Virtual Archaeology Review, **7**(15): 112-122, 2016
http://dx.doi.org/10.4995/var.2016.5834

Received: May 25, 2016
Accepted: September 29, 2016

EXPLORACIÓN DE LAS POSIBILIDADES DE LA LIBRERÍA THREE-JS EN PROYECTOS DE DIFUSIÓN DEL PATRIMONIO CULTURAL

EXPLORING THREE-JS LIBRARY POSSIBILITIES FOR THE DISSEMINATION OF CULTURAL HERITAGE PROJECTS

Jesús Palomar-Vázquez*, María José Viñals-Blasco

Departamento de Ingeniería Cartográfica, Geodesia y Fotogrametría, Universitat Politècnica de València, Camino de Vera s/n, 46022 Valencia, España. jpalomav@upvnet.upv.es; mvinals@cgf.upv.es

Resumen:

Para cualquier proyecto relacionado con el estudio y la conservación del patrimonio cultural, la fase de difusión de la información tridimensional (3D) debe ser una parte fundamental de todo el proceso. La mayoría de las plataformas existentes utilizan la tecnología WebGL para difundir contenido 3D a través de navegadores Web, aunque suelen ser de propósito genérico, con visores con bajo nivel de personalización. Como alternativa, el usuario dispone de una serie de librerías, la mayoría de código abierto, que permiten un alto grado de adaptación a las necesidades de un proyecto específico, si bien también exigen un nivel suficiente de conocimientos de programación para poder utilizarlas. En el presente artículo se estudiarán las características mínimas que debería tener un proyecto de difusión en el campo del patrimonio cultural y su implementación en una de las librerías más utilizadas hoy día en los entornos de código abierto para visualización 3D, como es Three-js. Asimismo se verán ejemplos de integración de Three-js con otro tipo de librerías que aprovechan las ventajas del lenguage HTML5 (*HyperText Markup Language*, versión 5) para enriquecer la experiencia de visualización y análisis dentro de un entorno Web.

Palabras clave: modelos 3D, difusión del patrimonio, WebGL, Three-js

Abstract:

For whatever project related with the study and conservation of cultural heritage, the dissemination stage of tree-dimensional (3D) information must be a key part of the whole process. Most of the existing web-based platforms are using WebGL technology to disseminate 3D content through Web navigators, despite they are normally generic, with a low level of viewers' customization. Alternatively, users can make use of several libraries, most of them open source, which allow a complete adaptation to the specific project features, even though enough level of programming skills are necessary to use them. In the present paper we will describe the minimum requirements that any cultural heritage dissemination project should contain and its implementation using Three-js, one of the most versatile open source libraries for 3D visualization. In addition, we will show several examples of Three-js integration with other libraries that take advantage from HTML5 (HyperText Markup Language, version 5) to enhance the user's experience on a Web platform.

Key words: 3D models, heritage dissemination, WebGL, Three-js

1. Introducción

Cualquier proyecto que tenga como objetivo el estudio de un bien de patrimonio cultural para su conservación, ha de pasar forzosamente por un proceso de documentación, análisis y difusión. En los últimos años, la posibilidad de crear reconstrucciones virtuales se ha convertido en un pilar fundamental de estos procesos, permitiendo el estudio y el análisis científico del elemento en cuestión, sin que sea necesario trasladarse hasta su emplazamiento físico y sin la necesidad de utilizar técnicas invasivas que pudieran alterar su estado original (Domingo, Carrión, Blanco, & Lerma, 2015; Koutsoudis et al., 2014). Una vez generado el modelo virtual, se pueden llevar a cabo multitud de estudios, ya sean análisis geométricos (acotaciones, secciones, volúmenes, etc.), de textura (patologías, grabados, dibujos, etc.) o de contexto (relaciones entre elementos diferentes dentro del mismo modelo o con elementos de su entorno). Es en este punto cuando comienza la fase de difusión, ya sea con propósito científico o divulgativo. En este sentido, el tándem Internet como vía de comunicación, en unión con las técnicas de representación virtual de gráficos por computador, se está convirtiendo en un estándar de difusión, debido sobre todo a la capacidad de alcance de Internet y a la naturalidad con la que los seres humanos entendemos los modelos 3D (Stefani et al., 2014).

Este artículo se centra en el proceso de difusión y concretamente en el estudio de las posibilidades de la librería de código abierto Three-js (TJS) basada en la tecnología WebGL, la cual permite incluir modelos 3D en entornos Web sin necesidad de instalar ningún tipo de programa adicional (*plug-in*) (Parisi, 2012). Repasaremos

* Corresponding Author: Jesús Palomar-Vázquez, jpalomav@upvnet.upv.es
DOI: http://dx.doi.org/10.4995/var.2016.5834

las diferencias y similitudes con otros tipos de tecnologías y plataformas de difusión de contenido 3D y describiremos las características mínimas que un proyecto de difusión de patrimonio cultural debería tener y cómo se implementa dentro de TJS.

2. Medios de difusión de contenido 3D

En primer lugar debemos distinguir entre las plataformas de creación de contenido y las de difusión. Las primeras están orientadas a la generación de modelos 3D a partir de distintos tipos de datos (nubes de puntos obtenidas mediante escáner láser o técnicas fotogramétricas SfM, planos de planta y alzado, etc.) o modelos creados desde cero. Entre ellas podemos citar productos como Maya®, 3D Max®, Zbrush® o Blender®. Las segundas son plataformas destinadas al almacenamiento y visualización en entornos Web de modelos 3D.

La difusión online de contenidos 3D tiene dos vías bien diferenciadas: por un lado encontramos las plataformas comerciales que ofrecen repositorios o almacenes de modelos y visores especializados con características básicas. Entre ellas podemos citar los casos de p3d.in, Verold o la más conocida de todas ellas, Sketchfab; por otro lado encontramos alternativas que nos permiten, a partir de librerías especializadas orientadas al renderizado de información 3D en entornos Web, la creación de visores adaptados a las necesidades de cada usuario. A este segundo grupo pertenecerían librerías como TJS, Babylon-js, PlayCanvas o Potree, la mayoría de ellas enmarcadas dentro del software de código abierto.

