Escuela Politécnica Superior de Jaén
Máster Universitario en Ingeniería Informática

Universidad de Jaén
Escuela Politécnica Superior de Jaén

TRABAJO FIN DE MÁSTER

GESTIÓN DE INFORMACIÓN DE YACIMIENTOS ARQUEOLÓGICOS A TRAVÉS DE HERRAMIENTAS GRÁFICAS EN 3D

Alumno

Alberto Calzado Martínez

Tutores

Lidia Ortega Alvarado
(Departamento de Informática)

Ángel Luis García Fernández
(Departamento de Informática)

Junio, 2021

(Página intencionalmente en blanco)

Universidad de Jaén

Departamento de Informática

Doña Lidia Ortega Alvarado y Don Ángel Luis García Fernández, tutores del Trabajo Fin de Máster titulado: '**Gestión de información de yacimientos arqueológicos a través de herramientas gráficas en 3D**', que presenta Don Alberto Calzado Martínez, otorgan el visto bueno para su entrega y defensa en la Escuela Politécnica Superior de Jaén.

Jaén, Junio de 2021

El alumno:

CALZADO MARTINEZ ALBERTO - 53912964K
Firmado digitalmente por CALZADO MARTINEZ ALBERTO - 53912964K
Fecha: 2021.06.22 15:09:13 +02'00'

Alberto Calzado Martínez

Los tutores:

ORTEGA ALVARADO LIDIA - 24241033E
Firmado digitalmente por ORTEGA ALVARADO LIDIA - 24241033E
Fecha: 2021.06.22 14:14:15 +02'00'

Lidia Ortega Alvarado

Firmado por GARCIA FERNANDEZ ANGEL LUIS - 26038790F el día 22/06/2021 con un certificado emitido por AC FNMT Usuarios

Ángel Luis García Fernández

(Página intencionalmente en blanco)

Agradecimientos

Quiero agrader a mis tutores, Lidia Ortega y Ángel Luis García por la oportunidad dada y por la ayuda ofrecida durante el desarrollo de este proyecto.

También quiero agredecer a mis padres, hermanas y amigos por el apoyo dado estos meses y por aguantar mis quejas, enfados, decepciones y alegrías que este proyecto ha causado en mí.

Por úlitmo, quiero agradecer a mis compañeros de máster, en especial a José Luis por permitirme continuar con el trabajo que él empezó, y a Jesús por ahorrarme tiempo de trabajo con su ayuda.

FICHA DEL TRABAJO FIN DE TÍTULO	
Titulación	Máster en Ingeniería Informática
Modalidad	Trabajo Teórico/Experimental
Idioma	Español
Tipo	General
TFM en equipo	No
Autor/a	Alberto Calzado Martínez
Fecha de asignación	19/10/2020
Descripción corta	La información de un yacimiento arqueológico tiene naturaleza dispar, lo cual dificulta la consulta y el análisis de dicha información por parte de los expertos. En este trabajo se plantea el diseño y la implementación de un prototipo para la gestión de dicha información a través de una interfaz gráfica o GUI, en la cual el usuario es capaz de realizar las operaciones de consulta más habituales a través de operaciones corrientes en los entornos virtuales en 3D, como son la navegación y la interacción. En la actualidad se cuenta con una base de datos de Cástulo, yacimiento situado en las cercanías de Linares. Sobre esta información se pretende obtener modelos 3D para ser utilizados en la aplicación. Sobre estos modelos integrados en la aplicación se realizarán consultas sobre dicha base de datos. También se podrán realizar operaciones de análisis, sobre todo para establecer relaciones espaciales entre los fragmentos encontrados.

Abstract	The diverse nature of the information from an archaeological site makes it difficult for experts to consult and analyze it. This project proposes the design and implementation of a prototype for the management of the information through a graphical interface or GUI, through which the user is able to perform the most common query operations by doing common operations in the 3D virtual environments, such as navigation and interaction. We already have a database from the archaeological site of Castulo, near Linares. Our intent is to combine this information with 3D models, so that the resulting 3D Environment allows us to query the database. Analysis operations can also be performed, especially to establish spatial relationships among the fragments found.

NORMAS APLICADAS EN ESTE DOCUMENTO	
LOCALES	
TFT-UJA:2017	Normativa de Trabajos Fin de Grado, Fin de Máster y otros Trabajos Fin de Título de la Universidad de Jaén (Normativa marco UJA aprobada en Consejo de Gobierno)
TFT-EPSJ:2017	Normativa sobre Trabajos Fin de Grado y Fin de Máster en la Escuela Politécnica Superior de Jaén (Normativa EPSJ aprobada en Junta de Escuela)
TFT-EPSJ	Criterios de evaluación y normas de estilo para TFG y TFM de la Escuela Politécnica Superior de Jaén
NACIONALES E INTERNACIONALES	
ISO 2145:1978	Documentación - Numeración de divisiones y subdivisiones en documentos escritos
UNE 50132:1994	Traducción de la ISO 2145
APA 6ª edición	Estilo de referencias y citas de APA (American Psychological Association)

NORMAS UTILIZADAS COMO BASE O REFERENCIA	
NACIONALES	
UNE 157001:2014	Criterios generales para la elaboración formal de los documentos que constituyen un proyecto técnico
UNE 157801:2007	Criterios generales para la elaboración de proyectos de sistemas de información
*Estas normas se han utilizado **como base o referencia** para la inclusión de algunos contenidos y definiciones sobre elaboración de proyectos, entendiendo como **proyecto** la documentación consensuada entre una empresa y un cliente, que da lugar al perfeccionamiento de un contrato para la elaboración de una obra o la prestación de un servicio. Por consiguiente, no debe esperarse la aplicación de estas normas en cuanto a la completitud de los contenidos ni a la organización de los mismos.*	

Contenido

Índice de ilustraciones

Índice de tablas

1 ESPECIFICACIÓN DEL TRABAJO

En este capítulo se presenta la especificación del trabajo, con una estructura y contenidos **inspirados** en los criterios y recomendaciones que establece la norma UNE 157801:2007 - "*Criterios Generales para la elaboración de proyectos de Sistemas de Información*".

A lo largo del documento se utilizarán términos y acrónimos cuya descripción aparecen en el apartado 7 (Definiciones y abreviaturas).

1.1 Introducción

La documentación es una de las tareas más importantes durante el proceso de excavación, incluida tanto en el momento en el que se desentierran los hallazgos, como una vez que estos están bajo estudio en un laboratorio. En todo caso, este proceso debe apoyarse en una metodología precisa que permita registrar toda la información relevante sobre los hallazgos, recopilando, por ejemplo, información acerca del material y/o datación. Partiendo de la premisa "excavación es destrucción", el objetivo es evitar cualquier pérdida de información futura.

Los métodos tradicionales, principalmente manuales, proporcionan información muy dispar, desde información textual hasta fotografías digitales o bocetos hechos a mano. Aunque estas entradas 2D son muy enriquecedoras, suponen un nivel de abstracción, al representar una realidad tridimensional de manera bidimensional. Aún así, estos métodos manuales se han visto enriquecidos en la actualidad con herramientas digitales para añadir la información a la base de datos.

Hoy en día, la limitación del 2D se ha resuelto en gran medida con técnicas basadas en captura 3D, como fotogrametría, escaneo láser o LiDAR (Light Detection and Ranging). Los modelos 3D ofrecen ventajas, principalmente en términos de visualización, pero también para el análisis y el intercambio de datos. Estas tecnologías pueden solucionar la falta de precisión en la geometría.

Sin embargo, la integración de información 3D con una base de datos tradicional sigue siendo un desafío. La recopilación 3D debe ir más allá de la visualización. La forma, posición y apariencia de los objetos virtuales deben reproducir la realidad que representan de la manera más fiel posible. Esto supone un nuevo

enfoque que genera conocimientos avanzados en comparación con los métodos clásicos.

1.2 Objetivos del trabajo

Los principales objetivos que se desea abordar en este trabajo son los siguientes:

- Adquirir y/o ampliar conocimientos relacionados sobre el ámbito arqueológico.
- Implementar una aplicación en RV (Realidad virtual) que permita la recreación virtual de un yacimiento arqueológico y la manipulación de modelos 3D de sus artefactos.
- Definir una forma de trabajo que permita el escaneado tanto del terreno como de las piezas en un sitio arqueológico real.

1.3 Antecedentes y estado del arte

Actualmente, las técnicas de documentación arqueológica están evolucionando hacia un enfoque multifacético. El objetivo principal es guardar el conocimiento que desaparece cuando se extrae un hallazgo para futuros análisis (Lucas, 2012). Para ello, los métodos tradicionales están añadiendo algunas de las siguientes características:

- Digitalizar la información proveniente de diferentes fuentes para formar parte de un mismo registro. Las imágenes 2D o los modelos detallados en 3D de los hallazgos son cada vez más frecuentes en la documentación de zonas arqueológicas (Cosmas et al., 2003).
- Compartir información con la comunidad científica a través de plataformas web (Galeazzi et al., 2016) o arquitecturas clientes-servidor. Esto permite un acceso eficiente y completo a expertos de cualquier parte del mundo. Además, permite acceder a los registros y participar en el proceso de análisis únicamente utilizando un ordenador conectado a Internet.

- Incluir todas las etapas involucradas en el proceso, desde la captura y documentación inicial hasta el análisis y difusión final, en un sistema de información integrado (Meghini et al., 2017). Esto significa que el proceso de captura *in situ* alimenta la base de datos, lo que inevitablemente implica el uso de dispositivos ubicuos (Austin, 2014).
- Desarrollar interfaces para acceder a la información lo más *amigables* posible, ya que a estos sistemas colaborativos acceden expertos con diferentes grados de conocimiento (Power et al., 2017).

Todas las características descritas anteriormente son deseables para recrear el yacimiento arqueológico. De esta manera, la expresión “excavación es destrucción” es sustituida por “excavación es reconstrucción digital”. Así, un experto que no participó en el proceso de excavación puede conocer todos los detalles del sitio como si hubiera estado allí, asi como el posicionamiento original de la pieza.

Hoy en día tenemos a nuestra disposición muchas posibilidades para capturar los detalles de los lugares y objetos en 3D, como escáneres láser (Martínez-Fernández et al., 2020), LiDAR (Richards-Rissetto, 2017) o fotogrametría (Eltner & Sofia, 2020) que son ampliamente utilizadas en arqueología (Armstrong et al., 2018; Galeazzi, 2016). Estas permiten determinar el material, la datación, la presencia de posibles inscripciones o examinar en detalle el deterioro de las piezas. Sin embargo, un objetivo adicional es capturar información espacial subyacente a la zona. La posición original de las piezas junto a su disposición en el terreno son necesarias para la reconstrucción virtual de la escena (D’Urso et al., 2018). La forma en la que se depositan los restos, por ejemplo, en cementerios humanos, puede proporcionar información acerca de las tradiciones religiosas o las relaciones sociales en un asentamiento.

En comparación con una representación 2D, la virtualización 3D enriquece la experiencia de los arqueólogos (Koller et al., 2010). La mayoría de los modelos 3D están pensados para ofrecer una mejor experiencia al usuario utilizando, por ejemplo, AR (Realidad Aumentada) (Bekele et al., 2018; Benko et al., 2004; Ridel et al., 2014). Además, se consigue una mayor precisión geométrica en comparación con los dibujos (Jensen, 2018). Las fotografías tomadas durante las excavaciones tienen limitaciones ya que se toman desde un ángulo específico y no muestran aquellas partes de las

piezas que aún están enterradas. Esto puede solucionarse utilizando una escena en 3D de la zona y permitiendo al usuario moverse libremente por ella.

Sin embargo, la virtualización 3D de un sitio arqueológico todavía tiene importantes deficiencias que abordar. Por un lado, un escaneo 3D de la zona produce modelos muy pesados, con millones de puntos, que son difíciles de manejar con el hardware de un ordenador estándar. Además, el software capaz de visualizar estos modelos no está integrado en el gestor de base de datos. Por tanto, el modelo 3D suele ser información complementaria que se encuentra fuera del sistema de información. Algo similar ocurre con los modelos 3D de las piezas que se desentierran. En los pocos casos en los que se integran, los modelos 3D solo se utilizan para visualización, pero aislados del lugar del descubrimiento (Seifert et al., 2017). Por lo tanto, no incluyen su posición original ni las relaciones espaciales con hallazgos cercanos. En resumen, en la literatura encontramos una disociación entre los modelos 3D y el resto del sistema de información (Griffo et al., 2019). De esta forma es difícil establecer la relación espacial entre los hallazgos y su entorno. No obstante, esto puede solucionarse añadiendo relaciones topológicas para permitir este análisis espacial. Esto permite recuperar información crucial que desaparece después de la excavación (Ohori et al., 2015; Zlatanova et al., 2004).

Una solución para añadir información espacial proviene de los Sistemas de Información Geográfica (SIG) que integran la visualización y gestión de datos (Caggiani et al., 2012; Wheatley & Gillings, 2013). Los SIG 3D superan a los sistemas 2D al incorporar datos espaciales con modelos tridimensionales bajo el mismo marco (Katsianis et al., 2021; Landeschi, 2019). Sin embargo, existen limitaciones subyacentes a estas herramientas, ya que aportan soluciones genéricas con un alto nivel de abstracción, es decir, alejadas de los procedimientos arqueológicos (Richards-Rissetto, 2017). Los SIG 3D todavía se encuentran en las primeras etapas de su desarrollo, con capacidades limitadas en términos de integración 3D, interfaces usables y ubicuidad. Por esta razón, muchos proyectos de investigación optan por desarrollar sus propias herramientas software (Austin, 2014).

Por todo ello, se propone una aplicación que permita la visualización 3D del yacimiento arqueológico para replicar la distribución original de las piezas. En ella, se facilitará la recolocación de los modelos 3D y se actualizará la base de datos con la información modificada. También se pretende incluir un sistema de referencia basado en imágenes o en modelos 3D de los terrenos para ayudar a la manipulación. La

obtención de los modelos 3D se llevará a cabo con dispositivos móviles o con escáneres portátiles mediante diversas tecnologías, considerándose un proceso ágil.

1.4 Requisitos iniciales

En este apartado se van a enumerar los requisitos de manera detallada. Entendiéndose como requisito una necesidad documentada de un producto o servicio. Los requisitos establecen **qué** debe hacer el sistema, pero **no cómo** hacerlo.

Se van a describir los siguientes tipos de requisitos:

- **Requisitos funcionales**: describen la funcionalidad del sistema, qué debe hacer o qué se espera que haga.
- **Requisitos no funcionales**: describen aspectos del sistema relacionados con el grado de cumplimiento de los requisitos funcionales.

1.4.1 Requisitos funcionales

A continuación, se listan los requisitos funcionales:

- La base de datos debe almacenar información relevante de cada zona de excavación, así como cualquier otra información que se considere oportuna.
- La aplicación debe permitir filtrar los datos que se muestran en cada momento en pantalla.
- La aplicación debe permitir un movimiento libre tan cómodo como sea posible.
- La aplicación debe mostrar la información de los datos cuando el usuario lo requiera.
- La aplicación debe permitir la manipulación de los hallazgos para posicionarlos y rotarlos, asi como almacenar estos cambios en la base de datos modificando la misma como consecuencia.
- El sistema debe permitir utilizar fotografías y modelos 3D del terreno como ayuda a la manipulación.

- La aplicación debe permitir la visualización de diferentes datos asociados a los hallazgos, como sus relaciones topológicas o la distancia respecto al suelo.

1.4.2 Requisitos no funcionales

A continuación, se listan los requisitos no funcionales:

- La aplicación debe minimizar el tiempo de respuesta. Sobre todo, al cargar modelos 3D desde ficheros.
- La aplicación debe utilizar la mínima cantidad de memoria posible.
- La interfaz gráfica asociada a la aplicación debe ser intuitiva y fácil de utilizar por cualquier usuario independientemente de su conocimiento y/o experiencia.
- El sistema debe ser tolerante a errores para evitar cierres bruscos.
- Los ficheros utilizados para la carga de modelos 3D deben ser lo más simples posible, tanto para su uso en la base de datos como en la aplicación.

1.5 Alcance

En este apartado se enumerar todos los entregables vinculados al proyecto de programación y solicitados por la guía del TFM:

- Documentación del proyecto con el objetivo de exponer todo lo desarrollado, incluidos los manuales de usuario necesarios para su replicación y ejecución.
- Código fuente de la aplicación donde se puede encontrar la implementación de todos aquellos comportamientos que se describen en este documento. Incluye tanto los ficheros C# (C Sharp) propios como aquellos ficheros pertenecientes a Unity. También incluyen los ficheros de los modelos 3D escaneados.
- Ejecutable del proyecto, el cual permite comprobar la solución desarrollada.

- Copia de respaldo de la base de datos junto al código de la API REST para poder replicarla en cualquier servidor.
- Vídeo de ejemplo de la aplicación, donde se muestra el funcionamiento de la misma.

1.6 Hipótesis y restricciones

El TFM se define como una asignatura de 12 créditos, lo que supone que la duración total del proyecto será de 300 horas, incluyendo todas las etapas del ciclo de vida, con la excepción del mantenimiento. Por consiguiente, la principal restricción aplicable es la limitación de la duración del trabajo.

Por otro lado, debido a la situación social actual causada por la pandemia de COVID-19, no ha sido posible trabajar en un yacimiento real, por lo que todas las escenas, incluyendo la mayoría de los modelos 3D utilizados, han sido simuladas por el autor del trabajo.

1.7 Tecnologías utilizadas

A continuación, se van a enumerar cada una de las tecnologías utilizadas durante el desarrollo del proyecto, junto a una breve descripción de las mismas.

En primer lugar, tenemos el software y bibliotecas utilizadas para la implementación del proyecto:

- **XAMPP**[1]: paquete de software libre que incluye el servidor web Apache y el intérprete para el lenguaje de script PHP.
- **Visual Studio Code**[2]: editor de código fuente que admite muchas funcionalidades. Utilizado para la programación en PHP.
- **PostgreSQL**[3]: sistema gestor de base de datos relacional. Junto a su extensión **PostGIS**[4], nos permite almacenar toda la información de un yacimiento.

[1] https://www.apachefriends.org/es/index.html
[2] https://code.visualstudio.com/
[3] https://www.postgresql.org/
[4] https://postgis.net/

- **Unity**[5]: motor de videojuegos que nos permite la creación de la aplicación aportando funcionalidades como la física, animaciones, etcétera.
- **Microsoft Visual Studio 2019**[6]: entorno de desarrollo (IDE, *Integrated Development Environment*) usado para la programación en distintos lenguajes de programación. En nuestro caso, lo hemos utilizado para la programación de los scripts de Unity en C#.
- **Assets de terceros** provenientes de la tienda de Unity[7] para dotar al proyecto con funcionalidad extra.

A este software se le debe añadir software utilizado para la creación o edición de los distintos modelos utilizados en el proyecto:

- **CloudCompare**[8]: software de procesamiento de nubes de puntos 3D y mallas. Utilizado para el recorte de las mallas del terreno.
- **Blender**[9]: software dedicado al modelado de objetos 3D. Utilizado para la simplificación de modelos 3D.
- **Qlone**, **SCANN3D y 3D Live Scanner**: aplicaciones móviles para escaneado de objetos.
- **Artec Studio y EXScan Pro**: software propietario para los escáneres portátiles con los que se obtienen los modelos 3D.

Por último, queda enumerar aquellos programas que se han utilizado para elaborar la documentación del proyecto:

- Edición de imágenes: **GIMP**[10] y **Draw.io**[11]. Para la creación de algunas ilustraciones.

[5] https://unity.com/es
[6] https://visualstudio.microsoft.com/es/vs/
[7] https://assetstore.unity.com/
[8] https://www.danielgm.net/cc/
[9] https://www.blender.org/
[10] http://www.gimp.org.es/
[11] https://app.diagrams.net/

- Documentación del proyecto: software del paquete **Microsoft Office**[12]: Microsoft Word y Microsoft Power Point. Software empleado para escribir la memoria del proyecto.
- Creación de diagramas UML: **Visual Paradigm**[13].
- Planificación temporal: **Gantt Project**[14]. Permite crear y asignar tareas en un diagrama de Gantt.

1.8 Metodología de desarrollo de software

Antes de comenzar la implementación del sistema que se ha propuesto, es necesario definir una metodología de trabajo. Las metodologías de trabajo están pensadas para estructurar, planificar y controlar el proceso de desarrollo de sistemas informáticos.

Actualmente existen varias metodologías a tener en cuenta para la elaboración de un proyecto software. A continuación, se explicarán de forma breve algunas de ellas, con sus ventajas y desventajas para finalmente decidir la metodología a seguir.

Empezando por los métodos tradicionales, tenemos al **modelo en cascada** (Aziz, 2012). Este se caracteriza por seguir un comportamiento lineal, en el que cada etapa se empieza al acabar la etapa anterior. Esto puede verse en la Ilustración 1.

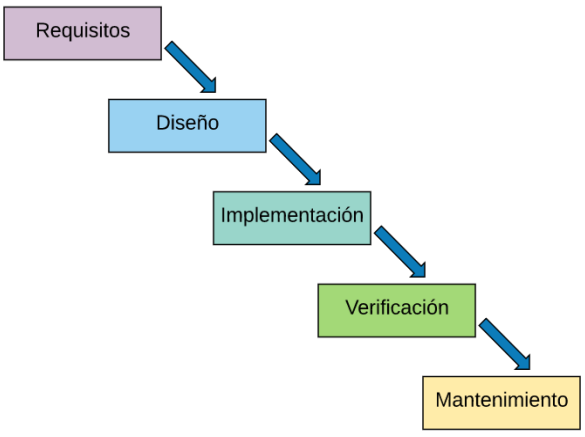


Ilustración 1: Etapas del modelo en cascada
(https://ivan395.github.io/Web/modelos.html)

[12] https://www.microsoft.com/es-es/microsoft-365
[13] https://www.visual-paradigm.com/
[14] https://www.ganttproject.biz/

Las principales ventajas de este modelo son:

- Los requisitos se identifican en las primeras etapas.
- Facilita la detección de errores.
- Es más fácil de implementar que otros métodos.

Los principales inconvenientes son:

- Una vez empezada una etapa no podemos volver a la etapa anterior.
- No es recomendable para cambios continuos de requisitos.
- Si se produce un cambio en alguna de las etapas, se debe volver a realizar las etapas anteriores.

El siguiente modelo es el **modelo basado en prototipos** (Carr & Verner, 2021). Este utiliza prototipos visuales para obtener la interfaz de usuario final. Posteriormente se desarrolla todo lo necesario para dotar al prototipo final de funcionalidad (ver Ilustración 2).

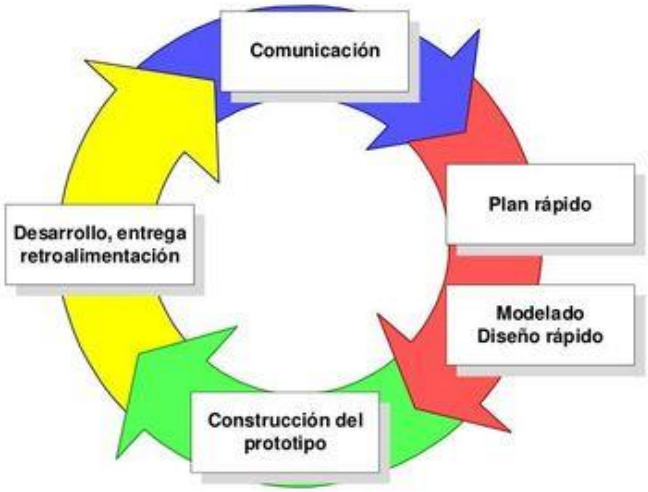


Ilustración 2: Etapas del modelo basado en prototipos
(https://ivan395.github.io/Web/modelos.html)

Las ventajas de este modelo son:

- Proporciona una idea clara al cliente sobre sus requisitos.
- Se reduce el número de errores al mostrar un prototipo del resultado final.
- Genera un entorno cooperativo entre cliente y desarrollador.

Los inconvenientes son:

- El usuario puede pensar que el prototipo es un resultado funcional del producto.
- La generación del prototipo final puede ser lenta si se incluyen muchos cambios en las reuniones para enseñar al cliente cada prototipo.

Otro modelo disponible es el **modelo incremental** (Aziz, 2012). En este se combinan elementos del modelo en cascada con un enfoque iterativo e incremental, en el que en cada incremento se añade una funcionalidad nueva hasta que todas las funcionalidades deseadas quedan implementadas (ver Ilustración 3).

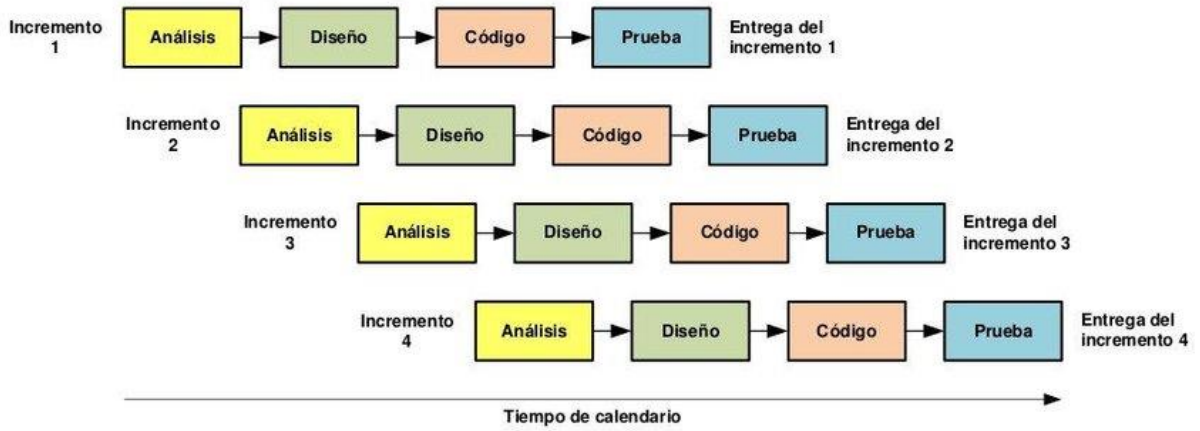


Ilustración 3: Etapas del modelo incremental
(https://www.researchgate.net/figure/Figura-10-Modelo-de-proceso-incremental-Fuente_fig6_326571456)

Las principales ventajas de este modelo son:

- Es flexible a cambios en los requisitos del cliente.
- Se consigue un producto funcional desde los incrementos iniciales.
- Presenta un menor riesgo de fallo comparado con otros modelos.

Los principales inconvenientes son:

- No es recomendable para sistemas de tiempo real, alto nivel de seguridad o de procesamiento distribuido.
- Requiere de mucha planificación.

Por último, tenemos el **modelo en espiral** (Aziz, 2012). Este modelo enlaza la naturaleza iterativa de la construcción de prototipos, pero conservando algunas

propiedades del modelo en cascada, con un fuerte énfasis en el análisis de riesgos. Sus etapas se muestran de forma gráfica en la Ilustración 4.

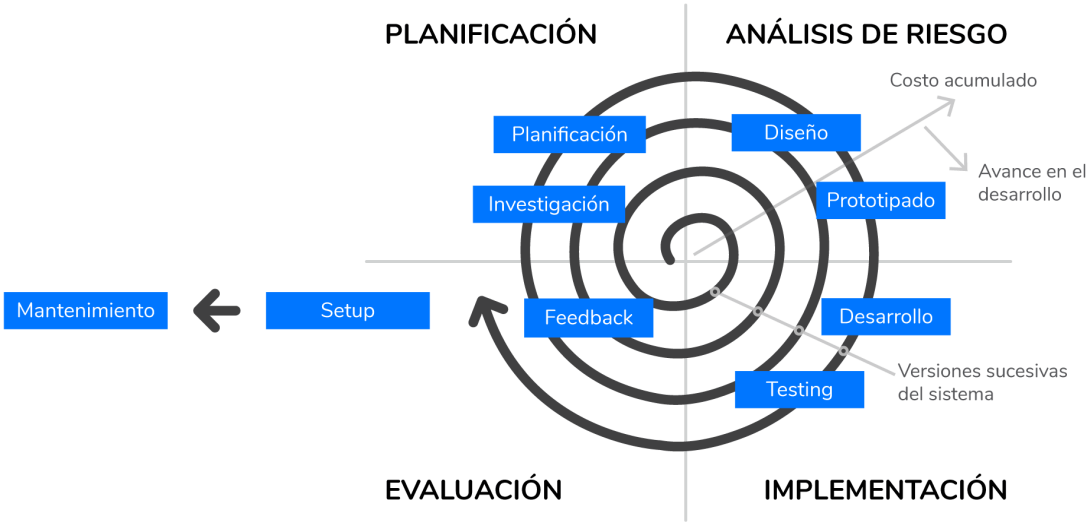


Ilustración 4: Etapas del modelo en espiral
(https://aspgems.com/metodologia-de-desarrollo-de-software-iii-modelo-en-espiral/)

Sus principales ventajas son:

- Se realiza un análisis de riesgos profundo, lo que permite reducir los riesgos considerablemente.
- Se puede añadir funcionalidad adicional en etapas posteriores.
- El desarrollo es rápido y las características se añaden de forma sistemática.

Sus inconvenientes son:

- Es muy costoso.
- El desarrollo del sistema dura demasiado tiempo.
- No es recomendable para proyectos pequeños.

En contraposición con los métodos tradicionales, aparecieron los **métodos ágiles** (Saravia & Elizabeth, 2012). En ellos, las necesidades y soluciones evolucionan a través de una colaboración estrecha entre equipos multidisciplinares. Es por esto que se caracterizan por enfatizar la comunicación frente a la documentación, por el desarrollo evolutivo y por su flexibilidad.

Generalmente, este tipo de metodología funciona con ciclos iterativos e incrementales, por lo que el software se entrega en partes pequeñas y utilizables a las que se va añadiendo funcionalidades hasta terminar el proyecto (ver Ilustración 5). Cada iteración comprende a su vez las etapas de análisis, diseño implementación y pruebas, repitiéndose en cada una de las iteraciones hasta obtener el producto completo.

Dentro de las metodologías ágiles nos podemos encontrar varios marcos de trabajo como, por ejemplo:

- **Programación extrema** (XP, *eXtreme Programming*): El marco de trabajo XP potencia las relaciones personales a través del trabajo en equipo fomentando la comunicación.
- **SCRUM**: Es el marco de trabajo más conocido y utilizado. Trabaja de forma incremental en iteraciones denominadas Sprint.
- **Kanban**: Caracterizado por la elaboración de un cuadro visual en el que se reflejan, generalmente, tres columnas de tareas (pendientes, en proceso y terminadas).

Ilustración 5: Etapas del desarrollo ágil
(https://www.tithink.com/es/2018/10/16/metodologias-agiles-que-son-y-para-que-sirven/)

Las ventajas de las metodologías ágiles son:

- Generan soluciones durante el proceso de trabajo sin necesidad de tener que esperar al final.
- La entrega del producto o servicio es más rápida.
- Al crear prioridades se optimizan los recursos.

Los principales inconvenientes son:

- Se depende en gran medida del líder del equipo.
- Puede haber faltas de documentación de los archivos del proyecto.
- Pueden surgir soluciones erróneas que conllevan grandes consecuencias en pleno trabajo de producción.

Una vez vistos todos los métodos de trabajo, vamos a elegir el que mejor se adapte a nuestro proyecto.

En nuestro caso, vamos a utilizar como núcleo de nuestra metodología de desarrollo, el **modelo incremental**, puesto que partimos de una aplicación ya desarrollada a la que le vamos a ir añadiendo nueva funcionalidad.

Aun así, no se va a seguir el esquema propuesto por el proceso incremental de forma rígida, ya que se pretenden utilizar varias técnicas ágiles que se utilizan en marcos de trabajo como SCRUM o Kanban.

Algunas de estas técnicas son:

- **Sprint**: es el núcleo de SCRUM. Un sprint representa cada uno de los ciclos o iteraciones por los que va a pasar el proyecto, los cuales suelen tener duraciones aproximadas, aunque puede considerarse una duración variable dependiendo del objetivo a desarrollar en cada iteración.
- **Tablero de tareas:** similar al tablero comúnmente utilizado en el marco de trabajo Kanban en el cual se visualiza el estado de cada una de las tareas. En este caso no se utilizará un tablero visual, pero se colocarán las tareas en un documento que dicte en qué fase se encuentra cada una.
- **Reuniones ágiles:** aunque el trabajo está desarrollado por una única persona, las reuniones se mantienen con los tutores que evalúan este trabajo. Por tanto, de forma periódica (por ejemplo, cada semana, dos semanas, etc.) se mantendrán reuniones con ellos para simular las reuniones llevadas a cabo en metodologías ágiles.

1.9 Análisis de riesgos

En este apartado vamos a identificar los riesgos que pueden surgir a lo largo de todo el desarrollo. Para ello utilizaremos una matriz de riesgos, ya que nos permite capturar la información más relevante de los riesgos identificados y evaluarlos según su probabilidad de ocurrencia y su nivel de impacto (Talbot, 2017)

Su objetivo es la visualización de los riesgos para lograr una gestión de los mismos que permita disminuir la probabilidad de que acontezcan, y si llegan a suceder, poder minimizar el impacto negativo que estos pueden llegar a tener en el proyecto.

Su construcción se basa en el uso de diversas tablas, y para ello se deben de seguir los siguientes pasos: identificación de los riesgos, evaluación de la probabilidad de ocurrencia, así como su impacto y/o consecuencias y la elaboración gráfica final de la matriz.

Una vez que se han analizado los posibles riesgos, se debe planificar una respuesta a cada uno de ellos para determinar las medidas que se deben adoptar o los recursos que se deben utilizar para reducir los daños que un riesgo pueda producir. Esta respuesta se conoce como plan de contingencia. Las decisiones a la hora de generar un plan de contingencia para un riesgo se basan principalmente en encontrar un balance entre el coste de desarrollar una respuesta y su impacto potencial

1.9.1 Identificación de los posibles riesgos

Los posibles riesgos que se han podido identificar son los siguientes:

- **Riesgo R1**: ocurre un desastre o catástrofe natural que provoca daños irreparables en la zona de trabajo.
- **Riesgo R2**: fallo hardware de algún dispositivo que se utilizará durante el desarrollo por lo que queda inutilizable durante un periodo de tiempo.
- **Riesgo R3**: el trabajador a cargo del proyecto tiene una lesión o enfermedad grave y se da de baja al menos un mes.
- **Riesgo R4**: el trabajador a cargo del proyecto tiene una lesión o enfermedad leve y se da de baja como máximo un mes.
- **Riesgo R5**: el trabajador no tiene los conocimientos o habilidades requeridas para llevar a cabo el proyecto.

- **Riesgo R6**: se producen cambios muy significativos en los requisitos, objetivos y/o alcance del proyecto.
- **Riesgo R7**: retraso en algunos de los incrementos en los que se divide el proceso de desarrollo.
- **Riesgo R8**: se parte de un proyecto mal diseñado y mal desarrollado que carece de calidad.

1.9.2 Construcción de la matriz

La matriz de riesgos que se va a construir es una matriz que relaciona la probabilidad de que ocurra el riesgo con el impacto que este puede producir al desarrollo del trabajo.

Se empieza por una definición cualitativa de cada una de las probabilidades que vamos a tener en cuenta, mostrada en la Tabla 1.

Probabilidad	
Improbable	Es improbable que ocurra
Raro	Poco probable que ocurra en algún momento
Posible	Es probable de que se produzca
Probable	Bastante probable de que ocurra en el tiempo
Frecuente	Casi certeza de que se produzca inmediatamente o en un corto periodo de tiempo de forma habitual

Tabla 1: Definición de los tipos de probabilidad

La Tabla 2 define el tipo de daño que puede producirse en el proyecto.

Daños	
Despreciable	Daño no perjudicial que puede ser subsanado de forma rápida
Mínimo	Daño mínimo que puede ser solucionado en un periodo corto de tiempo
Moderado	Daño significativo: lesiones, pérdida económica, incumplimientos legales, y/o pérdida de reputación
Peligroso	Daño grave: lesiones, pérdida económica, incumplimientos legales, y/o pérdida de reputación
Catastrófico	Daño irreparable

Tabla 2: Definición de los tipos de daños

Una vez definidas de forma cualitativa las probabilidades y los impactos, se deben definir los tipos de riesgos asociados (Tabla 3). Para una mejor visualización posterior, se definen con una serie de colores atendiendo a su peligrosidad.

Riesgo		
E	Extremo	Riesgos a los que hay que prestarle mucha atención por ser catastróficos o muy probables
A	Alto	Riesgos potenciales y probables. Es necesario aplicar técnicas para reducir el daño producido.
M	Moderado	Riesgos poco frecuentes o cuyo impacto no es peligroso
B	Bajo	Riesgos con probabilidad mínima o con daños sin importancia

Tabla 3: Definición de los tipos de riesgos

Como paso previo a la construcción de la matriz de riesgos, vamos a crear un esquema de colores para ver de forma rápida la clasificación de los riesgos atendiendo

a su probabilidad de ocurrencia y a su impacto. El esquema queda definido en la Tabla 4.

	Improbable	Raro	Posible	Probable	Frecuente
Catastrófico	M	A	A	E	E
Peligroso	M	M	A	A	E
Moderado	B	M	M	A	A
Mínimo	B	B	M	M	M
Despreciable	B	B	B	M	M

Tabla 4: Esquema de colores para la matriz de riesgos

Como puede verse en la Tabla 4, a medida que la probabilidad y el daño aumentan, los riesgos se vuelven cada vez más peligrosos, y por tanto es necesario tomar más medidas para disminuir o evitar el daño producido.

A continuación, utilizando el esquema anterior y los riesgos definidos en el apartado 1.9.1 (Identificación de los posibles riesgos), se construye la matriz de riesgos resultante (Tabla 5).

	Improbable	Raro	Posible	Probable	Frecuente
Catastrófico	R1				
Peligroso	R5, R8	R6			
Moderado		R2	R4	R7	
Mínimo		R3			
Despreciable					

Tabla 5: Matriz de riesgos

1.9.3 Planes de contingencia

Generalmente, existen cuatro respuestas o estrategias que podemos poner en marcha para un riesgo. Estas son:

- **Evitar**. Realizar cambios en el proyecto para evitar el riesgo. Puede implicar cambios en el cronograma o en el alcance del proyecto.
- **Transferir**. Trasladar la amenaza a un tercero junto con la responsabilidad de la respuesta.

- **Mitigar**. Disminuir la probabilidad y/o impacto de que se produzca el riesgo.
- **Aceptar**. No tomar ninguna medida a menos que el riesgo suceda. Pensada, normalmente, cuando no es viable o rentable abordar el riesgo de otra manera.

De este modo, las estrategias tomadas y los planes de contingencia para cada riesgo vienen definidos en la Tabla 6.

