El buen código es su mejor documentación"
> -- Steve McConnell

Para generar la documentación se ha hecho uso de PhpDocumentor, el cual a partir de los comentarios en el código es capaz de crear automáticamente toda la documentación para el código php. La documentación se encuentra en el apartado de anexos en el CD adjunto.

7 Conclusiones

> "Es mejor cojear por el camino que avanzar a grandes pasos fuera de él. Pues quien cojea en el camino, aunque avance poco, se acerca a la meta,mientras que quien va fuera de él, cuanto más corre, más se aleja."
> --San Agustín

Esta sección contiene una descripción con los objetivos del presente proyecto logrados y no logrados. Seguidamente una breve exposición de las conclusiones que se desprenden del proyecto y para finalizar una relación de posibles ampliaciones.

Como objetivos principales del proyecto se proponían la creación de un gestor de museos virtuales que permitiera realizar visitas virtuales a dichos museos. En mayor o menor medida se han logrado cumplir todos los objetivos marcados, si bien ello a implicado hacer una serie de concesiones en cuanto a funcionalidades, y cantidad de efectos gráficos que O3D permite. Respecto de la planificación se ha logrado completar el proyecto con un margen de tiempo cercano al previsto como muestra la figura con la planificación al finalizar el proyecto. La codificación de la fase 2 resultó necesitar muchas más horas de las previstas, no obstante no se requirieron horas para el modelado del museo 3d para la versión de demostración, pues se encontraron varios modelos hechos a través de Internet compensando el número de horas extra necesitadas durante el desarrollo.