A decir verdad, ambos tipos de posibilidades pueden usar tecnologías similares (una plataforma comercial puede estar basada en librerías de código abierto), pero normalmente, las del primer tipo siguen estrategias de mercadotecnia (*marketing*), ofreciendo almacenamiento y características de visualización y funcionalidades que se incrementan junto con el nivel de compromiso del cliente, representado por diferentes planes de pago.

De esta manera, la diferencia básica entre ambos grupos es que en el primero de ellos, el usuario ha de subir su modelo a un repositorio, por lo que "pierde" la propiedad del mismo. Por otro lado, si bien las características básicas del visor ofrecido cubren un espectro limitado de posibilidades, el usuario no tiene que preocuparse de nada más que de subir su modelo y de aplicar una serie de ajustes iniciales para su visualización.

En el caso de ser el propio usuario el que utilice librerías especializadas, éste podrá tener el modelo 3D en su propio servidor y bajo su propio dominio. En segundo lugar, la adaptación del visor a las necesidades del proyecto es total, pero por el contrario se necesitará un alto nivel de especialización a nivel de programación.

En cuanto a la tecnología que permite implementar la visualización de gráficos 3D en entornos Web, es el estándar WebGL (*Web Graphics Library*) el que se está imponiendo, debido sobre todo a su alta compatibilidad con el lenguaje HTML5, lo que le permite funcionar en múltiples navegadores, sin necesidad de instalar ningún tipo de plugin.

Muchas de las plataformas y librerías antes citadas como Babylon-js, Sketchfab o la propia librería TJS toman WebGL como base para su funcionamiento. En este sentido, podemos encontrar ejemplos de aplicación en el ámbito del patrimonio cultural en Jiménez, García, Revelles & Melero (2012) o en Robles, Feito, Jiménez & Segura (2012), donde se describen proyectos aplicados a la difusión del patrimonio escultórico o a la creación de visitas a museos virtuales en plataformas móviles, basados en el estándar WebGL.

Ahora vamos a pasar a estudiar el contexto en el que se enmarcan las tecnologías antes mencionadas.

2.1. Análisis del contexto

Como se comentó anteriormente, estas tecnologías abordan la difusión de contenido desde dos perspectivas claramente diferenciadas: por un lado encontramos las plataformas comerciales de almacenamiento y visualización, y por otro lado la posibilidad de que sea el propio usuario el que utilice librerías como las nombradas anteriormente para generar su propio entorno de visualización según sus necesidades. Básicamente, las diferencias entre ambos puntos de vista radican en la orientación de la difusión del contenido y el grado de libertad que tiene el usuario a la hora de adaptar las características particulares de su proyecto, y será el usuario quien decida el uso de uno u otro tipo de tecnología. En la Tabla 1 podemos ver algunas de estas diferencias entre estos tipos de plataformas.

Tabla 1: Diferencias entre los dos tipos de plataformas de difusión de contenido 3D

Tipo de plataforma	*Uso mayoritario*	*Nivel de adaptación a un proyecto específico*	*Grado de conocimientos exigidos*
Comercial	Redes sociales, publicidad, muestra de trabajos profesionales	Bajo	Bajo
Orientada al usuario	Divulgación científica, educación, juegos online, arte digital, realidad virtual	Alto	Medio - Alto

2.1.1. Plataformas comerciales

Al igual que ha sucedido con diversos tipos de contenidos como los blogs (Blogger o WordPress), las fotografías (Flickr, Instagram o Pinterest), o los vídeos (YouTube o Vimeo), los contenidos 3D han entrado en la Web de la mano de las redes sociales, comenzando a formar un ecosistema de plataformas de propósito general en las que tienen cabida desde aplicaciones para mostrar el trabajo de artistas digitales, pasando por empresas con servicios de impresión 3D o para la divulgación de un bien cultural como podría ser nuestro caso. Entre estas plataformas podemos citar ejemplos como p3d.in, Verold o Sketchfab, que pasamos a comentar a continuación.

- p3d.in (https://p3d.in). Plataforma que permite el almacenamiento de modelos 3D en varios tipos de formatos. Aunque no podemos alterar la geometría de los modelos, sí podemos editarlos desde la misma plataforma para que tengan el aspecto final que deseamos, añadiendo texturas, mapas de normales, posición y rotación inicial, fondo, iluminación, etc. Así mismo, podemos alterar las características del visor mediante petición URL (Fig. 1).

Figura 1: Peticiones URL: a) para una vista normal (https://p3d.in/O4Xyo); y b) para una vista de malla (https://p3d.in/O4Xyo +subd+wire). Fuente: galería de ejemplos de p3d.in.

- Verold. Esta plataforma, que ha sido recientemente adquirida por la empresa de servicios en la nube Box (https://www.box.com), está pensada para la creación de proyectos que integren contenidos 3D con capacidades de presentaciones multimedia. Su uso abarca proyectos de publicidad, educación o impresión 3D. Dispone de un editor en línea que permite cargar diversos tipos de modelos y vincular otro tipo de contenidos como texto (mediante anotaciones), vídeos, animaciones, recorridos de cámara configurados previamente, etc.

- Sketchfab (https://sketchfab.com). Actualmente, una de las plataformas más utilizadas para el almacenamiento y la visualización de contenido 3D, tiene una clara orientación de red social. Al igual que Verold o p3d.in, permite ajustar muchas características iniciales en cada modelo y tiene la capacidad de embeber el visor en cualquier página web, siendo también compatible con dispositivos móviles. Entre las características que permite incluir destacan el procesamiento de las texturas para mejorar aspectos como el brillo o el contraste, las anotaciones con contenido HTML, o la vista adaptada para dispositivos de realidad virtual. Al ser una plataforma con orientación a las redes sociales, dispone de un sistema de clasificación por votación y permite descargar o compartir el contenido (Fig. 2).

2.1.2. Librerías especializadas

WebGL es un estándar que permite el acceso directo a la GPU (*Graphics Processing Unit*), acelerando mucho el tiempo de cálculo intensivo que exige el renderizado 3D. Es una librería a bajo nivel, lo que exige también muchos conocimientos en cuanto a gráficos por computador y cálculo matricial para sacarle todo el partido. Es por ello, que han surgido diferentes tipos de librerías de más alto nivel que abstraen funcionalidades detalladas de WebGL, haciéndolas más accesibles y sencillas para los desarrolladores, lo que a su vez contribuye a la democratización del uso de esta

Figura 2: Visor de Sketchfab. Estela romana. Sagunto (España). Fuente propia (https://goo.gl/FvFXjQ).

tecnología. Entre estas iniciativas podemos destacar Three-js, Babylon-js, PlayCanvas o Potree.