Estrategias y planes de contingencia	
Riesgo 1 (catástrofes)	Aceptar
Puesto que es totalmente imprevisible y muy improbable que ocurra el riesgo, no se ha pensado en ningún plan de contingencia para ello.	
Riesgo 2 (fallo hardware)	Mitigar
Es recomendable utilizar el hardware de la manera correcta para evitar posibles daños por el uso. En caso de avería, contactar con el fabricante para recurrir a la reparación o en su caso, a la garantía.	
Riesgo 3 (enfermedad grave)	Mitigar
Utilizar parte del presupuesto dedicado a sobrecoste para contratar a otro trabajador que realice el trabajo.	
Riesgo 4 (enfermedad leve)	Aceptar
Esperar al trabajador durante el tiempo de baja. Es posible que se produzcan varios días de retraso con respecto al cronograma inicial.	
Riesgo 5 (conocimientos y/o habilidades)	Mitigar
Dotar al trabajador del conocimiento necesario para el uso de las tecnologías propuestas. Utilizar tecnologías conocidas por parte del trabajador también puede ser una buena medida.	
Riesgo 6 (cambios significativos)	Mitigar
Documentar los requisitos, el alcance y los objetivos al principio del inicio del proyecto para su aprobación por parte del equipo.	

Riesgo 7 (retrasos)	Evitar
Realizar un buen estudio de la planificación temporal del trabajo para adecuarse a los plazos aprobados. Planificar el desarrollo de las tareas de cada iteración atendiendo a su importancia.	
Riesgo 8 (mal diseño)	Aceptar
Debido a que el proyecto ya ha sido desarrollado por otro alumno, es bastante improbable que esté mal diseñado. Por este motivo, no se va a tomar ninguna medida contra este riesgo.	

Tabla 6: Estrategias y planes de contingencia

1.10 Estimación del tamaño y esfuerzo

Ya que el presente proyecto es un TFM, no existen restricciones de tipo económico, sino de tipo temporal (un número aproximado de horas). Por consiguiente, los cálculos de tamaño del proyecto están supeditados el tiempo disponible. En cuanto al esfuerzo, se dispone de tan un solo efectivo (la persona autora del trabajo).

Aun así, el cálculo de la estimación se puede llevar a cabo mediante múltiples modelos, siendo uno de los más conocidos el modelo COCOMO (COnstructive COst MOdel), desarrollado por Boehm en 1981.

COCOMO hace una distinción de tres modelos (Universidad del Pais Vasco, 2021):

- **Básico**: trata de determinar el coste del proyecto de una manera rápida en función de líneas de código estimadas.
- **Intermedio**: se utiliza para estimaciones más complejas, incluyendo hasta 15 atributos (clasificados en 4 categorías) para determinar el coste del proyecto.
- **Detallado**: incorpora los atributos del modelo intermedio y puede llevar a cabo una evaluación del coste en cada etapa del proyecto.

Además, distingue tres tipos de proyectos:

- **Orgánico**: proyecto relativamente sencillo, con pocos miles de líneas de código, el cual está desarrollado por programadores expertos en un entorno estable y familiar.

- **Empotrado**: proyectos complejos en los que apenas se tiene experiencia, con gran innovación técnica, requisitos muy restrictivos y gran volatilidad.
- **Semilibre**: proyecto con características de los otros dos.

Las ecuaciones utilizadas por los tres modelos de COCOMO son:

$$Esfuerzo = a * KLOC^{b} * M(X)$$

$$Tiempo\ de\ desarrollo = c * Esfuerzo^{d}$$

$$Personas = \frac{Esfuerzo}{Tiempo\ de\ desarrollo}$$

Los valores a, b, c y d dependen del tipo del proyecto. KLOC hace referencias a la cantidad de líneas de código (en miles) que se pretende escribir, mientras que la constante M(X) es un multiplicador utilizado en el modelo intermedio o detallado.

Descrito esto, es necesario seleccionar un modelo de COCOMO, así como el tipo de proyecto que mejor se ajuste al proyecto a desarrollar. En este caso, el modelo elegido es el **intermedio**, ya que el modelo básico es demasiado impreciso y el modelo detallado requiere de demasiado conocimiento de las etapas del producto. Para el tipo de proyecto se ha elegido el **semilibre**, puesto que el autor del trabajo no dispone de experiencia previa en algunas de las tecnologías a utilizar, pero se trabaja sobre un entorno más o menos estable.

Ya que se ha elegido el modelo intermedio, es necesario indicar la valoración en cada uno de los atributos que se evalúan. Esto está detallado en la Tabla 7. Los valores por debajo de 1 disminuyen el esfuerzo, mientras que los valores por encima de 1, lo aumentan. La tabla de la que se extraen los datos se encuentra en el apartado 6.3 (Tablas de referencia para el modelo COCOMO).

Atributos	Valor
Fiabilidad	Nominal (1)
Tamaño de la base de datos	Muy alto (1,16)
Complejidad	Nominal (1)
Restricciones de tiempo de ejecución	Alto (1,11)
Restricciones de memoria virtual	Nominal (1)
Volatilidad de la máquina virtual	Bajo (0,87)
Tiempo de respuesta	Alto (1,07)
Capacidad de análisis	Nominal (1)
Experiencia en la aplicación	Muy bajo (1,29)
Capacidad del programador	Muy alto (0,7)
Experiencia en la máquina virtual	Alto (0,90)
Experiencia en el lenguaje	Alto (0,95)
Técnicas actualizadas de programación	Nominal (1)
Utilización de herramientas de software	Muy alto (0,83)
Restricciones de tiempo de desarrollo	Muy alto (1,10)
Multiplicador	0,8449

Tabla 7: Multiplicadores del coste en el modelo COCOMO

Este factor multiplicador es lo que en las fórmulas propuestas por el modelo COCOMO se conoce como *M(X)* y se calcula como resultado de la multiplicación de la valoración dada a cada atributo.

Lo siguiente a calcular es el número de líneas de código. El proyecto en su totalidad se divide en código escrito para la parte del servidor (API REST) y la parte del cliente (aplicación VR). En ambos casos se utilizan códigos de terceros para aumentar las funcionalidades, aunque estas líneas no se han tomado en cuenta para este cálculo. Por tanto, estimamos que la API contendrá 3000 líneas de código, mientras que el proyecto en VR contendrá alrededor de 5500 líneas.

Los coeficientes a, b, c y d vinculados al proyecto semilibre en el modelo intermedio vienen referenciados en el apartado 6.3 (Tablas de referencia para el modelo COCOMO).

Aplicando estos valores a las formulas anteriores, obtenemos los siguientes resultados:

$$Esfuerzo = 3 * 8{,}5^{1{,}12} * 0{,}8449 = 27{,}85\ personas/mes$$

$$Tiempo\ de\ desarrollo = 2{,}50 * 27{,}85^{0{,}35} = 8{,}01\ \approx 8\ meses$$

$$Personas = \frac{27{,}85}{8{,}01} = 3{,}48\ \approx 4\ personas$$

Un proyecto de estas características necesitaría de 4 trabajadores durante aproximadamente 8 meses para completarse, aunque como se expuso al principio, las condiciones de ejecución no son viables debido a las restricciones de tiempo (300 horas) y personal (un único autor).

1.11 Planificación temporal

En este apartado, vamos a describir la planificación temporal del desarrollo del proyecto. La planificación temporal se lleva a cabo siguiendo una metodología de desarrollo iterativa, por lo que se le asignará a cada iteración una cantidad de días de trabajo equivalente al esfuerzo estimado para cada una de ellas. La estimación queda de la siguiente manera:

- **Iteración 0**: Estudio del proyecto de partida. Duración estimada de 15 días.
- **Iteración 1**: Implementación de mejoras y cambios propuestos a partir de la iteración anterior. Duración estimada de 20 días
- **Iteración 2**: Implementación de gizmos para la manipulación de los elementos. Duración estimada de 30 días.
- **Iteración 3**: Lectura de los modelos 3D tanto en la base de datos como en Unity. Duración estimada de 10 días
- **Iteración 4**: Simulación de una nueva disposición. Duración estimada de 15 días.
- **Iteración 5**: Simulación de la metodología durante el proceso de excavación. Duración estimada de 25 días.

- **Documentación**: Elaboración de la memoria final. Duración estimada de 25 días.

Debemos hacer especial hincapié en la parte de documentación. Esta no se considera como una iteración tal cual, sino más bien una tarea que está incluida en todas las iteraciones tenidas en cuenta. No obstante, se concluirá el proyecto dedicando algunos días a dar forma a la documentación final.

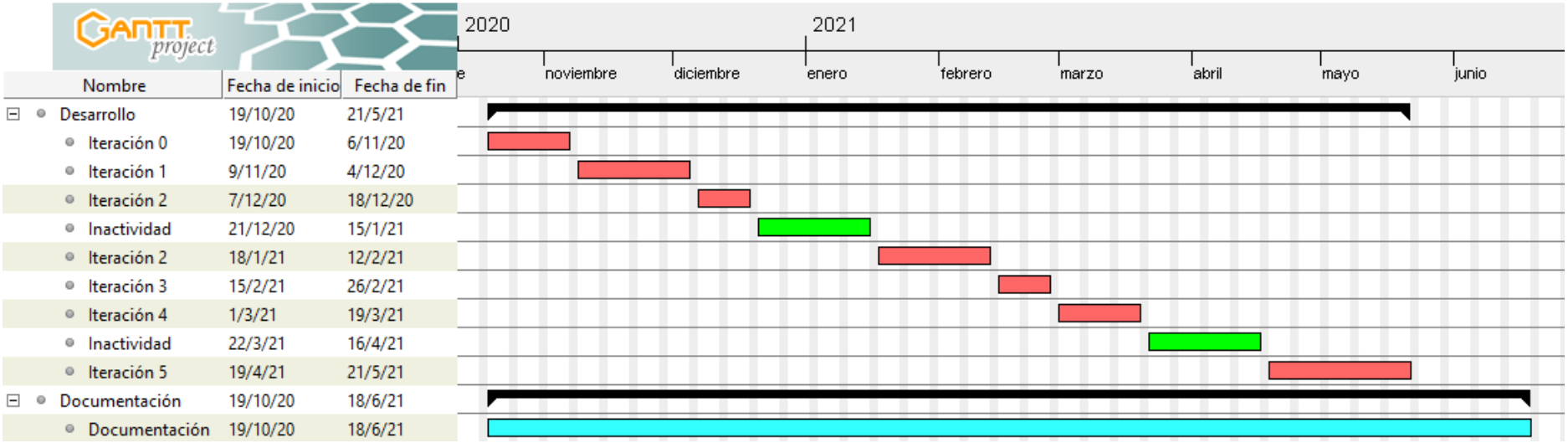
Para la asignación temporal se han descontado las festividades que caen entre semana y periodos de descanso.

- **Fiestas locales**: 25 de noviembre, 11 de junio.
- **Fiestas nacionales y autonómicas**: 2 de noviembre, 7 de noviembre, 8 de noviembre, 1 de marzo.
- **Periodos de descanso**: del 21 de diciembre al 15 de enero, del 22 de marzo al 16 de abril.

Si las estimaciones se cumplen, la asignación final puede verse en la Ilustración 6. Por tanto, la asignación temporal puede ser la siguiente:

- **Iteración 0**: Comienza: 19/10/2020. Finaliza: 06/11/2020
- **Iteración 1**: Comienza: 09/11/2020. Finaliza: 04/12/2020
- **Iteración 2**: Comienza: 07/12/2020. Finaliza: 12/02/2021. Inactividad durante su desarrollo.
- **Iteración 3**: Comienza: 15/02/2021. Finaliza: 25/02/2021
- **Iteración 4**: Comienza: 01/03/2021. Finaliza: 19/03/2021
- **Iteración 5**: Comienza: 19/04/2021. Finaliza: 21/05/2021
- **Documentación**: Durante todo el desarrollo extendiéndose hasta el 18/06/2021.

Con una extensión total de 8 meses.



Nombre	Fecha de inicio	Fecha de fin
Desarrollo	19/10/20	21/5/21
Iteración 0	19/10/20	6/11/20
Iteración 1	9/11/20	4/12/20
Iteración 2	7/12/20	18/12/20
Inactividad	21/12/20	15/1/21
Iteración 2	18/1/21	12/2/21
Iteración 3	15/2/21	26/2/21
Iteración 4	1/3/21	19/3/21
Inactividad	22/3/21	16/4/21
Iteración 5	19/4/21	21/5/21
Documentación	19/10/20	18/6/21
Documentación	19/10/20	18/6/21

Ilustración 6: Diagrama de Gantt completo

1.12 Presupuesto

Este apartado tiene como objetivo calcular los costes resultantes de la ejecución del proyecto, dividiéndose en los siguientes apartados.

- **Costes hardware**: costes asociados a los sistemas informáticos tangibles.
- **Costes software**: costes asociados a las herramientas software y bibliotecas.
- **Costes de personal**: costes asociados a la contratación del personal.
- **Costes indirectos**: costes asociados al alquiler del local, luz, agua, conexión a Internet, sobrecostes, etc.

1.12.1 Costes hardware

En primer lugar, como costes hardware, tenemos el ordenador que se ha utilizado de manera íntegra para el proyecto, desglosado por componentes. Los costes están basados en una factura real (Ilustración 7) y se muestran en la Tabla 8.

Componente	Descripción	Coste estimado
Intel Core i7.7700 3.60 4.20GHz LLGA1151 KABY	Procesador	350.80 €
Gigabyte B250M Gaming 3 1151 DDR4 HDMI M2M	Placa base	111.20 €
Seagate HDD 3.5 1TB 7200rpm 64MB Sata3 Barra	Disco duro de carácter general	47.50 €
Crucial DDR4 16GB 2400MHz Ballistix Sport Lt	Memoria RAM	129.00 €
Nox Semitorre Hummer ZS S.Fuente USB3.0 negr	Carcasa torre	47.40 €
Scythe Refrig. Katana 4 Intel AMD	Ventilación	32.10 €

Asus Turbo GTX 1070 8G 8GB GDDR5 PCIE3.0 HDMI	Tarjeta gráfica	522.00 €
Asus Grabadora 24X Sata Energy DR W24F1ST	Grabadora	14.80 €
Sandisk plus disco SSD 240 GB interno	Disco duro SSD para el SO	86.00 €
Nox FA 650W ATX 12V 2.2 PFC activo 12cm	Fuente de alimentación	49.50 €
Hacer KA240H monitor led24 1920 x 1080 full	Monitor	138.20 €
Pack Mars Gaming 2[15] (no incluido en la factura)	Ratón y teclado	15.95
Montaje y testeo del equipo	Profesional que se encarga del montaje	39.50 €
Total		1.583,95 €

Tabla 8: Estimación de costes hardware para el ordenador de sobremesa

[15] https://es.marsgaming.eu/es/combos_m10

Factura Simplificada nº : 595

Ventas Mostrador

Fecha : 24/09/2017

APP_

www.appinformatica.com

	DESCRIPCIÓN	UNIDADES	PRECIO	TOTAL
(40)	INTEL CORE I7-7700 3,60 4,20GHZ LGA1151 KABY	1	350,80	350,80
(49)	GIGABYTE B250M-GAMING 3 1151 DDR4 HDMI M.2 M	1	111,20	111,20
(40)	SEAGATE HDD 3,5 1TB 7200RPM 64MB SATA3 BARRA	1	47,50	47,50
(49)	CRUCIAL DDR4 16GB 2400MHZ BALLISTIX SPORT LT	1	129,00	129,00
(49)	NOX 9LM - ORNU HUMMER Z8 S.FUENTE U600 0 NEGR	1	47,40	47,40
(40)	SCYTHE REFRIG. KATANA 4 INTEL AMD	1	32,10	32,10
(49)	ASUS TUF B0 GTX1070-8G 8GB GDDR5 PCIE3.0 HDMI	1	522,00	522,00
(50)	ASUS GRABADORA 24X SATA ENERGY DR-W24F1ST	1	14,80	14,80
(1)	MONTAJE Y TESTEO DE EQUIPO	1	39,50	39,50
(55)	SANDISK PLUS DISCO SSD 240 GB INTERNO	1	88,00	88,00
(49)	NOX FA 600W A. X 12V 2.2 PFC ACTIVO 12CM	1	49,50	49,50
(50)	ACER KA240H MONITOR LED 24 1920 X 1080 FULL	1	136,20	136,20

Base Imponible	% IVA	Importe IVA	Total Venta
1.295,87	21	272,13	1.568,00

¡CONSERVE ESTA FACT. SIMPLIFICADA! Será necesaria su presentación a efectos de GARANTÍA del producto

Apúntate al boletín de App: www.appinformatica.com

Mª INES ORTEGA CABEZUDO - PASEO DE LA ESTACION 35, BAJO - Tel: 953 895 625
23008 - JAEN

tu tienda de informática_

Ilustración 7: Factura real de ordenador de sobremesa

Por otro lado, tenemos los costes hardware asociados a los demás dispositivos que se han utilizado durante la ejecución del proyecto (Tabla 9).

Componente	Descripción	Coste estimado
Artec Eva	Escáner para el terreno	13.700 €
EinScan Pro 2X	Escáner para las piezas	5.015 €
Teléfono Xiaomi Mi 9	Dispositivo móvil para la toma de imágenes	449 €
Total		19.164 €

Tabla 9: Estimación de costes hardware para otros dispositivos

No todo el coste del hardware está asociado al proyecto ya que debe amortizarse para el tiempo que ha durado.

Por su sencillez se ha decidido aplicar una amortización lineal. Aplicando este método, el gasto por amortización anual es el mismo y se calcula asignándole un porcentaje anual fijo al precio de adquisición o asignándole una vida útil.

En este caso se va a utilizar el cálculo por porcentaje. Estos porcentajes vienen definidos por la agencia tributaria[16]. Y la fórmula es la siguiente:

$$Amortización\ anual = valor\ de\ adquisición * coeficiente\ lineal$$

A continuación, en la Tabla 10, se calcula la amortización para el hardware desglosado en dispositivos.

Activo	Valor de adquisición	Coeficiente lineal	Amortización anual
Ordenador	1.583,95 €	20 %	316,79 €
Escáneres	18.715 €	20 %	3.743 €
Teléfono móvil	449 €	20 %	89,8 €
Total			4.149,59 €

Tabla 10: Amortización del hardware

16 https://www.agenciatributaria.es/AEAT.internet/Inicio/_Segmentos_/Empresas_y_profesionales/Empresas/Impuesto_sobre_Sociedades/Periodos_impositivos_a_partir_de_1_1_2019/Base_imponible/Amortizacion/Tabla_de_coeficientes_de_amortizacion_lineal_.shtml

La amortización total debe calcularse atendiendo a los 8 meses de su duración. El cálculo es el siguiente:

$$Amortización\ hardware = \frac{4.149{,}59\ €}{12\ meses}\ x\ 8\ meses = 2.766{,}39\ €$$

1.12.2 Costes software

En los costes software se incluyen todas aquellas herramientas que han posibilitado la ejecución del proyecto (Tabla 11).

Todos los precios se han consultado a junio de 2021.

Componente	Tipo de pago	Coste estimado
XAMPP	-	0 €
Visual Studio Code	-	0 €
PostgreSQL	-	0 €
PostGIS	-	0 €
Unity	-	0 €
Visual Studio 2019 Community	-	0 €
Assets de Unity	Pago único	116.14 €
CloudCompare	-	0 €
Blender	-	0 €
SCANN3D	Versión de prueba	0 €
3D Live Scanner	-	0 €
GIMP	-	0 €
Microsoft Office Personal	Mensual	7 € x 8 = 56 €
Visual Paradigm	Licencia de estudiante	0 €
Gantt Project	-	0 €
Windows 10 Home	Pago único	145 €
Total		317,14 €

Tabla 11: Estimación de costes para el software

Al igual que para los costes hardware, los costes software deben amortizarse. También se utiliza una amortización lineal basada en porcentajes. En la Tabla 12.

Activo	Valor de adquisición	Coeficiente lineal	Amortización anual
Software	317,14 €	33 %	104,66 €
Total			104,66 €

Tabla 12: Amortización del software

El cálculo de la amortización también debe calcularse para los 8 meses de duración del proyecto.

$$Amortización\ software = \frac{104{,}66\ €}{12\ meses}\ x\ 8\ meses = 69{,}77\ €$$

1.12.3 Costes de personal

Costes derivados de la contratación del personal. En este caso, al tratarse de un TFM, el personal está formado únicamente por una persona (autor del proyecto). El rol seleccionado es el de **Analista Programador**, para el cual se recupera el salario en vigor según el **XVII Convenio colectivo estatal de empresas de consultoría y estudios de mercado y de la opinión pública**[17] publicado en el BOE en el año 2018, aunque la información está actualiza a diciembre de 2019.

Los cálculos se expresan en la Tabla 13.

Componente	Salario anual	Salario mensual	Coste estimado
Analista programador	24.640,37 €	2.053,36 €	2.053,36 x 8 = 16.426,88 €

Tabla 13: Estimación de costes para el personal

1.12.4 Costes indirectos

En estos costes se reflejan los costes derivados del proyecto (Tabla 14). Dada la dificultad para calcular algunos de los costes, se utilizará un porcentaje del coste total acumulado hasta ahora que es de 19.263,04 €

[17] https://www.boe.es/boe/dias/2018/03/06/pdfs/BOE-A-2018-3156.pdf

Componente	Descripción	Coste estimado
1Gbps Fibra Óptica y llamadas ilimitadas (Vodafone)[18]	Acceso a Internet	40.99 € x 8 = 327,92 €
Sobrecostes	Para cubrir agua, luz, alquiler, etc.	10% x 19.263,04 € = 1.926,30 €
Total		2.254,22 €

Tabla 14: Estimación de costes para gastos indirectos

1.12.5 Coste total

Por último, en la Tabla 15 unificamos los costes descritos en apartados anteriores en una sola tabla.

Coste	Coste estimado
Costes hardware	2.766,39 €
Costes software	69,77 €
Costes de personal	16.426,88 €
Costes indirectos	2.254,22 €
Total	21.517,26 €

Tabla 15: Costes totales

1.13 Normas y referencias

A continuación, se presentan las normas, reglamentos y referencias de cualquier tipo que han sido de aplicación en la elaboración del trabajo y/o en la puesta en práctica del mismo.

[18] https://www.vodafone.es/c/particulares/es/productos-y-servicios/fibra-optica-adsl/

1.13.1 Métodos, herramientas, modelos, métricas y prototipos

En este apartado se contemplan todos aquellos métodos, prototipos, métricas, programas, modelos y herramientas que se han empleado para obtener las estimaciones anteriores: costes, tiempos, etc.

1.13.1.1 Presupuesto

La elaboración del presupuesto se ha basado únicamente en observaciones de las fuentes antes referenciadas. Para el cálculo de la amortización durante el tiempo de desarrollo del proyecto se han utilizado los períodos de años y coeficientes aportados por la Agencia Tributaria.

1.13.2 Mecanismos de control de calidad

El aseguramiento de la calidad en la redacción del trabajo implica la verificación de la completitud (falta de omisiones), la integridad de la documentación, así como una redacción clara, concisa y entendible por todos los participantes e interesados en dicho trabajo.

Al estar el trabajo desarrollado por una sola persona y verificado por los tutores, no es necesario llevar un documento de control de edición y revisión de la documentación. De esta forma, se han utilizado mecanismos básicos para la verificación de la integridad y completitud, que incluyen un control de versiones a nivel de documento y un control sencillo de la trazabilidad de especificaciones y requisitos.

2 DISEÑO INICIAL

En este apartado se va a describir el estado inicial del proyecto antes de la primera iteración. Posteriormente, en cada una de las iteraciones descritas en el apartado 3 (Desarrollo) se describirá el diseño propuesto con más detalle en caso de sufrir algún cambio.

2.1 Análisis y diseño del sistema

Partimos de un proyecto ya desarrollado al que vamos a añadirle funcionalidad extra, por lo que podemos considerar el diseño inicial de este proyecto como el diseño final del proyecto desarrollado anteriormente. Las futuras modificaciones de análisis y diseño se describirán en apartados posteriores.

A continuación, se va a detallar el diseño para cada uno de los diferentes elementos que componen el proyecto, desglosado en subsecciones.

2.1.1 Arquitectura inicial del sistema

Lo primero a describir es la arquitectura del sistema (ver Ilustración 8). La arquitectura nos ayudará a obtener una visión global, permitiéndonos visualizar todas las conexiones existentes entre cada uno de los elementos principales. Además, nos permite ver los datos con los que se trabaja en cada fase y el paso de información entre ellas.

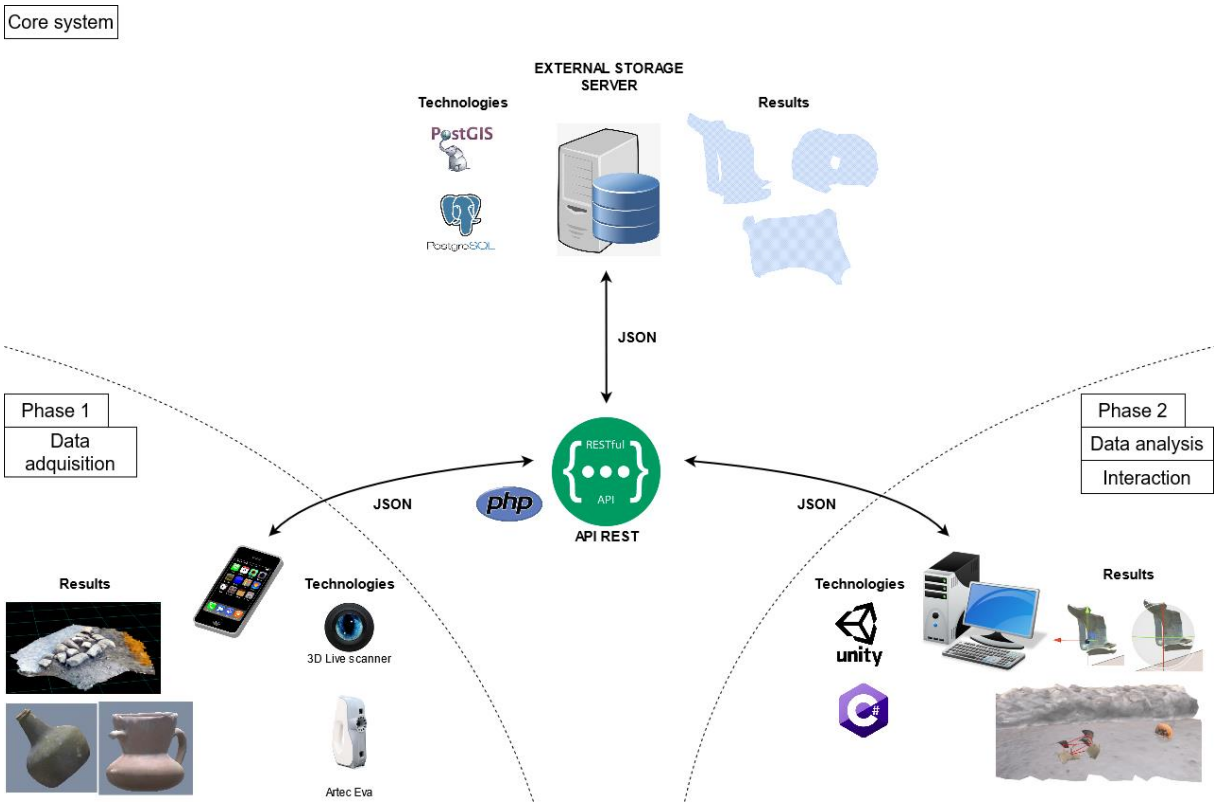


Ilustración 8: Arquitectura del sistema

El sistema se divide en cuatro elementos principales: API REST, base de datos, herramienta gráfica para la visualización de los mismos y herramientas de adquisición de datos.

La idea es obtener los modelos 3D a partir de aplicaciones de escáner o fotogrametría de distintos dispositivos móviles (etapa 1), almacenar la geometría en la base de datos para la realización de operaciones sobre esta y, por último, permitir la interacción y análisis mediante la herramienta gráfica en 3D, y si se dispone de los dispositivos adecuados, en realidad virtual (etapa 2).

Toda la información necesaria para conectar las distintas aplicaciones con la base de datos a través de la API se realiza con información codificada en formato JSON (JavaScript Object Notation).

A continuación, se explica con más detalle cada una de las partes en las que se divide la arquitectura del sistema y el software elegido para cada una de ellas.

2.1.1.1 Sistema gestor de base de datos

Actualmente se cuenta con una base de datos desplegada en un servidor PostgreSQL con la extensión PostGIS. Gracias a esta extensión podemos almacenar

datos geométricos y realizar un gran conjunto de operaciones tanto en 2D como en 3D. En definitiva, tenemos un software bastante potente para trabajar con bases de datos espaciales.

Algunas de las características de PostgreSQL y PostGIS (Ferrari & Prizozzi, 2020; Obe & Hsu, 2015) son:

- Es un software libre y de código abierto.
- Alta concurrencia y soporte para sistemas distribuidos.
- Compatibilidad con los estándares de OGC (Open Geospatial Consortium).
- Ofrece datos espaciales tanto en 2D como en 3D y una gran cantidad de operaciones geométricas con estos datos.
- Gran número de clientes SIG de escritorio para visualizar los datos.

2.1.1.2 Herramientas móviles de escaneado 3D

En este proyecto se utilizan las herramientas para obtener los modelos 3D tanto del terreno en el que los arqueólogos se encuentran trabajando, como los restos arqueológicos que se desentierren. Todo esto con la idea de obtener las geometrías que se almacenarán en la base de datos y se visualizarán en la herramienta gráfica.

No está dentro de nuestro alcance el diseño y desarrollo de estas aplicaciones, por lo que el objetivo es el estudio de varias aplicaciones y escáneres para decidir cuál es la que mejor se adapta a la metodología de trabajo de los especialistas. El estudio llevado a cabo y los resultados obtenidos están explicados con mayor detalle en el apartado 3.6 (Iteración 5).

2.1.1.3 Herramienta gráfica para Realidad Virtual

La herramienta de realidad virtual ha sido programada con el motor de videojuegos Unity (Okita, 2014). El lenguaje de programación utilizado por Unity es C#.

Algunas de las características de este motor de videojuegos son:

- Renderizado 2D y 3D.
- Integración con VR y AR.
- Bastante completo en cuanto a funciones: animaciones, sonido, físicas, inteligencia artificial, entre otras.

- Multisistema y multiplataforma independientemente de su sistema operativo: ordenadores de escritorio (Windows, Mac), dispositivos móviles (Android), consolas y web.

El objetivo es ofrecer a un usuario experto, aunque no necesariamente un arqueólogo, una visualización en realidad virtual no inmersiva del estado del yacimiento arqueológico en el momento de su excavación, con la posibilidad de que este realice los análisis que estime oportunos.

2.1.1.4 Comunicación y paso de información

La comunicación entre todas las partes anteriores se realiza mediante el uso de una API REST. La API está diseñada y escrita en el lenguaje de programación PHP puesto que es un lenguaje bastante popular para el desarrollo web.

El paso de información se codifica en formato de texto JSON, siendo un formato de texto bastante extendido y conocido y que puede integrarse con las demás partes de la arquitectura sin ningún tipo de problema.

2.1.2 Diseño inicial de la base de datos

A continuación, se va a explicar cada uno de los subsistemas en los que se divide la base de datos, para finalmente mostrar el diseño completo.

Todas las tablas que se utilizan tienen como punto de partida las tablas utilizadas por Cástulo en su base de datos, añadiendo y modificando aquellas que eran necesarias.

Aunque, el diseño está completo, algunos subsistemas no se utilizaron el en proyecto anterior y tampoco se utilizarán en el desarrollo de este, por lo que puede dejarse para trabajo futuro. Estos son, por ejemplo, el sistema de datación y el sistema de gestión de usuarios.

Entendiendo como subsistema un conjunto de tablas que almacenan información relacionada, los diferentes subsistemas son:

1. **Conceptos principales**: tablas básicas utilizadas para almacenar la información de las áreas, volúmenes, capas y restos arqueológicos de un yacimiento.
2. **Coordenadas**: sistema de coordenadas para indicar la posición de cada resto arqueológico.

3. **Registro**: sistema de registro de cada uno de los conceptos principales en la base de datos.
4. **Materiales**: diferentes materiales de los restos arqueológicos.
5. **Topología**: relaciones topológicas entre restos arqueológicos de una misma capa.
6. **Imágenes**: sistema de imágenes relacionadas con las capas y los restos arqueológicos.
7. **Datación**: sistema de datación de los restos arqueológicos en un periodo y/o época determinados
8. **Usuarios**: sistema de registro de usuarios.

Es necesario hacer especial hincapié en el **sistema topológico**. Su función es conectar los elementos que podrían estar relacionados en base a su localización espacial. Cada elemento puede estar asociado con varios de ellos.

Para rellenar las tablas se va a hacer uso del sistema de intersección DE-9IM (Dimensionally Extended 9-Intersection Model) para 2D (Clementini et al., 1993). Este es un modelo topológico que describe las relaciones espaciales entre dos geometrías en 2D. Las relaciones espaciales expresadas en el modelo son invariables a las transformaciones de rotación, traslación y escalado de la geometría que se compara.

El resultado es una matriz de 3 x 3 en el que se computa la dimensión de la intersección entre el interior, borde y exterior de cada geometría (Strobl, 2016). Las dimensiones que puede dar son:

- 0: la intersección es un punto.
- 1: la intersección es una línea.
- 2: la intersección es un área.
- F: no existe intersección.

A modo de ejemplo, dadas dos geometrías (a y b), la matriz resultante es la que viene dada en la Ilustración 9.

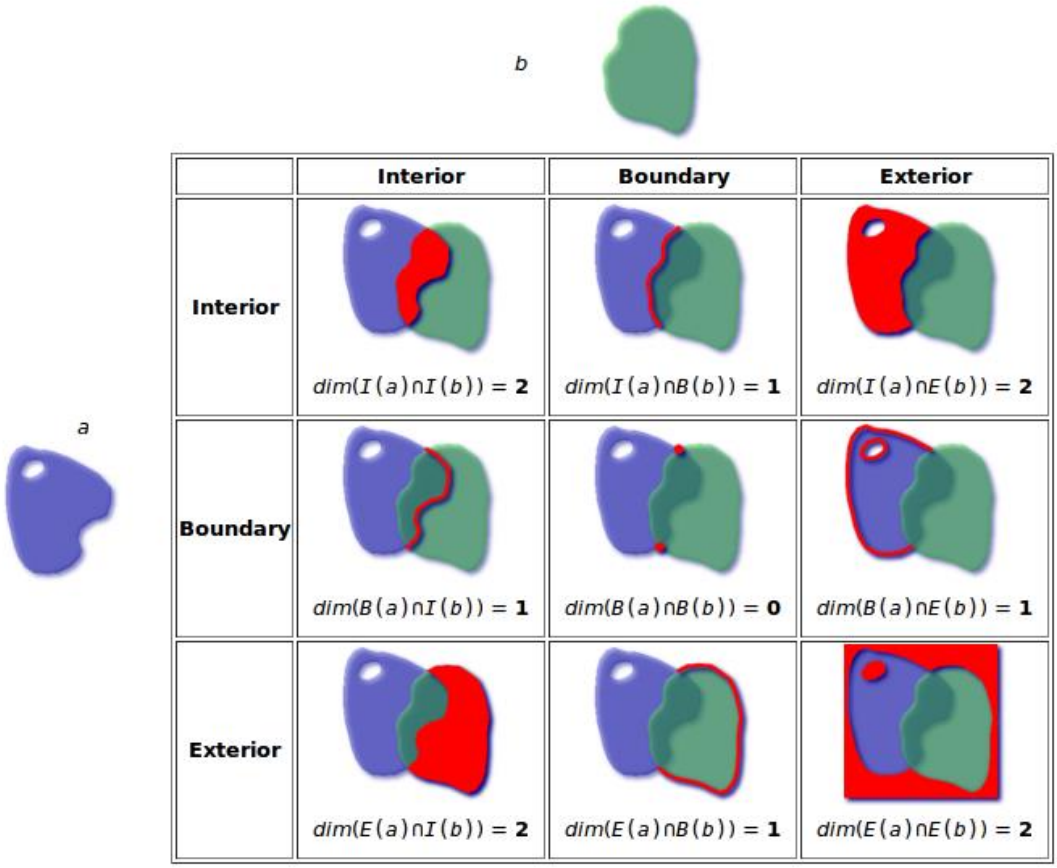


Ilustración 9: Matriz que proporciona el modelo DE-9IM
(https://geotalleres.readthedocs.io/es/latest/postgis-relaciones-espaciales/relaciones_espaciales.html)

Todo esto es aplicable a 2D, pero es necesario extrapolar la matriz a 3D. Por este motivo existe el método DE-25IM que añade una nueva dimensión denominada "cuerpo" (polígono en 3D) y dando como resultado una matriz de 5x5 (Zhou & Guan, 2019).

En nuestro caso, el método DE-9IM en 3D consiste en aplicar el algoritmo 2D en cada una de las proyecciones de la geometría a comparar en cada uno de los planos principales del espacio 3D (XY, XZ o YZ).

2.1.2.1 Conceptos principales

Estas tablas almacenan la información de los conceptos fundamentales de un yacimiento arqueológico: áreas, volúmenes, capas y elementos. Las tablas resultantes se describen a continuación, y su diseño puede verse en la Ilustración 10.

- Tabla **Area**: encargada de almacenar información de un área de excavación. Sus campos son:

  - **id** (entero): identificador único del área y clave primaria de la tabla.
  - **archaeological_site_id** (entero): clave foránea que identifica al yacimiento arqueológico al que pertenece. Un área solo puede estar contenida en un yacimiento.
  - **entry_info_id** (entero): clave foránea que identifica el registro temporal asociado.
  - **name** (cadena de caracteres): nombre del área.
  - **description** (cadena de caracteres): breve descripción.
  - **corners** (geometría): geometría 2D que forma el polígono cerrado que define al área. Cada punto está dado en coordenadas UTM (Universal Transverse Mercator).
  - **is_active** (valor booleano): Indica si el área está activa o no.

- Tabla **Volume**: encargada de almacenar la información de un volumen de excavación. Sus campos son:
  - **id** (entero): identificador único del volumen y clave primaria de la tabla.
  - **area_id** (entero): clave foránea que identifica al área donde se encuentra incluido el volumen. Un volumen solo puede estar incluido en un área.
  - **entry_info_id** (entero): clave foránea que identifica el registro temporal asociado.
  - **corners** (geometría): geometría 2D que forma el polígono cerrado que define al volumen. Cada punto está dado en coordenadas UTM.

- Tabla **Layer**: encargada de almacenar la información de una capa de excavación. Sus campos son:
  - **id** (entero): identificador único de la capa y clave primara de la tabla.

  - **volume_id** (entero): clave foránea que identifica al volumen donde se encuentra incluida la capa. Una capa solo puede estar incluido en un volumen.
  - **entry_info_id** (entero): clave foránea que identifica el registro temporal asociado.
  - **description** (cadena de caracteres): breve descripción.
  - **z_start** (vector de números reales): coordenadas Z que indican la altura superior de la capa y representa la altura de un terreno. Hay tantos valores como puntos tenga el polígono.
  - **z_end** (vector de números reales): igual que el campo anterior, pero representando la altura inferior de la capa.