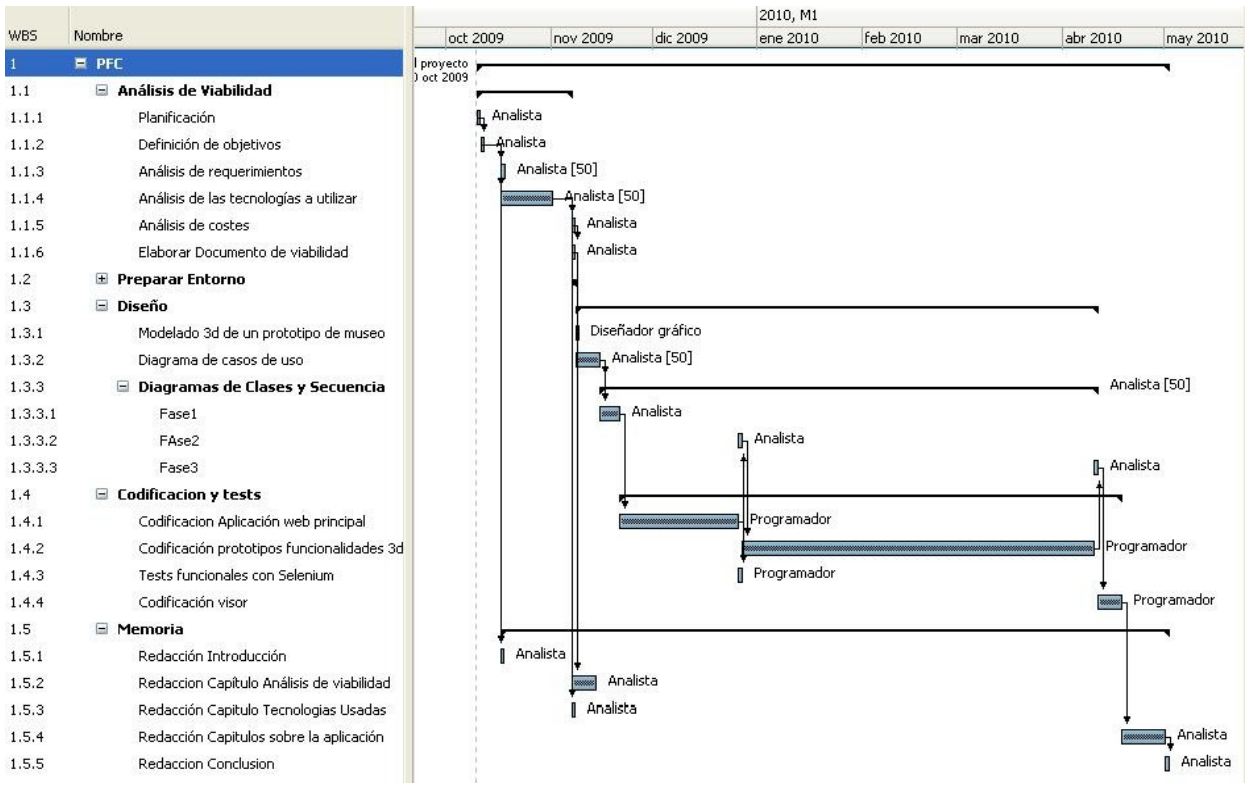


Figura 30: Diagrama de Gantt, muestra la planificación al termino del proyecto

A continuación se exponen las conclusiones que se han extraído del proyecto.

Históricamente se ha intentado repetidas veces juntar 3d y web, y en todos los intentos el enfoque ha sido el de intentar revolucionar las interfaces actuales, conocidas por los usuarios, sustituyéndolas por modelos innecesariamente complejos.

En todos estos casos dos factores han influido negativamente en el asentamiento de dichas tecnologías, el primero es la complejidad tanto a nivel de implementación para el desarrollador, como para el usuario; el segundo eran las limitaciones técnicas impuestas por las tecnologías de conexión que no permitían muchos alardes en términos de calidad gráfica. Así tecnologías como VRML97 quedaron en el olvido.

El hecho de que el proyecto trate acerca de museos no es sino una escusa para tratar de unir el mundo del 3d y el mundo web una vez más. Esta vez, no obstante, dando un enfoque orientado a dar vida a proyectos con salida comercial, buscando en todo momento que la interfaz sea lo más intuitiva posible para los usuarios. Es decir salvando los dos obstáculos que anteriormente hicieron fracasar toda tentativa.

El desarrollo de proyectos de dicha índole con las herramientas que dispone el mercado es, sin duda alguna, posible hoy en día, y existen infinidad de áreas comerciales que podrían hacer uso de dicha tecnología, desde museos y salones de exposiciones hasta galerías de moda, probadores de ropa virtuales para tiendas de ropa *online*, pasando por juegos *online* y salones de chat virtuales. El límite lo ponen la creatividad y un espíritu emprendedor por parte del desarrollador.

Para el proyecto, durante la fase del análisis de viabilidad se barajaron varias tecnologías para sustentar técnicamente al mismo. Se optó por Google O3D. Las alternativas basadas en flash parecían condenadas al fracaso. Solamente WebGL se presentaba como alternativa realista a largo plazo y al mismo tiempo resultaba demasiado inmadura como para ser usada. En el momento del análisis, solamente Mozilla Firefox 3.7prealpha soportaba WebGL, y era una implementación llena de *bugs* que hubiere entorpecido el desarrollo del proyecto. No obstante, WebGL se estaba posicionando como el estándar de facto para el 3d.

Pues bien, recientemente, Google que no era ajena a este dato, ha sustituido su *plugin* O3D por una implementación nueva de la librería que continúa llamando O3D, la cual se sostiene sobre WebGL. Por tanto no se requiere un *plugin* externo al explorador que era la mayor desventaja de O3D y pasa a estar soportado nativamente por los exploradores que decidan dar soporte a WebGL (Firefox, WebKit y Chrome confirmados). Esto significa a su vez que el uso de *shaders* basado anteriormente en HLSL y Cg pasa a usar únicamente GLSL. Salvo este cambio las aplicaciones creadas anteriormente con el *plugin* continuarán funcionando con la implementación basada en WebGL con un mínimo esfuerzo por parte del desarrollador y con la dos grandes ventajas de pasar a no depender de un Sistema Operativo determinado y de la necesidad de que el usuario deba instalar un *plugin*. Sin duda alguna Google se gana con este movimiento el posicionarse con la tecnología que esta llamada a ser estándar de facto para unificar 3d y web. Todo aquel que se quiera subir al tren, seguirá el camino de O3D.