- Three-js (http://threejs.org). Es una librería que permite generar gráficos 3D en entornos Web aprovechando directamente la capacidad de aceleración de la GPU. Para ello utiliza una serie de funciones en JavaScript que trabajan con WebGL y todo ello sin tener que instalar ningún tipo de plug-in. El flujo estándar de trabajo comienza con la creación de un nodo inicial, denominado escena, donde se van añadiendo una serie de elementos de forma jerárquica: una cámara, una fuente de luz y una malla (objeto geométrico que a su vez consta de una geometría y un material). Todo el trabajo de renderizado se vuelca, sobre el componente <canvas> de HTML5 para poder ser visto por el usuario. TJS nació en 2010 y es la más veterana de todas estas librerías, contando con una amplia comunidad de usuarios y un API bien documentado, siendo concebida como una librería de propósito general y una de las más extendidas (Evans, Romeo, Bahrehmand, Agenjo, & Blat, 2014). En cuanto a los aspectos más negativos podemos citar que no está optimizada para la creación de juegos y que las sucesivas versiones no mantienen una compatibilidad total con las anteriores, con lo que hay que poner especial cuidado en el mantenimiento de los proyectos que se creen con TJS.

- Babylon-js (http://www.babylonjs.com). Esta librería nacida en 2013, es más moderna que TJS y, sobre todo, está concebida específicamente para tener un alto rendimiento en juegos 3D para entornos Web. De igual manera que TJS, Babylon dispone de un API JavaScript al que acceder a WebGL, si bien, esta API es de mayor nivel, lo que la hace algo más fácil de manejo para el usuario. Esto, sin embargo, puede ser una pequeña limitación para alguien que posea conocimientos más avanzados sobre gráficos por computador a la hora de sacarle más partido a la librería. En cuanto al modo de funcionamiento, es

muy similar a TJS, contando también con todos los elementos típicos: escenas, cámaras, geometrías, materiales, animaciones o un motor físico.

- PlayCanvas (https://playcanvas.com). Creada por Microsoft en 2014, esta librería tiene dos tipos de licencias, ya que se considera de código abierto, pero para proyectos comerciales mantiene una licencia propietaria y un servicio de almacenamiento de pago. Como Babylon-js, PlayCanvas está orientada a la creación de juegos en plataforma Web, tanto con WebGL como con HTML5. Dispone de un API JavaScript bien documentado y una plataforma de desarrollo bien pensada y orientada a un rápido aprendizaje por parte del usuario.

- Potree (http://www.potree.org). La primera versión de esta librería es de diciembre de 2014. Esto nos da una idea de su juventud. A pesar de ello, esta librería merece una mención especial, ya que viene a cubrir un hueco interesante, sobre todo dentro del ámbito del patrimonio cultural: la gestión y visualización de nubes de puntos (Fig. 3). La librería nació como un proyecto independiente a partir de la reescritura de TJS para adaptarla al manejo y visualización de grandes cantidades de puntos, tanto de forma local como en línea, a través de la transmisión continua de información (*streaming*).

Como podemos observar por todo lo dicho anteriormente, a pesar de que con cualquiera de estas librerías podríamos desarrollar un proyecto dentro del ámbito de la difusión del patrimonio cultural, descartamos las librerías Babylon-js y PlayCanvas por estar más orientadas a la creación de juegos y Potree, por estar enfocada únicamente a la gestión de nubes de puntos. En este sentido, escogeremos TJS por ser una librería más madura, de propósito general y disponer de una batería enorme de ejemplos, una comunidad activa y un API bien documentado. Por todo ello, TJS cumple ampliamente los requisitos que un proyecto de esta índole puede necesitar.

Es en este marco sobre el que se desarrolla el resto de este artículo, ya que a continuación se irán describiendo los requisitos mínimos que un proyecto de difusión de patrimonio cultural debería tener y cómo se implementarían dentro de TJS.

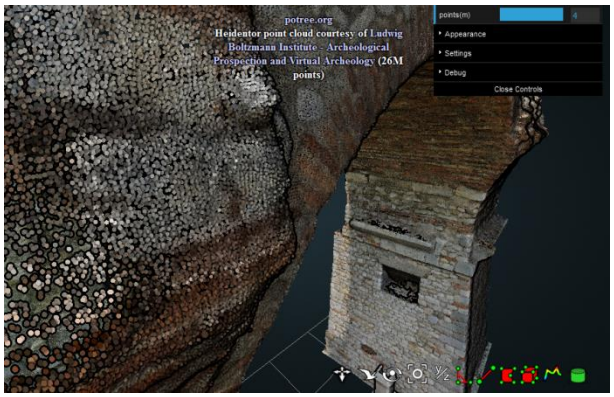


Figura 3: Proyecto realizado con Potree (26 millones de puntos). Fuente: galería de ejemplos de Potree (http://goo.gl/pyociq).

3. Escenario de proyecto base con TJS

3.1. Caso de estudio

Para el desarrollo de este proyecto se ha escogido uno de los vestigios que todavía se conservan dentro de la línea defensiva de trincheras Puig-Carassols (Durbán, 2014), dentro del paraje conocido como La Vallesa de Mandor, en el término municipal de Paterna (España). Concretamente, se trata de una estructura defensiva de hormigón (búnker) que dispone de un túnel interior y varias troneras para ametralladoras. Aunque la parte posterior de la estructura muestra un estado cercano a la ruina, la parte anterior está bien conservada, si bien presenta gran cantidad de grafitis (Fig. 4).

Figura 4: Vistas del búnker objeto de estudio.