- Tabla **Element**: encargada de almacenar la información de un resto arqueológico como, por ejemplo: piezas de cerámica, huesos humanos o cualquier otro hallazgo que se considere importante. Sus campos son:
  - **id** (entero): identificador único del elemento y clave primaria de la tabla.
  - **layer_id** (entero): clave foránea que identifica la capa en la que fue desenterrado el elemento. Un elemento solo puede estar contenido en una capa.
  - **material_id** (entero): clave foránea que identifica el material del elemento. Un elemento solo puede tener asociado un material.
  - **entry_info_id** (entero): clave foránea que identifica el registro temporal asociado.
  - **weight** (número real): peso en kilogramos del elemento.
  - **qr_image** (cadena de caracteres): URL (Uniform Resource Locator) de la imagen que representa un código QR (Quick Response) que identifica al elemento.
  - **original_mesh** (cadena de caracteres): URL de la malla de triángulos correspondiente al modelo 3D de un elemento.

- **simplified_mesh** (cadena de caracteres): URL de la malla de triángulos simplificada correspondiente al modelo 3D de un elemento.
- **original_contour** (geometría): polígono cerrado en 2D que representa el contorno del elemento tal y como se encontró en el yacimiento.
- **convex_hull** (geometría): geometría en 2D que representa la envolvente convexa del contorno original.
- **xy_projection** (geometría): polígono cerrado en 2D que representa la silueta del objeto en 3D proyectada en los planos XY.
- **xz_projection** (geometría): igual que el campo anterior, pero sobre los planos XZ.
- **yz_projection** (geometría): igual que el campo anterior, pero sobre los planos YZ.
- **elements_related** (vector de enteros): conjunto de números enteros representando los identificadores de los elementos con los que se relaciona topológicamente el elemento.
- **others** (json): fichero en formato JSON para guardar información adicional.

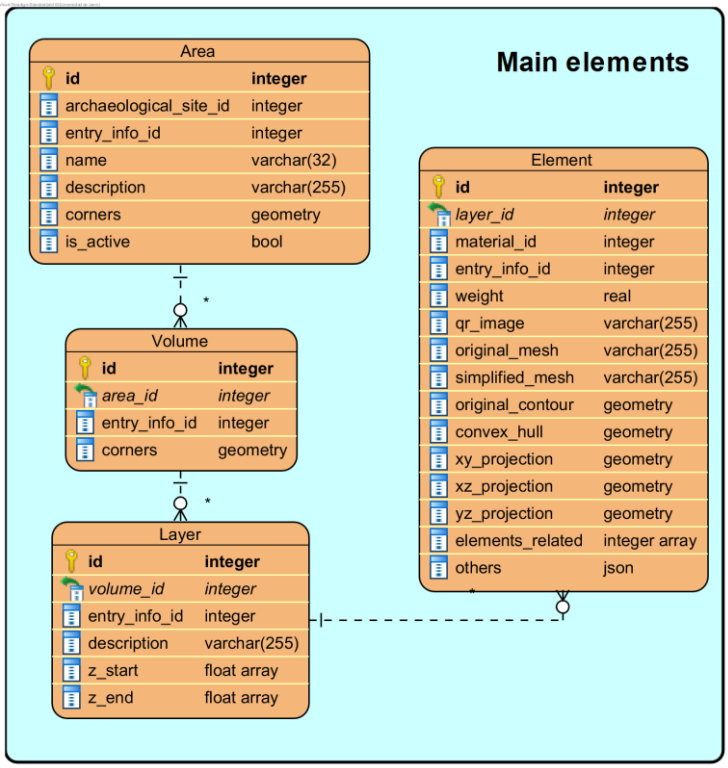


Ilustración 10: Conceptos principales

2.1.2.2 Coordenadas

Sistema compuesto por las tablas necesarias para almacenar las coordenadas que indican la posición exacta del resto arqueológico descubierto (ver Ilustración 11).

Actualmente la base de datos puede almacenar dos sistemas de posicionamiento: GPS (*Global Positioning System*) y el sistema de coordenadas UTM, siendo este último el utilizado en el proyecto.

Las tablas propuestas utilizan una característica de PostgreSQL, similar a la herencia utilizado en los lenguajes de programación, que permite a una tabla heredar las columnas de la tabla padre y, además, tener sus propias columnas (PostgreSQL, 2021).

Por tanto, la tabla padre, y sus campos, es:

- Tabla **Coordinates**: encargada de almacenar la información común para los dos sistemas de posicionamiento.
    - **id** (entero): identificador único del conjunto de coordenadas y clave primaria de la tabla.

- **element_id** (entero): clave foránea que identifica al elemento arqueológico que se está posicionando. Un conjunto de coordenadas solo puede pertenecer a un elemento.

Mientras que las tablas hijas son:

- Tabla **CoordinatesGPS**: encargada de almacenar las coordenadas necesarias para su posicionamiento mediante el formato WGS84 (World Geodetic System 1984)
  - **latitude** (número real): coordenada angular que posiciona al elemento en la dirección norte-sur
  - **longtitude** (número real): coordenada angular que posiciona al elemento en la dirección este-oeste.
  - **altitude** (número real): distancia vertical del elemento respecto al nivel del mar.
- Tabla **CoordinatesUTM**: encargada de almacenar las coordenadas UTM necesarias para su posicionamiento.
  - **x** (número real): coordenada X para posicionar al elemento dada en metros.
  - **y** (número real): coordenada Y para posicionar al elemento dada en metros.
  - **height** (número real): indica la altura en la que se encontró el elemento.
  - **quadrant** (cadena de caracteres): indica el cuadrante al que pertenece el elemento teniendo en cuenta la división en husos y zonas UTM.

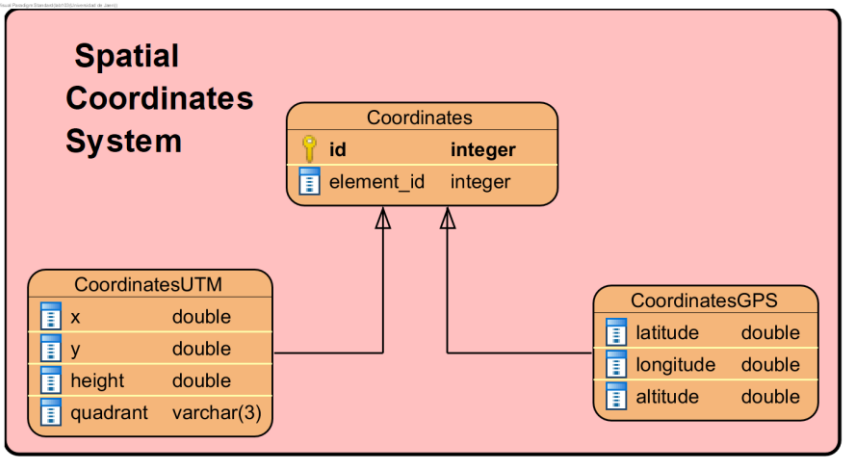


Ilustración 11: Sistema de coordenadas

2.1.2.3 Registro

Estas tablas almacenan la información relacionada con el registro del yacimiento arqueológico y el registro de cada uno de los conceptos principales en la base de datos. Su diseño puede verse en la Ilustración 12 y las tablas resultantes se describen a continuación:

- Tabla **ArchaeologicalSite**: encargada de almacenar la información de los yacimientos arqueológicos. Sus campos son:
  - **id** (entero): identificador único de un yacimiento y clave primaria de la tabla.
  - **entry_info_id** (entero): clave foránea que identifica el registro temporal asociado.
  - **name** (cadena de caracteres): nombre del yacimiento arqueológico.
  - **description** (cadena de caracteres): breve descripción del yacimiento arqueológico.
  - **municipality** (cadena de caracteres): municipio en la que se encuentra situado el yacimiento arqueológico.
- Tabla **Campaign**: encargada de almacenar la información de una campaña, que no es más que un rango de tiempo en el que se realizan excavaciones. Sus campos son:

- **id** (entero): identificador único de una campaña y clave primaria de la tabla.
- **name** (cadena de caracteres): nombre de la campaña.
- **start_date** (fecha): fecha de inicio de la campaña.
- **end_date** (fecha): fecha de fin de la campaña.

- Tabla **EntryInfo**: encargada de almacenar la fecha en la que algún concepto principal fue introducido en la base de datos. Sus campos son:
  - **id** (entero): identificador único de un registro y clave primaria de la tabla.
  - **campaign_id** (entero): clave foránea que identifica la campaña a la que pertenece el registro.
  - **date** (fecha): fecha en la que se introdujo la información del concepto en la base de datos.

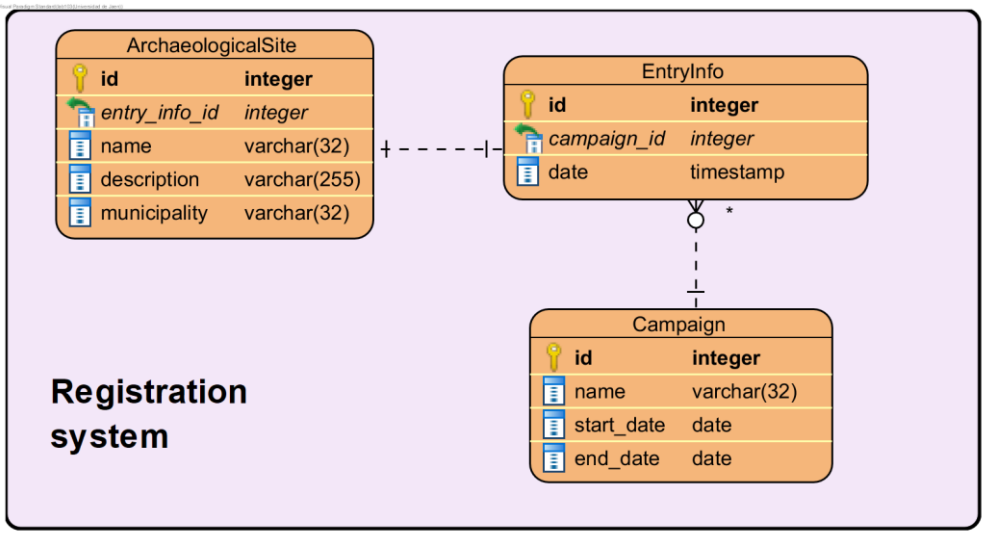


Ilustración 12: Sistema de registro

2.1.2.4 Materiales

El sistema de registro de los materiales está compuesto por una única tabla (ver Ilustración 13), la cual está formada por los siguientes campos:

- Tabla **Material**: encargada de almacenar la información de los materiales.

- **id** (entero): identificador único de un material y clave primaria de la tabla.
- **type_material** (cadena de caracteres): nombre del material. Aunque no es una clave primaria, no permite valores repetidos.
- **description** (cadena de caracteres): breve descripción del material en cuestión.

Esta tabla está pensada únicamente para restos arqueológicos, pero podría extenderse a cada capa siempre que se tenga información del material de cada estrato.

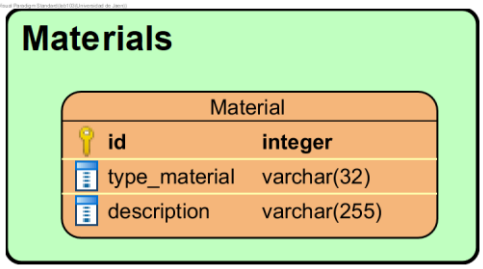


Ilustración 13: Materiales

2.1.2.5 Topología

El sistema topológico pretende conectar elementos que podrían estar relacionados en base a su localización espacial. Cada elemento puede estar relacionado con varios de ellos.

El sistema está formado por una única tabla (ver Ilustración 14) cuyos campos son:

- Tabla **EE_9IM**: encargada de almacenar las matrices que determinan la posible conexión entre dos restos arqueológicos.
  - **elementA_id** (entero): clave foránea que identifica un elemento y a su vez es parte de la clave primaria de la tabla.
  - **elementB_id** (entero): es la clave foránea que identifica un elemento (diferente que el identificado por el atributo anterior) y a su vez es parte de la clave primaria de la tabla.
  - **xy_matrix** (cadena de caracteres): matriz 9IM en el plano XY.

- **xz_matrix** (cadena de caracteres): matriz 9IM en el plano XZ.
- **yz_matrix** (cadena de caracteres): matriz 9IM en el plano YZ.

La idea es que cada vez que se inserta un nuevo elemento arqueológico en el sistema, se comprueba la relación con el resto de elementos en la misma capa y se insertan las tuplas en esta tabla. Además, en caso de estar relacionados, deben modificarse los campos correspondientes en la tabla **Element.**

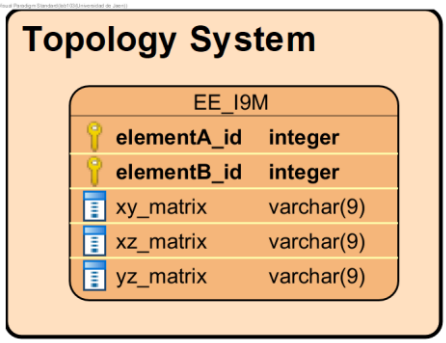


Ilustración 14: Sistema topológico

2.1.2.6 Imágenes

Para los arqueólogos es bastante importante replicar el estado del yacimiento a partir de imágenes y, por tanto, se ofrece la posibilidad de almacenarlas en la base de datos.

Este sistema está compuesto por una tabla de la cual heredan otras dos. Una de ellas se centra en el almacenamiento de imágenes de las distintas capas con la información adicional que esto conlleva, mientras que la segunda se centra en el almacenamiento de las imágenes de los elementos arqueológicos y su información correspondiente. Su diseño se presenta en la Ilustración 15.

La tabla padre, junto a sus atributos, es:

- Tabla **Image**: encargada de almacenar la información común para todas las imágenes disponibles en la base de datos.
  - **id** (entero): identificador único de la imagen y clave primaria de la tabla.
  - **path_image** (cadena de caracteres): URL de la imagen.

Las tablas derivadas y sus campos son:

- Tabla **Image_Layer:** encargada de almacenar la información sobre qué capa pertenece la imagen.
    - Campos heredados de la tabla padre.
    - **layer_id** (entero): clave foránea que identifica la capa a la que pertenece la imagen. Una imagen únicamente puede pertenecer a una capa.
- Tabla **Image_Element:** encargada de almacenar la información sobre a qué elemento pertenece la imagen.
    - Campos heredados de la tabla padre.
    - **element_id** (entero): clave foránea que identifica al resto arqueológico al que pertenece la imagen. Una imagen únicamente puede pertenecer a un elemento.

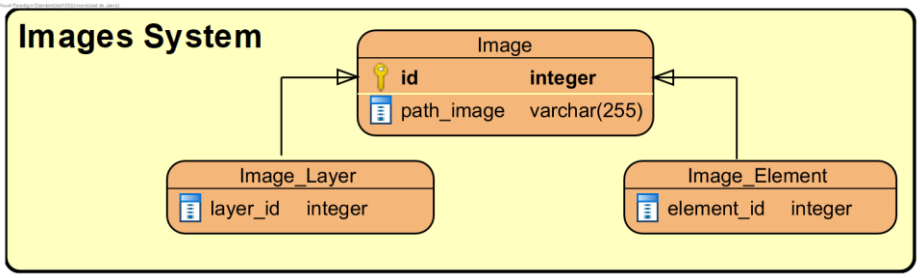


Ilustración 15: Sistema de imágenes

2.1.2.7 Datación

El sistema de datación se utiliza para el estudio posterior por parte de los arqueólogos con el objetivo de determinar la época a la que pertenece cada resto arqueológico encontrado en la campaña de excavación.

La datación puede estar basada en eras cuando no se puede dar una fecha concreta como, por ejemplo, que una pieza pertenezca al siglo 3 o pertenezca a la época Ibera Tardía. Por otro lado, la datación también puede estar basada en fechas más o menos concretas.

Al igual que en el apartado anterior, las tablas utilizan herencia para almacenar información de las eras o de las fechas que se pueden utilizar. Su diseño puede verse en la Ilustración 16.

La tabla padre, junto a sus atributos, es:

- Tabla **Dating**: encargada de almacenar la información común para el sistema de datación.
    - **id** (entero): identificador único de la datación y clave primaria de la tabla.
    - **element_id** (entero): clave foránea que identifica al elemento que ha sido datado. Cada elemento puede datarse más de una vez en caso de establecer un intervalo de tiempo.

Mientras que las tablas derivadas son:

- Tabla **EraDate**: tabla específica para las eras.
    - Campos heredados de la tabla padre.
    - **era_id** (entero): clave foránea que identifica a la era en cuestión.
- Tabla **CalendarDate**: tabla específica para las fechas.
    - Campos heredados de la tabla padre.
    - **calendar_id** (entero): clave foránea que identifica al calendario al que está asociado.
    - **day** (entero): día de la datación.
    - **month** (entero): mes de la datación.
    - **year** (entero): año de la datación.

Puesto que el sistema permite que el usuario pueda añadir nuevas eras y fechas, así como utilizar las ya existentes, existen dos tablas más que permiten realizar este propósito.

- Tabla **Era**: encargada de almacenar la información de cada una de las eras introducidas por el usuario.
    - **id** (entero): identificador único de la era y clave primaria de la tabla.
    - **name** (cadena de caracteres): nombre descriptivo de la era.
    - **start_day** (entero): día en el que se establece el inicio de la era.
    - **start_month** (entero): mes en el que se establece el inicio de la era.

- **start_year** (entero): año en el que se establece el inicio de la era.
- **end_day** (entero): día en el que se establece el fin de la era.
- **end_month** (entero): mes en el que se establece el fin de la era.
- **end_year** (entero): año en el que se establece el fin de la era.

- Tabla **Calendar**: encargada de almacenar la información de cada calendario que puede utilizarse en el sistema.
  - **id** (entero): identificador único del intervalo y clave primaria de la tabla.
  - **name** (cadena de caracteres): nombre descriptivo del intervalo.
  - **start_day** (entero): día en el que se establece el inicio del calendario.
  - **start_month** (entero): mes en el que se establece el inicio del calendario.
  - **start_year** (entero): año en el que se establece el inicio del calendario.

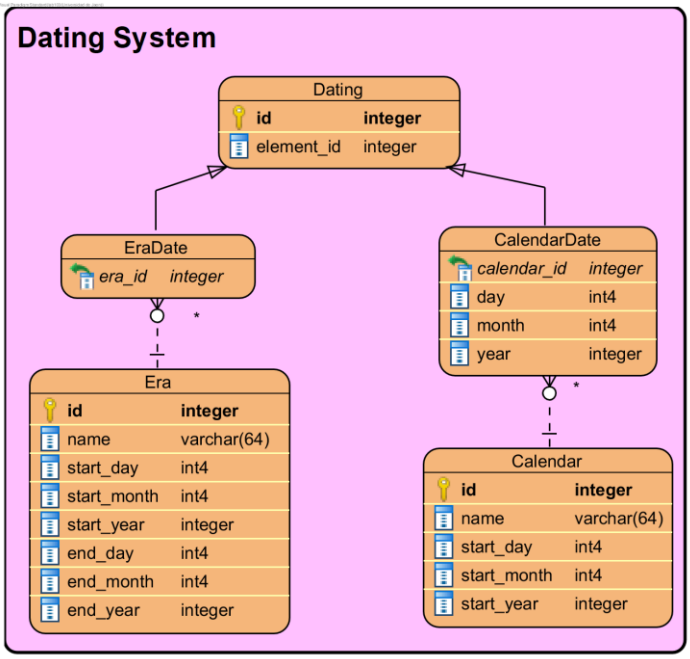


Ilustración 16: Sistema de datación

2.1.2.8 Usuarios

Por último, se describen ahora las tablas que forman parte del sistema de registro de usuarios (ver Ilustración 17). Además de guardar la propia información de los usuarios, se pretende almacenar el registro de comentarios que estos realizan sobre los elementos arqueológicos descubiertos. Por tanto, las tablas son:

- Tabla **User**: encargada de almacenar la información relativa a los usuarios.
    - **id** (entero): identificador único del usuario y clave primaria de la tabla.
    - **alias** (cadena de caracteres): nombre reducido que representa a cada usuario.
    - **email** (cadena de caracteres): correo electrónico del usuario.
    - **full_name** (cadena de caracteres): nombre completo del usuario.
    - **password** (cadena de caracteres): contraseña encriptada.
    - **read_permission** (booleano): indica si el usuario tiene permiso de lectura en la base de datos.
    - **write_permission** (booleano): indica si el usuario tiene permiso de escritura en la base de datos.
    - **delete_permission** (booleano): indica si el usuario tiene permiso de eliminación en la base de datos.
- Tabla **Comments**: encargada de almacenar la información relativa a los comentarios realizados para expresar aclaraciones, dudas u opiniones sobre los elementos.
    - **id** (entero): identificador único del comentario y clave primaria de la tabla.
    - **element_id** (entero): clave foránea que identifica al elemento sobre el que se ha realizado el comentario. Un elemento puede recibir muchos comentarios.

- **user_id** (entero): clave foránea que identifica al usuario que ha realizado el comentario. Un usuario puede realizar varios comentarios.
- **text** (cadena de caracteres): comentario del usuario.

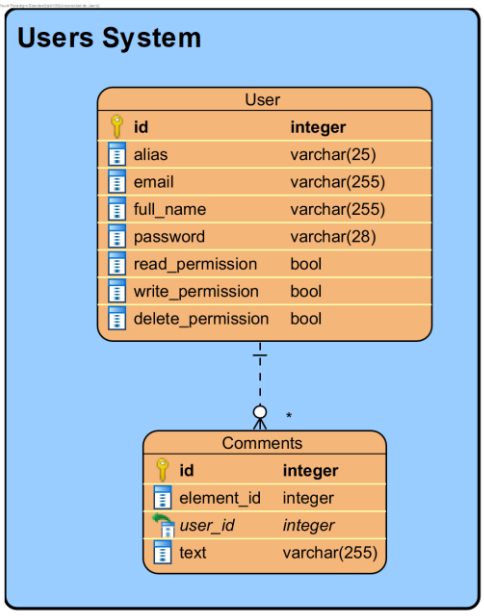


Ilustración 17: Sistema de usuarios

Por último, se muestra en diagrama completo en la Ilustración 18.

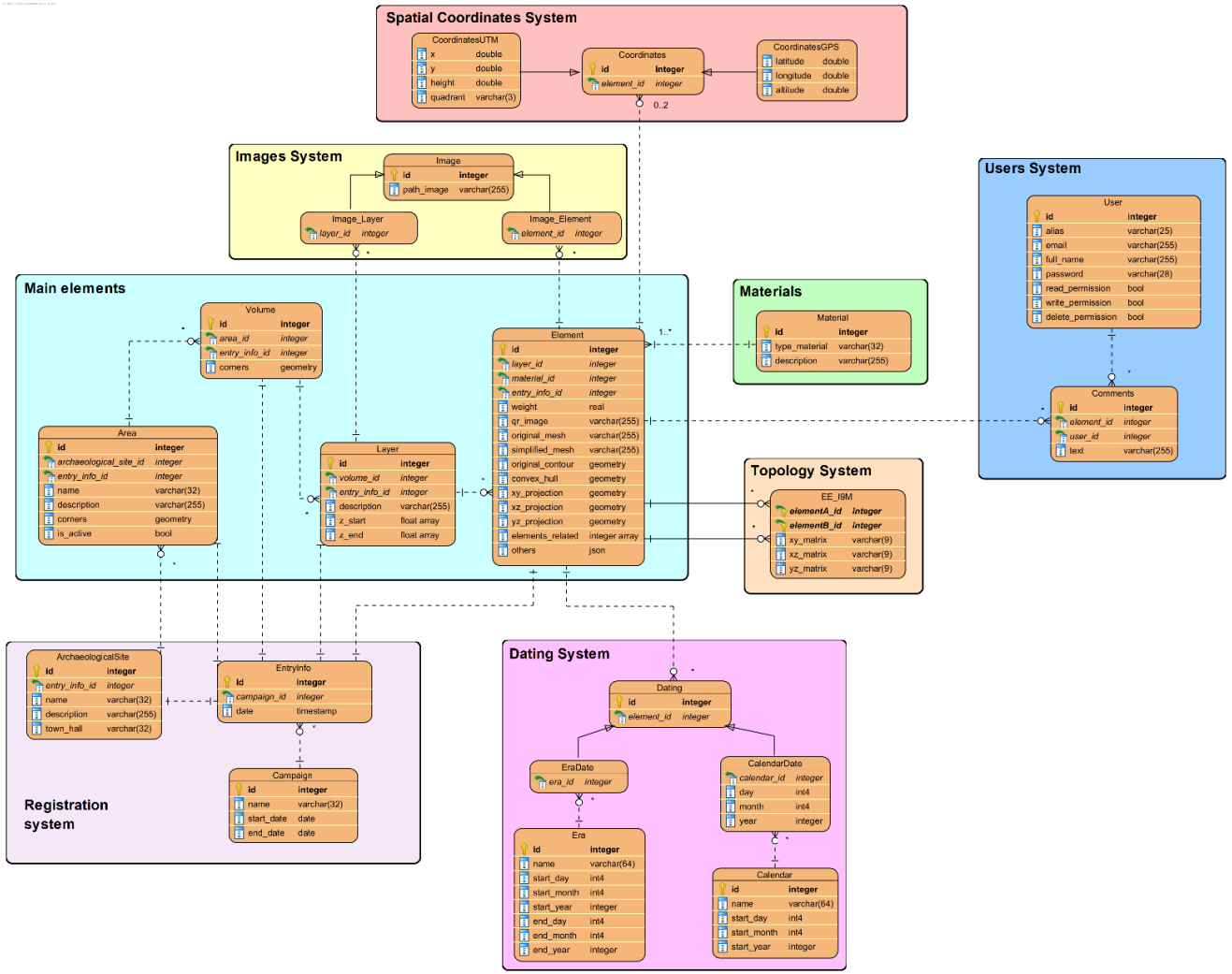


Ilustración 18: Diseño completo de la base de datos

2.1.3 Diseño inicial de la API REST

La API REST está diseñada para dar cobertura a dos tipos de comunicaciones: comunicación con la base de datos y comunicación con la aplicación. Ambas comunicaciones son bidireccionales y por tanto la API permite mandar y recibir información.

El diseño se ha dividido en cuatro partes, una por cada concepto principal de la base de datos ya que, en este caso, la API solo ofrece servicios para los conceptos principales.

Además, existen clases auxiliares que utilizan estos servicios y que se detallarán más adelante.

2.1.3.1 Servicios asociados al concepto: Área

En primer lugar, tenemos los servicios asociados a la información del área. Estos servicios son:

- Lectura:
  - Método: GET.
  - Entrada: nada.
  - Respuesta:
    - Código 200 OK: devuelve una cadena de texto en formato JSON con la información de las áreas (ver Ilustración 19).
    - Código 400: devuelve un mensaje de error.
- Inserción:
  - Método: POST.
  - Entrada: cadena de texto en formato JSON con toda la información de un área (ver Ilustración 20).
  - Respuesta:
    - Código 200 OK: devuelve un mensaje indicando que la inserción ha tenido éxito.
    - Código 400: devuelve un mensaje indicando que la inserción no ha tenido éxito.

- Actualización:
  - Método: PUT.
  - Entrada: cadena de texto en formato JSON con la información actualizada de un área. Solo se pueden actualizar los campos: **name** y **description**. El campo **id** se utiliza para identificar al área (ver Ilustración 21).
  - Respuesta:
    - Código 200 OK: devuelve un mensaje indicando que la actualización ha tenido éxito.
    - Código 400: devuelve un mensaje indicando que la actualización no ha tenido éxito.


```
[
    {
        "id": 2,
        "archaeological_site_id": 1,
        "name": "Área 2 - Edificio de los Mosaicos y Edificio Paleocristiano",
        "description": "None",
        "corners": [{
            "x": 445127,
            "y": 4209811
        }, {
            "x": 445200,
            "y": 4209811
        }, {
            "x": 445200,
            "y": 4209991
        }, {
            "x": 445127,
            "y": 4209991
        }, {
            "x": 445127,
            "y": 4209811
        }],
        "is_active": true,
        "campaign_name": "Cástulo 20",
        "entry_info": "2020-06-14 11:46:28.737981"
    }
]
```


Ilustración 19: Ejemplo de respuesta de la lectura de las áreas

```
{
    "id": 2,
    "archaeological_site_id": 1,
    "description": "None",
    "corners": [{
        "x": 445127,
        "y": 4209811
    }, {
        "x": 445200,
        "y": 4209811
    }, {
        "x": 445200,
        "y": 4209991
    }, {
        "x": 445127,
        "y": 4209991
    }, {
        "x": 445127,
        "y": 4209811
    }],
    "is_active": true
}
```


Ilustración 20: Datos de entrada para la inserción de un área


```
{
    "id": 2,
    "name": "Área 2 - Edificio de los Mosaicos y Edificio Paleocristiano",
    "description": "None",
}
```


Ilustración 21: Datos de entrada para la actualización de un área

2.1.3.2 Servicios asociados al concepto: Volumen

En segundo lugar, tenemos los servicios asociados a la información del volumen. Estos servicios son:

- Lectura:
    - Método: GET.
    - Entrada: nada.
    - Respuesta:
        - Código 200 OK: devuelve una cadena de texto en formato JSON con la información de los volúmenes (ver Ilustración 22).
        - Código 400: devuelve un mensaje de error.

- Inserción:
    - Método: POST.
    - Entrada: cadena de texto en formato JSON con toda la información de un volumen (ver Ilustración 23).
    - Respuesta:
        - Código 200 OK: devuelve un mensaje indicando que la inserción ha tenido éxito.
        - Código 400: devuelve un mensaje indicando que la inserción no ha tenido éxito.
- Actualización:
    - Método: PUT.
    - Entrada: cadena de texto en formato JSON con la información actualizada de un volumen. No se dispone de campos que puedan ser actualizados.
    - Respuesta:
        - Código 200 OK: devuelve un mensaje indicando que la actualización ha tenido éxito.
        - Código 400: devuelve un mensaje indicando que la actualización no ha tenido éxito.

```
[
    {
        "id": 1,
        "area_id": 1,
        "corners": [{
            "x": 445220.8,
            "y": 4209930
        }, {
            "x": 445234.7,
            "y": 4209936
        }, {
            "x": 445233,
            "y": 4209956
        }, {
            "x": 445223.1,
            "y": 4209946
        }, {
            "x": 445220.8,
            "y": 4209930
        }],
        "campaign_name": "Cástulo 20",
        "entry_info": "2020-06-14 11:47:01.369913"
    }
]
```


Ilustración 22: Ejemplo de respuesta de la lectura de los volúmenes


```
{
    "id": 1,
    "area_id": 1,
    "corners": [{
        "x": 445220.8,
        "y": 4209930
    }, {
        "x": 445234.7,
        "y": 4209936
    }, {
        "x": 445233,
        "y": 4209956
    }, {
        "x": 445223.1,
        "y": 4209946
    }, {
        "x": 445220.8,
        "y": 4209930
    }]
}
```


Ilustración 23: Datos de entrada para la inserción de un volumen

2.1.3.3 Servicios asociados al concepto: Capa

En tercer lugar, tenemos los servicios asociados a la información de la capa. Estos servicios son:

- Lectura:
  - Método: GET.
  - Entrada: nada.
  - Respuesta:
    - Código 200 OK: devuelve una cadena de texto en formato JSON con la información de las capas (ver Ilustración 24).
    - Código 400: devuelve un mensaje de error.
- Inserción:
  - Método: POST.
  - Entrada: cadena de texto en formato JSON con toda la información de una capa (ver Ilustración 25).
  - Respuesta:
    - Código 200 OK: devuelve un mensaje indicando que la inserción ha tenido éxito.
    - Código 400: devuelve un mensaje indicando que la inserción no ha tenido éxito.
- Actualización:
  - Método: PUT.
  - Entrada: cadena de texto en formato JSON con la información actualizada de una capa. Solo se puede actualizar el campo **description**. El campo **id** se utiliza para identificar a la capa (ver Ilustración 26).
  - Respuesta:
    - Código 200 OK: devuelve un mensaje indicando que la actualización ha tenido éxito.
    - Código 400: devuelve un mensaje indicando que la actualización no ha tenido éxito.

```
[
    {
        "id": 1,
        "volume_id": 1,
        "description": "None",
        "z_start": [0, 0, 0, 0],
        "z_end": [4.29643, 3.297756, 4.348565, 4.332405],
        "campaign_name": "Cástulo 20",
        "entry_info": "2020-06-14 11:48:39.356478"
    }, {
        "id": 2,
        "volume_id": 1,
        "description": "None",
        "z_start": [4.29643, 3.297756, 4.348565, 4.332405],
        "z_end": [8.592859, 6.595511, 8.69713, 8.66481],
        "campaign_name": "Cástulo 20",
        "entry_info": "2020-06-14 11:48:40.352063"
    }
]
```


Ilustración 24: Ejemplo de respuesta de la lectura de las capas


```
{
    "id": 1,
    "volume_id": 1,
    "description": "None",
    "z_start": [0, 0, 0, 0],
    "z_end": [4.29643, 3.297756, 4.348565, 4.332405]
}
```


Ilustración 25: Datos de entrada para la inserción de una capa


```
{
    "id": 1,
    "description": "None"
}
```


Ilustración 26: Datos de entrada para la actualización de una capa

2.1.3.4 Servicios asociados al concepto: Elemento

Por último, tenemos los servicios asociados a la información de los elementos. Estos servicios son:

- Lectura:
  - Método: GET.
  - Entrada: nada.
  - Respuesta:

        - Código 200 OK: devuelve una cadena de texto en formato JSON con la información de los elementos (ver Ilustración 27).
        - Código 400: devuelve un mensaje de error.

- Inserción:
    - Método: POST.
    - Entrada: cadena de texto en formato JSON con toda la información de un elemento (ver Ilustración 28).
    - Respuesta:
        - Código 200 OK: devuelve un mensaje indicando que la inserción ha tenido éxito.
        - Código 400: devuelve un mensaje indicando que la inserción no ha tenido éxito.
- Actualización:
    - Método: PUT.
    - Entrada: cadena de texto en formato JSON con la información actualizada de un elemento. Solo se pueden actualizar los campos: **material_id** y **weight**. El campo **id** se utiliza para identificar al elemento (ver Ilustración 29).
    - Respuesta:
        - Código 200 OK: devuelve un mensaje indicando que la actualización ha tenido éxito.
        - Código 400: devuelve un mensaje indicando que la actualización no ha tenido éxito.

```
[
    {
        "id": 1623,
        "layer_id": 215,
        "material_id": 1,
        "material": "Cerámica                              ",
        "weight": "4.07731",
        "qr_image": null,
        "original_mesh": "Cuenco3
        "campaign_name": "Cástulo 20",
        "entry_info": "2020-06-14 12:28:29.837364",
        "coordinates": [{
            "x": "445171.2",
            "y": "18.92852",
            "z": "4209875"
        }]
    }
]
```


Ilustración 27: Ejemplo de respuesta de la lectura de los elementos


```
[
    {
        "id": 1623,
        "layer_id": 215,
        "material_id": 1,
        "weight": "4.07731",
        "original_mesh": "Cuenco3
        "coordinates": [{
            "x": "445171.2",
            "y": "18.92852",
            "z": "4209875"
        }]
    }
]
```


Ilustración 28: Datos de entrada para la inserción de un elemento


```
[
    {
        "id": 1623,
        "material_id": 1,
        "weight": "4.07731"
    }
]
```


Ilustración 29: Datos de entrada para la actualización de un elemento

2.1.3.5 Clases auxiliares

Además de los ficheros necesarios para el uso de la API, que se centran en el paso de información entre las aplicaciones, se dispone de un conjunto de clases que

implementan los servicios de la API. El objetivo de estas clases es la lectura, actualización e inserción de la información en la base de datos. Todas ellas se implementarán tanto en cliente como en servidor en sus lenguajes de programación correspondientes.

Las clases son las siguientes (ver Ilustración 30):

- **Database**: encapsula todo lo relacionado con la base de datos y permite mantener las conexiones a través de un PDO (PHP Data Objects). La información que se almacena es la necesaria para establecer una conexión a la base de datos.
- **ArchaeologicalSite**: encapsula todo lo relacionado con la información de un yacimiento arqueológico y la transformación de dicha información. La operación más importante es:
  - **read:** lee de la base de datos y devuelve toda la información en un array de yacimientos.
- **Area**: encapsula todo lo relacionado con la información de las áreas y la transformación de dicha información. Las operaciones más importantes son:
  - **read:** lee de la base de datos y devuelve toda la información en un array de áreas.
  - **create:** inserta una nueva área en la base de datos a partir de su información en formato JSON.
  - **update:** actualiza un área existente a partir de su información en formato JSON.
- **Volume**: encapsula todo lo relacionado con la información de los volúmenes y la transformación de dicha información. Las operaciones más importantes son:
  - **read:** lee de la base de datos y devuelve toda la información en un array de volúmenes.
  - **create:** inserta un nuevo volumen en la base de datos a partir de su información en formato JSON.

    - **update:** actualiza un volumen existente a partir de su información en formato JSON.

- **Layer**: encapsula todo lo relacionado con la información de las capas y la transformación de dicha información. Las operaciones más importantes son:
    - **read:** lee de la base de datos y devuelve toda la información en un array de capas.
    - **create:** inserta una nueva capa en la base de datos a partir de su información en formato JSON.
    - **update:** actualiza una capa existente a partir de su información en formato JSON.
- **Element**: encapsula todo lo relacionado con la información de los elementos y la transformación de dicha información. Las operaciones más importantes son:
    - **read:** lee de la base de datos y devuelve toda la información en un array de elementos.
    - **create:** inserta un nuevo elemento en la base de datos a partir de su información en formato JSON.
    - **update:** actualiza un elemento existente a partir de su información en formato JSON.
- **Point2D:** clase auxiliar para almacenar puntos en 2D. El método más relevante es:
    - **Project:** obtiene la proyección de un punto 3D con respecto a un plano: XY, YZ o XZ.
- **Point3D:** clase auxiliar para almacenar puntos en 3D.
- **Mesh:** clase auxiliar para construir una malla de triángulos a partir de un fichero OBJ.