Ser pionero en la aventura de la web 3d, no solamente ha sido emocionante, sino que me

ha permitido adquirir una serie de conocimientos que en un entorno de competencia profesional, ofrece una ventaja competitiva clara, permitiendo brindar a los clientes un producto diferencial, el cual, la competencia esta a años luz de poder ofrecer.

Uno de los requerimientos no funcionales del proyecto era el de escribir un proyecto con código reutilizable. Esto responde a la intención de continuar el desarrollo del mismo ampliando la base creada a lo largo del proyecto con nuevas funcionalidades, diseños más robustos si cabe, interfaces más sencillas, un mayor uso de O3D, más efectos gráficos, y dar el salto a la implementación WebGL del *plugin* O3D. Todo esto con el objetivo en mente de explotar económicamente dichas ampliaciones en proyectos con salida comercial a medio – largo plazo. Algunas de dichas ampliaciones se discuten a continuación.

7.1 Ampliaciones futuras

> "Preguntarse cuándo los ordenadores podrán pensar es como preguntarse cuándo los submarinos podrán nadar"
> -- Edsger W. Dijkstra

A continuación se muestra una relación de ampliaciones a realizar al trabajo del proyecto.

- La primera de las ampliaciones que están en la lista de futuras ampliaciones es el paso a usar la implementación de O3D basada en WebGL. Este paso incluye sustituir todos los *shaders* , actualmente escritos en el lenguaje del *plugin* y escribirlos en GLSL. Ello implica usa serie de cambios en algunos de los objetos como BitmapTexture. También implica empezar a testear la aplicación en diferentes entornos, incluyendo pero no limitado a Linux, y MacOS X puesto que lograr hacer ver sin errores las aplicaciones usando shaders GLSL en dichos entornos tiene más posibilidades de éxito que las que se tenían con el *plugin*. Implicaría no obstante un estudio previo sobre que extensiones de OpenGL son más comunes de encontrar implementadas en la mayor variedad posible de *hardware* gráfico a largo plazo para evitar incompatibilidades.

- Mejorar el soporte para animaciones. Actualmente el modelo del actor solamente dispone de una única animación. Poder usar un grafo estados con tantas animaciones como transiciones de estado en el grafo, y algún sistema que permita configurar fácilmente los eventos que disparen un salto de un estado a otro y por lo tanto la reproducción de una animación u otra.

- Crear herramientas de *debug* para acelerar el desarrollo de nuevas funcionalidades.

- Otras mejoras menores, pero no por ello menos importantes, son el hecho de hacer un uso más extenso de ideas que imiten las interfaces a las que los usuarios se encuentran acostumbrados. Por falta de tiempo, por ejemplo , el puntero del ratón no cambia de icono al pasar por encima de los botones para las herramientas del gestor, y las imágenes de estos no cambian al ser pulsados.

- La mejor manera de vender un producto es que entre por los ojos. Así pues, donde ahora encontramos un museo sombreado, sin ningún fondo, sin luces, sin objetos dinámicos con los que interactuar, sin fuentes de las que brote agua, si antorchas de las que mane una llama, sin guías que nos expliquen la historia detrás de cada cuadro, se podría crear todo esto y mucho más. Se podrían crear efectos de *enviroment mapping*, refracción, reflexión, *normal mapping, cell shading*, *high dynamic range(HDR)*, etcétera. La figura 31 muestra algunos efectos de entre la infinidad que se podrían usar a tal efecto.