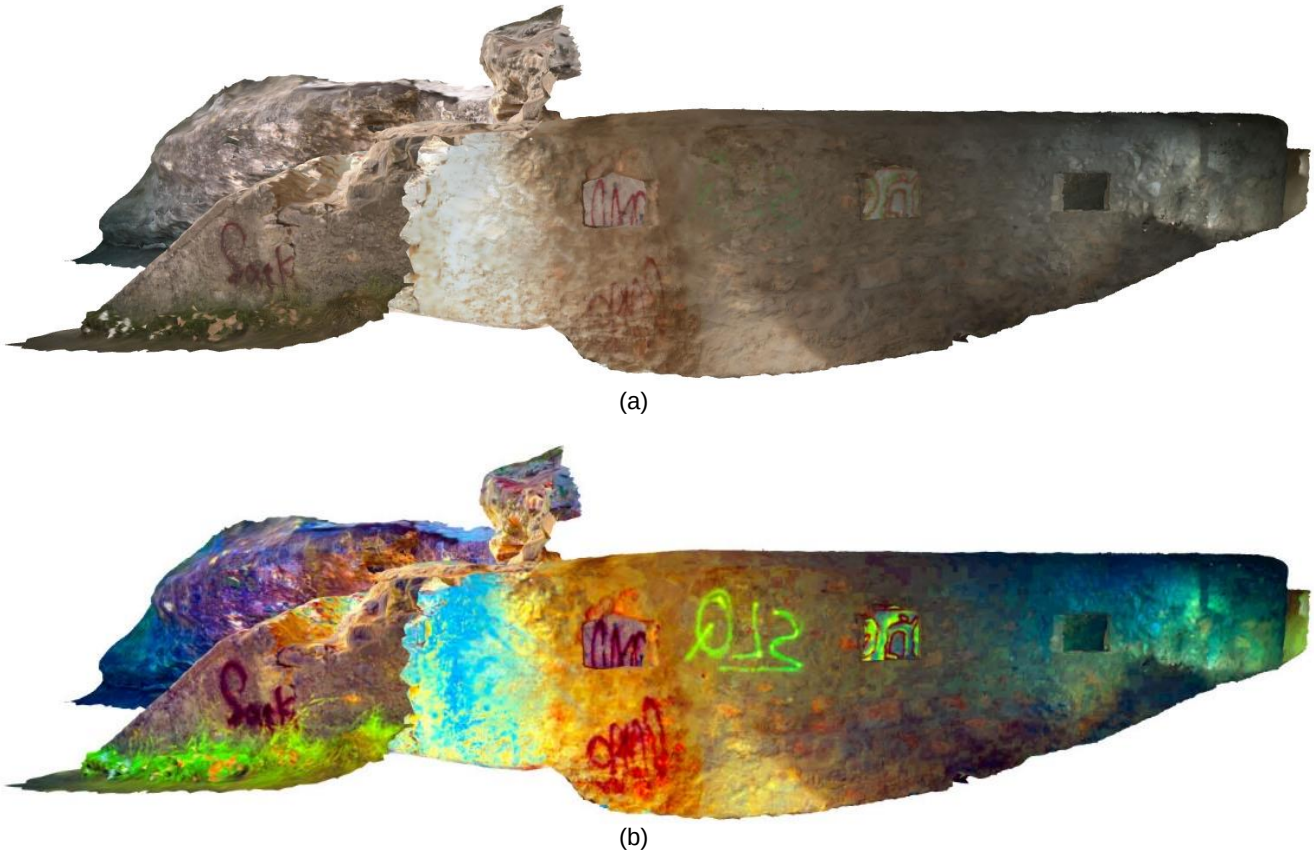
El objetivo principal de este proyecto es por un lado el modelado 3D de la estructura y por otro el estudio de la situación y tipología de los grafitos. Finalmente, se abordará la implementación de un visor personalizado basado en Three-js donde se mostrarán los resultados en un entorno Web (http://goo.gl/2X4rhN). Los datos originales utilizados para el visor pueden ser obtenidos en Palomar-Vázquez & Viñals-Blasco (2016).

Por último, decir que la captura de la información 3D, así como el mallado y el texturizado, se han realizado utilizando el software Agisoft Photoscan® a partir de una colección de fotografías tomadas del objeto. Ejemplos del uso de este software en el ámbito del patrimonio cultural podemos encontrarlos en Peinado, Fernández & Agustín (2014).

Atendiendo a las necesidades del proyecto en cuestión, se implementarán las siguientes características: carga del modelo, aplicación dinámica de texturas, visualización de distintos modos de geometría, puntos de vista, paseos virtuales, puntos de información, visibilidad y opacidad de los modelos, medidas, secciones y editor de imágenes.

3.2. Carga del modelo

Three-js (TJS) dispone de unas clases denominadas "Loaders" que permiten trabajar con multitud de formatos, entre los que podemos citar OBJ, JSON, MTL o COLLADA. Este último formato es el que se ha escogido para el proyecto, entre otros motivos porque funciona bien con texturas asociadas y permite la integración de animaciones. La clase que maneja la carga del modelo



(a)

(b)

Figura 5: Vista lateral de la estructura interior del búnquer: a) textura original; b) textura procesada mediante Decorrelation Stretch (algoritmo LDS)

dispone de una función *Load* en cuya sintaxis encontramos los parámetros necesarios: la ruta del archivo collada (con extensión DAE), una función que se ejecuta una vez cargado el modelo en memoria y una última función que se ejecuta mientras el modelo se está cargando. Es en esta última función donde se implementa el feedback de carga, ofreciendo al usuario información dinámica del porcentaje del modelo que se está cargando. En el anexo puede verse un ejemplo de esto último (Anexo, Ejemplo 1).

Una cuestión de otra índole pero no menos importante es el tema de la seguridad. Al cargar el modelo en la Web lo exponemos a una posible descarga por parte de cualquier usuario. No es el objetivo de este artículo tratar en profundidad este tema, pero entre las posibles soluciones deberíamos citar la posibilidad de utilizar formatos comprimidos y protegidos por contraseña (Jiménez et al., 2012) o algo menos sofisticado pero igual de efectivo, como el que se pudiera lleva a cabo la manipulación por código de la geometría de nuestro modelo alterando el contenido original.

3.3. Aplicación dinámica de texturas

En el caso que nos ocupa, es necesario el estudio de los grafitis distribuidos por toda la superficie del búnker. Para conseguir un mayor realce de estos trazos, se ha sometido a la textura original del modelo a diferentes filtros utilizando el plug-in Decorrelation Stretch sobre el software libre ImageJ (Harman, 2005). Estas texturas se pueden cargar y aplicar dinámicamente sobre el modelo para apreciar en cada caso el mejor resultado obtenido con estos filtros (Fig. 5). Dentro del modelo de datos de TJS, una malla (*mesh*) se compone de una geometría y de un material, el cual, entre sus atributos, permite la aplicación de una textura. En el Ejemplo 2 del Anexo podemos ver el código necesario para ello.