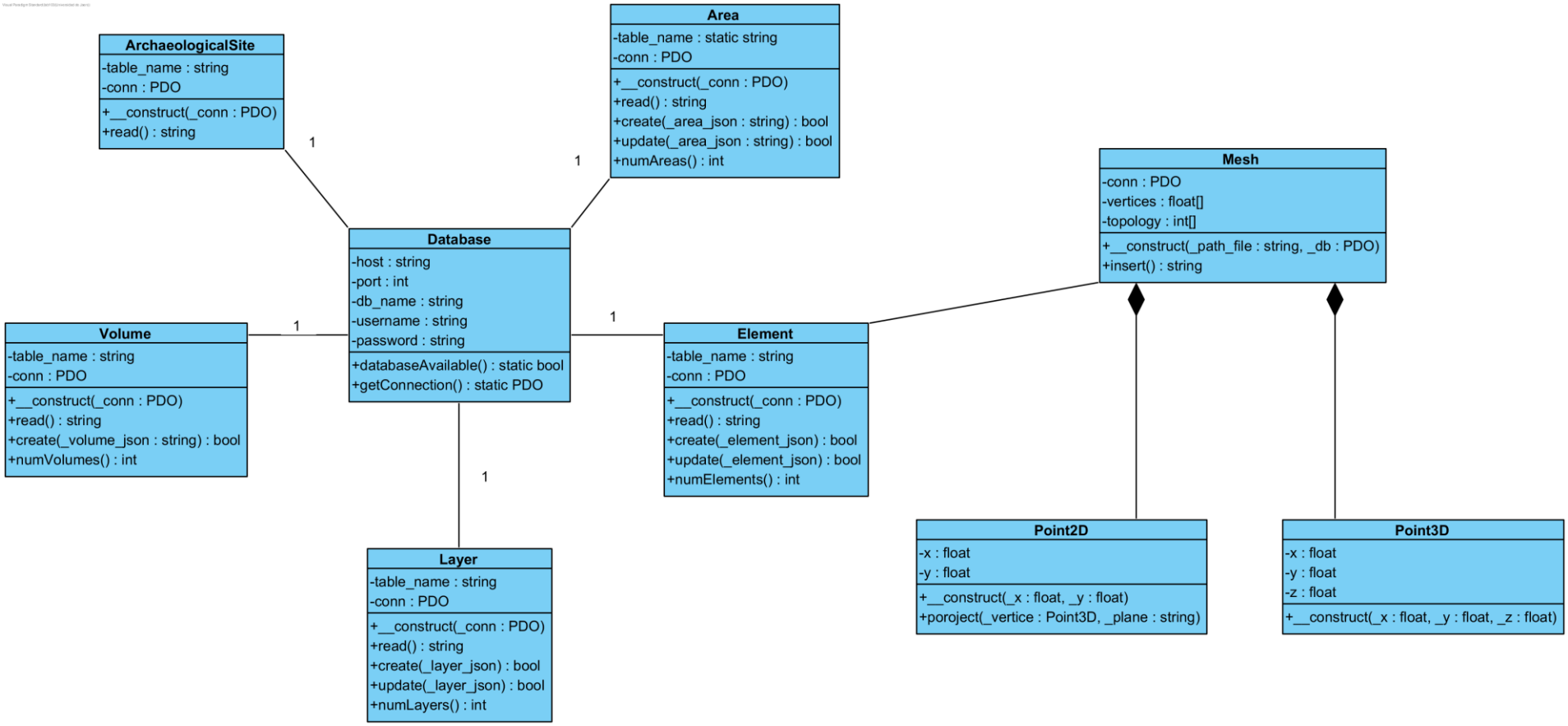


Ilustración 30: Diagrama de clases de la API

2.1.3.6 Sistema de carpetas

Una vez visto todo el diseño de la API, es hora de ver el sistema de carpetas del servidor, para dar una visión global de cómo está construido. La distribución de carpetas y archivos puede verse en la Ilustración 31.

En primer lugar, tenemos la carpeta **api** encargada de almacenar todos los ficheros que implementan los servicios web de la API REST. Contiene cuatro carpetas, una por cada concepto, y estas a su vez contienen un fichero PHP por cada uno de los servicios disponibles (read, insert y update).

En segundo lugar, tenemos la carpeta **objects** donde se almacenan las clases que encapsulan la transformación de la información en la base de datos.

Como ayuda a las clases de la carpeta anterior, disponemos de una carpeta donde se almacenan las clases que contienen funciones geométricas, denominada **geometric.**

Por último, disponemos de la carpeta **config** encargada de encapsular todo lo relacionado con la configuración para la conexión a la base de datos.

Como puede verse en la Ilustración 31, el sistema de carpetas está bastante estructurado y modularizado, lo que permite añadir de forma fácil y en cualquier momento nuevos servicios al proyecto.

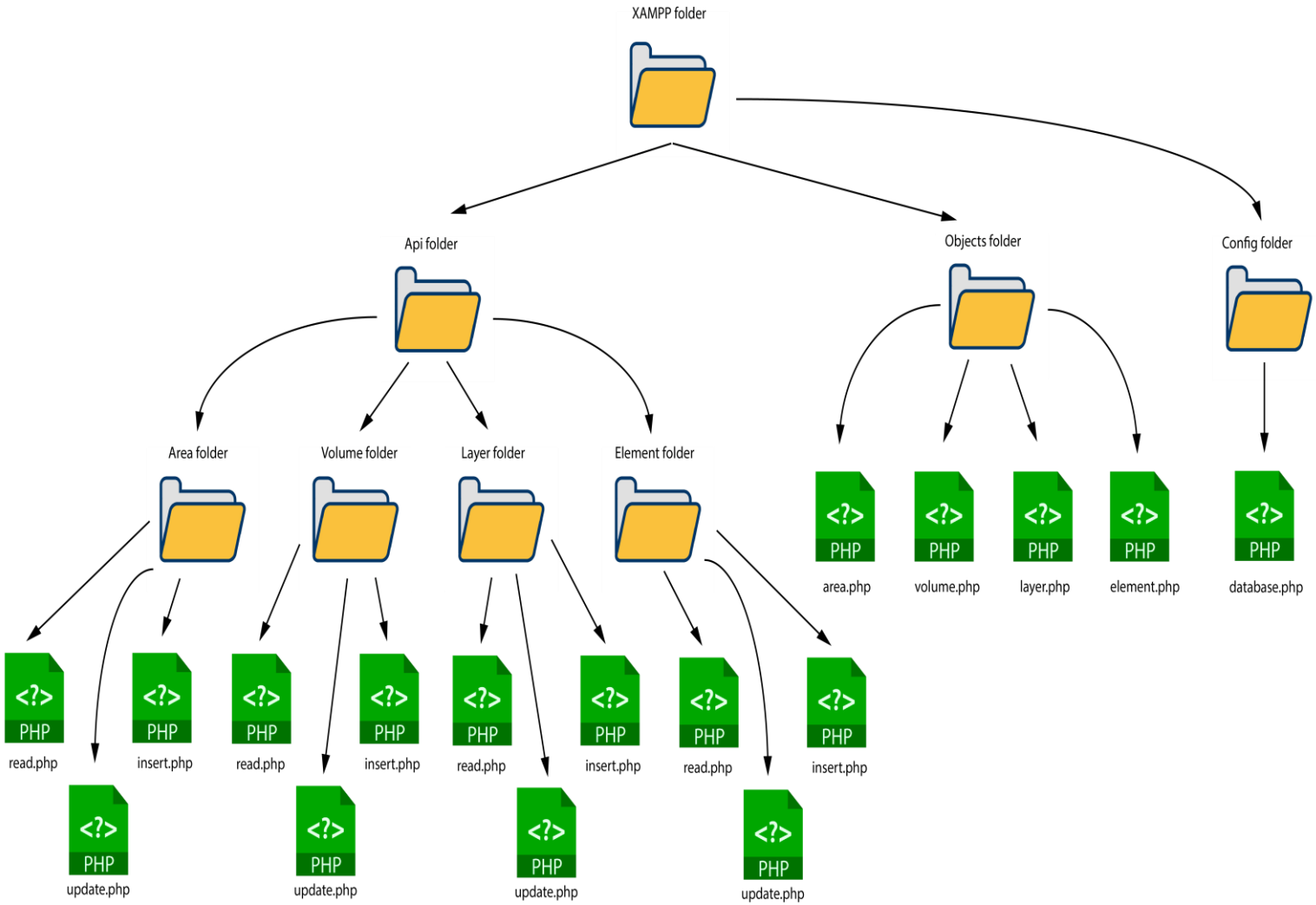


Ilustración 31: Sistema de carpetas de la API REST

2.1.4 Diseño inicial de la herramienta gráfica de realidad virtual

De igual manera que se explicó el diseño de la base de datos, se puede explicar el diseño de la herramienta de VR mediante el uso de paquetes donde se asocian clases con funcionalidad similar. Estos paquetes son:

- **Patrones de diseño**: clases de apoyo para implementar patrones de diseño.
- **Base de datos**: encapsula todo lo relacionado con el envío y recepción de información a la base de datos y la información de las clases principales.
- **Lector de ficheros**: clases que encapsulan la lógica de la lectura de ficheros.
- **Interfaz**: encapsula todo lo relacionado con las acciones del usuario en la interfaz.
- **Visualización**: encapsula todo lo relacionado con el comportamiento interno del proyecto.
- **Editor**: conjunto de clases que añaden funcionalidad al editor de Unity. Estas clases solo están disponibles cuando el proyecto se abre desde el editor de Unity.
- **Assets de terceros**: conjunto de assets de terceros utilizados en el proyecto para reutilizar assets ya existentes.

Pero antes de detallar el uso de cada uno de los paquetes, se va a detallar el diseño gráfico y visual de la aplicación. Desde la Ilustración 32 a la Ilustración 38 se muestran capturas de pantalla del estado inicial de la herramienta de visualización.

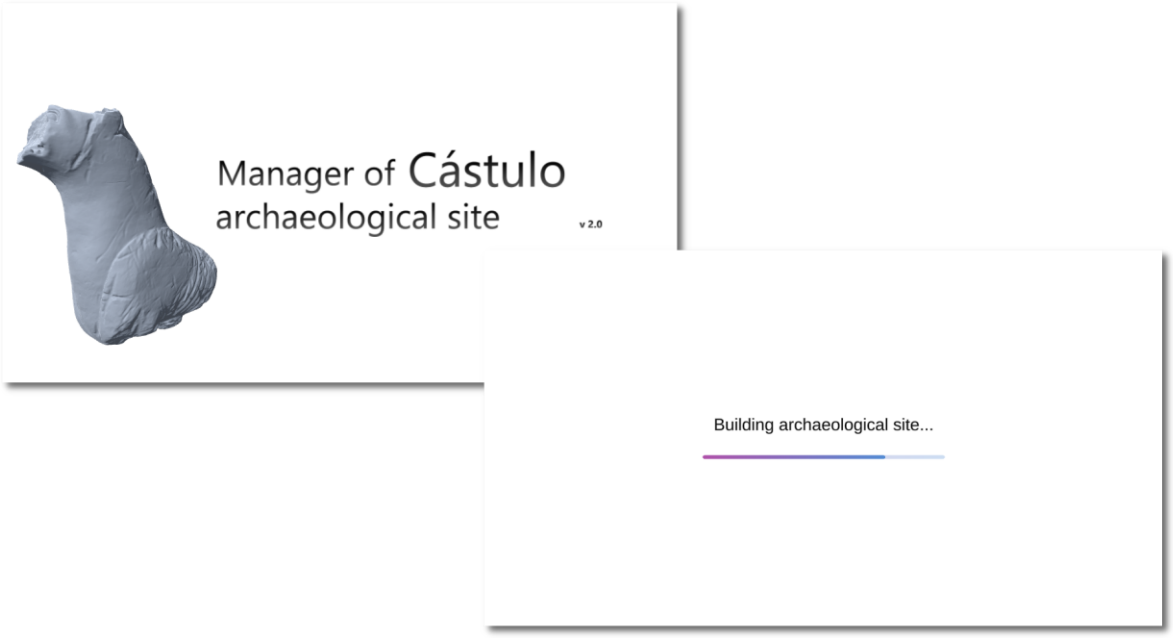


Ilustración 32: Pantallas de carga

Ilustración 33: Vista del yacimiento arqueológico

Ilustración 34: Uso de la funcionalidad de filtrado disponible en la aplicación

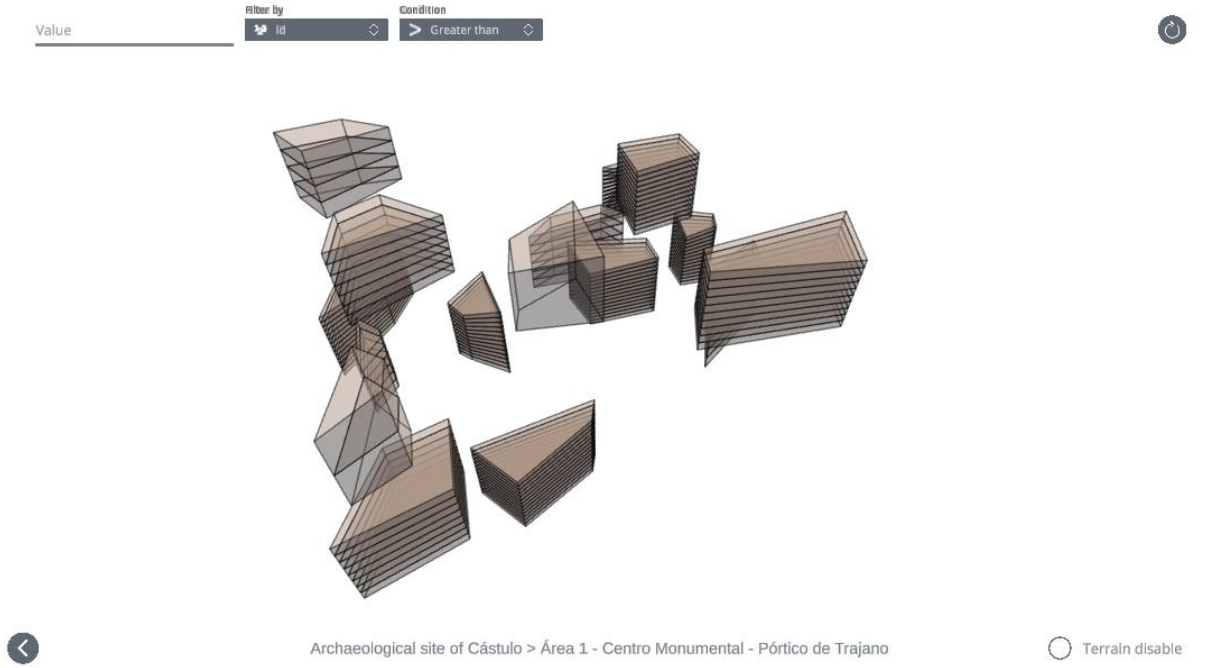


Ilustración 35: Visualización de los volúmenes y capas pertenecientes a un área

Ilustración 36: Información del concepto: Layer

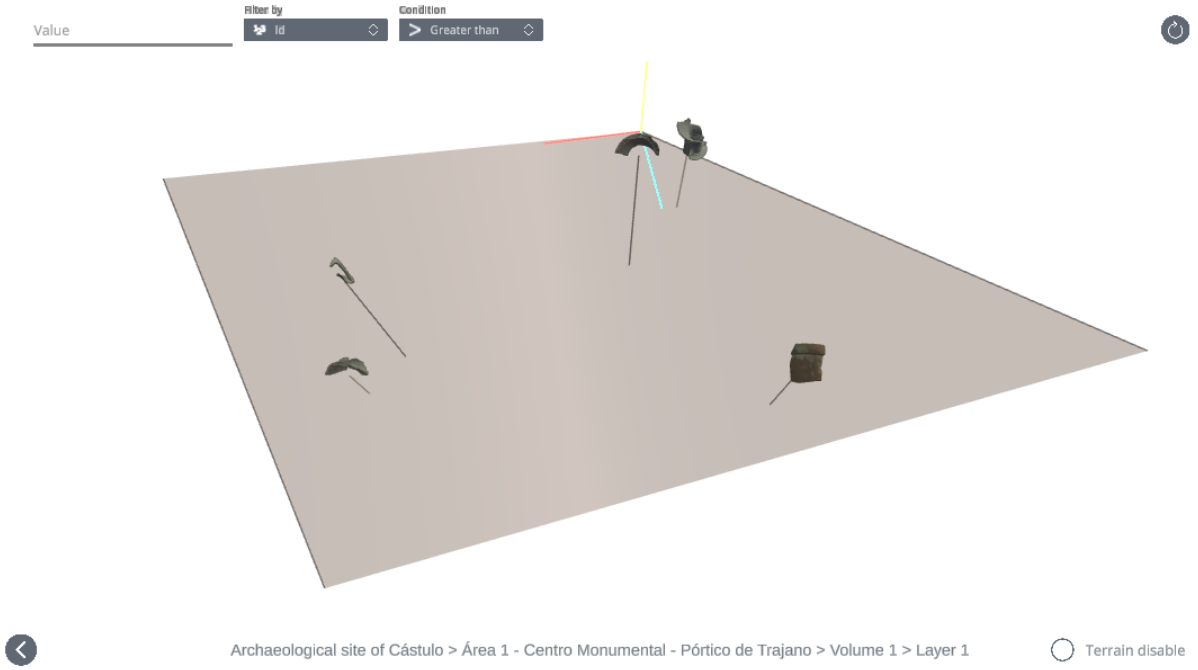


Ilustración 37: Visualización de los elementos en una capa

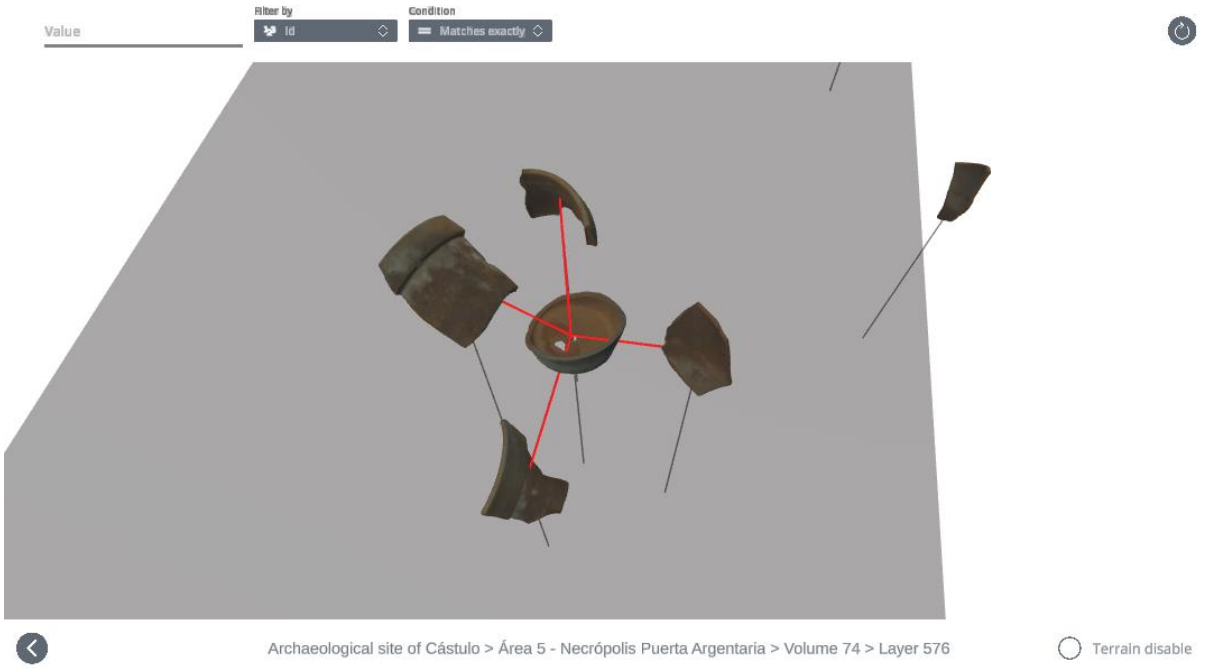


Ilustración 38: Simulación de las relaciones topológicas de los elementos en una capa

2.1.4.1 Patrones de diseño

Este paquete contiene las clases encargadas de implementar dos patrones de diseño (Pree, 1995): el patrón observador y el patrón DAO (Data Access Object) (ver Ilustración 39).

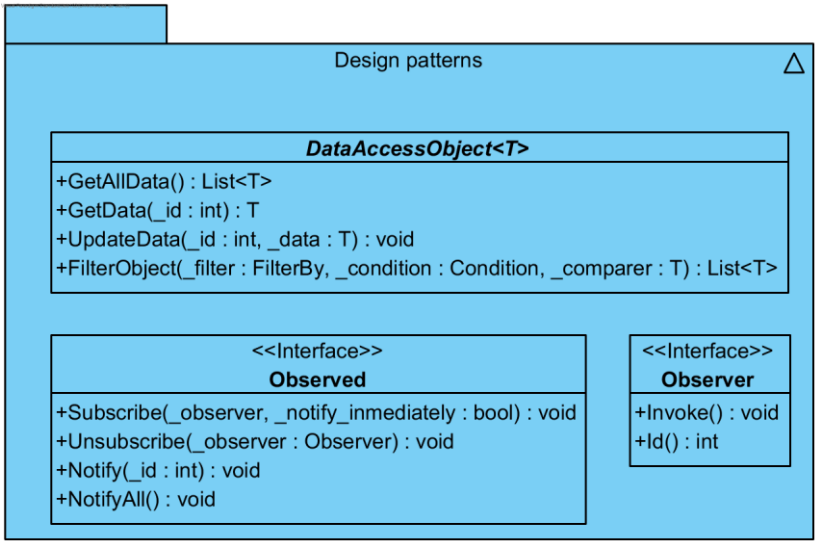


Ilustración 39: Diagrama de clases para los patrones de diseño

El patrón observador define una dependencia del tipo “uno a muchos” de manera que cuando uno de los objetos cambia su estado, este notifica el cambio a todos los dependientes.

El patrón DAO se utiliza para separar la lógica de acceso a los datos, de tal forma que el DAO encapsula esta lógica al resto de la aplicación.

2.1.4.2 Base de datos

En este paquete nos encontramos con las clases que se encargan de acceder a la base de datos a través de los servicios ofrecidos por la API REST. Hay una clase por cada tipo de concepto y todas ellas implementan el patrón DAO y el patrón observador. El diagrama de clases viene reflejado en la Ilustración 40.

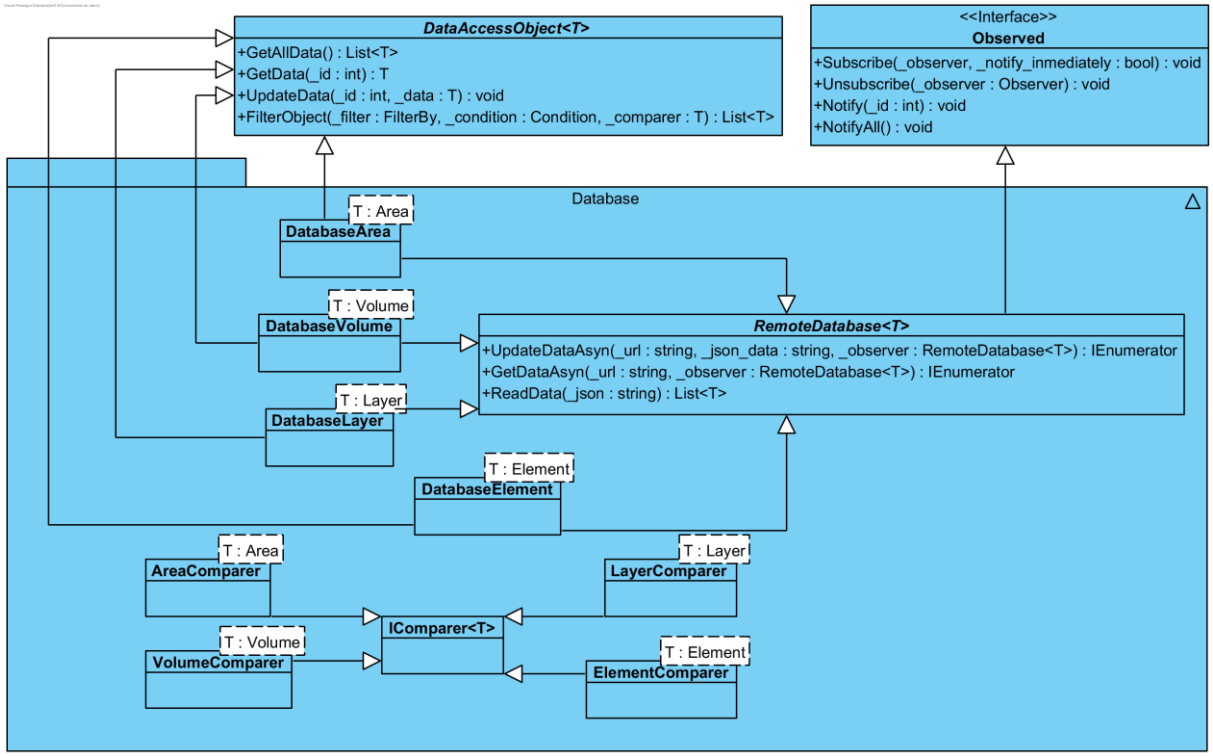


Ilustración 40: Diagrama de clases para las clases de la base de datos

Para poder almacenar la información es necesario crear las clases para el modelo de datos principal. Estas clases poseen como atributos los campos existentes de la base de datos. Las instancias de estas clases se obtienen mediante la transformación de la información en formato JSON.

Todo esto queda mejor detallado en la Ilustración 41.

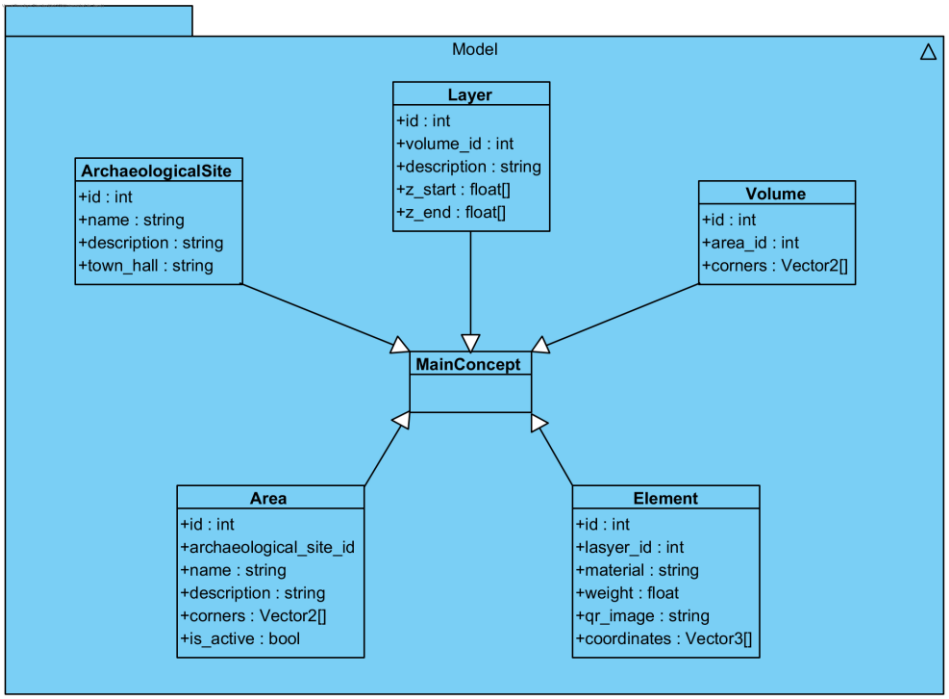


Ilustración 41: Diagrama de clases para el modelo de datos

2.1.4.3 Lector de ficheros

Este paquete contiene clases que permiten leer cualquier tipo de fichero. Aunque solo se ha diseñado para leer ficheros ASC y poder construir un DEM (Digital Elevation Model), al estar diseñado con herencia (ver Ilustración 42) es posible adaptar una clase hija para leer cualquier otro tipo de fichero y dotarla de un comportamiento personalizado.

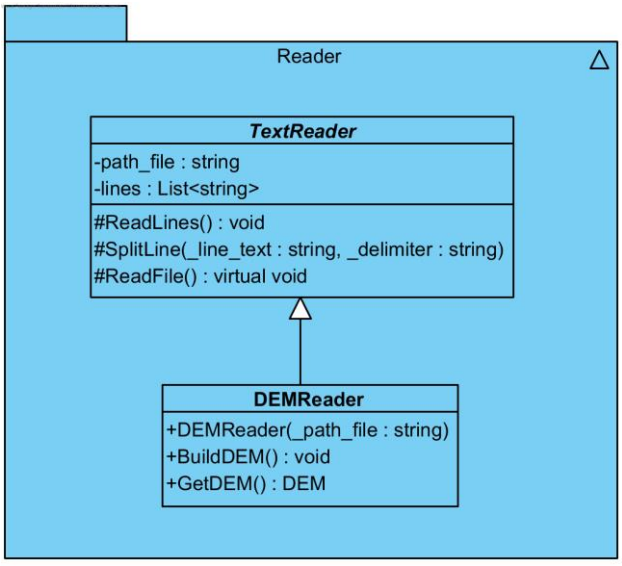


Ilustración 42: Diagrama de clases para la lectura de ficheros

2.1.4.4 Interfaz

Las clases que forman este paquete encapsulan la lógica de los comportamientos de la interfaz 2D (paneles, botones, entradas de texto, etc.).

Entre estas clases, encontramos la clase más importante: **InterfaceController**. Esta se encarga de:

- Recoger la información de entrada de los filtros.
- Notificar el comportamiento establecido para los botones presentes en la interfaz.
- Notificar los cambios en los controladores de listas y los cambios en la clase principal de la visualización, **VisualizerSite**.

Las demás clases sirven de apoyo a la principal o como puente para conectar unas clases con otras, como puede observarse en la Ilustración 43.

Algunas clases implementan el patrón observador como, por ejemplo, la clase **ListController**; mientras que otras clases hacen uso de las implementaciones del patrón DAO como, por ejemplo, la clase **AreaListAdapter**.

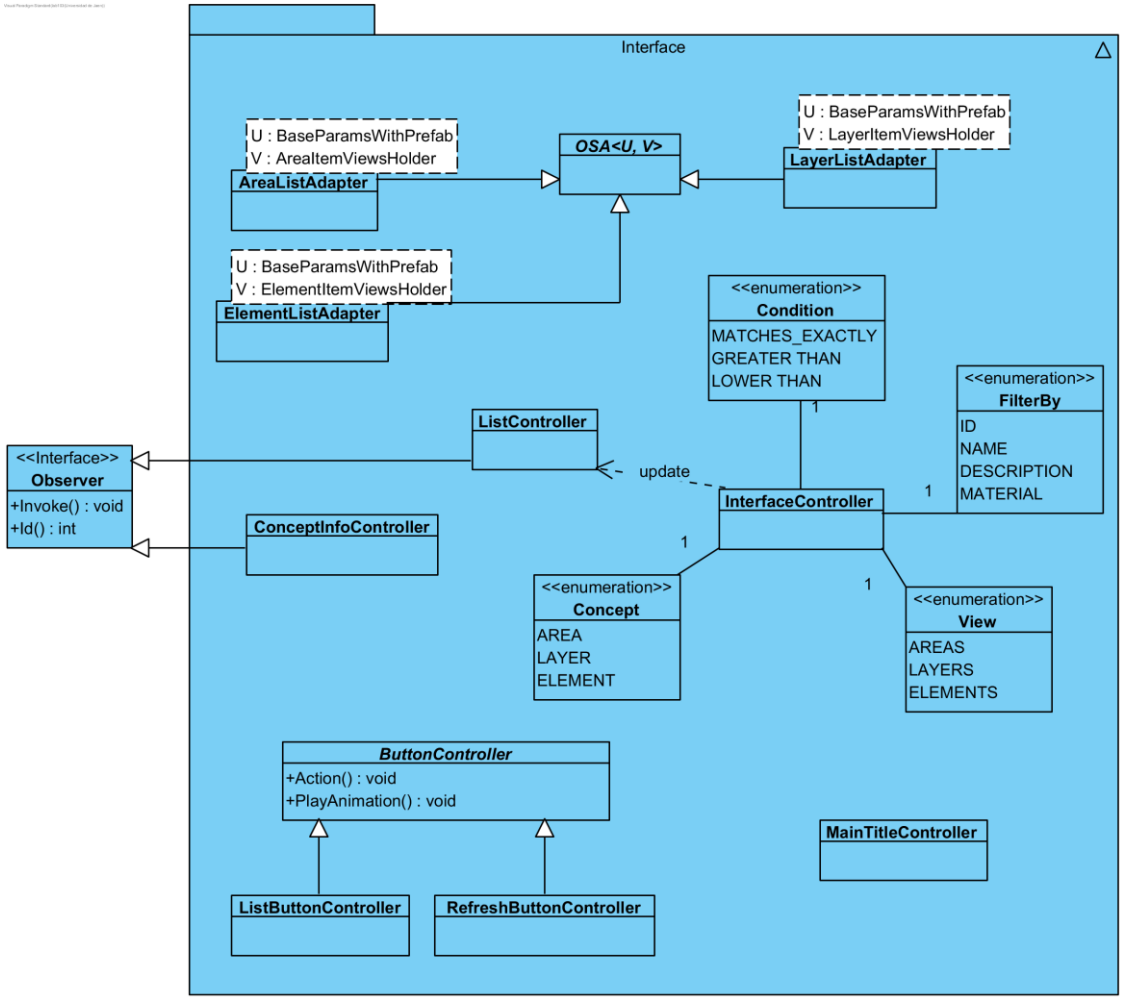


Ilustración 43: Diagrama de clases para el control de la interfaz

2.1.4.5 Visualización

Este paquete está formado por clases que se encargan de la creación y visualización de todos los objetos 3D, incluyendo las mallas de triángulos que forman cada uno de los conceptos principales: áreas, volúmenes, capas y elementos. Para los tres conceptos iniciales se utiliza la clase **Polygon3D**, mientras que, para los elementos, se utiliza la clase **Element3D**.

La clase principal es **VisualizerSite**, aunque existen clases auxiliares que ayudan en sus funciones (Ilustración 44). Estas clases auxiliares se caracterizan por contener pequeños scripts con comportamientos sencillos (a excepción de la clase principal).

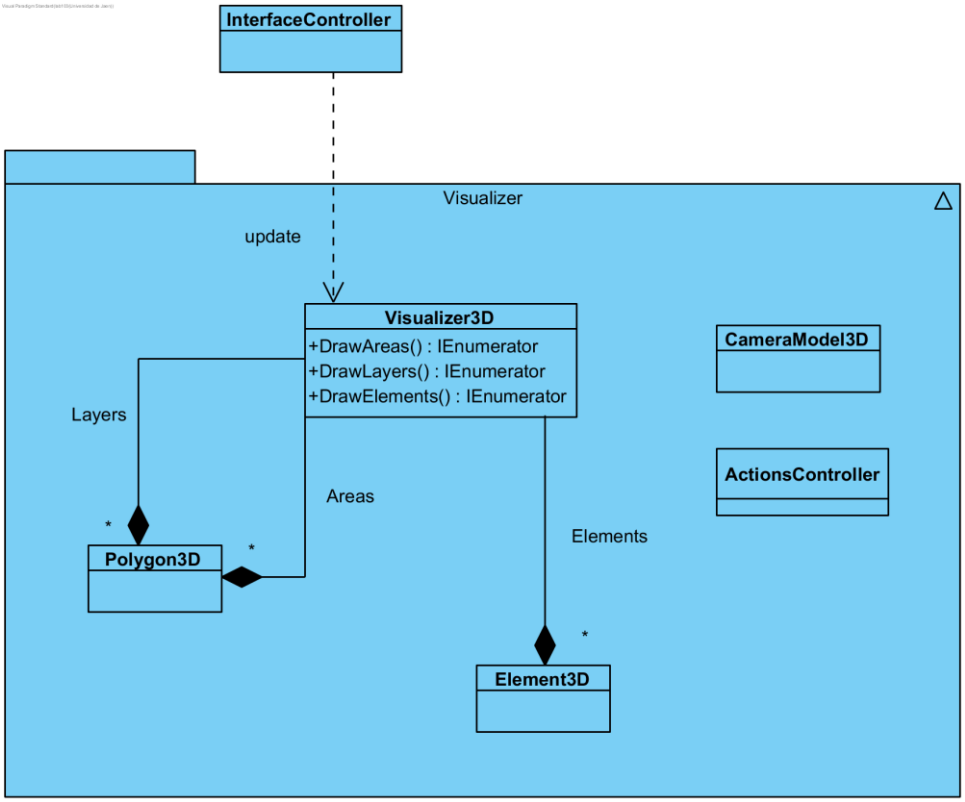


Ilustración 44: Diagrama de clases para el paquete de visualización

2.1.4.6 Editor (Unity)

Cuando se crea un script en Unity, este únicamente es ejecutado en el modo ejecución. Para poder ejecutar código sin la necesidad de ejecutar el proyecto, Unity permite a los usuarios la posibilidad de crear editores personalizados y añadir funcionalidad extra.

Por tanto, este paquete contiene las clases que se están utilizando para añadir funcionalidad extra al propio editor que por defecto utiliza Unity. El diagrama puede verse en la Ilustración 45.

Las funciones extras son:

- **DEM Builder**: crea el terreno a partir de un fichero ASC y lo almacena como prefab para poder utilizarlo en el proyecto.
- **Screenshot**: utilizado para tomar capturas de pantalla de lo que está visualizando la cámara.
- **Procedural archaeological site**: crea una base de datos completa y desde cero utilizando como base el conjunto de áreas.

En la versión inicial del proyecto, algunos de los scripts que se están utilizando no funcionan correctamente, por lo que estas funcionalidades extra no están disponibles por el momento.

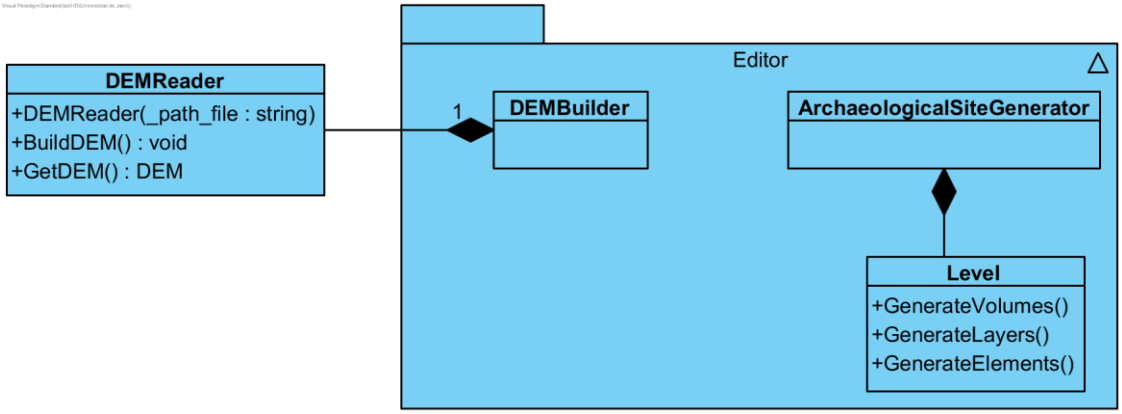


Ilustración 45: Diagrama de clases para las clases del editor

2.1.4.7 Assets de terceros

En la versión inicial del proyecto se ha utilizado software que está a disposición de cualquier desarrollador accediendo a la tienda oficial de Unity.

Entre los utilizados en este proyecto, encontramos:

- **Modern UI Pack**[19]: conjunto de assets y scripts que aportan un estilo moderno para la interfaz junto a sus comportamientos.
- **OSA (Optimized Scroll View Adapter)**[20]: conjunto de scripts para mantener listas eficientes que aparecen en la parte izquierda de la aplicación.
- **Runtime Transform Gizmos**[21]**:** API para el control y movimiento de la cámara, transformación de objetos en tiempo de ejecución**,** sistema de rehacer/deshacer, entre otros.