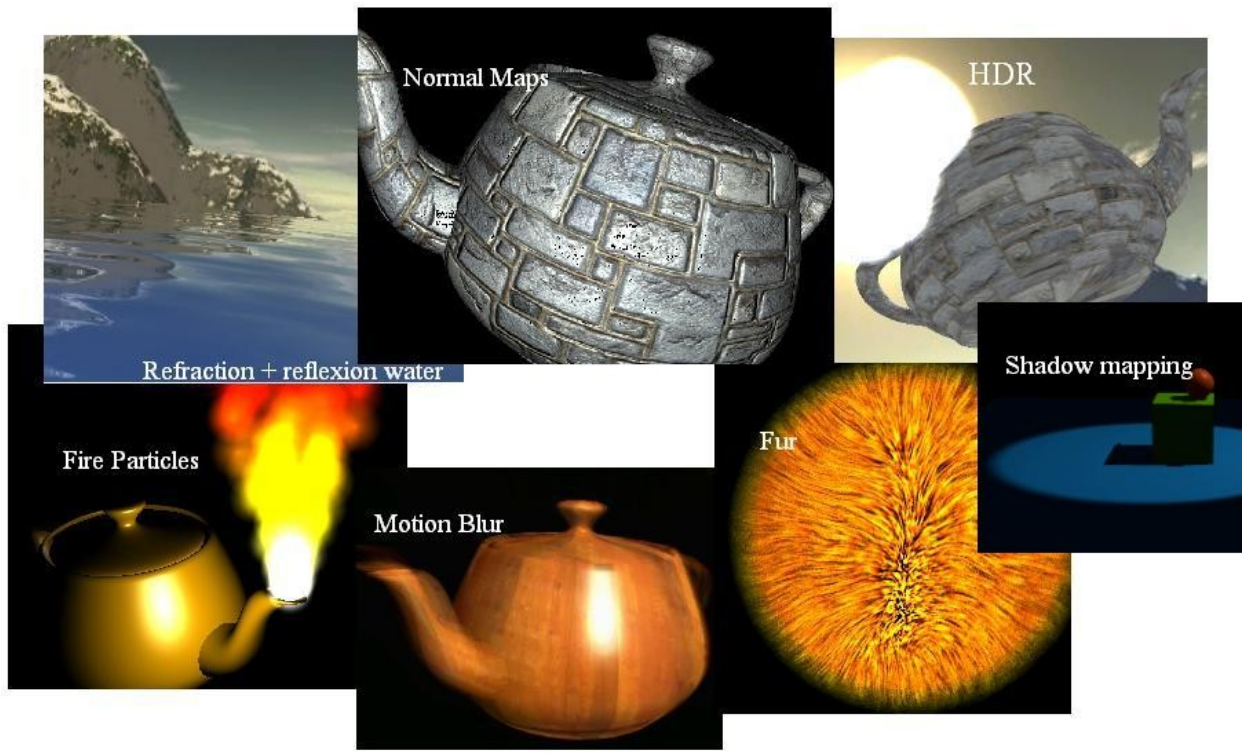


Figura 31: Efectos gráficos

8 Bibliografía

> Si cerca de la biblioteca tenéis un jardín ya no os faltará de nada.
> – Marco Tulio Cicerón

A continuación se detalla la bibliografía básica:

1. CROCKFORD Douglas. *JavaScript: The Good Parts*. O'Reilly Media, (2008). 153 páginas. ISBN:0596517742.
2. Manual de documentación jQuery [en linea] [Consulta: Agosto 2010] http://docs.jquery.com/Main_Page .
3. Manual de documentación jQueryUI [en linea] [Consulta: Agosto 2010] http://jqueryui.com/demos/ .
4. Manual de documentación MySql [en linea] [Consulta: Agosto 2010] http://dev.mysql.com/doc/ .
5. Manual de documentación O3D [en linea] [Consulta: Agosto 2010] http://code.google.com/intl/es-ES/apis/O3D/docs/index.html .
6. Manual de documentación PHP [en linea] [Consulta: Agosto 2010] http://www.php.net/ .
7. Manual de documentación Selenium [en linea] [Consulta: Agosto 2010] http://seleniumhq.org/docs/ .
8. O3D-Announce [lista de correo] [Consulta: Agosto 2010] O3D-announce .
9. SHREINER Dave, MASON Woo, NEIDER Jackie, DAVIS Tom. *OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL(R), Version 2.1.* 6ª edición. Addison-Wesley Professional, (2007). 928 páginas. ISBN:0321481003.
10. SHREINER Dave, khronos OpenGL ARB working group. *OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL(R), Version 3.0 and 3.1.* 7ª edición. Addison-Wesley Professional, (2009). 936 páginas. ISBN:0321552628.
11. STEFANOV, Stoyan. *Object-Oriented JavaScript: Create scalable, reusable high-quality JavaScript applications and libraries*. Packt Publishing (2008) . 356 páginas. ISBN:1847194141.
12. ST LAURENT, Sebastien. *Shaders for game programmers and artists*. Course Technology PTR (2004) . 483 páginas. ISBN:1592000924
13. BASORA Jordi, JANÉ Àngela, GUITERAS Josep M, *Matemàtiques batxillerat credits 4,5,6,* Mc Graw Hill, 263 páginas. ISBN: 8448112989.

8.1 Otros enlaces

> "¿Internet? ¿Todavía anda eso por ahí?"
> -- Homer Simpson

En esta sección se citan algunas herramientas útiles en el mundo del desarrollo web citadas a lo largo de la memoria sin ser necesariamente usadas en el proyecto:

1. Google translate api, permite traducir de forma dinámica el contenido de tus páginas. [en linea] [Consulta: Agosto 2010] http://code.google.com/intl/es-ES/apis/ajaxlanguage/
2. Google ajax api, permite una forma de cargar librerías y frameworks ajax desde un CDN (*Content delivery network*, dicho rápido y mal, una nube de servidores dedicados) de Google, logrando reducir la latencia al cargar estas librerías, aumentando la capacidad de cargar librerías en paralelo y mejorando el soporte de *catching*(efecto memoria del explorador web, que evita volver a cargar una librería si ya se ha cargado con anterioridad).[en linea] [Consulta: Agosto 2010] http://code.google.com/intl/es/apis/ajaxlibs/
3. Google gears, tecnología que permite explorar webs dinámicamente sin necesidad de estar conectado a la red, que una vez te conectas puedes usar para sincronizar los contenidos en linea con los guardados localmente en el cliente. Nota: HTML 5 soportaría nativamente el mismo tipo de funcionalidad.[en linea] [Consulta: Agosto 2010] http://gears.google.com/
4. Google closure tools, herramientas para optimizar código JavaScript, reduciendo el tamaño de los archivos,[en linea] [Consulta: Agosto 2010] http://code.google.com/intl/es-ES/closure/
5. Google search api,búsquedas transparentes en tu web con el motor de búsquedas de Google[en linea] [Consulta: Agosto 2010], http://code.google.com/intl/es-ES/apis/ajaxsearch/
6. Google maps api, funcionalidades de geolocalización en tu web,[en linea] [Consulta: Agosto 2010] http://code.google.com/intl/es-ES/apis/maps/
7. Google visualization api, soporte para mostrar diagramas y gráficos en tu web,[en linea] [Consulta: Agosto 2010] http://code.google.com/intl/es-ES/apis/visualization/
8. away3d, motor 3d basado en flash [en linea] [Consulta: Agosto 2010] http://away3d.com/
9. papervision3d, motor 3d basado en flash [en linea] [Consulta: Agosto 2010] http://blog.papervision3d.org/
10. smarty, motor de plantillas HTML muy usado y potente [en linea] [Consulta: Agosto 2010] http://www.smarty.net/.