3.4. Modos de geometría

Es común en este tipo de proyectos dar al usuario la posibilidad de visualizar el modelo 3D en sus tres tipos de geometría: malla de caras, aristas (wireframe) o nube de puntos (Fig. 6). TJS permite alternar fácilmente entre los dos primeros modos, ya que la clase *MeshBasicMaterial* tiene entre sus atributos la propiedad *wireframe*, la cual permite asignar este modo de visualización (Anexo, Ejemplo 3).

El caso de la visualización de la nube de puntos no es directo y requiere el uso de otro tipo de geometrías soportada por TJS: el sistema de partículas. Un sistema de partículas es una malla especial formada únicamente por una geometría de tipo vértices y un material, el cual especificará, entre otros atributos, el tamaño y el color de las partículas (Anexo, Ejemplo 4).

Por último, decir que la estrategia seguida para la obtención de secciones de modelos 3D complejos como es éste y que abordaremos más adelante, se basará en la gestión de este sistema de partículas.

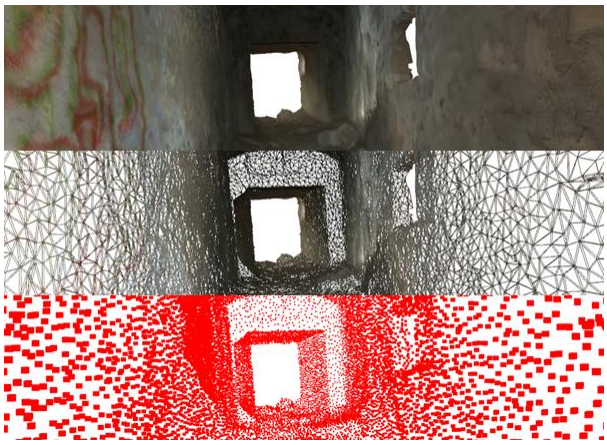


Figura 6: De arriba abajo: textura, malla de alambre y nube de puntos del interior del túnel del búnker.

3.5. Puntos de vista

Para la visualización correcta de las distintas partes del modelo es muy útil la posibilidad de navegar rápidamente con la cámara hasta una vista predefinida con anterioridad. La cámara permite ver la escena a través de ella y la manera en que se nos presentan los objetos depende de dos parámetros: posición y rotación. La posición define el punto 3D en el espacio de la escena donde se coloca la cámara (punto de anclaje); la rotación indicará el ángulo de la cámara respecto de los tres ejes de rotación sobre el punto de anclaje (Anexo, Ejemplo 5).

3.6. Paseos virtuales

En el caso del proyecto que nos ocupa, se desea crear una animación que se introduzca por el interior del túnel y salga por el extremo posterior de la estructura. TJS permite este tipo de animaciones basándose en ir cambiando de forma consecutiva el punto de vista de la cámara (ver Apartado 3.5). De esta forma, la estrategia consiste en predefinir primero unos puntos de paso obligado en el trayecto de la cámara (con sus posiciones y rotaciones) para luego densificarlos. Esta densificación de los puntos de paso de la trayectoria permitirá incrementar la suavidad de la animación. Para ello TJS dispone de una función que interpola puntos dentro de una curva tipo *spline* generada a partir de una serie de puntos de control previamente definidos (Anexo, Ejemplo 6). Finalmente, haremos que la cámara cambie de vista cada vez que se renderize un nuevo *frame*, actualizando los valores de posición y rotación de la cámara dentro de la función *requestAnimationFrame* de TJS.

3.7. Puntos de información

Esta característica es muy interesante, ya que permite añadir información complementaria (comentarios, imágenes, vídeos, etc.) asociada a una posición del espacio. Para conseguir ésto la estrategia a seguir se basa en colocar unas esferas en una posición cercana al punto de interés. En el siguiente ejemplo se puede ver cómo se crea una de estas esferas y se añade a la escena (Anexo, Ejemplo 7).

Una vez añadidas las esferas, el reto consiste en poder seleccionarlas. Para ello, TJS dispone de un mecanismo de emisión de rayos (*ray casting*), por el cual se determina el primer objeto del mundo 3D intersectado por un rayo. Esta técnica permite renderizar (o proyectar) una escena 3D sobre la pantalla 2D. De esta manera, si utilizamos el proceso inverso, podremos, a partir de las coordenadas 2D dadas por el usuario sobre la pantalla, obtener un rayo (vector 3D) que intersecte con los elementos 3D de la escena, por lo cual podremos saber sobre qué esfera está pulsando el usuario. Una vez disponemos de esta información, podremos visualizar la información adicional asociada a dicha esfera. En nuestro caso, se ha utilizado la librería popupmodal-js, una librería JavaScript que permite la visualización de contenido HTML dentro de una ventana modal flotante (Fig. 7 y Anexo, Ejemplo 8).

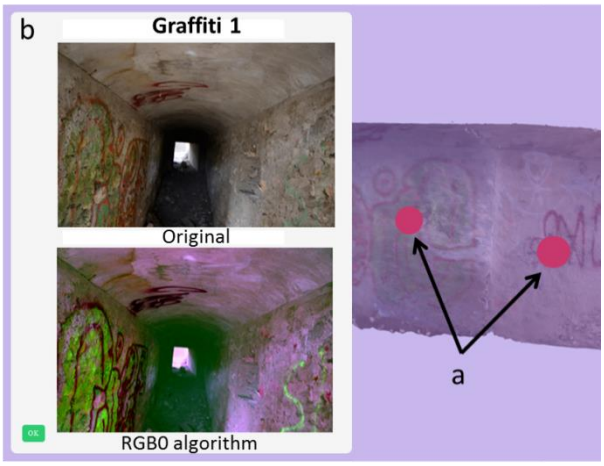


Figura 7: a) Puntos de información (esferas rojas); b) ventana de información

3.8. Visibilidad y opacidad

En el escenario de este proyecto nos interesaba mucho separar la visualización de la parte interior y exterior del túnel. Para ello, primero se cargaron ambas partes como modelos independientes. Esto permite hacer visible a voluntad la zona exterior del búnker y, por otra parte, darle un nivel de transparencia, lo que hace aún más efectiva y atractiva la comprensión de la estructura de toda la escena (Fig. 8). Ambos objetivos son fácilmente resueltos mediante el ajuste de las propiedades de los modelos y sus materiales (Anexo, Ejemplo 9).