[19] https://assetstore.unity.com/packages/tools/gui/modern-ui-pack-150824
[20] https://assetstore.unity.com/packages/tools/gui/optimized-scrollview-adapter-68436
[21] https://assetstore.unity.com/packages/tools/modeling/runtime-transform-gizmos-125537

2.1.5 Diseño inicial de la página web

La web está diseñada para:

- Mostrar la información almacenada en la base de datos, en cualquier momento y a cualquier persona. La información se muestra de forma clara y concisa mediante el uso de tablas. Actualmente la información que puede contemplarse es la relativa a los conceptos principales de un yacimiento arqueológico: área, volumen, capa y elemento.
- Generar una base de datos a partir de la lectura de ficheros CSV (Comma Separated Values) provenientes de la base de datos de Cástulo e introducir la geometría de los modelos.

En cuanto al estilo visual, que se puede ver con más detalle desde la Ilustración 46 a la Ilustración 48, la web está compuesta por un panel izquierdo con dos opciones. La primera de ellas permite visualizar el contenido de la base de datos ayudándose de tablas que se muestran en la parte derecha, pudiendo intercambiar entre las tablas correspondientes a cada concepto. La segunda opción permite generar la base de datos con una selección muy simple de la información que se va a añadir.

Es destacable que la generación de la base de datos fue desarrollada para introducir la información sobre el yacimiento de Cástulo. Aunque se dispone de esta funcionalidad, no se le va a dar uso, puesto que como veremos más adelante (ver apartado 3.2.7 (Generación procedural de la base de datos)) se dispone de un mecanismo de generación de la base de datos de manera procedural integrado en Unity.

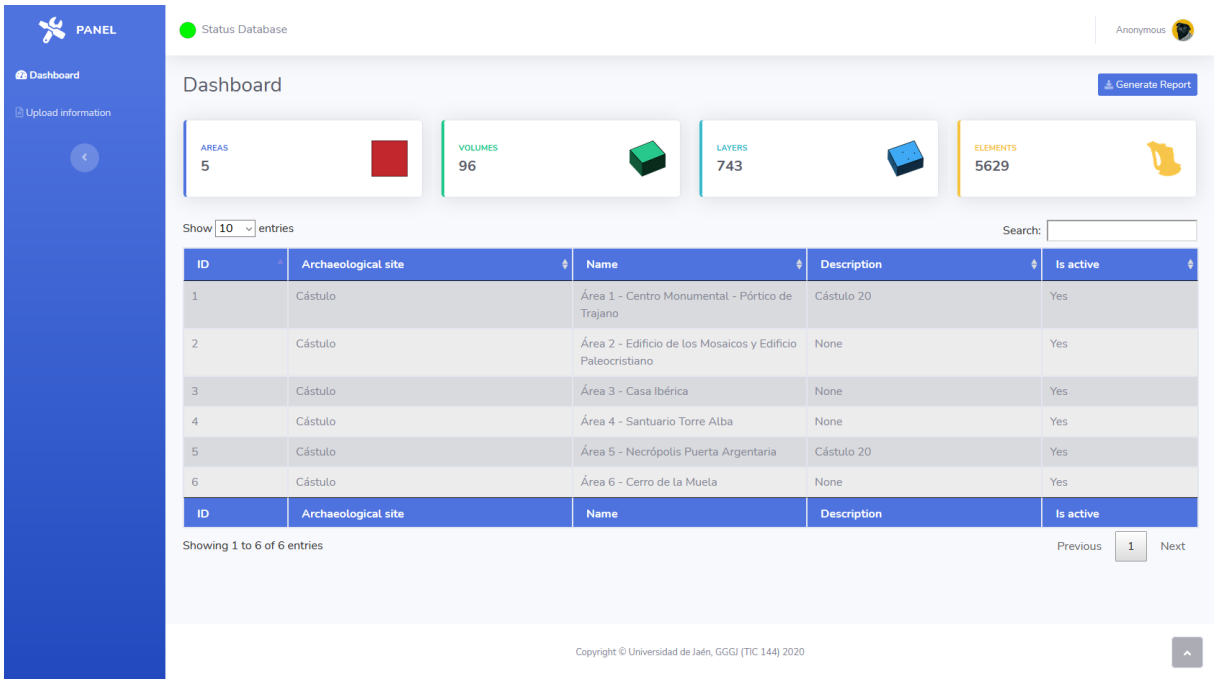


Ilustración 46: Aspecto visual de la web. Tabla áreas

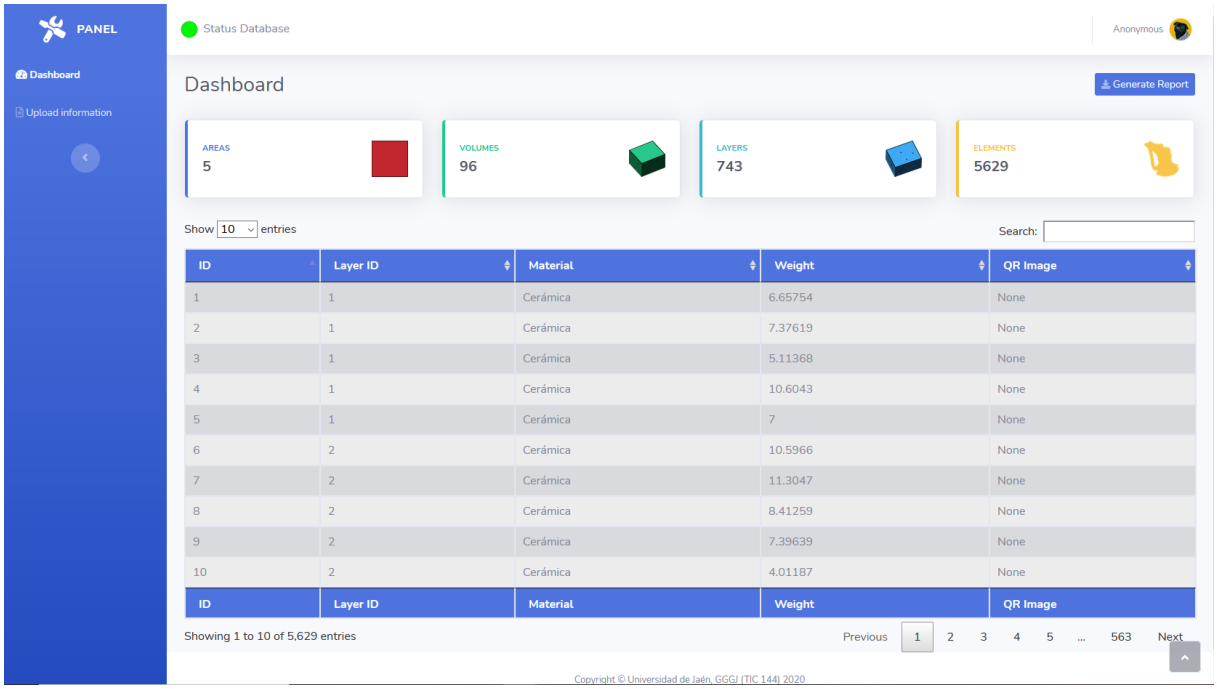


Ilustración 47: Aspecto visual de la web. Tabla elementos

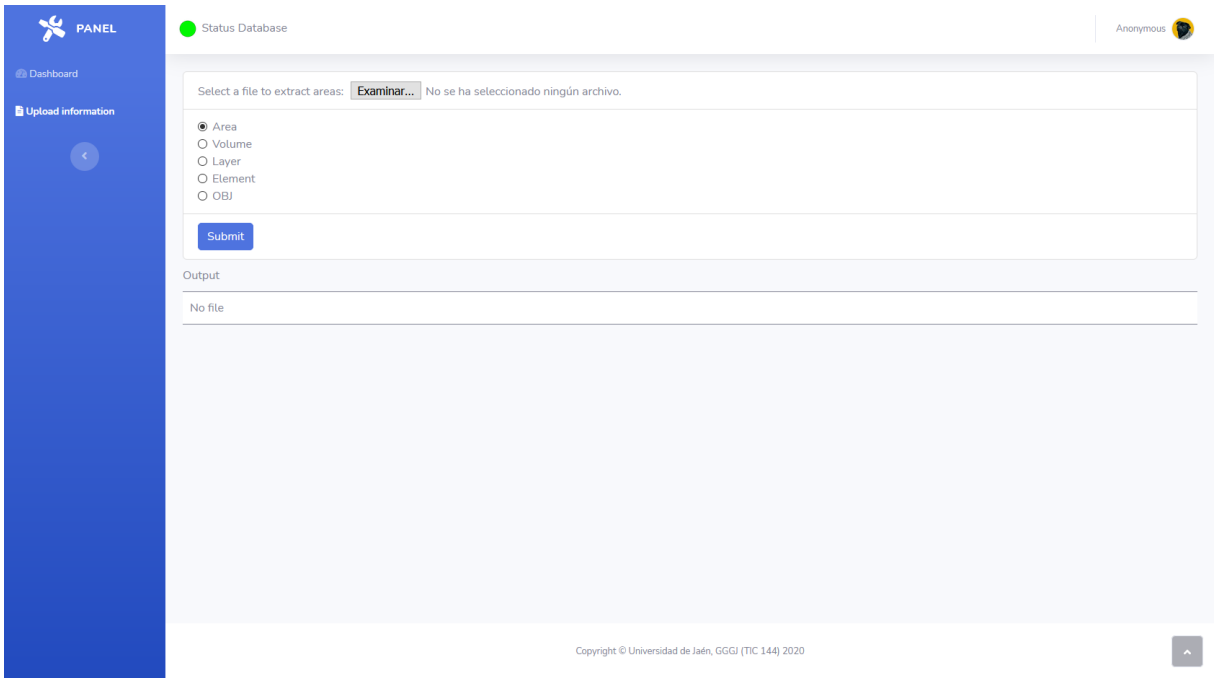


Ilustración 48: Aspecto visual de la web. Inserción de datos

Por último, tenemos que destacar que la continuación de la web no forma parte de este TFM, por lo que no se diseñará ni desarrollará nada relacionado con ella. Únicamente se proporcionará el mantenimiento necesario para la visualización rápida de todo el conjunto de datos de los que se dispone.

3 DESARROLLO

Este apartado tiene como punto de partida el diseño descrito en el apartado anterior, al cual se le han añadido o cambiado funcionalidades conforme se avanzaba en el desarrollo del proyecto. A continuación, se describen todas las iteraciones por las cuales ha pasado el desarrollo del proyecto.

3.1 Iteración 0

Está destinada al estudio del proyecto del que se ha partido, por lo que podemos nombrarla como la iteración cero, dado que es necesaria para comenzar el propio desarrollo del proyecto. Se ha llevado a cabo durante los días 19/10/2020 y 10/11/2020.

La realización de esta iteración ha permitido describir el diseño explicado en el apartado 2 (Diseño inicial), por lo que los resultados obtenidos durante el desarrollo de esta iteración ya han sido expuestos en este trabajo de forma detallada.

Además, al realizar el estudio para cada una de las partes en las que se divide el proyecto se ha podido encontrar las carencias y/o errores de los que sufre cada parte. Por tanto, en esta iteración, se van a listar aquellas deficiencias encontradas.

En esta iteración también se comprobó el software necesario para el correcto funcionamiento del proyecto, explicado en el apartado 2.1.1 (Arquitectura inicial del sistema), así como la instalación de aquel software que se necesitara y no estuviera en el apartado mencionado.

En cuanto a la **base de datos**, se realizó el estudio de cada una de las tablas que la forman, así como las relaciones entre ellas. Esto dio lugar a los subsistemas en los que se divide su diseño inicial.

Además, se comprobó la validez de los datos mediante una serie de consultas sencillas. Y dado la inexperiencia del autor del trabajo con el gestor de bases de datos PostgreSQL, la realización de consultas ayudó a la familiarización con dicho gestor.

Aunque la base de datos se ha generado de manera procedural, se dispone de la base de datos inicial utilizada por el equipo que trabaja en el yacimiento arqueológico de Cástulo. Los datos con los que finalmente se ha trabajado no se corresponden con los contenidos en la base de datos de dicho yacimiento, pero se ha

utilizado esta información como base para la creación de las tablas de la base de datos actual.

En la base de datos no se han encontrado carencias o errores que hicieran falta solucionar.

En cuanto al **servidor web**, este está formado por la página web que muestra un resumen de los datos almacenados y la API para el intercambio de datos.

El estudio de la web no supone mayor problema. Tan solo es necesario entender la navegación entre las distintas páginas web. Como se ha mencionado en apartados anteriores, no se a modificar ningún comportamiento de la web. No obstante, conviene conocer su comportamiento.

En cuanto a la API REST, sí es necesario comprender su funcionamiento, puesto que, al servir como puente para la inserción y actualización de datos, es necesario saber cuáles son las tablas que se pueden actualizar y qué campos deben ser obligatorios para insertar o actualizar una fila en la base de datos.

Al igual que con la base de datos, no ha sido necesario resolver ningún error ni ha hecho falta ninguna mejora.

Por último, nos centramos en la **aplicación gráfica**. Es en este punto donde se ha dedicado la mayor parte del tiempo empleado durante el transcurso de esta iteración, ya que la aplicación en Unity se considera la parte más importante del proyecto y donde se enfocarán las siguientes iteraciones y, por tanto, es necesario un correcto entendimiento de la misma.

En este caso se ha hecho especial énfasis en la estructura de carpetas de Unity para conocer el estado de cada uno de los objetos. De esta manera podemos conocer cuáles son los assets y prefabs se están utilizando y cuáles no. También podemos conocer cómo están formados cada uno de ellos. De igual manera, se han repasado todos los scripts (no pertenecientes a paquetes de terceros) para encontrar fallos en la lógica de programación mientras se entendía su funcionamiento.

A diferencia de los casos anteriores, esta vez sí se han detectado errores a solucionar y ha surgido la necesidad de añadir nuevas funcionalidades. A continuación, se enumeran:

1. Limpieza de código y objetos poco útiles.
2. Distinción en la visualización entre volumen y capa.
3. Solución y/o implementación de filtros.

4. Listas para la búsqueda y localización rápida de conceptos.
5. Permitir retroceder en la jerarquía de visualización hacia el concepto anterior.
6. Mejoras en el control de la cámara y su sensibilidad al movimiento.
7. Integración de la generación procedural del yacimiento.
8. Cierre de la aplicación.

Todos estos cambios se abordarán en la siguiente iteración.

3.2 Iteración 1

La primera iteración recoge las mejoras y cambios propuestos a partir del estudio realizado en la iteración anterior. El desarrollo abarca desde el día 11/11/2020 al día 04/12/2020.

Apenas existen limitaciones por las cuales una mejora debe ser desarrollada antes de comenzar la siguiente, por lo que cada una de ellas es totalmente independiente del resto. Motivo por el cual cada una de ellas se explica en un apartado distinto.

3.2.1 Limpieza

El estudio completo del proyecto software nos permite realizar una limpieza del código en el mismo. Con el término “limpieza” nos referimos a la eliminación de comentarios innecesarios y pequeños trozos de código que nunca se han llegado a utilizar, asi como la escritura de comentarios donde se considere necesario. Cabe destacar que la documentación del proyecto es extensa y completa, pero esto no impide que haya partes del software no documentadas o comentarios innecesarios que, para facilitar la lectura, se ha decidido eliminar.

Aprovechando esta tarea se pretende eliminar cualquier archivo, fichero, o carpeta que no se utilice en el proyecto. Por tanto, se pretende eliminar algunos de los objetos existentes en Unity que no han sido usados en ningún momento por la aplicación. Estos incluyen modelos repetidos pertenecientes a la creación del terreno a partir de archivos DEM, así como pequeños scripts y bibliotecas que no se han

utilizado durante el desarrollo del proyecto. De igual manera, se han movido diferentes archivos de una carpeta a otra con el fin de reestructurar el sistema de ficheros.

Aunque esta tarea no se considera necesaria ni importante para abordar las siguientes tareas, se ha visto conveniente invertir tiempo en la realización de la misma para facilitar el desarrollo posterior.

3.2.2 Visualización de volúmenes y capas

En esta tarea nos vamos a centrar en mejorar la visualización propuesta por la aplicación para los conceptos de Volumen y Capa. Es importante dejar claro que un volumen está formado por un conjunto de capas, una debajo de la otra. A su vez, una capa contiene varios elementos.

La situación inicial del proyecto con respecto a la visualización de estos dos conceptos se muestra en la Ilustración 49 y en la Ilustración 50.

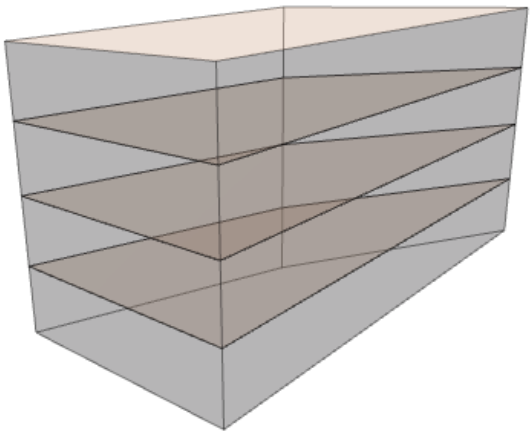


Ilustración 49: Visualización inicial de un volumen

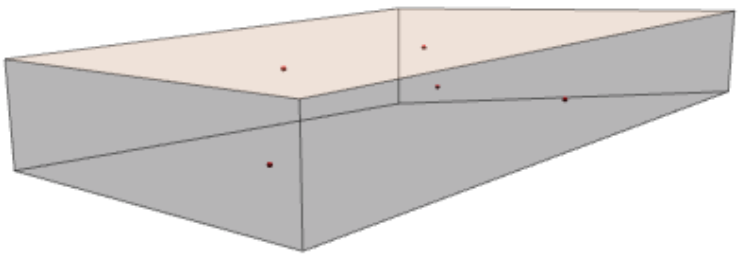


Ilustración 50: Visualización inicial de una capa

Se pretende mejorar esta visualización para facilitar el uso de la herramienta por parte del usuario final, puesto que se piensa que esta visualización resulta confusa.

A nivel de base de datos se realiza una clara distinción entre lo que es un volumen y lo que es una capa. Sin embargo, esto no está tan claro a nivel externo (parte visual) y a nivel interno (programación) en la herramienta gráfica. La mejora en la visualización se propone debido a la clara estructura de los conceptos en un yacimiento arqueológico que debe ser necesario que quede reflejada en la herramienta.

El volumen como tal no tiene altura inicial ni altura final, sino que se obtienen de acuerdo al tamaño de cada una de sus capas. Por lo que su construcción se asemeja a un prisma. Estas bases coinciden con la forma geométrica del volumen almacenada en la base de datos, por lo que tenemos almacenados sus vértices para su construcción (Ilustración 51). La construcción de las caras del prisma no resulta difícil, ya que estos vértices vienen dados por las coordenadas almacenadas en la base de datos (siendo necesaria su conversión) y las alturas máximas y mínimas referentes a las capas que forman al volumen.

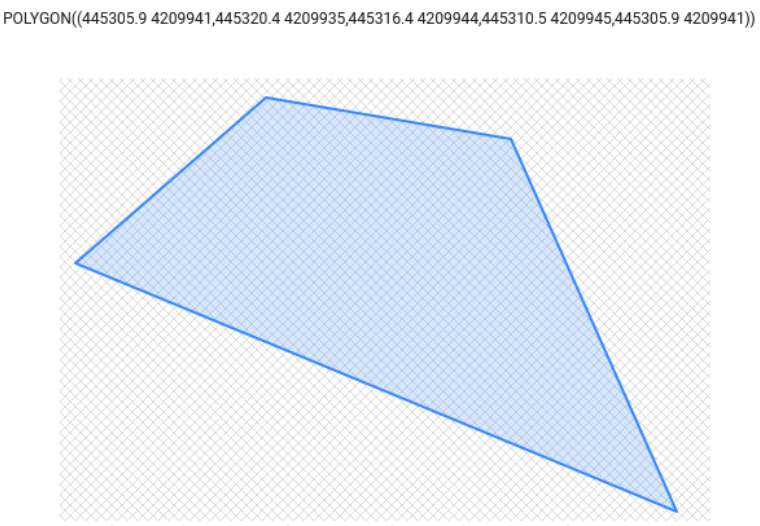


Ilustración 51: Geometría del volumen almacenada en la base de datos

Para crear la malla de triángulos que forma el prisma tan solo es necesario construir cada una de sus caras (cuadriláteros), teniendo en cuenta que los vértices

deben estar dispuestos en sentido antihorario, tal y como se muestra en la Ilustración 52.

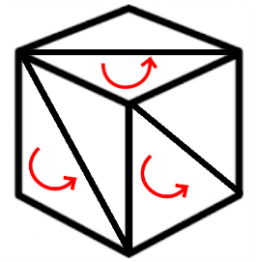


Ilustración 52: Orientación de los triángulos

Todo esto está orientado a la parte visual del proyecto. Pero también es necesario cambiar la lógica para dar soporte a los scripts necesarios que implementan este comportamiento.

Actualmente la aplicación está diseñada mediante vistas, por lo que cada concepto representa una vista. En el estado inicial, las vistas de las que disponemos son: Area, Layer y Element. Por tanto, es necesario introducir una nueva vista: Volume.

No es necesario modificar el diagrama de clases para esta tarea. Esto es debido a que los scripts ya están preparados para cambiar de una vista a otra y tan solo es necesario añadir la nueva vista. La adición de esta únicamente consiste en ejecutar el mismo código ya escrito, pero teniendo en cuenta que ahora debemos trabajar con lo que hemos denominado volumen.

El resultado final en el que queda la nueva jerarquía de vistas para los volúmenes y las capas viene representado en la Ilustración 53. De este modo se representa de una manera más clara la distinción realizada entre cada uno de los conceptos. Además, aunque no es necesario para su implementación, el diseño actual facilita el desarrollo del apartado 3.2.5 (Retroceso en la jerarquía de conceptos).

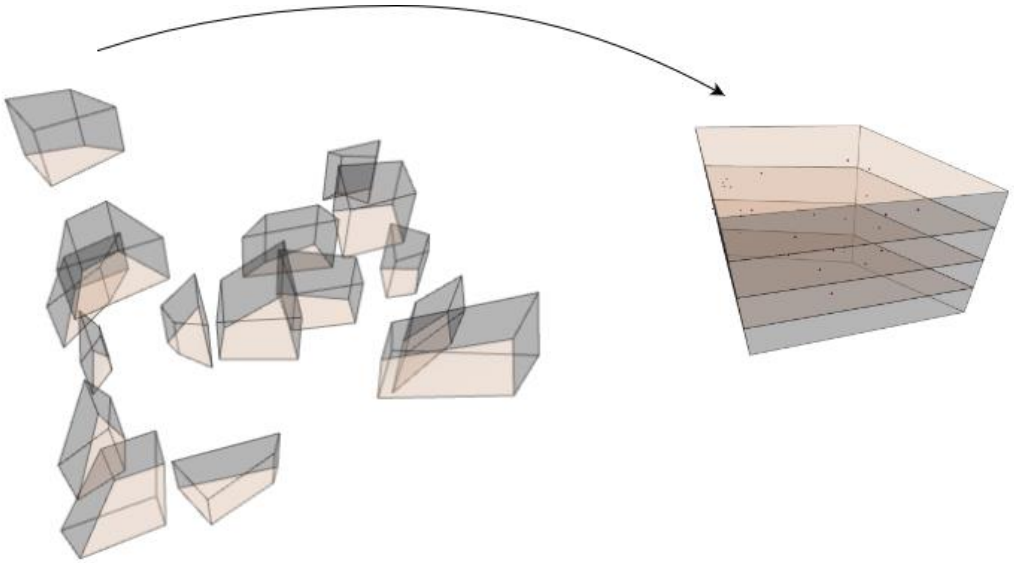


Ilustración 53: Nueva distinción entre volumen y capa

3.2.3 Filtros

En primer lugar, se van a explicar los filtros y condiciones de las cuales se dispone.

En el caso de los filtros, estos son:

- **ID**: permite búsquedas por el id del concepto almacenado en la base de datos.
- **Name**: permite búsquedas por el nombre asociado al concepto. Únicamente utilizado para áreas.
- **Description**: permite búsquedas por la descripción dada al concepto. Únicamente utilizado para volúmenes.
- **Material**: permite búsquedas por el material de construcción de un elemento arqueológico. Únicamente utilizado para elementos.

Como complemento a estos filtros se utilizan condiciones de búsqueda, las cuales son:

- **Matches exactly**: devuelve aquellos conceptos que coinciden exactamente con el filtro utilizado.
- **Greater than**: devuelve aquellos conceptos con valores mayores al indicado en el filtro. Sólo disponible con búsquedas numéricas.

- **Lower than**: igual que el anterior, pero para valores menores. También está disponible únicamente para búsquedas numéricas.

Teniendo en cuenta lo anterior, esta funcionalidad se encuentra presente en cualquiera de las vistas expuestas. En este caso se ha comprobado que la implementación no es correcta, como puede verse en la Ilustración 54 en la que se ve un estado que no debería darse puesto que no todos los filtros están disponibles para todos los conceptos y, por tanto, esto es lo primero a comprobar.

Ilustración 54: Selección del filtro a aplicar

Una vez que las opciones de filtrado se muestran correctamente sobre los conceptos correspondientes, es hora de implementar aquellas funciones que permitan realizar el filtrado.

El comportamiento de los filtros se debería de aplicar sobre la vista en la que se encuentra el usuario pero, en la Ilustración 55 podemos ver como se está intentando aplicar un filtro de tipo “id menor que 10” en la vista de los volúmenes sin éxito, ya que podemos acceder al volumen con id 16.

La solución consiste en copiar el código que funciona correctamente para las áreas y replicarlo para los demás conceptos, cambiando lo necesario para que funcione en cada uno de ellos. Todo ello resulta fácilmente programable puesto que el código ya está preparado para dichos cambios.

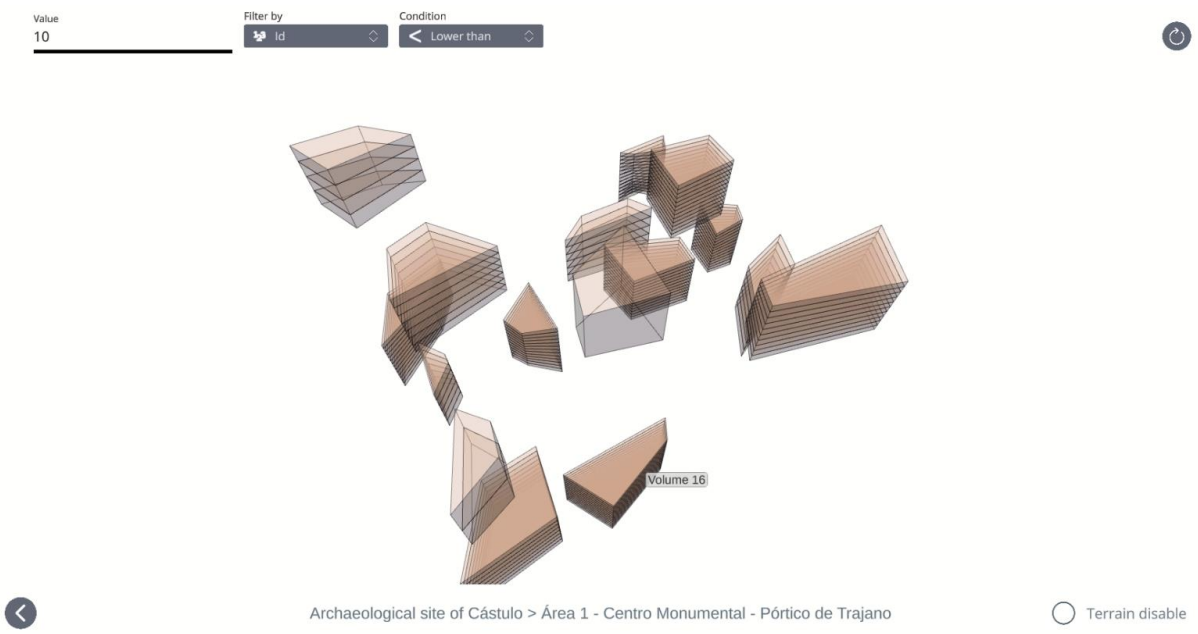


Ilustración 55: Ejemplo de un mal funcionamiento de los filtros

Una vez reprogramados los filtros ya obtenemos el comportamiento adecuado que se buscaba desde un principio: aplicar los filtros correspondientes dependiendo de la vista actual que se muestra en la aplicación. Esto se muestra en la Ilustración 56 con un ejemplo.

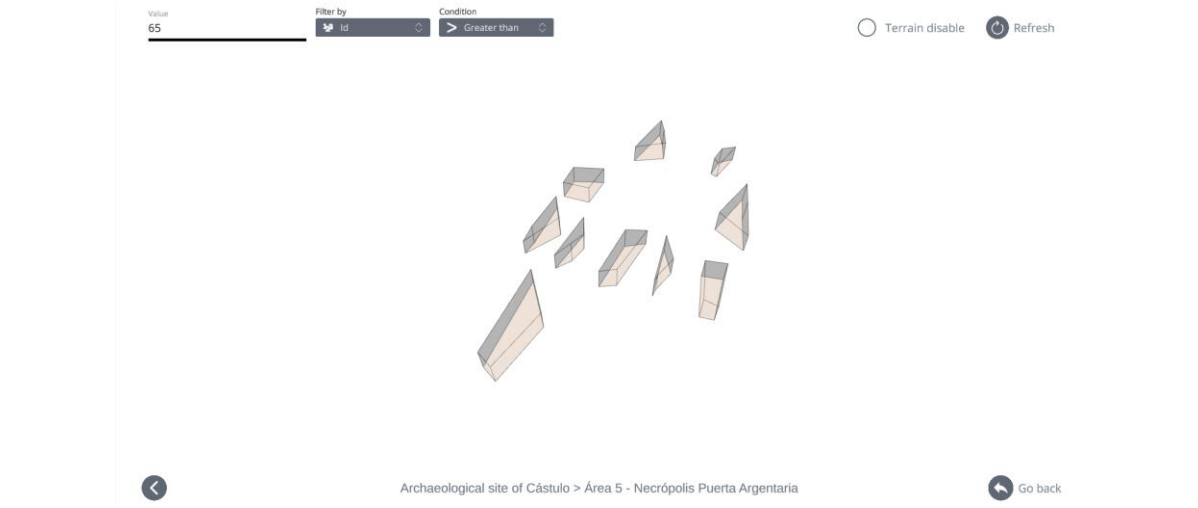


Ilustración 56: Ejemplo del correcto funcionamiento de los filtros

3.2.4 Listas para los conceptos

Esta tarea está bastante relacionada con el apartado anterior. No obstante, dentro del proyecto se encuentra programada de manera independiente, por lo que

no se ha incluido en el apartado anterior y se va a tratar como si fuera otra funcionalidad.

La característica de esta funcionalidad es la inclusión de una lista de elementos cuyo objetivo es mostrar aquellos conceptos que se están visualizando en cada momento. Un ejemplo del funcionamiento de esta lista se ve en la Ilustración 57.

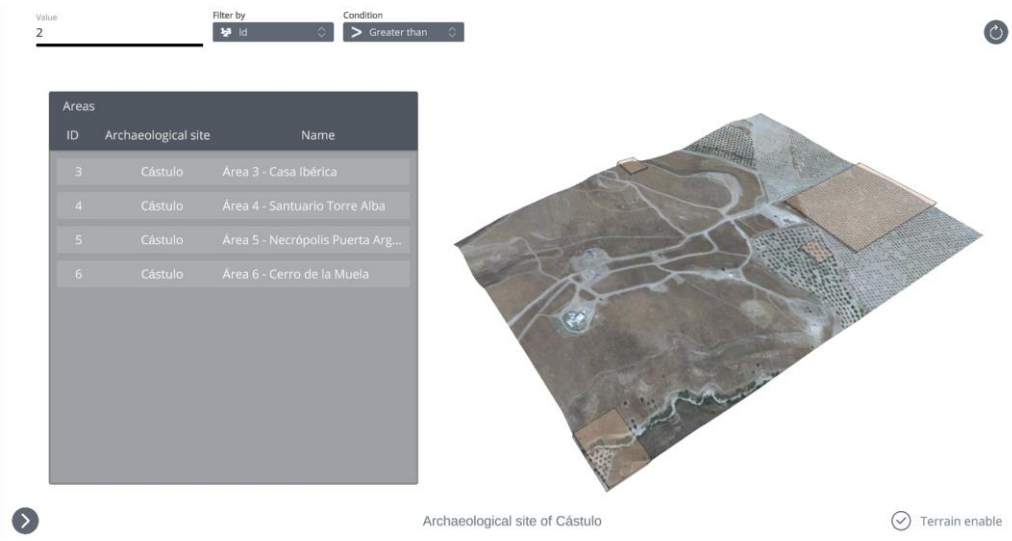


Ilustración 57: Ejemplo del funcionamiento de la lista de conceptos

No obstante, aunque, según la ilustración anterior, parece que las listas funcionan correctamente, este funcionamiento es incorrecto. Como puede verse en la Ilustración 58, la lista sigue manteniendo la información de las áreas y, sin embargo, los polígonos de la derecha hacen referencian a los volúmenes.

Ilustración 58: Ejemplo del mal funcionamiento de la lista de conceptos

Esto lleva a pensar que, en la situación actual del proyecto, no se dispone del cambio entre una lista y otra. El objetivo de esta tarea es dar solución a este error.

El primer paso es comprobar que tenemos creadas de la manera correcta cada una de las listas que en tiempo de ejecución se intercambiarán dependiendo del concepto. En este punto, se aprovecha para crear la lista perteneciente a los volúmenes, ya que no está creada, y, por consiguiente, modificar el diseño inicial para añadir una clase que actúe de controlador para la nueva lista. Esta modificación se muestra en la Ilustración 59.

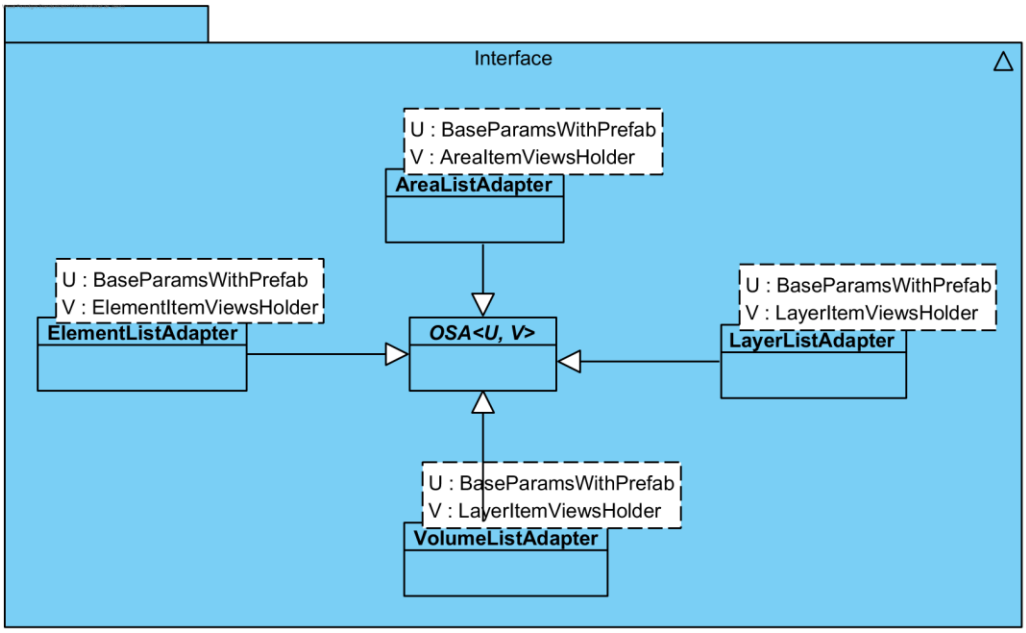


Ilustración 59: Diagrama de clases para añadir el nuevo controlador de la lista

El proceso, al igual que en los casos anteriores, es bastante simple de realizar. Ya tenemos código que funciona bien para mantener y crear la lista de áreas y, por tanto, sólo tiene que ser replicado para añadir la nueva lista creada.

Para dar por finalizada esta tarea únicamente se tiene que cambiar la lista dependiendo de la vista actual. Este comportamiento está bastante completo por lo que solo es necesaria una pequeña modificación para solventar esta tarea.

El resultado final tal y como se tenía previsto que funcionara se muestra en la Ilustración 60.

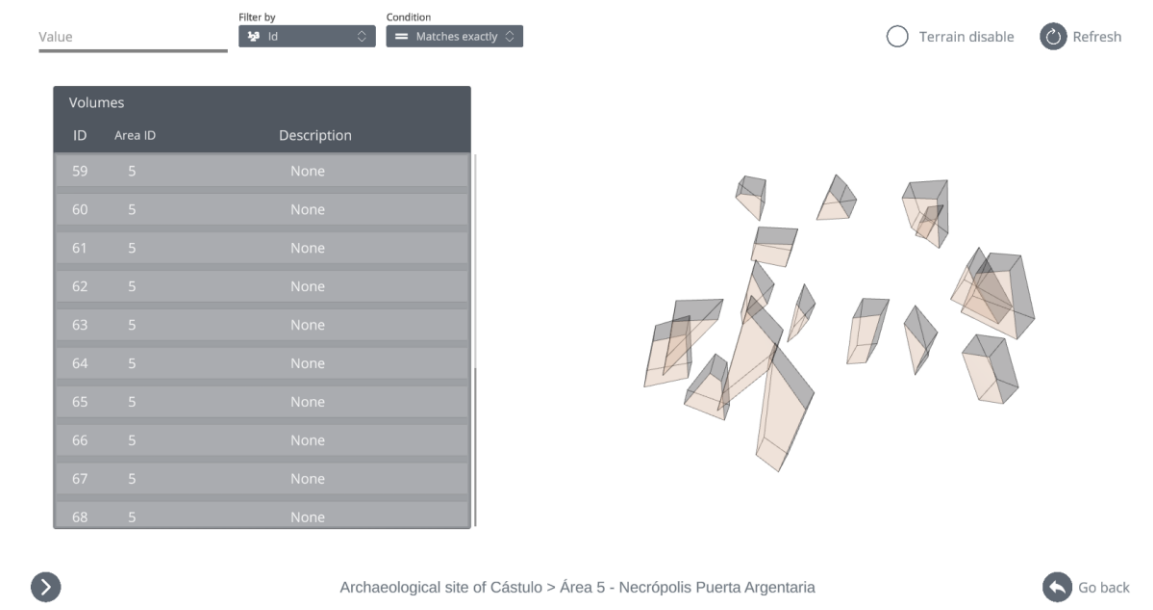


Ilustración 60: Resultado final del estado de las listas

3.2.5 Retroceso en la jerarquía de conceptos

Esta sección se puede considerar como la tarea más importante a desarrollar de entre todas las planteadas. Actualmente, el proyecto está pensado para descender en la jerarquía de conceptos hasta llegar a la última vista en la cual se muestra una capa con todos los elementos encontrados en ella.

El mayor inconveniente que este proceso genera es el hecho de que es necesario volver a ejecutar el proyecto desde el principio en caso de querer seleccionar otro concepto, con el tiempo de espera que esto conlleva.

Aunque normalmente el arqueólogo que utilice la aplicación únicamente estará centrado en una capa, pueden darse diferentes motivos que lo lleven a elegir otra capa distinta, como pueden ser equivocarse al seleccionar un concepto o terminar el trabajo en la capa actual y tener la necesidad de trabajar en otra capa.

Y, por tanto, se pretende implementar un comportamiento de retroceso en la jerarquía de los conceptos. El comportamiento estará inspirado en el comportamiento ya programado para descender en la jerarquía, pero se programará para retroceder por ella. Para ello es necesario reprogramar el comportamiento actual para que trabaje en ambos sentidos.

Para el diseño inicial se plantea utilizar una pila como estructura de datos que nos permita almacenar información sobre el concepto anterior. Las pilas son listas ordenadas que nos permiten almacenar y recuperar datos siendo el modo de acceso

de tipo LIFO (Last In, First Out) por lo que en cada momento solamente se tiene acceso a la parte superior de la lista.

Esta estructura de datos es la que mejor se adapta a la solución a implementar. Y en nuestro caso vamos a utilizar una pila para almacenar el concepto anterior antes de descender sobre la jerarquía. Al almacenar el concepto anterior, tenemos acceso al id del mismo y por tanto podemos restaurar la vista anterior mostrando aquellos conceptos que estén relacionados con el concepto en la parte superior de la pila. La Ilustración 61 muestra el planteamiento teórico de esta solución.

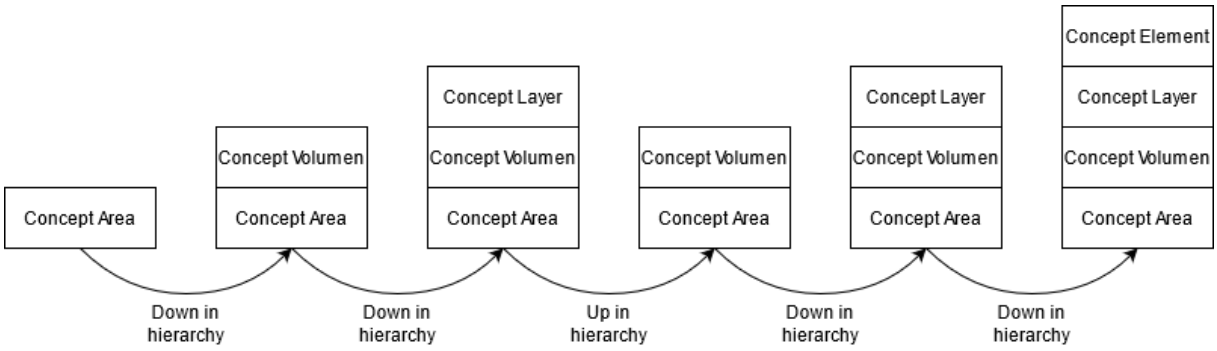


Ilustración 61: Utilización de una pila como estructura de datos

Por otro lado, el movimiento entre vistas está totalmente controlado y en cada momento conocemos en qué vista nos encontramos. Y dada la jerarquía en las vistas impuesta por el proyecto, conocemos en todo momento cuál es la vista anterior y la vista posterior.

Para indicar al usuario que dispone de tal funcionalidad, también es necesario adaptar la interfaz gráfica a un nuevo diseño, mostrado en la Ilustración 62, en la que se han resaltado en color azul los cambios más representativos.

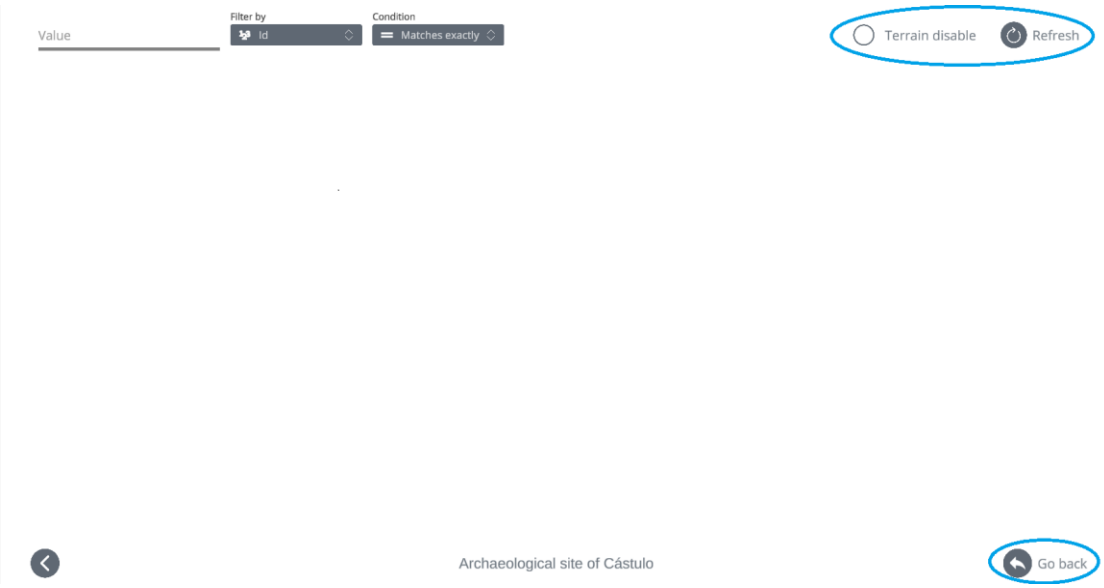


Ilustración 62: Diseño de la nueva interfaz gráfica

El nuevo diseño varia poco con respecto al diseño original, pero se adapta a la implementación de esta tarea al mismo tiempo que prepara la interfaz gráfica para la adición de futuras características, siempre que se vea conveniente.

3.2.6 Control y movimiento de la cámara

En este punto vamos a tratar y mejorar el control de la cámara centrándonos en los controles necesarios para los distintos movimientos que nos permiten mover la cámara libremente, así como los ajustes de velocidad y sensibilidad de la misma.

En primer lugar, se indican los movimientos permitidos son los siguientes:

- **Zoom**: modificación de la distancia focal lo que provoca un acercamiento o un alejamiento hacia el punto al que está orientada la cámara, sin cambiar la posición de la misma.
- **Pan**: rotación de la cámara respecto a su vertical, es decir: la cámara se orienta hacia la derecha o hacia la izquierda, sin desplazarse.
- **Tilt**: rotación de la cámara respecto a su eje X, es decir: la cámara se orienta hacia arriba o hacia abajo, sin desplazarse.
- **Dolly**: modificación de la coordenada X de la posición de la cámara.
- **Truck**: modificación de la coordenada Z de la posición de la cámara.
- **Pedestal**: modificación de la coordenada Y de la posición de la cámara.

En la Ilustración 63 puede verse un resumen de los mismos.

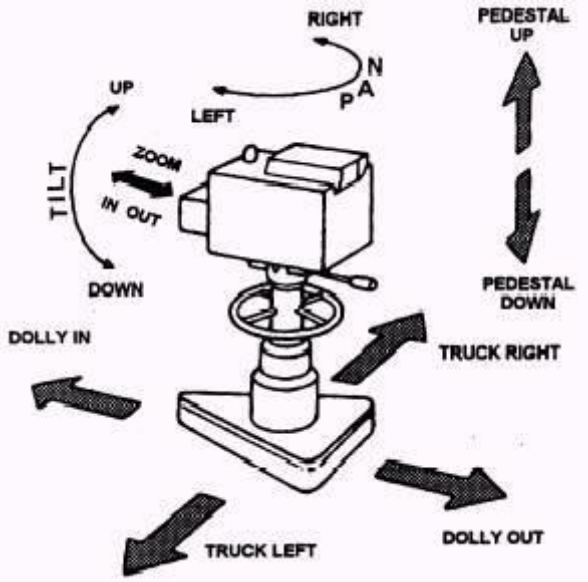


Ilustración 63: Movimientos de la cámara
(https://docsjo.info/2016/10/14/movimientos-de-camara/)

Todos ellos están implementados gracias a un asset descargable desde la tienda de Unity, con los controles y velocidad que se definieron en un principio. Por defecto se utiliza tanto el teclado como el ratón para realizar dichos movimientos, llegando a ser necesario el uso de ambos al mismo tiempo para realizar alguno.

Para facilitar al usuario el movimiento de cámara, se ha llevado a cabo el cambio de controles, simplificando su uso únicamente al teclado o al ratón, sin la necesidad de utilizar ambos periféricos al mismo tiempo. Las teclas que proporcionan los nuevos controles estarán descritas en el apartado 6.2 (Manuales de usuario).

Después del cambio de controles, es hora de cambiar la sensibilidad de los movimientos. El objetivo del cambio es mejorar la precisión que se obtiene del movimiento en los últimos niveles de la jerarquía de conceptos.

En nuestro caso, utilizamos un área de visualización bastante grande, necesaria para dar cabida a la visualización de las áreas del yacimiento que se muestran en ella. Cuando descendemos sobre los conceptos, estos son cada vez más pequeños por lo que el control de movimientos se vuelve un poco más complicado en cada nivel.

Para dar solución a este problema se ha propuesto un cambio de sensibilidad que se ajuste a la vista actual. Los valores exactos se han obtenido mediante la técnica

de prueba y error hasta conseguir unos valores que se consideren correctos según la percepción del desarrollador. Estos nuevos valores muestran un movimiento fluido que se mantiene constante en cada una de las vistas.

Para terminar, se va a implementar un comportamiento que mueva la cámara desde la posición inicial a una posición fija y preestablecida al descender por la jerarquía de conceptos.

El comportamiento inicial de la cámara se puede ver en la Ilustración 64. En ella podemos ver que al descender un nivel en la jerarquía conceptual (en el caso de la figura, de áreas a volúmenes), la cámara permanece en la posición en la que se encontraba antes de ese descenso, lo que podía hacer necesario que el usuario tuviera que hacer un largo desplazamiento de cámara para ver correctamente la información seleccionada. Esto dificulta la selección del siguiente concepto y su visualización.

Ilustración 64: Vista lejana de los conceptos

Para seguir facilitando el control al usuario, se ha decidido implementar dicho comportamiento entendiendo que, una vez que el usuario decide descender hacia un concepto, es porque se quiere centrar en el mismo para realizar ciertas tareas.

La implementación de este comportamiento nos permite mover la cámara hacia un punto en concreto. Podemos reutilizarlo para añadir un botón a la interfaz gráfica

que permita posicionar la cámara en su posición inicial, puesto que el usuario tiene libertad absoluta en cuanto al movimiento.

La estructura de datos más adecuada para este comportamiento también es una pila. En ella se almacenarán las posiciones de la cámara conforme se desciende en la jerarquía de conceptos, de manera que podemos recuperar su estado inicial, tanto si subimos en la jerarquía como si nos mantenemos en ella.

3.2.7 Generación procedural de la base de datos

La base de datos se ha construido de manera procedural a partir de información real de la base de datos utilizada por el equipo que trabaja en el sitio arqueológico de Cástulo.

En el estado inicial del proyecto no se consiguió integrar la generación procedural en el propio editor de Unity de manera que facilitara el desarrollo al autor del trabajo. Como solución se propuso la creación de un proyecto independiente que mantuviera la conexión con la base de datos para dotarla de información.

El proyecto independiente funciona sin problemas y se encuentra disponible para su acceso. No obstante, se ha decidido integrar ambos proyectos, tal y como se pretendía en el trabajo anterior. Aunque no se considera una tarea importante y no se le ha dedicado mucho tiempo.

El código está basado en el código fuente disponible, pero adaptado al trabajo en el editor, el cual merece algunas explicaciones:

- Se ha trabajado con corrutinas en el editor para permitir que el comportamiento suceda cada cierto tiempo. De esta manera simulamos el descubrimiento de restos arqueológicos a través del tiempo. Además, es necesario el uso de corrutinas para ejecutar código sin necesidad de ejecutar el proyecto.

- Para las opciones de configuración de la generación procedural se han utilizado lo que Unity denomina **Propiedades** (ver Ilustración 65), que no son más que ajustes que pueden ser visualizados y editados desde las ventanas del editor de Unity.

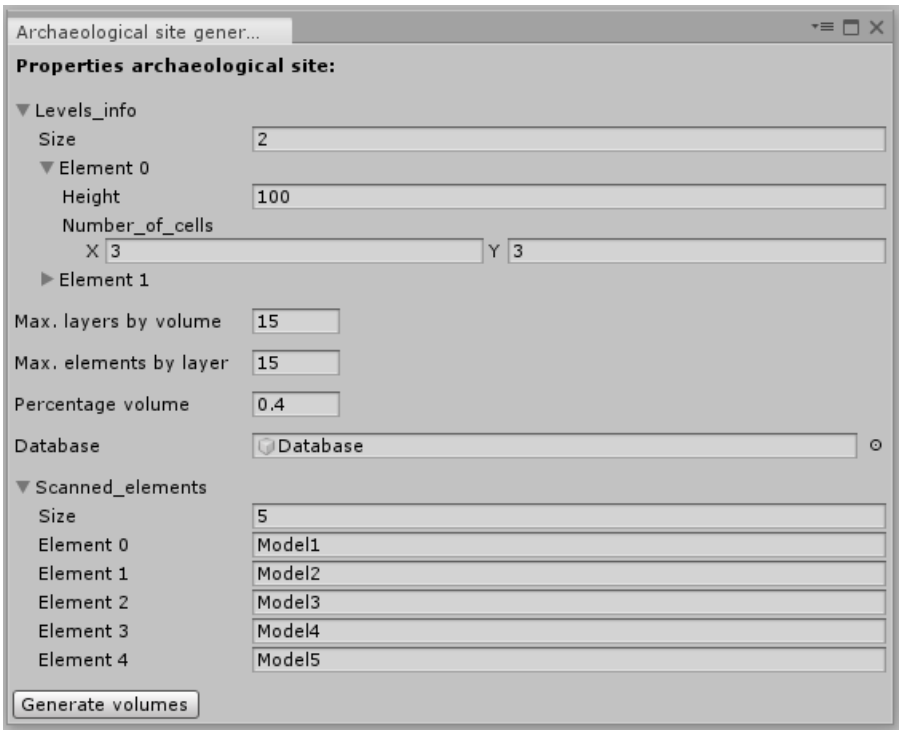


Ilustración 65: Configuración de la generación del terreno

A continuación, se van a explicar cada una de las opciones de configuración y su relación dentro de la generación procedural.

- **Levels_info**: array de estructuras propias que se utilizan para almacenar información sobre la altura total de los volúmenes, así como su disposición en un área.
- **Max. layers by volume**: número máximo de capas en un volumen. Su valor por defecto es 1.
- **Max. elements by layer**: número máximo de elementos en una capa. Su valor por defecto es 1
- **Percentage_volume**: probabilidad de que en cada elemento del array se genere un volumen. Su valor por defecto es 0,4.
- **Database**: objeto que contiene los scripts para la conexión con la base de datos.
- **Scanned_elements**: array de cadenas de texto para referenciar al nombre del modelo escaneado.

La primera propiedad nos permite dividir cada una de las áreas en niveles que se dividen a su vez en varias zonas, denominadas **celdas**. Cada celda puede contener un volumen (según la probabilidad) contenido dentro de los límites del área y con la altura especificada.

Dentro de todos los volúmenes se generan las capas. Teniendo en cuenta que se pueden generar desde una capa hasta el máximo indicado. La altura de las mismas se calcula a partir de la altura del volumen y el número de capas del mismo.

Por último, para cada capa se generan un número aleatorio de elementos (sin exceder del máximo). Al conocer los límites de las capas, no tenemos problema en colocar el elemento en una posición aleatoria de la misma. El modelo 3D asociado al elemento viene dado también de manera aleatoria por el array de nombres.

3.2.8 Cerrar aplicación

Por último, mencionar la manera de abandonar la aplicación. Hasta ahora, no existe manera de abandonarla una vez que la aplicación ha sido exportada y está siendo ejecutada.

Para solventar esto se propone dar la posibilidad al usuario de abandonar la ejecución con la pulsación de una tecla. Para que la finalización no sea extremadamente brusca y no se haya producido por error del usuario, se implementará una ventana modal para confirmar la salida de la aplicación.

Su diseño inicial y final pueden verse en la Ilustración 66 y en la Ilustración 67 respectivamente.

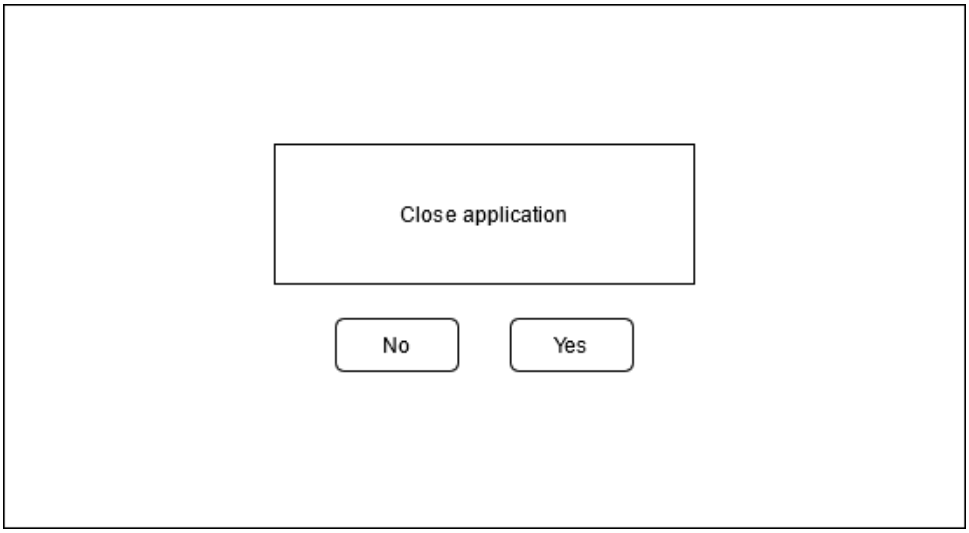


Ilustración 66: Diseño inicial de la ventana modal del cierre de la aplicación

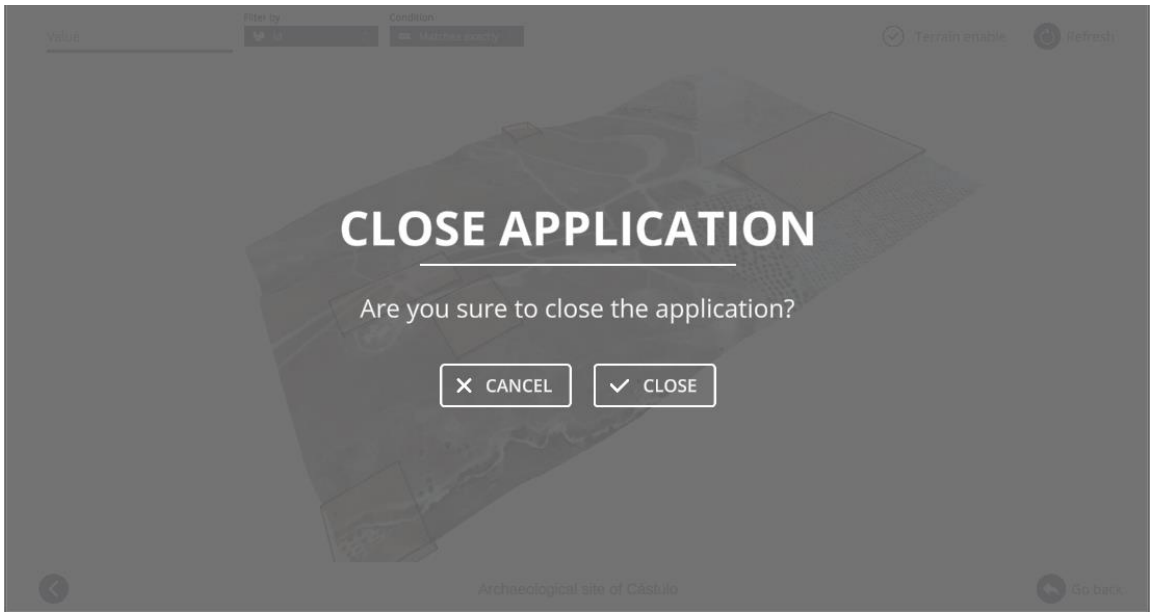


Ilustración 67: Diseño final de la ventana modal del cierre de la aplicación

3.3 Iteración 2

Esta iteración recoge la implementación de gizmos en tiempo de ejecución. Estos gizmos deben permitir realizar rotaciones y traslaciones de los elementos, de tal forma que se pueda reconstruir el estado original del yacimiento del que se extrajeron.

Para facilitar la recolocación se hará uso de imágenes. La idea es fotografiar al elemento en su posición original y utilizar estas imágenes como guía para la recolocación virtual.

Este desarrollo abarca del día 07/12/2020 al día 09/02/2021.

3.3.1 Gizmos

El concepto principal durante esta iteración es el gizmo, por lo que es conveniente dejar clara una definición desde el principio. Un **gizmo** es un elemento visual que se coloca en la vista de escena para dar información adicional de los objetos presente en ella (Manual Unity, s. f.-b). Los gizmos se encuentran muy presentes en Unity (Ilustración 68), sobre todo en el modo de edición de la escena. No obstante, Unity pone a disposición de los desarrolladores herramientas que posibilitan el trabajo con gizmos en tiempo de ejecución mediante código.

En este trabajo el objetivo consiste en simular los gizmos que Unity proporciona para trabajar en la escena a la hora de trasladar, rotar y escalar los objetos, de manera

que el usuario final también pueda posicionar y rotar los restos arqueológicos para colocarlos en su posición original.

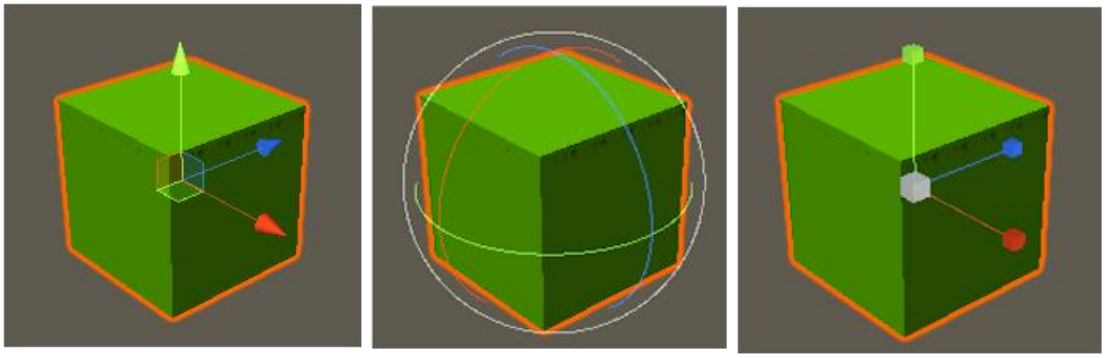


Ilustración 68: Gizmos de Unity

Debido a la escasa experiencia por parte del autor en la utilización de gizmos, se ha procedido a buscar software creado por desarrolladores externos que nos faciliten el trabajo a realizar.

Los assets que se han tomado en cuenta son los siguientes:

- Unity3DRuntimeTransformGizmo desarrollado por el usuario HiddenMonk y disponible en GitHub[22]. Este complemento es ligero y fácil de utilizar.
- RuntimeTransformGizmos desarrollado por el usuario Octamodius y disponible en la Unity Store. Es mucho más completo, incluyendo otras funcionalidades. Está integrado en el proyecto, aunque no en su totalidad.

El primero se puede utilizar de una manera sencilla para implementar el comportamiento que vamos buscando. La integración con el proyecto es fácil al igual que su utilización en tiempo de ejecución, ya que tan solo es necesario añadir un script a la escena, más concretamente al objeto que contiene la cámara. No obstante, la manera en la que está desarrollado nos impide intercambiar fácilmente el objeto sobre el que aplicar los gizmos, lo que nos lleva a desistir de su uso.

El segundo asset es mucho más completo que el primero y partimos de la ventaja de que ya está incorporado en el proyecto. El uso e integración de este

[22] https://github.com/HiddenMonk/Unity3DRuntimeTransformGizmo

complemento también se considera fácil y en este caso, debido a su implementación, se puede tener un control absoluto sobre qué gizmo se está utilizando en cada momento por lo que sí se nos permite el intercambio entre diferentes gizmos al mismo tiempo que podemos aplicarlo a múltiples objetos. No obstante, el uso de este asset lleva consigo el inconveniente de que no es compatible con el paquete de Unity LWRP (Lightweight Render Pipeline), que está siendo utilizado en el proyecto para mejorar la calidad gráfica del mismo y que debe ser eliminado dada su incompatibilidad. La eliminación de este paquete lleva pérdidas en algunas de las texturas que se utilizan en el proyecto. Esto se debe a que se pierden las referencias a los shaders que se utilizan para visualizar las texturas. Además, se pierden las animaciones que se utilizan para filtrar los conceptos ya que las mismas hacen uso de estos shaders.

El asset que utilizaremos para implementar el comportamiento es el último citado ya que está incluido en el proyecto y dispone del código necesario para implementar el comportamiento que se desea de una manera fácil y simple, aunque perdamos parte de las texturas y animaciones que en tareas posteriores deben solucionarse.

3.3.2 Pipeline de rendering

El pipeline de rendering, conocido en español como tubería de renderizado, es la secuencia de pasos que un programa lleva a cabo para convertir una escena 3D en una imagen 2D. OpenGL y Direct3D son dos estándares gráficos que proporcionan un pipeline y además permiten la programación de shaders para modificar su comportamiento.

Unity implementa el renderizado a partir de shaders que definen cómo se ve un objeto (sus propiedades materiales) y cómo reacciona a la luz. Por defecto, Unity utiliza el pipeline más básico de entre todos los que dispone, pero se puede configurar para utilizar cualquier otro pipeline, incluso personalizados. Es más, Unity posee dos pipelines personalizados (Manual Unity, s. f.-a):

- **LWRP (Lightweight Render Pipeline)**: Esta tecnología ofrece gráficos escalables a plataformas móviles, consolas y ordenadores de gama alta. Se puede lograr un renderizado rápido con una alta calidad sin necesidad de utilizar mucha tecnología. LWRP utiliza materiales e

iluminación simplificada y optimiza el rendimiento en tiempo real en varias plataformas.

- **HDRP (High Definition Render Pipeline):** Creado para plataformas modernas. HDRP utiliza técnicas de iluminación basadas en la física, iluminación lineal e iluminación HDR. De manera resumida, se ofrecen las herramientas necesarias para crear aplicaciones con un alto estándar gráfico.

Al dejar de utilizar LWRP por la inclusión de los gizmos, se han perdido las texturas que se estaban utilizando (Ilustración 69). En este punto se ha hecho necesario utilizar los shaders básicos que Unity pone a disposición del desarrollador. Puesto que es necesario cambiar las animaciones, los shaders básicos que se van a utilizar son shaders que permitan la transparencia.

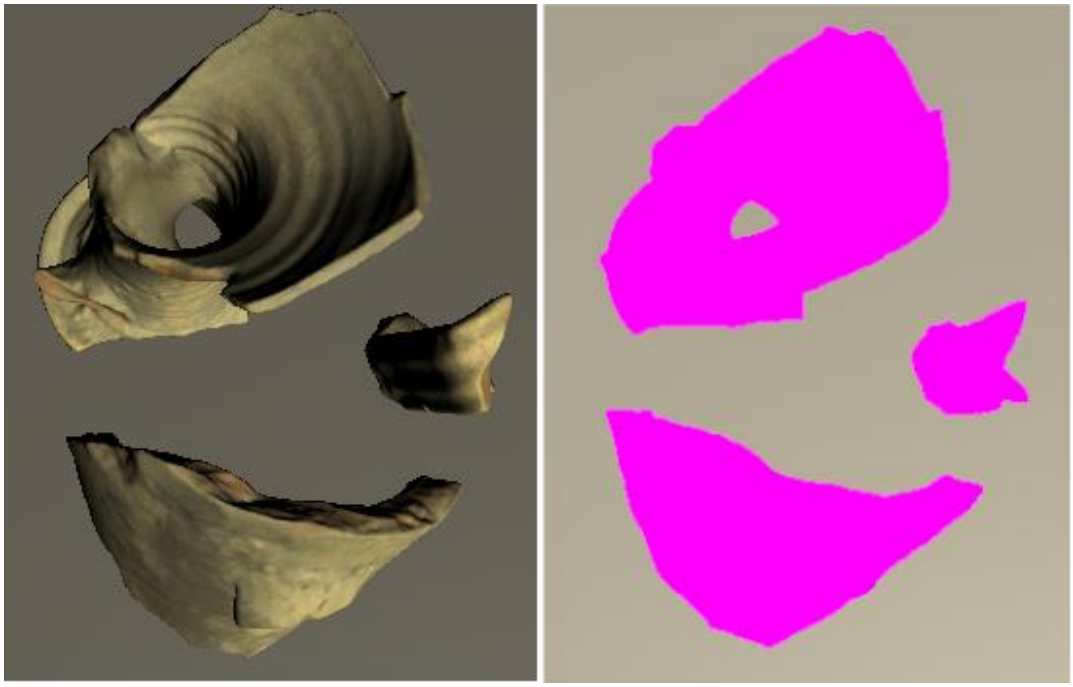


Ilustración 69: Pérdida de las texturas

Una vez renderizadas las texturas es necesario cambiar las animaciones que se activan cuando aparecen o desaparecen los conceptos, ya sea por la bajada o subida en la jerarquía, o por el uso de los filtros.

Se han desechado todas las animaciones ya existentes puesto que estaban pensadas para ser compatibles con los shaders del LWRP. Las nuevas animaciones

son mucho más simples que las anteriores, siendo necesario únicamente un cambio en el canal de transparencia del color asociado al material.

Ya que se están cambiando las animaciones, se aprovecha para añadir una nueva animación. Esta tiene como objetivo recalcar las áreas, los volúmenes o las capas cuando se pulsa sobre ellas en la lista de conceptos (Ilustración 70). Con esta funcionalidad se propone una búsqueda rápida con únicamente hacer un clic sobre su información en la lista.

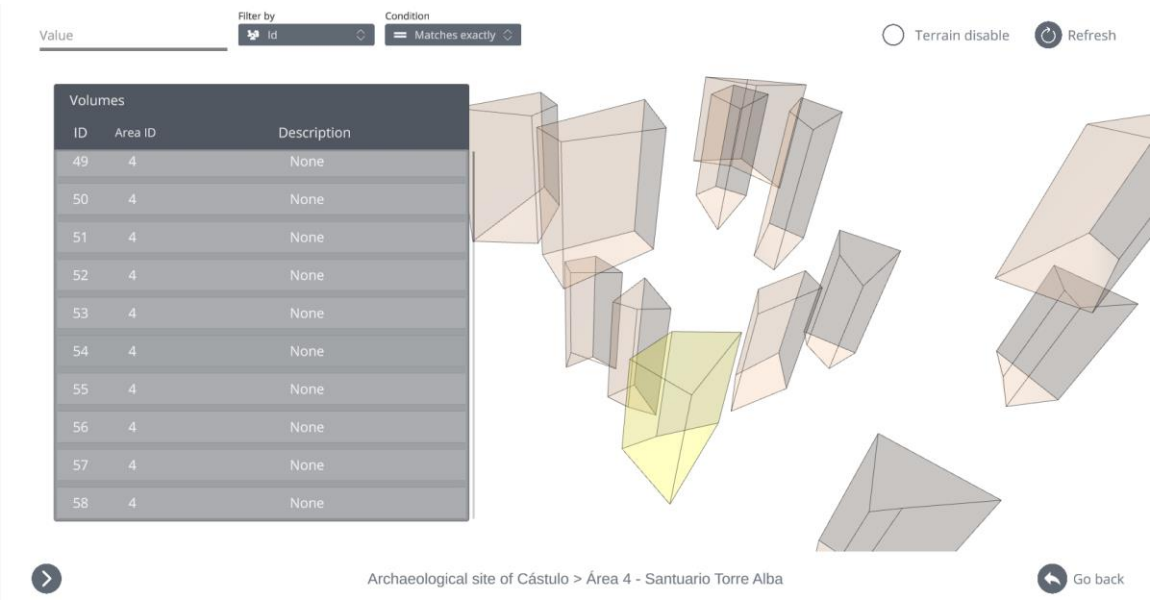


Ilustración 70: Recalcado de un concepto

Por último, se han añadido dos shaders nuevos que añaden el comportamiento perdido por la eliminación de LWRP en dos objetos de Unity. Ambos shaders están basados en shaders escritos por desarrolladores ajenos al proyecto, pero modificados y adaptados al mismo.

Uno de ellos pertenece al terreno. El comportamiento del shader consiste en dibujar lo que se considera la parte posterior del polígono de forma que el objeto no se vea transparente por dicha parte. La diferencia entre usar y no usar el shader se muestra en la Ilustración 71.

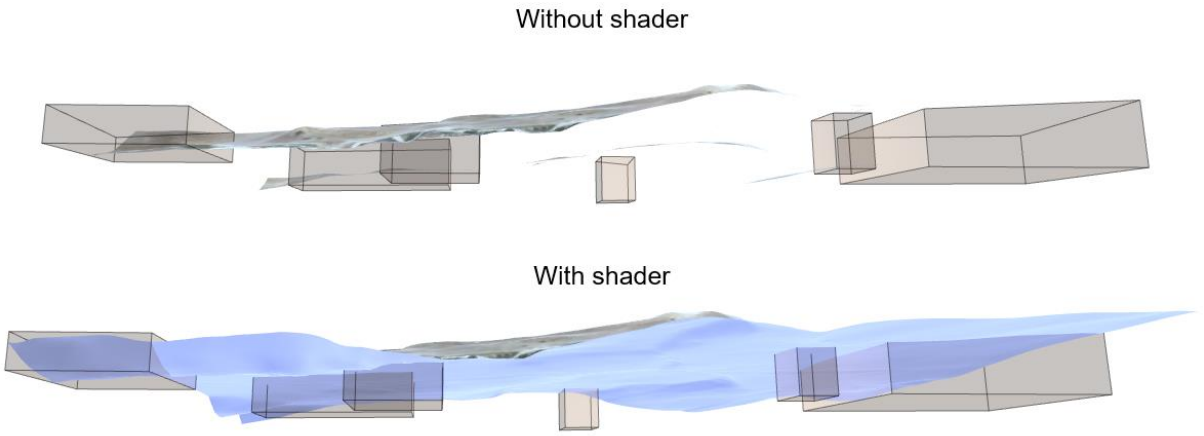


Ilustración 71: Parte posterior del terreno sin shader y con shader

El otro pertenece a la estatua que se muestra en la pantalla del inicio. Esta estatua tenía la peculiaridad de que se disolvía al realizar la carga de la aplicación tal y como se muestra en la Ilustración 72.

Ilustración 72: Carga del modelo de la estatua en la versión original

Para volver a simular este comportamiento es necesario utilizar un shader específico. Este shader hace uso de una textura en escala de grises que hace la función de máscara. El shader disuelve el objeto al completo, pero la máscara indica qué partes del mismo se disuelven antes, de tal manera que la disolución del objeto se puede modificar al cambiar de máscara.

3.3.3 Manipulación

Para permitir la manipulación de los elementos, se va a hacer uso de una clase controladora cuyo diagrama de clases se muestra en la Ilustración 73.

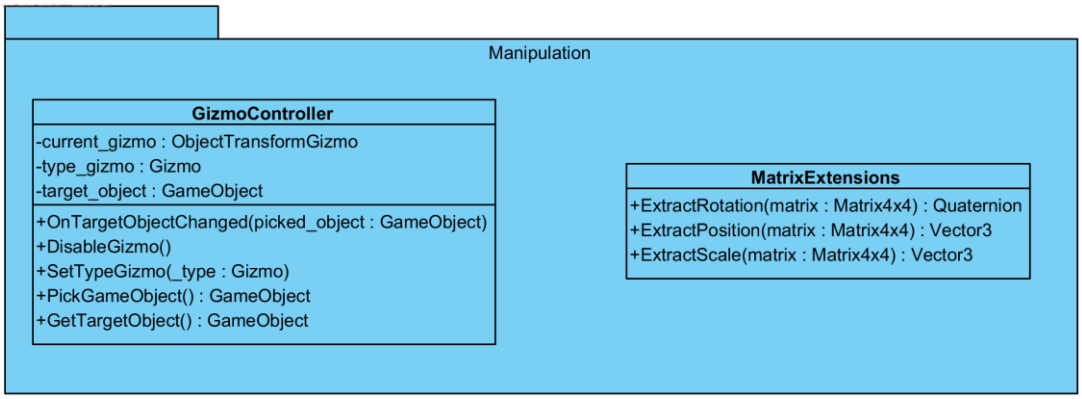


Ilustración 73: Diagrama de clases de GizmoController

Esta clase nos permite controlar qué gizmo se está utilizando en cada momento y a qué elemento pertenece. Por cada cambio tanto en el gizmo a utilizar como en el elemento, es necesario eliminar el gizmo utilizado y crear uno nuevo. El objetivo de este script es detectar estos cambios y generar el nuevo gizmo correspondiente.

Por otro lado, el controlador nos permite intercambiar entre gizmos que nos permitan manipular la posición, la rotación y la escala. Aunque el proceso está preparado para ser totalmente funcional con la modificación de la escala, esta ha sido deshabilitada, ya que consideramos que viene perfectamente definida en los modelos obtenidos y que se corresponde con la escala real del elemento en el yacimiento arqueológico.

Aclarar que para que los gizmos estén activos y visibles, antes debe seleccionarse el elemento a recolocar. Un ejemplo de su utilización se puede ver en la Ilustración 74.

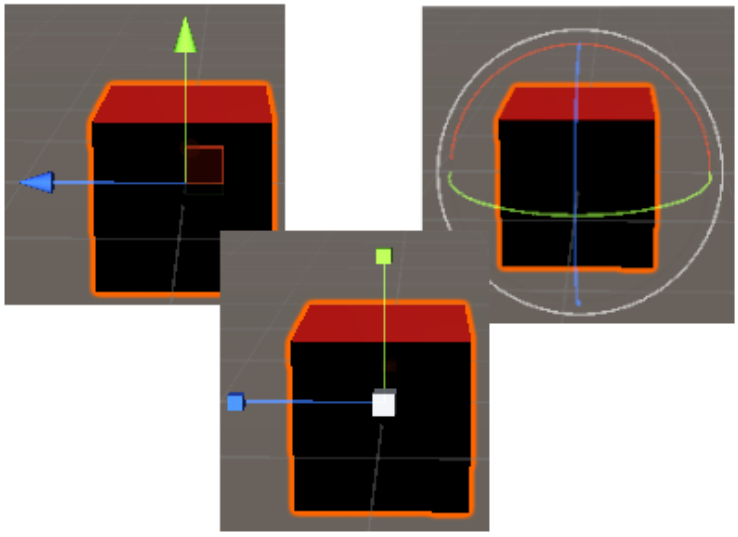


Ilustración 74: Gizmos desarrollados

Al terminar la manipulación se da la posibilidad de almacenar la nueva posición y rotación en la base de datos.