Figura 8: Vista cenital del búnquer: a) modelo opaco; b) opacidad al 50% (permite ver la estructura interna del túnel).

3.9. Medidas

Otra de las características más interesantes es la posibilidad de poder medir directamente sobre el modelo. Para ello nos apoyamos otra vez en la técnica de *Ray Casting*, la cual nos permitirá seleccionar las coordenadas 3D de dos puntos cualesquiera sobre el modelo y calcular su distancia. Para conseguir un manejo funcional de esta característica, se ha implementado el código añadiendo los eventos "*click*" y "*mousemove*" al documento html (Fig. 9). De esta manera podemos controlar cuándo y qué tipo de botón (derecho, central o izquierdo) se ha pulsado para después programar el mecanismo adecuado que obtenga los dos puntos seleccionados sobre el modelo. Así, el usuario ha de seleccionar cada una de las dos esferas (verde o azul) que representan los extremos de la regla y arrastrarlos hasta la posición de intersección con el modelo que deseen, mostrándose en la pantalla la distancia de esta regla. En el Ejemplo 10 del Anexo podemos ver cómo se obtiene el punto de intersección con el modelo.

Figura 9: Las esferas verde y azul representan los puntos seleccionados por el usuario de forma interactiva.

Figura 10: Sección transversal sobre el eje X vista sobre el modelo con transparencia.

3.10. Secciones

Un aspecto interesante y a veces necesario a la hora de presentar resultados gráficos es la posibilidad de crear secciones por planos. Aunque TJS dispone de métodos para generar operaciones de tipo CSG (*Constructive Solid Geometry*) como adiciones, uniones o sustracciones entre geometrías, debido a la complejidad del modelo utilizado (228000 caras y 114000 puntos), se ha optado por presentar secciones a partir de la manipulación de la nube de puntos. De esta forma la visualización de una sección es inmediata, ya que se limita a seleccionar vértices que se encuentran dentro de un rango en cada uno de los tres ejes de coordenadas (Anexo, Ejemplo 11). Para este proyecto solamente se permiten planos perpendiculares, aunque se puede seleccionar el ancho del perfil (Fig. 10).

3.11. Editor de imágenes

Hasta aquí, se han implementado algunas de las características básicas que TJS nos permite hacer para conseguir un buen visor de contenido 3D para un proyecto de patrimonio cultural. Como se demostró en el Apartado 3.7, si utilizamos otro tipo de librerías podemos añadir características nuevas que enriquezcan mucho más las funcionalidades de nuestro proyecto. En este caso, se ha implementado un editor de imágenes, previsto para obtener una copia de la textura del modelo y aplicar sobre ella diferentes tipos de filtros para ofrecer otros tipos de análisis o realce de las imágenes. Concretamente se han implementado unos filtros de detección de bordes y de manipulación del brillo y del contraste. Esto es posible debido al componente Canvas de HTML5. Este componente es un contenedor sobre el que podemos escribir y leer información matricial correspondiente a una imagen. De esta manera la estrategia parte por copiar la vista 3D actual del modelo al componente Canvas y transferir la información del mismo a una nueva página web que contiene a su vez otro Canvas sobre el que se escribe la imagen. Una vez ahí, podremos aplicar cualquier tipo de operación matricial para manipular el contenido de la imagen. En el siguiente ejemplo de código (Anexo, Ejemplo 12) se puede ver cómo se transfiere la imagen de la página principal del visor a una nueva página y un ejemplo de filtro de detección de bordes (Fig. 11).

Figura 11: Filtro de bordes aplicado sobre la imagen de una vista concreta del interior del túnel del búnker.

3.12. Interface gráfica

La interacción del usuario con el modelo es fundamental a la hora de conseguir una experiencia de navegación y consulta adecuada. En cuanto a la navegación por el modelo, TJS aporta el módulo OrbitControls, el cual permite la rotación de la cámara alrededor de un punto de la escena, movimientos de zoom y desplazamientos. Por otro lado, para la creación de una interfaz gráfica que permita interactuar con el modelo, se ha utilizado dat.GUI, una librería simple pero eficiente para modificar variables en JavaScript. Esta librería permite incluir controles básicos como botones, selectores booleanos, barras deslizadoras, cajas de texto, selectores de color, listas desplegables, etc., además de la agrupación de los mismos dentro de carpetas (Fig. 12).

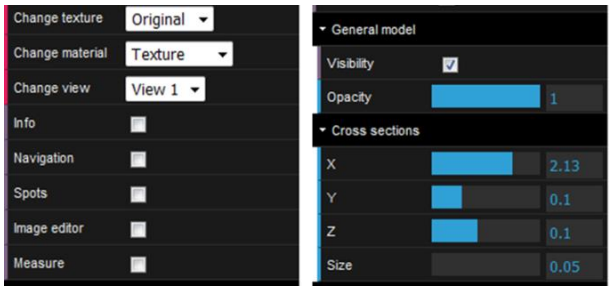


Figura 12: Interfaz gráfica diseñada para el proyecto con la librería dat.GUI.

4. Discusión y conclusiones

La primera idea a destacar es que a la hora de elegir un tipo u otro de tecnología debemos tener claras la forma y la orientación del proyecto de difusión que queramos abordar. De esta forma, si se trata por ejemplo de un proyecto de catalogación de piezas arqueológicas de pequeño tamaño, con comentarios adjuntos, y donde necesitemos un visor básico pero funcional, podemos emplear claramente plataformas como Sketchfab. Si por el contrario deseamos crear un proyecto integral, con un visor que soporte más capacidades (distintos tipos de vistas, recorridos, mediciones, secciones, asociación de información multimedia, interfaz personalizada, etc.) y mantener siempre la propiedad de nuestros datos, la opción más adecuada será emplear una librería como TJS.