Para almacenar el cambio realizado por el usuario durante la ejecución de la aplicación, se va a hacer uso de una matriz de transformación en la que se almacena la traslación y rotación realizada. Unity posee herramientas que nos permiten crear este tipo de matrices, pero no posee las herramientas que nos permiten obtener directamente la información desde este tipo de matrices. Es decir, a partir de una posición y los ángulos de rotación podemos obtener una matriz de transformación, pero no podemos obtener dicha posición y ángulos a partir de una matriz de transformación. Esto último debe ser implementado por el autor.

Al trabajar con Unity, podemos abstraer la descomposición de la matriz a unas cuantas llamadas de los métodos que Unity ofrece.

Siendo m una matriz de transformación como la siguiente:

$$m = \begin{bmatrix} m00 & m01 & m02 & m03 \\ m10 & m11 & m12 & m13 \\ m20 & m21 & m22 & m23 \\ m30 & m31 & m32 & m33 \end{bmatrix}$$

La traslación se extrae a partir de la cuarta columna, formada por el vector:

$$t = [m03 \quad m13 \quad m23]$$

En cuanto a la rotación, es necesario disponer de dos vectores que indiquen la dirección hacia delante y hacia arriba. Estos se consiguen a partir de la tercera y segunda columna, respectivamente:

$$r1 = [m02 \quad m12 \quad m22]$$
$$r2 = [m01 \quad m11 \quad m21]$$

Por último, el escalado viene definido para cada eje, siendo la escala en el eje X la primera columna, la escala en el eje Y la segunda columna y la escala en el eje Z la tercera columna. Para todas ellas es necesario obtener el módulo del vector resultante.

$$sx = \|[m00 \quad m10 \quad m20 \quad m30]\|$$
$$sy = \|[m01 \quad m11 \quad m21 \quad m31]\|$$
$$sx = \|[m02 \quad m12 \quad m22 \quad m32]\|$$

De esta manera, podemos aplicar la traslación y rotación extraída de la matriz de transformación al modelo en Unity, cambiando la posición y rotación originales del modelo.

Todo esto afecta a la matriz de transformación de los elementos en Unity. Para que su uso sea permanente debe almacenarse en la base de datos. Para ello se va a modificar la tabla **Element** para que albergue un nuevo campo al que vamos a llamar **trs_matrix**.

El diseño de la nueva tabla se puede ver en la Ilustración 75.

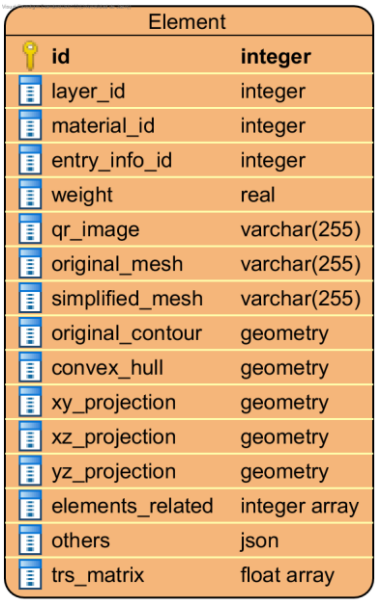


Ilustración 75: Nueva tabla para los elementos

Aunque la matriz de transformación se corresponde con un array de dos dimensiones, la manera de almacenarlo en la base de datos va a ser con un array de

una dimensión. PostGIS admite tipos de datos en forma de matrices, pero su acceso se complica en comparación con el uso de arrays. Por tanto, para la facilidad por parte del autor, mediante el uso de la API se pretende convertir la matriz de transformación 4x4 en un array de 16 posiciones tal y como muestra la Ilustración 76.

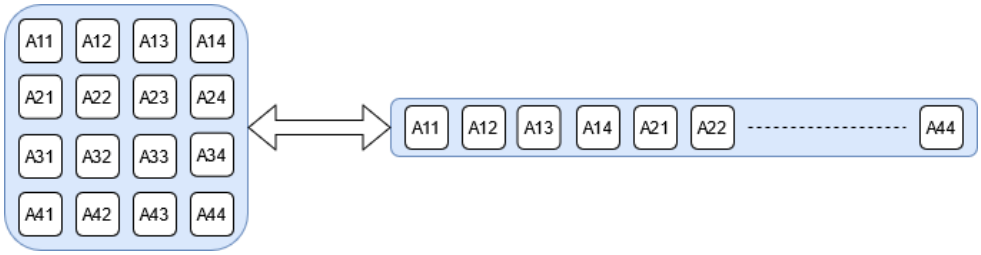


Ilustración 76: Conversión matriz-array

Para ello es necesario modificar únicamente el método encargado de guardar los cambios realizados en los elementos, para que también guarde la matriz de transformación. Por otro lado, también es necesario modificar el método encargado de leer la información de la API para modificar la posición y rotación local de los objetos en Unity, según indique la matriz almacenada en la base de datos.

3.3.4 Imágenes de referencia

Para ayudar en la recolocación de los elementos en sus posiciones originales, se puede hacer uso de imágenes que se utilicen como guía.

En principio, bastaría con utilizar tres imágenes ortogonales entre ellas tal y como muestra la Ilustración 77. Una por cada eje de coordenadas, que equivaldría a realizar tomas desde las vistas de planta, alzado y perfil, de tal forma que se pueda observar el mismo elemento desde todos los ángulos posibles. No obstante, no se va a limitar la cantidad de imágenes.

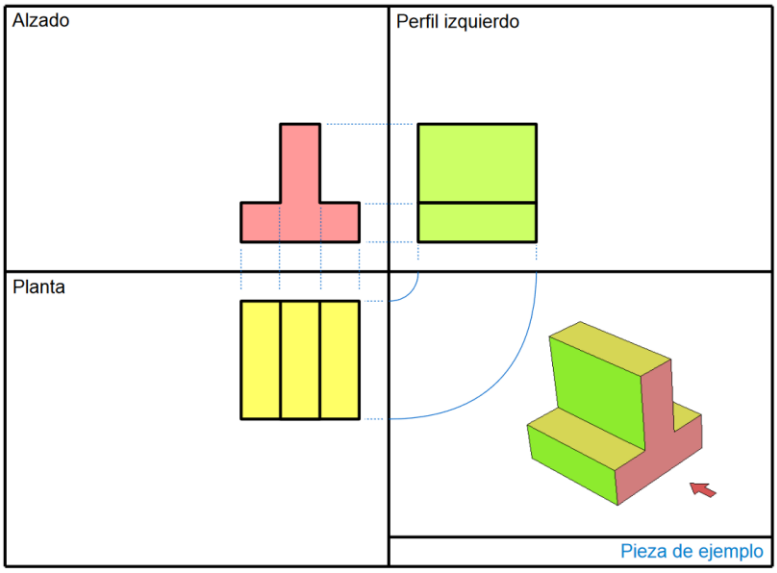


Ilustración 77: Imágenes ortogonales (planta, alzado y perfil)
(https://studylib.es/doc/5318909/alzado-planta-perfil---gobierno-de-canarias)

La base de datos está preparada para almacenar la URL del almacenamiento externo de cada imagen. Pero a esta información es necesario añadir información del plano donde se incluye la imagen, es decir:

- **direction**: dirección normal del plano. Se utiliza para calcular la rotación del mismo.
- **distance**: distancia hacia el objeto. Se utiliza para posicionarlo en el espacio 3D.

De esta manera el nuevo esquema para la tabla **Image_Element** queda representado en la Ilustración 78.

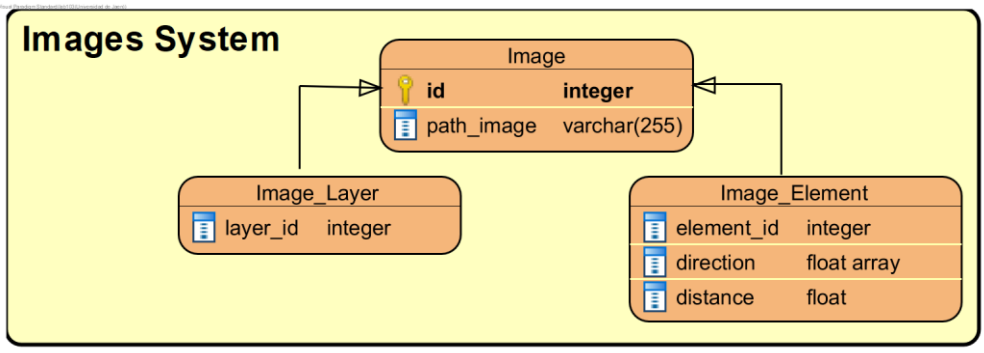


Ilustración 78: Nueva tabla Image_Element

En cuanto a la API, es necesario modificar la lectura de la base de datos para que obtenga estos nuevos campos, por lo que también es necesario modificar los accesos en Unity para que sea capaz de recibir esta información y tratarla de la manera pertinente.

3.3.5 Relaciones topológicas

Actualmente, las relaciones topológicas están siendo simuladas mediante código, eligiendo a conciencia un elemento con un id definido y añadiéndole a mano dichas relaciones. El objetivo de esta tarea radica en eliminar esta simulación y obtener las relaciones topológicas desde la base de datos, puesto que la tabla Element guarda información de las mismas.

En la base de datos se almacenan, para cada elemento, una lista con los elementos con los que se relaciona. Esto provoca que existan relaciones dobles. Esta duplicidad es necesario tenerla en cuenta.

En primer lugar, se va a diseñar una clase que sirva como controladora de las relaciones topológicas que se vayan leyendo desde la base de datos. Su diagrama puede verse en la Ilustración 79.

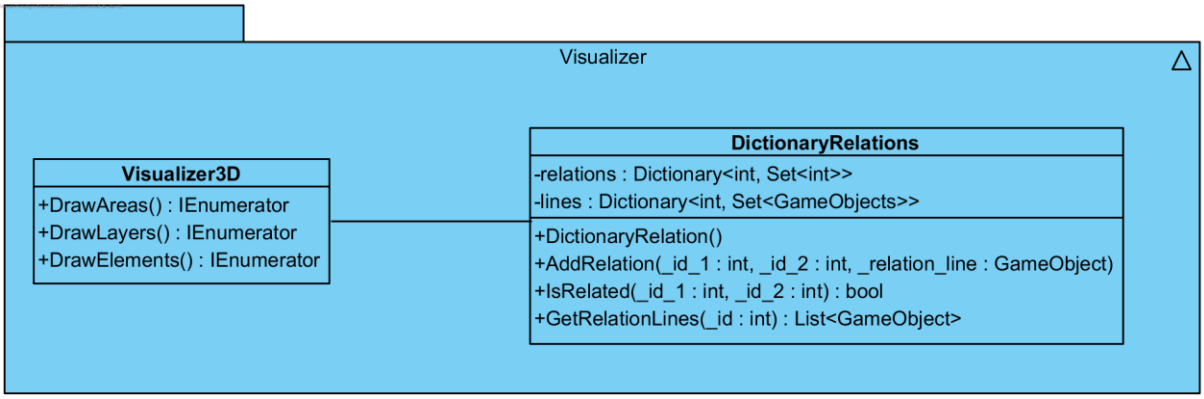


Ilustración 79: Diagrama de clases para DictionaryRelation

La clase descrita anteriormente contiene dos diccionarios, utilizados para almacenar las relaciones topológicas y los objetos asociados a ellas. Ambos diccionarios mantienen la misma estructura, por lo que solo se considera necesario explicar uno de ellos. Como clave se mantiene el id del elemento y como valor se mantiene una lista con los id de los elementos que se relaciona, tal y como podemos

extraer de la base de datos. De esta manera la continua búsqueda de relaciones se realiza de una manera más eficiente.

Con todo esto preparado, la aplicación es capaz de leer desde la base de datos todas las relaciones topológicas existentes y dibujarlas en Unity para que puedan ser visualizadas.

Esto es totalmente independiente del procedimiento para sugerir relaciones basado en el DE-9IM, que se aplica para cada nuevo elemento introducido en la base de datos.

3.3.6 Movimiento de los elementos

Al implementar el movimiento mediante gizmos estamos desplazando únicamente el elemento por la escena pero se mantienen en la posición original tanto las relaciones topológicas que este pudiera tener como las guías que nos muestran su altura en la capa, tal y como podemos ver en la Ilustración 80.

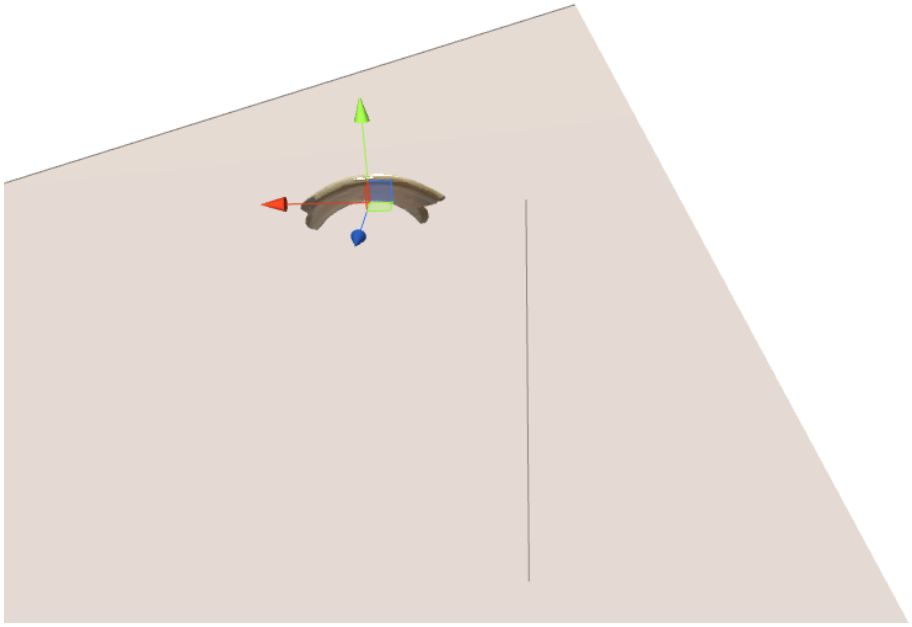


Ilustración 80: Elemento movido mediante gizmos sin mover las demás líneas

Por lo tanto, es necesario detectar el desplazamiento producido por el elemento activo a través del gizmo y comprobar cada cierto tiempo su nueva posición. Para una tarea de este tipo se van a utilizar las corrutinas de Unity. Estas nos permiten detener una parte del código en cualquier momento y continuar con su ejecución pasado un tiempo. De esta forma podemos detectar cada cierto tiempo el desplazamiento llevado

a cabo, y en caso de haberse realizado desplazamiento, desplazar a su vez todas aquellas relaciones o guías que se consideren oportunas.

La granularidad y precisión de la corrutina depende del tiempo establecido. A menor tiempo, antes se detecta movimiento, en caso de que hubiese, y antes se proporciona una respuesta visual al usuario final.

La rotación producida por el gizmo no presenta este inconveniente puesto que se realiza sobre el punto central del objeto y no existe desplazamiento alguno.

3.4 Iteración 3

El objetivo de esta iteración es cargar los elementos en tiempo de ejecución y eliminar los prefabs utilizados en Unity procedentes del proyecto inicial. Permitir la lectura de un fichero OBJ desvincula la aplicación de la simulación obtenida hasta ahora.

Además, se debe incluir la geometría de los modelos en la base de datos para la realización de cálculos futuros sobre ella.

El tiempo que abarca esta iteración transcurren desde el día 10/02/2021 al día 23/02/2021.

3.4.1 Geometría en la base de datos

La primera tarea consiste en la carga de la geometría de los objetos en la base de datos. Como la geometría está asociada a los elementos arqueológicos, la tabla Element va a ser la encargada de almacenar dicha geometría. La modificación de esta tabla consiste en añadirle un campo **geometry**, que como su propio nombre indica, se utiliza para almacenar la geometría. De esta manera disponemos de los siguientes campos en el que almacenar geometría:

- **xy_projection**: proyección de la malla de polígonos del modelo 3D del elemento en el plano “XY".
- **xz_projection**: proyección de la malla de polígonos del modelo 3D del elemento en el plano “XZ".
- **yz_projection**: proyección de la malla de polígonos del modelo 3D del elemento en el plano “YZ".
- **geometry**: malla de polígonos del modelo 3D del elemento.

Para rellenar estos campos con información, se hará uso de la página web. Esta nos permite leer un fichero OBJ e introducirlo en la base de datos. Con algunas modificaciones, podemos conseguir que de esta lectura extraiga tanto las proyecciones 2D como la geometría en 3D.

Los ficheros OBJ son bien conocidos, pero pueden tener una estructura compleja. Sin embargo, solo es necesario un subconjunto de esta información para almacenar la geometría en la base de datos.

La estructura de un fichero de este tipo, de manera muy simplificada, se muestra en la Ilustración 81.

```
v 1.000000 1.000000 -1.000000
v 1.000000 -1.000000 -1.000000
v 1.000000 1.000000 1.000000
v 1.000000 -1.000000 1.000000
vt 0.625000 0.250000
vt 0.875000 0.750000
vt 0.625000 1.000000
vn -1.0000 0.0000 0.0000
vn 0.0000 -1.0000 0.0000
vn 1.0000 0.0000 0.0000
vn 0.0000 0.0000 -1.0000
f 5/1/1 3/2/1 1/3/1
f 3/2/2 8/4/2 4/5/2
f 7/6/3 6/7/3 8/8/3
f 2/9/4 8/10/4 6/11/4
```

Ilustración 81: Estructura de un fichero OBJ

Cada letra inicial de una fila representa diferente información:

- Un vértice se representa mediante la letra **V** al principio de la línea, seguido de las coordenadas x, y, z.
- La coordenada de textura se representa con las letras **VT**, seguido de las coordenadas u, v.
- Las normales se representan con las letras **VN**, seguidas de las coordenadas x, y, z.
- Por último, las caras representadas con la letra **F**. En este caso existen varias maneras de establecer una cara.
    - f v1 v2 v3. Siendo v1, v2 y v3 índices a vértices.

- f v1/vt1 v2/vt2 v3/vt3. Siendo v1, v2 y v3 índices a vértices, y vt1, vt2 y vt3 índices a coordenadas de textura.
- f v1/vt1/vn1 v2/vt2/vn2 v3/vt3/vn3. Siendo v1, v2 y v3 índices a vértices, vt1, vt2 y vt3 índices a coordenadas de textura, y vn1, vn2 y vn3 índices a los vectores normales.

En este caso, tan solo es necesario obtener información de los vértices (representados por la letra V) y las caras (representadas por la letra F).

Con toda esta información leída y almacenada se deben construir las sentencias SQL que nos permitan almacenar la geometría en la base de datos. En PostGIS se permite la inserción de geometría a partir de múltiples funciones preparadas para ello. En nuestro caso se ha hecho uso de dos tipos de funciones:

- ST_MakePolygon, ST_MakeLine, ST_MakePoint: funciones que nos permiten introducir la geometría a partir de sus coordenadas X, Y, Z en el caso de introducir puntos, o a partir de un conjunto de puntos, en el caso de querer introducir un polígono o una línea.
- ST_GeomFromEWKT: función que nos permite introducir cualquier tipo de geometría a partir de su descripción en formato WKT[23] (Well Know Text)

Ambos casos son totalmente válidos, pero no se pueden intercalar entre ellos. Es decir, no se pueden utilizar las funciones ST_Make junto a la función ST_GeomFromEWKT.

Para comprobar el funcionamiento de estos scripts se han introducido pequeños objetos geométricos fáciles de describir con estos métodos, como pueden ser pirámides o cubos. Nótese que se introducen objetos en 3D de los cuales se obtienen las proyecciones 2D correspondientes. El cálculo de las proyecciones no lo realiza PostGIS, sino que el desarrollador debe generar las sentencias SQL que permitan obtener dichas proyecciones.

Un ejemplo del resultado obtenido con la malla de triángulos de un elemento arqueológico real puede verse en la Ilustración 82. Las proyecciones calculadas servirán como entrada al algoritmo basado en el DE-9IM.

[23] http://www.geoapi.org/3.0/javadoc/org/opengis/referencing/doc-files/WKT.html

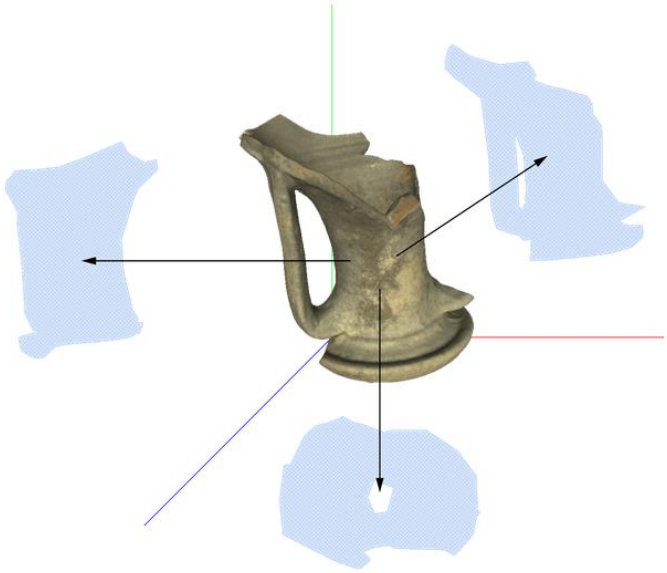


Ilustración 82: Proyecciones en la base de datos

No es posible visualizar el objeto en 3D desde la web o desde la base de datos en PostgreSQL por lo que deben utilizarse herramientas externas. Sin embargo, mediante funciones de PostGIS se puede comprobar que el objeto está almacenado correctamente. Además, las proyecciones que se obtienen de dicho objeto 3D son correctas y pueden ser visualizadas con la herramienta gráfica pgAdmin[24] (herramienta gráfica para PostgreSQL).

A continuación, se procede a introducir en la base de datos la geometría de todos los modelos con los que se está trabajando actualmente en la aplicación. Aunque nos encontramos con el inconveniente de que los modelos son demasiado pesados para ser introducidos en la base de datos, ya que algunos de ellos superan hasta los 50MB de almacenamiento en disco, con decenas de miles de caras.

El problema no reside en la base de datos, ya que esta está preparada para almacenar tanta información como sea necesario, sino que reside en la capacidad de la página web de leer el fichero OBJ y convertirlo en la sentencia SQL que permita introducir los datos en la base de datos. Esto hace que sea inviable trabajar con modelos de estas características en nuestra base de datos y por tanto debe ser necesario una simplificación de los mismos, entendiendo como simplificación, la eliminación de caras siempre y cuando se mantenga la forma y contorno originales de la pieza.

[24] https://www.pgadmin.org/

La solución propuesta es mantener dos copias del mismo objeto (ver Ilustración 83):

- El objeto original con toda la resolución posible, para su uso en la aplicación.
- El objeto simplificado con una resolución menor pero que mantiene la forma original para trabajar con ella en la base de datos y/o en Unity.

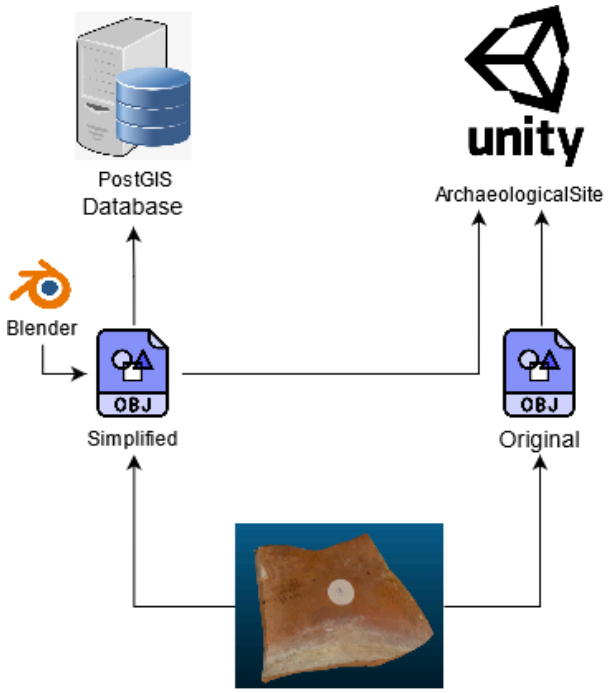


Ilustración 83: Duplicidad de modelos 3D

Para la simplificación de los modelos se ha utilizado la herramienta **Blender**. Este software trabaja con modificadores que pueden ser aplicados a los modelos, siendo uno de ellos el encargado de reducir la geometría.

Aunque la reducción final del número de caras y vértices depende del modelo original, trabajar con un máximo de 2000-3000 triángulos por modelo ha sido más que suficiente para mantener el contorno original y no sobrecargar la lectura por parte del servicio web.

En la Tabla 16 se muestra el resultado obtenido tras aplicar la reducción a un conjunto de modelos. Las imágenes se muestran sin textura, puesto que así se aprecia mejor la simplificación.

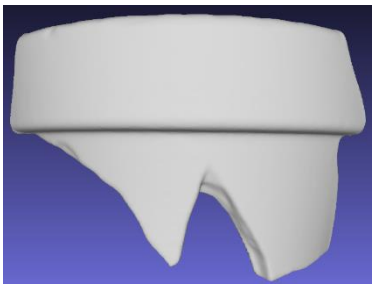
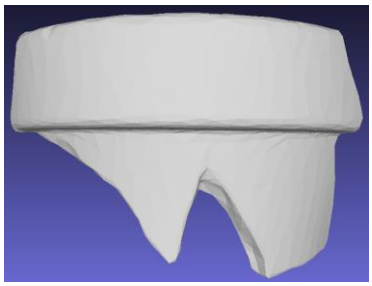
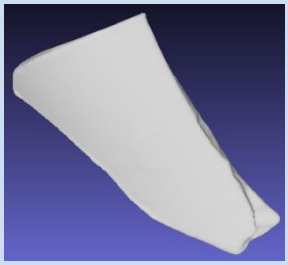
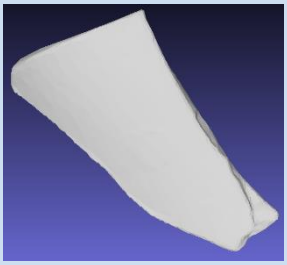
Nombre	Modelo original	Modelo simplificado
Ceramica1		
	4.774 vértices / 9.544 caras	1.433 vértices / 2.862 caras
Ceramica3		
	50.907 vértices / 101.810 caras	1.200 vértices / 2.036 caras
Ceramica5		
	20.483 vértices / 40.962 caras	1.026 vértices / 2.048 caras

Tabla 16: Reducción de algunos modelos

Una vez que la geometría está disponible en la base de datos se pueden aplicar cálculos a la misma como, por ejemplo, aplicar el algoritmo DE-9IM sobre sus proyecciones. Para mejorar la eficiencia de los cálculos, así como añadir nuevas funcionalidades, se puede hacer uso de la extensión de SFCGAL disponible para PostGIS.

La biblioteca SFCGAL está basada en la biblioteca CGAL (Computational Geometry Algorithms Library). Para instalarla es necesario contar con otras bibliotecas. Y para utilizarla en PostGIS es necesario volver a instalar la extensión cambiando la configuración desde un principio.

Se ha intentado compilar y enlazar la biblioteca con PostGIS con las siguientes versiones de las bibliotecas necesarias:

- CMake 3.15.0 (herramienta para compilación)

- Boost 1.75
- CGAL 4.13

No obstante, debido a problemas de compilación por dependencias entre archivos, como la eliminación de estos, el cambio de localización y/o de nombre, no se ha podido ni compilar ni enlazar la biblioteca, desistiendo de su uso puesto que ahora mismo no es necesaria.

3.4.2 Lectura de ficheros OBJ

En esta tarea se pretende eliminar los prefabs del proyecto original que se están utilizando en Unity para leer en tiempo de ejecución los modelos que deben ser cargados en la escena. Esto permite gestionar una solución real, mientras que, de la antigua manera, se gestionaba una simulación. También se abordarán diversas propuestas de lectura y/o almacenamiento para dotar a la aplicación de una lectura rápida de los modelos.

Además de la evidente eliminación de los prefabs creados en Unity, el valor del campo **original_mesh** debe ser cambiado en todos los elementos, puesto que estos referenciaban al nombre dado al prefab. Este cambio involucra la creación de una ruta específica dentro de la aplicación donde almacenar los ficheros OBJ para su posterior lectura.

Unity empaqueta todos los assets que se utilizan en el proyecto cuando este es exportado. Sin embargo, en ocasiones es útil colocar ficheros en una ruta de fácil acceso y que no se empaqueten con los demás archivos. Para este caso, Unity proporciona la carpeta **StreamingAssets** y pone a disposición del desarrollador funciones para su acceso. Al compilar y empaquetar la aplicación, Unity se encarga de copiar todos los archivos en esa carpeta a la misma ruta en la que se encuentra el ejecutable. La estructura de carpetas queda como muestra en la Ilustración 84, encontrándose dentro de la carpeta **StreamingAssets** la carpeta **Models**

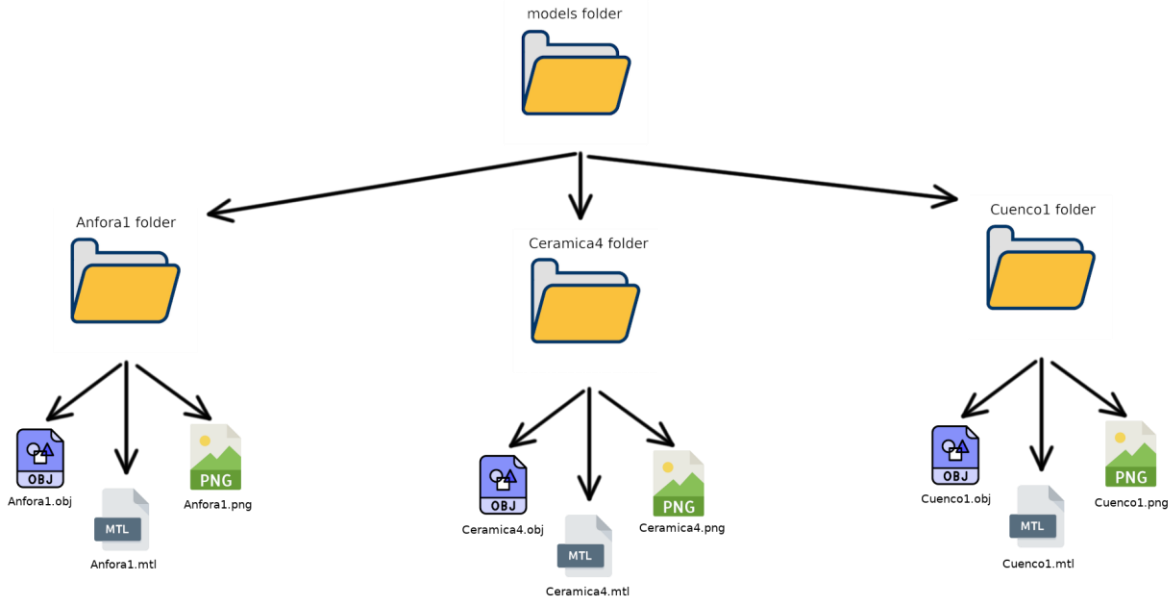


Ilustración 84: Estructura de la carpeta Models dentro de la carpeta StreamingAssets

Todos los ficheros incluidos en cada una de las subcarpetas contienen el mismo nombre (el nombre de la subcarpeta). Este nombre se corresponde con el valor del campo **original_mesh** de la tabla **Element**.

Como se explicó anteriormente, la lectura de ficheros OBJ puede ser lenta y tediosa, y por tanto se va a hacer uso de un asset de terceros que permita su lectura, llamado Runtime OBJ Importer[25] y disponible en la tienda de Unity. Además de leer la geometría, este asset también nos permite leer el fichero MTL asociado al modelo para obtener su textura.

Normalmente una capa puedo contener entre 20 y 50 elementos. En el estado actual del proyecto es inviable esperar mucho tiempo para visualizar la capa al renderizarse demasiada geometría.

Para disminuir tanto el tiempo de carga como la geometría, se pueden utilizar los modelos simplificados. Los mismo están disponibles puesto que son los utilizados para adquirir la geometría introducida en la base de datos, tal y como se ha explicado en el apartado anterior. Los modelos enriquecidos se pueden dejar para la visualización en alta resolución siempre y cuando el usuario lo requiera.

[25] https://assetstore.unity.com/packages/tools/modeling/runtime-obj-importer-49547

No obstante, para intentar mejorar el tiempo de carga, se proponen dos alternativas:

- Almacenamiento serializado de cada fichero OBJ.
- Almacenamiento de los modelos cargados en un diccionario.

La serialización es el proceso automático de transformar una estructura de datos u objetos en un formato que Unity puede almacenar y leer de forma rápida y eficiente. Algunas utilidades de Unity lo utilizan, como la ventana del inspector o el propio uso de prefabs. Además, está ampliamente estandarizado para el guardado de juegos.

Con el almacenamiento de objetos serializados, el fichero OBJ únicamente se leería la primera vez que se cargue en el sistema. Posteriormente se almacenaría en dicho formato utilizando funciones específicas de Unity, y a partir de la segunda vez que sea necesario cargar el mismo fichero, esta carga se realizaría desde el archivo serializado.

No obstante, según las pruebas realizadas, el tiempo que se tarda en leer el fichero serializado es muchísimo mayor que el tiempo empleado en leer el fichero OBJ. Además, dada la gran cantidad de vértices presentes en cada modelo se utiliza demasiada memoria para ello. Y, por tanto, tras probar su uso y comprobar que no mejoraba los resultados, ha dejado de considerarse como opción a implementar.

La otra opción que nos queda es utilizar un diccionario proporcionado por Unity para almacenar en forma de GameObjects los modelos ya cargados. De esta manera, cuando un nuevo modelo necesite ser cargado en el sistema, se comprueba si ya había sido cargado previamente y en tal caso, únicamente se realiza una copia del mismo, minimizando bastante el tiempo de carga para ese modelo. Si es la primera vez que se carga en el sistema, se almacena en el diccionario para futuros accesos.

Aunque esta opción reduce significativamente el tiempo de carga, únicamente se le saca provecho cuando los modelos se repiten. Esta situación solo se da en nuestra aplicación, puesto que estamos simulando los elementos de cada una de las capas con un conjunto limitado de modelos diferentes. Sin embargo, en un yacimiento real no sería útil esta solución ya que no se repetirían los elementos arqueológicos que se encuentran. Por lo tanto, su uso está limitado a nuestra simulación.

Así pues, para reducir el tiempo en posteriores iteraciones vamos a trabajar con los modelos simplificados y con un diccionario.

3.5 Iteración 4

En esta iteración vamos a simular de una manera más realista el resultado que se obtendría en una excavación. Para ello utilizaremos una capa previamente seleccionada como capa de pruebas. Se añadirán nuevos modelos exclusivamente para esta capa.

El desarrollo de esta iteración abarca desde el día 24/02/2021 al día 19/03/2021.

3.5.1 Nuevos modelos 3D

Para conseguir una mayor variedad de modelos con los que poder realizar una simulación más realista, se van a descargar modelos gratuitos de Internet. Para ello es necesario realizar una búsqueda de repositorios y páginas web desde los que obtener los modelos de manera gratuita[26,27,28].

En la Ilustración 85 se pueden ver ejemplos de estos nuevos modelos.

Ilustración 85: Nuevos modelos

Tal y como se ha llevado a cabo en anteriores iteraciones, se han simplificado los nuevos modelos y se han tomado imágenes de ellos para utilizarlas en la aplicación

[26] https://free3d.com/
[27] https://cults3d.com/
[28] https://sketchfab.com/

final con las funcionalidades ya incluidas. También se ha añadido su correspondiente geometría y proyecciones a la base de datos.

A todos estos modelos hay que añadirle un nuevo modelo que se ha partido utilizando Blender. La rotura se ha llevado a cabo utilizando un complemento llamado **Cell Fracture**[29]. Este utiliza un sistema de partículas para producir roturas realistas. Es ampliamente configurable, por lo que queda a elección del autor del trabajo el resultado final. La Ilustración 86 muestra el resultado final obtenido tras varias pruebas.

Ilustración 86: Rotura del modelo: Cuenco Catawba

3.5.2 Modelo 3D del terreno

En los repositorios visitados había también modelos de yacimientos arqueológicos reales. Hemos procedido a la descarga de algunos de ellos para intentar incorporarlos a nuestro proyecto.

El modelo que vamos a utilizar pertenece al parque arqueológico de Carranque[30] situado en Castilla-La Mancha[31].

Desafortunadamente, el modelo contiene millones de triángulos que dificultan mucho su visualización y manipulación (ver Ilustración 87). Además, no necesitamos un terreno tan grande, por lo que conviene recortar una zona que se adapte a alguna de las capas.

[29] https://docs.blender.org/manual/es/2.91/addons/object/cell_fracture.html
[30] https://sketchfab.com/3d-models/roman-villa-of-carranque-6d2feccad01c40a5a272bd37ece16231
[31] https://cultura.castillalamancha.es/patrimonio/parques-arqueologicos/carranque

Ilustración 87: Modelo 3D del terreno

El recorte se ha llevado a cabo con el software CloudCompare. Se ha utilizado este software y no Blender ya que este último dificultaba demasiado el recorte. La zona recortada y el contorno del área deben asemejarse lo máximo posible para aumentar el realismo de la simulación, tal y como muestra la Ilustración 88.

Una vez que el modelo se ha recortado correctamente, se simplifica de la misma manera que se ha simplificado cualquier otro modelo anterior.

Por último y como parte del objetivo de esta iteración, el modelo ***Cuenco Catawba*** se ha colocado de tal forma que parece estar semienterrado en el modelo del terreno. De esta manera, podemos simular la excavación del mismo y su descubrimiento en dicho proceso. Los demás modelos se colocarán en la misma capa de una manera más o menos aleatoria.

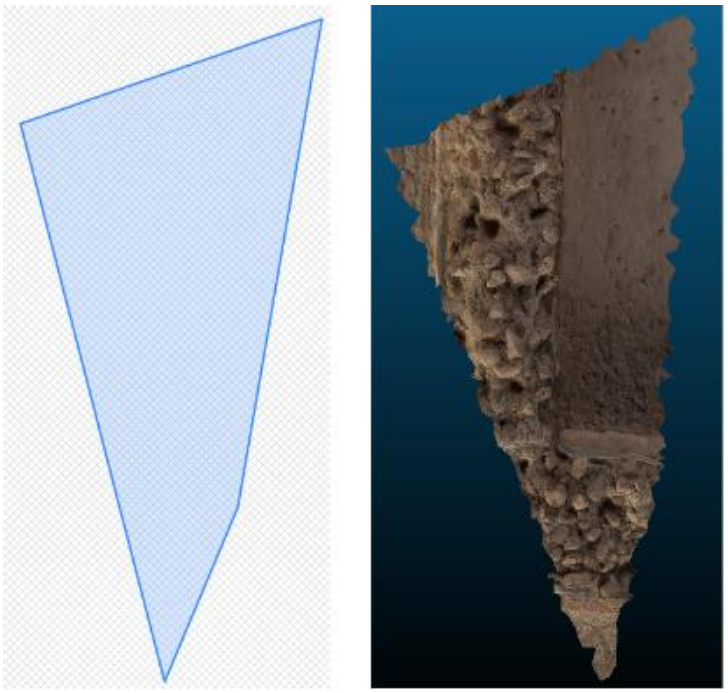


Ilustración 88: Comparación del modelo 3D del terreno recortado y la capa elegida

3.5.3 Simulación

Para colocar los modelos en la capa, se han sustituido los elementos que había previamente en ella por los nuevos. El resultado inicial puede verse en la Ilustración 89.