Por otro lado, también debemos sopesar el grado de experiencia que tengamos. De nuevo, si no disponemos de algunos conocimientos de programación, lo más rápido y eficiente es utilizar una plataforma orientada al usuario como las que hemos visto. De esta manera el usuario solamente se encarga de subir el modelo y ajustar los valores de visualización iniciales. Por el contrario, si tenemos experiencia en la programación de gráficos, podemos conseguir fácilmente resultados espectaculares y muy atractivos usando alguna de las librerías descritas.

Respecto a este último aspecto, no se puede concluir cuál de estas librerías o frameworks se impone a las otras ya que tampoco hay ninguna que integre y recoja todas las características deseables para un proyecto de visualización de información 3D. Como se comenta en Krämer & Gutbell (2015), algunas de estas librerías soportan características que otras no tienen (sistemas de referencia, transmisión contínua de la información, encriptación de los modelos, etc.), pero, respecto a TJS, se destaca la posibilidad de gestionar información de datos masivos en forma de nubes de puntos (a través de potree) y la gestión efectiva de la memoria en el caso de grandes volúmenes de datos gracias a su acceso directo a WebGL.

A lo largo de este artículo hemos ido viendo cómo a través de la librería TJS podemos establecer las bases mínimas que todo proyecto de difusión de contenido 3D asociado al patrimonio cultural debería tener. Queda demostrada la potencia de esta librería y cómo en unión con HTML5 y otro tipo de librerías, enriquecen las capacidades del visor.

También han quedado patentes las distintas posibilidades que nos encontramos a la hora de difundir nuestro contenido 3D y las diferencias entre utilizar una plataforma externa o generar nuestra propia plataforma, quedando claro que, aunque exige unos conocimientos moderados en programación, si queremos total independencia, librerías como TJS nos ofrecen gran cantidad de posibilidades.

Referencias

Babylon JS. 3D engine based on WebGL/Web audio and JavaScript. Retrieved form http://www.babylonjs.com

Domingo, I., Carrión, B., Blanco, S., & Lerma, J. L. (2015). Evaluating conventional and advanced visible image enhancement solutions to produce digital tracings at el Carche rock art shelter. *Digital Applications in Archaeology and Cultural Heritage*, *2*, 79–88. http://doi.org/10.1016/j.daach.2015.01.001

Durbán, J. (2014). Tipologías y estrategias en la defensa de Valencia. Estudio del punto de apoyo de San Antonio de Benagéber en el centro de resistencia de la Vallesa de Mandor. *La Linde, revista digital de arqueología*, *3*. Retrieved form http://www.lalindearqueologia.com/index.php/crea-articulo/52-edicion-numero-3/arqueologia-de-la-guerra-civil-3/125-san-antonio-de-benageber

Evans, A., Romeo, M., Bahrehmand, A., Agenjo, J., & Blat, J. (2014). 3D graphics on the web: A survey. *Computers & Graphics*, *41*, 43–61. http://doi.org/10.1016/j.cag.2014.02.002

Jiménez, J., García, M., Revelles, J., & Melero, F. (2012). Digitalización 3D y difusión en web del patrimonio de las universidades andaluzas mediante X3D y WebGL. *Virtual Archaeology Review, 3*(7), 55–59. http://dx.doi.org/10.4995/var.2012.4386

Harman, J. (2005). Using decorrelation stretch to enhance rock art images. *American Rock Art Research Association Annual Meeting*, Sparks, Nevada.

Koutsoudis, A., Vidmar, B., Ioannakis, G., Arnaoutoglou, F., Pavlidis, G., & Chamzas, C. (2014). Multi-image 3D reconstruction data evaluation. *Journal of Cultural Heritage*, *1*, 73–79. http://doi.org/10.1016/j.culher.2012.12.003

Krämer, M., & Gutbell, R. (2015). A case study on 3D geospatial applications in the Web using state-of-the-art WebGL frameworks. In *Proceedings of the 20th International Conference on 3D Web Technology* (pp. 189–197). New York, USA: ACM. http://dx.doi.org/10.1145/2775292.2775303

Palomar-Vázquez, J., & Viñals-Blasco, M. J. (2016). Replication data for: Exploración de las posibilidades de la librería Three-js en proyectos de difusión del patrimonio cultural. *Harvard Dataverse*, V1. http://dx.doi.org/10.7910/DVN/Q3H81Y

Parisi, T. (2012). *WebGL: Up and Running. Building 3D Graphics for the Web*. Sebastopol, USA: O'Reilly Media.

Peinado, Z., Fernández, A., & Agustín, L. (2014). Combination of low cost terrestrial and aerial photogrammetry: three-dimensional survey of the church of San Miguel in Ágreda (Soria). *Virtual Archaeology Review, 5*(10), 51–58. http://dx.doi.org/10.4995/var.2014.4210

PlayCanvas. 3D HTML5 & WebGL Game engine. Retrieved form https://playcanvas.com

Potree. WebGL pointcloud renderer. Retrieved form http://www.potree.org

Robles, M., Feito, F., Jiménez, J., & Segura, R. (2012). Tecnologías para museos virtuales en dispositivos móviles. *Virtual Archaeology Review*, *3*(7), 102–108. http://dx.doi.org/10.4995/var.2012.4402

Stefani, C., Brunetaud, X., Janvier-Badosa, S., Beck, K., De Luca, L., & Al-Mukhtar, M. (2014). Developing a toolkit for mapping and displaying stone alteration on a web-based documentation platform. *Journal of Cultural Heritage*, *15*(1), 1–9. http://doi.org/10.1016/j.culher.2013.01.011

Three js. Javascript 3D library. Retrieved form http://threejs.org

Anxo: Ejemplos de código

Ejemplo 1. Carga del modelo

```
1 //creación de un objeto de la clase "Loader" (en este caso para la carga de un modelo collada)
2 loader = new THREE.ColladaLoader();
3 //Carga del modelo. La función onProgress se ejecuta durante la carga
4 loader.load( url, onLoad(collada), onProgress(progress) );
5 //onProgress se utiliza para mostrar el porcentaje de carga del modelo en pantalla
6 function onProgress(progress){
7       loading = (progress.loaded / progress.total * 100).toFixed(2)+' % loaded';
8       text.innerHTML = loading;
9}
```

Ejemplo 2. Carga de una textura y aplicación de un material

```
1 //carga de la textura almacenada en un fichero
2 texture = THREE.ImageUtils.loadTexture("collada/bunker2/texturas/Original.jpg");
3 //creación del material mediante la textura cargada
4 material = new THREE.MeshBasicMaterial( { map: texture } );
```

Ejemplo 3. Activación del modo wireframe

```
1 //cambio del valor de la propiedad "wireframe" del material
2 mesh.material.wireframe = true;
3 //refresco del material para que se apliquen los cambios
4 mesh.material.needsUpdate = true;
```

Ejemplo 4. Visualización de los vértices mediante un sistema de partículas

```
1 //Creación del material para las partículas (atributos: color y tamaño de cada partícula)
2 material = THREE.ParticleBasicMaterial( { color: 0xff0000, size:0.02 } )
3 //Creación del Sistema de partículas (la lista de vértices representa la geometría)
4 particles = new THREE.ParticleSystem([vertices], material );
```

Ejemplo 5. Asignación de la posición y rotación de una cámara

```
1 //La posición de la cámara se asigna mediante un vector 3D
2 camera.position.set(new THREE.Vector3(0.493,1.183,1.819 ));
3 //La rotación de la cámara se indica también mediante un vector 3D
4 camera.rotation = new THREE.Vector3(-0.302,0.788,0.217 );
```

Ejemplo 6. Densificación de los puntos de la trayectoria de una cámara

```
1 //Número de puntos interpolados en el recorrido de la cámara
2 divisions = 1000;
3 //Vector de posiciones y rotaciones de la cámara en los puntos de paso del recorrido
4 positions = [[0.493,1.183,1.819],...,[[0.853,1.234,1.467]];
5 rotations = [[-0.302,0.788,0.217],...,[-0.108,0.654,0.135]];
6 //Obtención de las listas de posiciones y rotaciones interpoladas en el recorrido
7 splinePositions = THREE.SplineCurve3(positions).getPoints(divisions);
8 splineRotations = THREE.SplineCurve3(rotations).getPoints(divisions);
```

Ejemplo 7. Creación de una esfera para un punto de información

```
1 //Geometría de tipo esfera (atributos: radio y número de segmentos horizontales y verticales)
2 geo_sphere = new THREE.SphereGeometry( 0.1, 32, 32 );
3 //Material de la esfera (atributos: color)
4 mat_sphere = new THREE.MeshBasicMaterial( {color: 0xFF0000} );
5 //Creación de la malla (atributos: geometría y material)
6 sphere1 = new THREE.Mesh( geo_sphere, mat_sphere );
7 //Asignación de la posición y de un nombre identificativo a la malla
8 sphere1.position.set(-0.1995,1.0570,0.0454);
9 sphere1.name = 'sphere1';
10 //Se añade la malla a la escena
11 scene.add( sphere1 );
```

Ejemplo 8. Selección de una esfera mediante ray casting

```
1 //Objeto de la clase "Raycaster" (en este caso se utiliza para seleccionar un objeto con el ratón)
2 rayCaster = new THREE.Raycaster();
3 //la trayectoria del rayo vendrá indicada por la posición de la cámara y la posición del ratón
4 rayCaster.setFromCamera( mouse, camera );
5 //Cálculo de las intersecciones del rayo con las mallas 3D de la escena
6 intersects = rayCaster.intersectObjects( scene.children );
7 //Recorrido por los objetos intersectados por el rayo
8 for ( var i = 0; i < intersects.length; i++ ) {
9         //Si el nombre del objeto coincide con 'sphere1'
10        if(intersects[i].object.name == 'sphere1'){
11                  //contenido del mensaje para la ventana de información (formato HTML)
12                  info_popup = '<h3>Graffiti 1</h3> <br>
13                  <img src="images/g1_w.jpg"
14                  style="width:350px;"><br>Original<br>
15                  <img src="images/g1_ds_w.jpg"
16                  style="width:350px;"><br>RGB0 algorithm';
17                  //muestra la ventana de información
11                  popup.alert({content : info_popup});
12 }
```

Ejemplo 9. Visibilidad y opacidad de un modelo

```
1 //Búsqueda de una malla de la escena por su nombre
2 malla = scene.getObjectByName('exterior',true);
3 //Asignación de la propiedad "visible"
4 malla.visible  = true;
5 //la opacidad de la malla se asigna a su material
6 malla.material.opacity = 0.5;
```

Ejemplo 10. Obtención del punto de intersección con el modelo mediante ray casting

```
1 //Cálculo de las intersecciones del rayo con las mallas 3D de la escena
2 intersects = rayCaster.intersectObjects( scene.children );
3 //acceso a las coordendas del punto de intersección entre el rayo y la primera malla
4 point = intersects[0].point;
```

Ejemplo 11. Selección de los vértices de una sección

```
1 //Creación de una geometría vacía
2 section = new THREE.Geometry;
3 //Si el eje elegido para la sección es el eje "X"
4 if (axis == "x"){
5         //Recorrido por los vértices de las partículas
6        for (var i=0; i < particles.geometry.vertices.length; i++){
7                 //Si los vertices están dentro del ancho de la sección (variable "size")
8                 if( (pos_x-size <= particles.geometry.vertices[i].x) &&
9                 (particles.geometry.vertices[i].x<= pos_x+size)){
10                //se añaden a la geometría vacía (sustituye a la geometría original)
11                section.vertices.push(particles.geometry.vertices[i]);
12                }
13       }
14 }
```

Ejemplo 12. Transferencia de los datos de una imagen entre dos páginas

```
1 //Codificación de la imagen en una cadena de texto
2 url = canvas.toDataURL( "image/"+'jpg' );
3 //Se abre una nueva ventana del navegador
4 w = window.open("texture_editor.html");
5 //La función "onload" se ejecuta cuando se carga la página
6 w.onload = function(){
7        //Se copian las dimensiones del canvas de origen al canvas de destino
4        mcvas = w.document.getElementById("mycanvas");
5        mcvas.width = canvas.width;
6        mcvas.height = canvas.height;
11       //Se accede al contexto del canvas (funciones para renderizar gráficos en 2D en este caso)
7        ctx = mcvas.getContext("2d");
13       //Se crea una nueva imagen vacía y se le asigna su fuente (codificada)
8        img = new Image();
16       img.src = url;
17       //Al cargar la imagen se dibuja en el canvas
9        img.onload = function(){
10                ctx.drawImage(img, 0, 0);
11       }
13 }
```