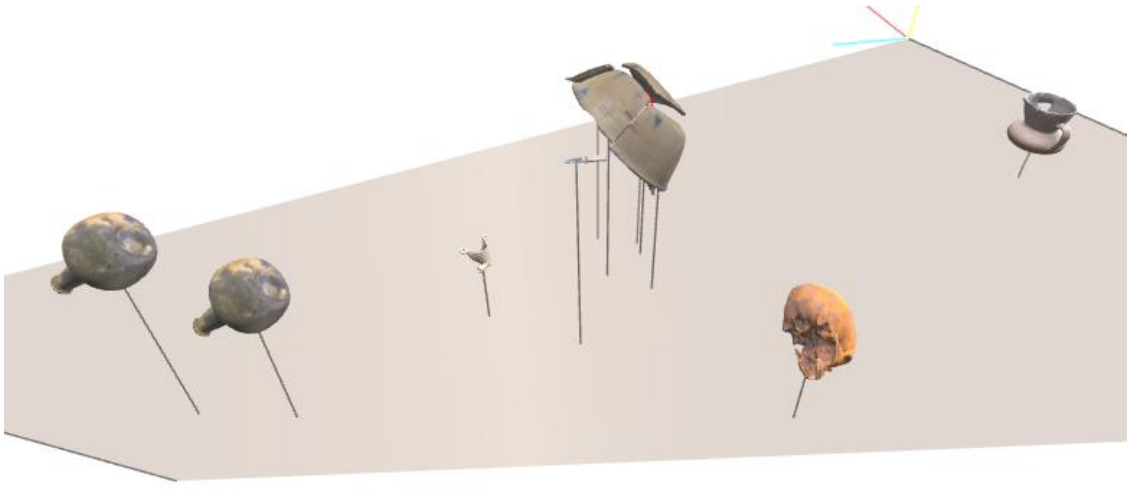


Ilustración 89: Nueva disposición de la capa

Continuando con la simulación, es necesario recolocar de manera externa los objetos, ya que tal cual se encuentra en este momento la capa, la disposición de los modelos sueltos no se relaciona con la disposición del terreno. Para ello, se modifican

las coordenadas UTM en la base de datos de todos aquellos elementos que así lo requieren, para dar como resultado el estado mostrado en la Ilustración 90.

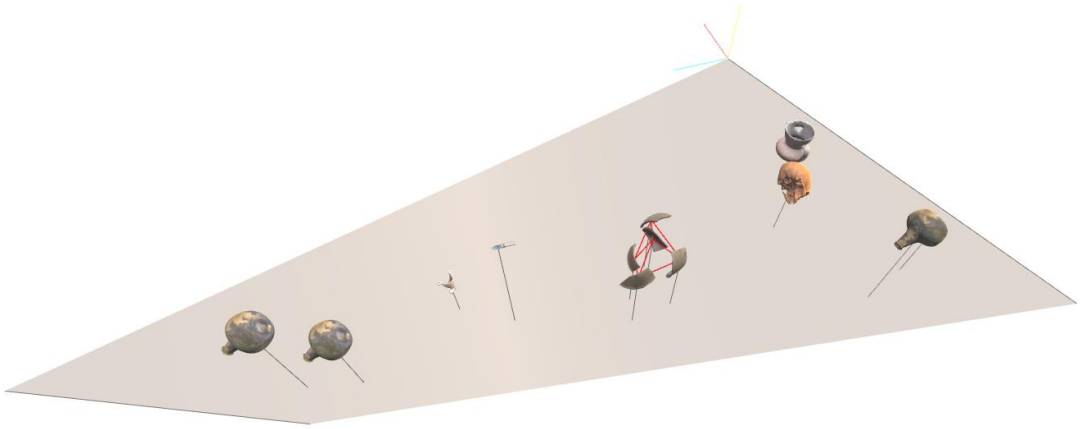


Ilustración 90: Disposición final de la capa

Por último, para completar la simulación, se añaden las relaciones topológicas y las imágenes a la base de datos. Las relaciones topológicas se han añadido para que cada pieza del ***Cuenco Catawba*** se relacione con las piezas que estaban a su lado, según la Ilustración 86.

Para diferenciar aquellos modelos que no pertenecen al terreno y que por tanto pueden moverse y girarse con la funcionalidad implementada en el apartado 3.3 (Iteración 2), se añade transparencia al modelo del terreno (ver Ilustración 91). De esta manera, los modelos transparentes se utilizan como guía (al igual que se utilizaban las imágenes) para reposicionar correctamente los modelos sueltos.

Ilustración 91: Modelos del terreno y elementos

3.6 Iteración 5

Hasta ahora se ha trabajado con los modelos que ya se poseían o con modelos descargados de repositorios públicos en Internet. Esta última iteración tiene como objetivo el escaneo de modelos 3D para utilizarse en el sistema.

Con esta iteración se pretende simular la metodología de trabajo que debe realizar un arqueólogo cuando añada nuevos modelos 3D a la base de datos. Como vimos en el apartado 2.1.1 (Arquitectura inicial del sistema), el procedimiento se divide en dos fases: adquisición de datos y análisis e interacción. Este apartado está centrado en la primera fase, por lo que, a continuación, se van a explicar las tecnologías probadas.

Su desarrollo abarca desde el día 12/04/2021 al día 18/05/2021.

3.6.1 Aplicaciones para dispositivos móviles

El uso de aplicaciones para dispositivos móviles se justifica porque el trabajo de los arqueólogos se desarrolla en las zonas de excavación, por lo que disponer de un dispositivo móvil para realizar el proceso de escaneo agiliza las tareas.

Se han estudiado y probado distintas aplicaciones para dispositivos con el sistema operativo Android, aunque es posible que algunas de ellas estén disponibles para iOS. En concreto, hemos probado las aplicaciones Qlone, SCANN3D y 3D Live Scanner. Todas ellas se han probado en los lugares habilitados para las prácticas de la Universidad de Jaén.

3.6.1.1 Qlone

Disponible de forma gratuita en la Play Store[32] de Android, aunque hay una versión de pago (versión premium) de la misma.

Para nuestro caso de estudio, la aplicación puede usarse para el escaneo de restos arqueológicos pequeños una vez que estos han sido extraídos para su estudio.

Para este proceso, es necesaria una plantilla (Ilustración 92, izquierda) con un patrón cuadriculado donde se coloca el objeto. De manera opcional se da la posibilidad de utilizar otra plantilla colocada de manera vertical para mejorar el proceso de escaneado (Ilustración 92, derecha).

[32] https://play.google.com/store/apps/details?id=com.eyecue.qlone

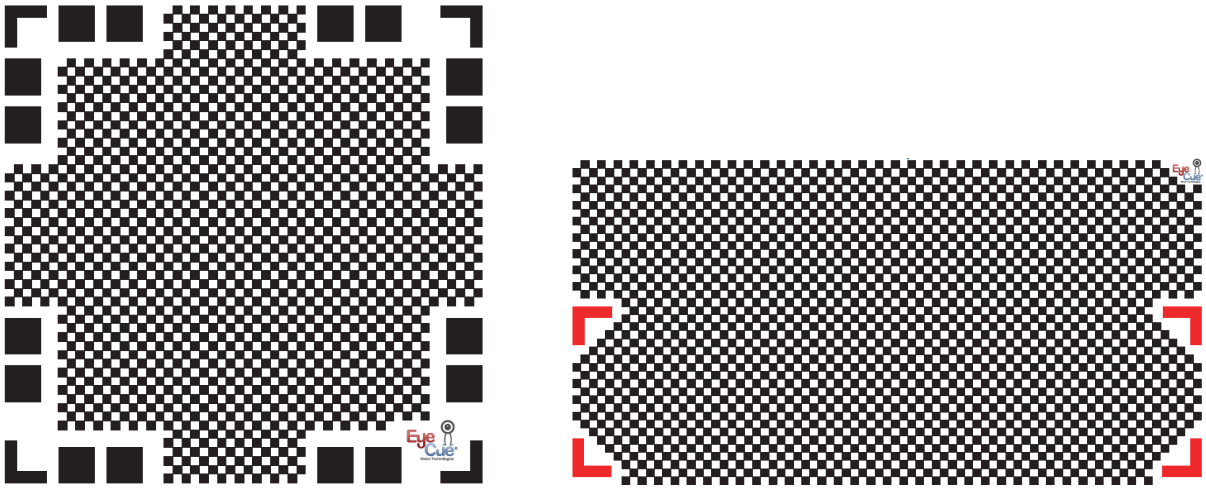


Ilustración 92: Izquierda: plantilla para la base del objeto. Derecha: plantilla vertical para mejorar el proceso

El escaneo del modelo se presenta como una cúpula dividida en franjas horizontales y verticales, de manera que cada zona hace referencia a cada una de las imágenes que la aplicación toma internamente para obtener el modelo 3D. Este proceso se realiza enfocando el objeto con la cámara trasera del móvil y moviéndolo alrededor de la cúpula.

La Ilustración 93 y la Ilustración 94 muestran capturas de pantalla del proceso de escaneado con un objeto de prueba.

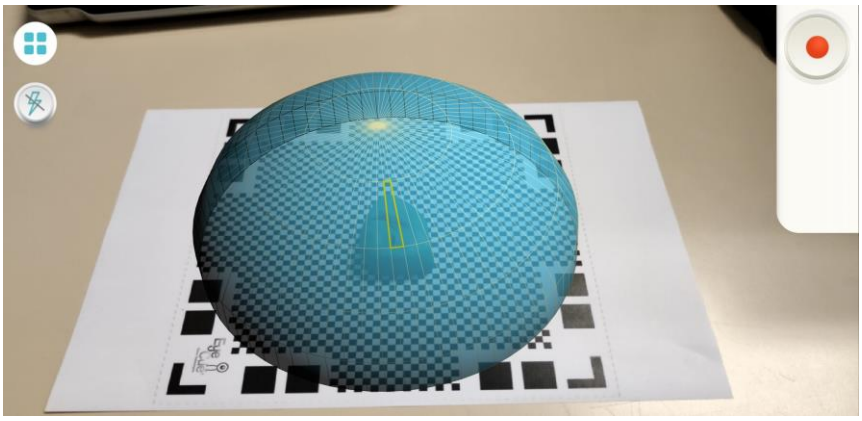


Ilustración 93: Captura de pantalla al inicio del proceso de escaneo con Qlone

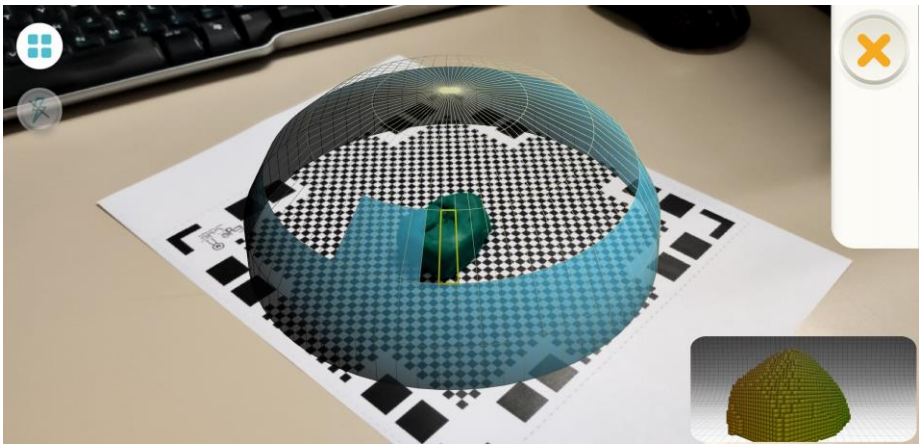


Ilustración 94: Captura de pantalla durante el proceso de la aplicación Qlone

El resultado del proceso se puede ver en la Ilustración 95.

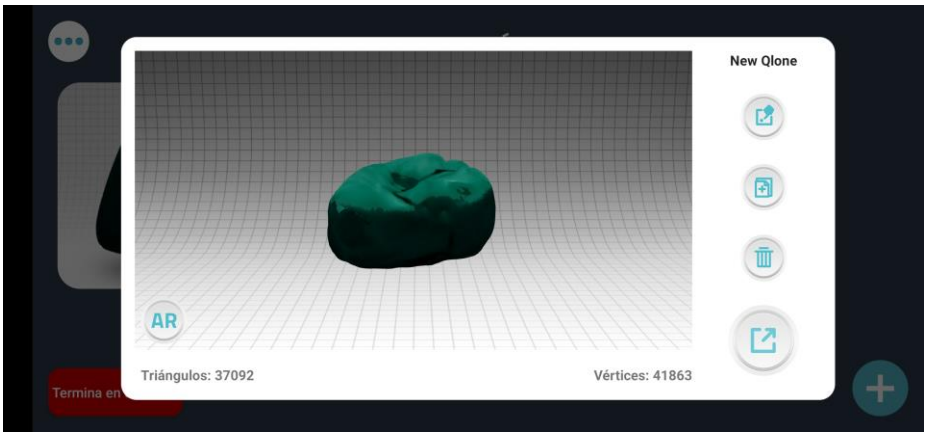


Ilustración 95: Resultado del proceso de escaneado con Qlone

Todo este proceso es totalmente gratuito. Como inconveniente, podemos decir que para las demás funciones: edición, visualización en AR y exportación a los formatos 3D más comunes se requiere la versión de pago.

A modo de conclusión, señalar que el proceso es bastante preciso y sencillo pero muchas funcionalidades quedan limitadas a la versión de pago.

3.6.1.2 SCANN3D

Al igual que la aplicación anterior, esta se encuentra disponible de forma gratuita en la Play Store[33] de Android. La aplicación cuenta con una suscripción

[33] https://play.google.com/store/apps/details?id=com.smartmobilevision.scann3d

mensual que añade características y funcionalidades a la versión gratuita. También existe la posibilidad de probar la aplicación sin restricciones durante tres días.

SCANN3D utiliza técnicas de fotogrametría para obtener la representación tridimensional del modelo a escanear. Por tanto, se necesita un conjunto de imágenes para realizar este proceso.

Al utilizar la aplicación se nos ofrece la posibilidad de escanear un nuevo objeto a partir de un conjunto de imágenes que se tomarán en ese instante, o importar un conjunto de imágenes ya tomadas (únicamente disponible en la versión de pago).

Para el proceso de escaneo se nos ofrece un modo guía donde nos indica cuándo debemos tomar la siguiente captura en base a los puntos más característicos de la imagen anterior. Este proceso puede verse en la Ilustración 96.

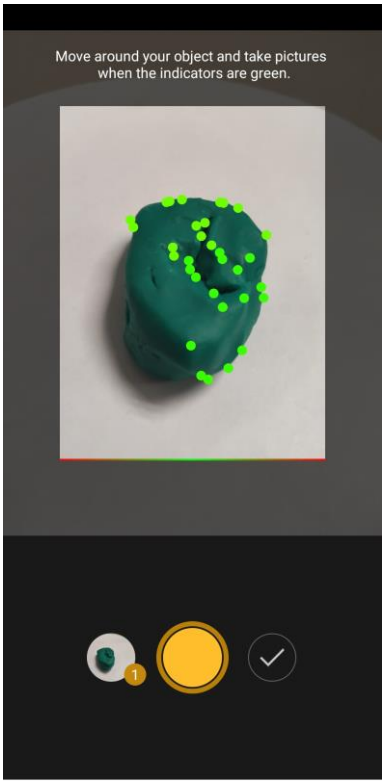


Ilustración 96: Captura de pantalla del proceso de escaneado con SCANN3D

El resultado de este proceso se ve en la Ilustración 97.

Ilustración 97: Resultado del proceso de escaneo con SCANN3D

Al ser una aplicación orientada a fotogrametría, la calidad del modelo viene determinada en gran medida por la calidad y cantidad de las imágenes tomadas. Por otro lado, con esta aplicación es posible obtener tanto los modelos 3D del terreno arqueológico como de los restos arqueológicos que se encuentren en él.

3.6.1.3 3D Live Scanner

Por último, disponemos de está aplicación de manera gratuita en la Play Store[34] de Android. Al igual que las anteriores, existe una aplicación de pago que extiende las características y funcionalidades de la versión gratuita.

A diferencia de las aplicaciones anteriores, esta se encuentra en desarrollo. Además, utiliza el sensor ToF (Time of Flight) disponible en las nuevas cámaras de algunos teléfonos Android, por lo que está pensada para ser utilizada en interiores y/o exteriores.

Estos sensores se utilizan como sistemas de lectura de profundidad basados en haces de luz infrarroja, en concreto en haces de luz de 20 MHz (Amad-ud-Din et al., 2009). La cámara del teléfono se encarga de medir el tiempo entre la emisión y la

[34] https://play.google.com/store/apps/details?id=com.lvonasek.arcore3dscanner

recepción del haz de luz. De esta manera, es capaz de medir la profundidad completa de una escena con un disparo único, soportando distancias de hasta 200 metros (Amad-ud-Din et al., 2009). Al utilizar haces de luz infrarroja, los datos que proporciona son instantáneos y precisos.

En relación a este proyecto, podemos utilizar esta aplicación únicamente para el escaneo del terreno, debido a que la aplicación dificulta el escaneo de piezas pequeñas.

3.6.2 Escáneres 3D portátiles

Otra de las tecnologías que se pueden utilizar es un escáner 3D portátil.

Al utilizar escáneres portátiles tenemos la posibilidad de poder escanear tanto el terreno como los elementos que sobresalgan de él con mucha más precisión.

Para este proyecto se han utilizados dos escáneres distintos**: Artec Eva** y **EinScan Pro 2x**.

3.6.2.1 Artec Eva

El escáner Artec Eva[35] es fabricado por la compañía Artec3D. Entre sus especificaciones encontramos:

- Escáner manual para objetos de tamaño mediano a grande.
- Precisión de puntos de hasta 0.1 mm y resolución de hasta 0.2 mm.
- Captura de texturas.
- Algoritmos de procesamiento de datos basados en geometría y/o textura.

Para el proceso de escaneo es necesario el software propietario, denominado Artec Studio y disponible para Windows, incluido en la licencia de compra del escáner. También se necesita conectar el escáner a un ordenador para procesar todos los datos que este recoge.

[35] https://www.artec3d.com/es/portable-3d-scanners/artec-eva-v2

3.6.2.2 EinScan Pro 2X

El escáner EinScan Pro 2X[36] pertenece a la compañía Shining 3D y algunas de sus especificaciones son:

- Precisión de hasta 0.04mm.
- Velocidad de escaneo de hasta 30 FPS (Frames Per Second).
- Captura de texturas.
- Diferentes modos de escaneo: portátil y fijo.
- Alineación mediante marcadores, geometría y/o textura (dependiendo del modo de escaneo)

Al igual que el escáner anterior, es necesario el software propietario, denominado EXScan Pro y disponible para Windows, incluido en la licencia de compra del escáner, por lo que también es necesario conectar el escáner a un ordenador para realizar todo el procesamiento.

Sin embargo, a diferencia del escáner anterior, este dispone de una plataforma giratoria para escanear de manera automática cualquier objeto colocado sobre la misma, por lo que, aunque el escáner es portátil, también puede usarse de manera fija.

3.6.3 Obtención del modelo del terreno

Dada la situación social vivida en estos meses no ha sido posible obtener datos reales de excavaciones arqueológicas hasta ahora. Para simular la metodología seguida por un usuario experto en el uso de la herramienta, se va a escanear el terreno de una de las zonas del campo de prácticas de los alumnos de la Universidad de Jaén pertenecientes al Grado Interuniversitario en Arqueología.

En la Ilustración 98 se muestra el campo de prácticas. Está separado mediante filas de piedras. En este trabajo se ha elegido una zona al azar para realizar en ella todas las tareas necesarias.

[36] https://www.einscan.com/handheld-3d-scanner/einscan-pro-2x/

Ilustración 98: Campo de prácticas del Grado en Arqueología

Dentro de las aplicaciones móviles explicadas anteriormente, se van a utilizar las dos últimas (SCANN3D y 3D Live Scanner), ya que ambas nos permiten obtener los modelos 3D de una superficie. Una de ellas a partir de fotogrametría y la otra a partir del sensor ToF incluido en la cámara.

Recordemos que **SCANN3D** obtiene el modelo mediante fotogrametría, así que es necesario conseguir un conjunto de imágenes de la zona. Para obtener un buen resultado, se requiere una cantidad significativa de fotografías tomadas desde un número de puntos de vista suficientemente diferentes. Además, es necesario cierto grado de superposición entre los contenidos visibles en cada fotografía.

El recorrido típico es girar alrededor de todo el objeto mientras se le toman fotos cada cierto ángulo. Un ejemplo del recorrido para la toma de imágenes se puede ver en la Ilustración 99.

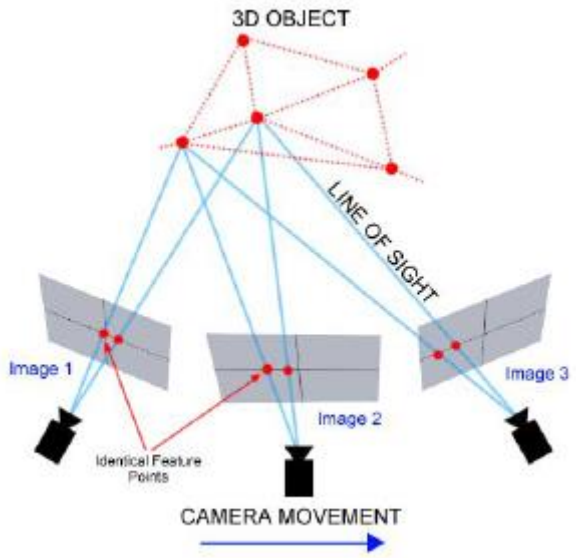


Ilustración 99: Captura de imágenes para la recreación del modelo mediante fotogrametría
(https://thehaskinssociety.wildapricot.org/photogrammetry)

Una vez tomadas todas las imágenes necesarias (ejemplos de las tomas en la Ilustración 100), es hora de generar el objeto en 3D. Para ello, basta con cargar el conjunto de imágenes a la aplicación y esperar a que esta cree el modelo 3D a partir de las mismas.

Ilustración 100: Distintas tomas del terreno

El resultado obtenido de esta aplicación móvil es bastante preciso y puede verse en la Ilustración 101 y en la Ilustración 102.

Ilustración 101: Resultado de la aplicación SCANN3D

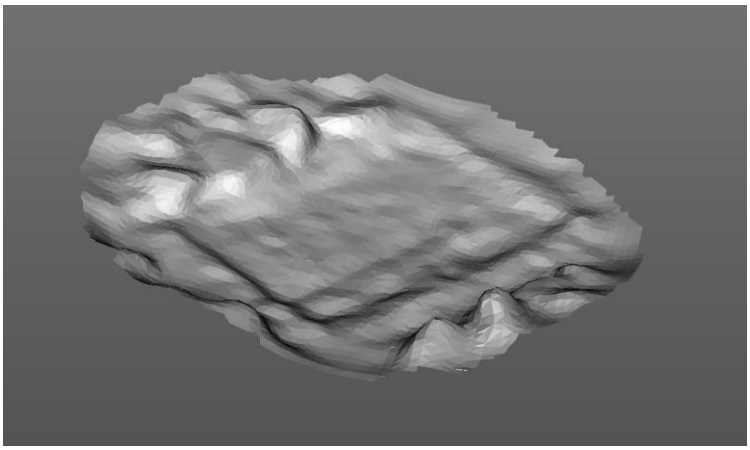


Ilustración 102: Resultado en malla de triángulos de la aplicación SCANN3D

En el caso de utilizar **3D Live Scanner**, el proceso es más sencillo y ágil. Únicamente basta con seleccionar si la resolución elegida es para interior o exterior (recordemos que esta aplicación está enfocada en el escaneo de interiores y/o exteriores) e inmediatamente la aplicación nos permite realizar el proceso de escaneo (ver Ilustración 103).

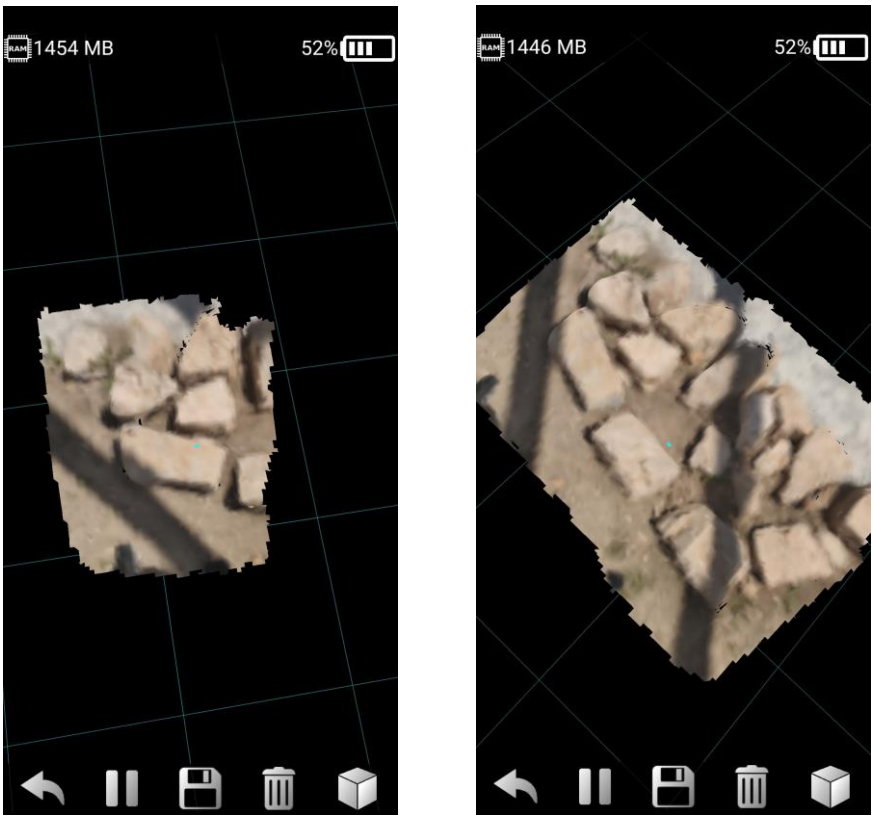


Ilustración 103: Proceso de escaneo del terreno con la aplicación 3D Live Scanner

Por tanto, el resultado se consigue de manera inmediata (ver Ilustración 104).

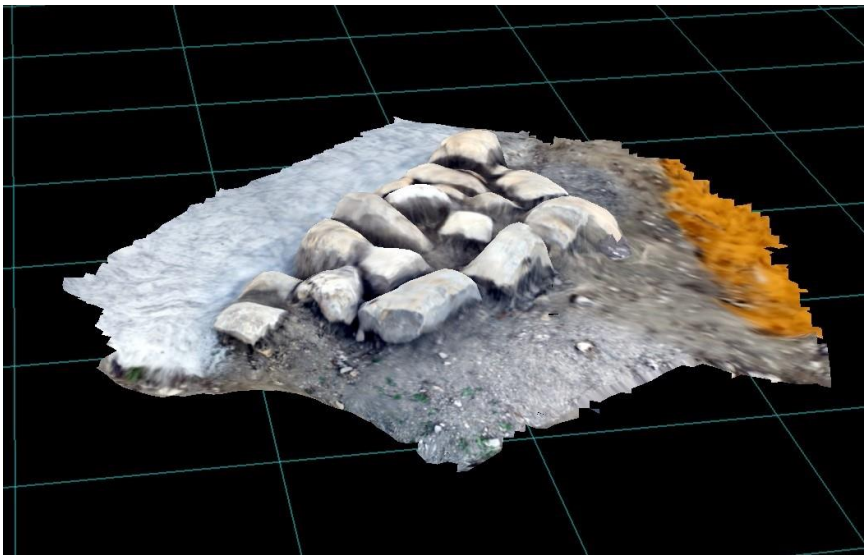


Ilustración 104: Resultado con la aplicación 3D Live Scanner

Ambas aplicaciones funcionan perfectamente a la hora de obtener el modelo en 3D del terreno. Por tanto, se presentan dos herramientas válidas para su uso en yacimientos arqueológicos.

Por último, vamos a comparar ambas aplicaciones con el escáner de mano **Artec Eva**. Aunque normalmente estos tipos de escáneres están pensados para utilizarse con objetos de tamaño medio-grande, también es posible escanear el terreno.

Es cierto que nuestro campo de prácticas es suficientemente pequeño para no dar problemas con la escala del modelo y para ser escaneado en una única pasada con el escáner, situación que puede no darse en un yacimiento arqueológico. Como solución a este último inconveniente podemos escanear el terreno a trozos, obteniendo los modelos correspondientes para cada trozo y utilizar su software para unir los distintos modelos.

A la hora de proceder con el escaneo del campo de pruebas, nos encontramos con un gran inconveniente: el escáner no detecta el terreno debido a la luminosidad del ambiente. Esto dificulta la obtención del modelo 3D mediante el uso del escáner y tan solo es posible utilizarlo con condiciones lumínicas específicas.

En nuestro caso, el proceso de escaneo se produce un día nublado en el cual las condiciones lumínicas no entorpecen el proceso. La Ilustración 105 y la Ilustración 106 muestran el proceso de escaneado.

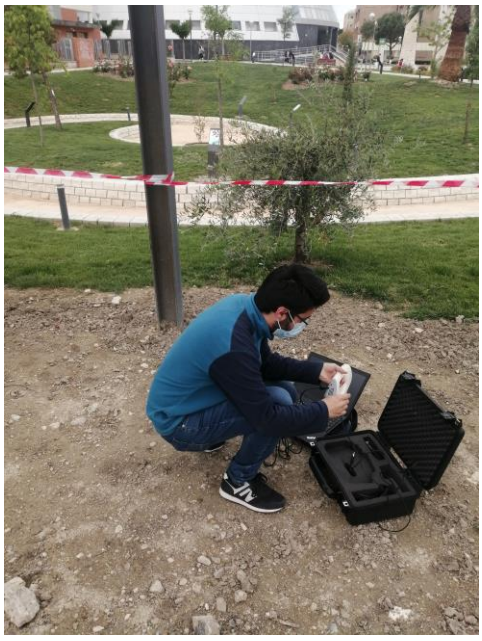


Ilustración 105: Preparación del escáner

Ilustración 106: Proceso de escaneado

Una vez obtenidas las tomas necesarias, se procede a unirlas mediante el software incluido en la licencia del escáner. El software es bastante potente y capaz de realizar todo el postprocesamiento básico por sí mismo, a excepción de la limpieza de puntos.

El resultado obtenido se muestra en la Ilustración 107.

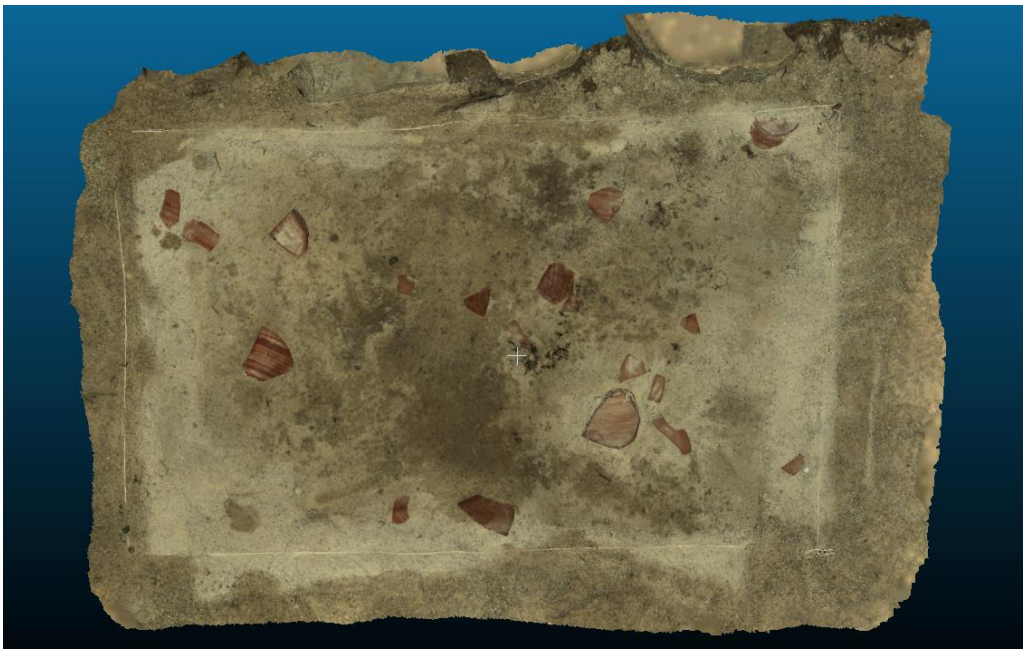


Ilustración 107: Resultado del escaneo con el escáner Artec Eva

Como era de esperar, el resultado es más preciso que los obtenidos con las aplicaciones móviles. Sin embargo, el uso del escáner está limitado a las condiciones lumínicas de la zona, sin contar todo el equipo necesario (cables, portátil y batería) para su funcionamiento, lo que lo convierte en una tecnología no tan ágil como los dispositivos móviles.

3.6.4 Obtención de los modelos de los objetos

Por último, una vez obtenido el modelo 3D del terreno, es necesario obtener los modelos 3D de cada una de las piezas enterradas en el terreno. La extracción de las piezas es un proceso lento, ya que deben ser dibujadas, medidas y analizadas en la situación en la que se encontraron por parte de los expertos.

No obstante, una vez que los expertos extraigan las piezas estas pueden ser escaneadas con el fin de introducirlas en la herramienta de Realidad Virtual. Dicho escaneo debe ser rápido y lo más automático posible, ya que disponemos de muchas piezas.

El proceso se llevará a cabo con el escáner **EinScan Pro 2X** el cual permite el escaneo sobre una plataforma giratoria.

El proceso llevado a cabo se muestra en la Ilustración 108 y en la Ilustración 109.

Ilustración 108: Escaneado con el escáner EinScan Pro 2X

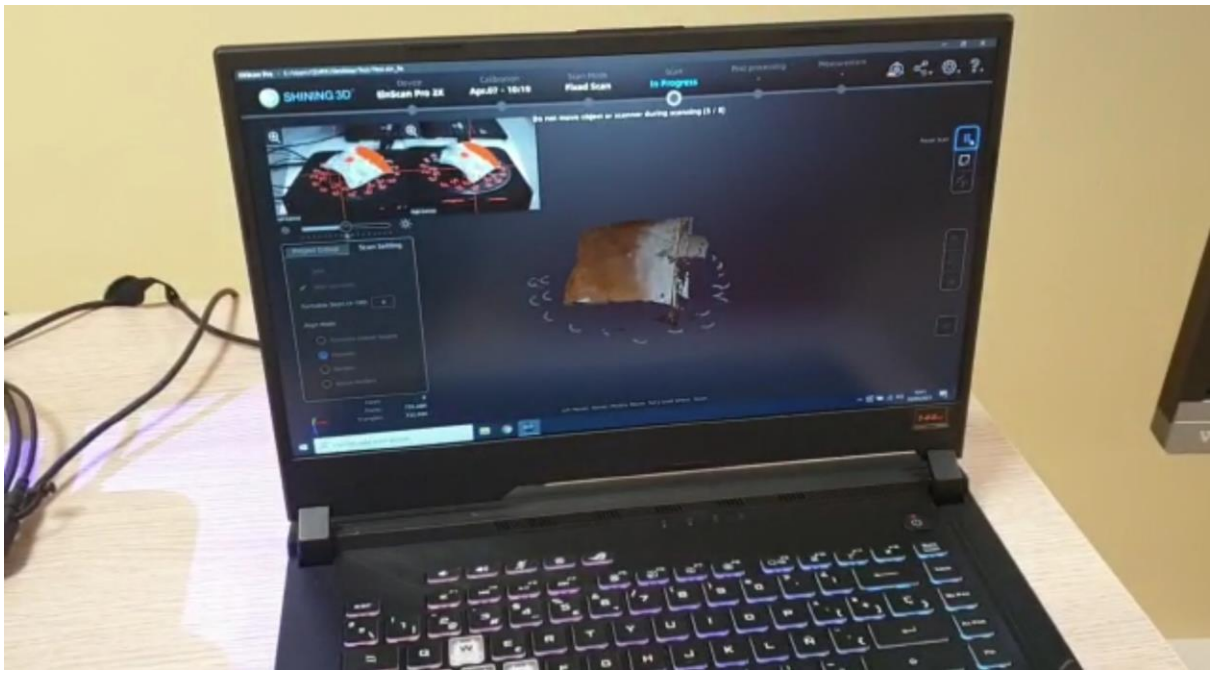


Ilustración 109: Visualización en tiempo real de la construcción del modelo con el software EXScan Pro

Al igual que con el escáner Artec Eva, se dispone de un software propietario en el cual se realiza todo el proceso de escaneo y facilita todo el procesamiento posterior. El resultado obtenido puede verse en la Ilustración 110.

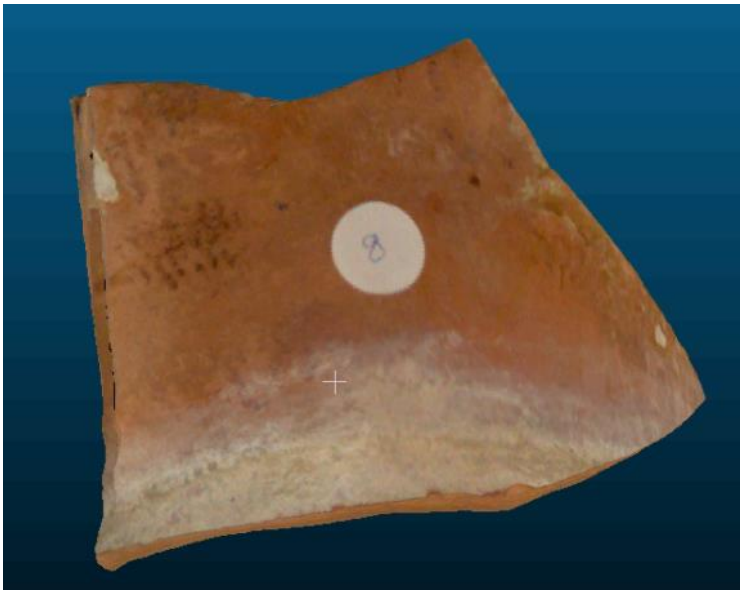


Ilustración 110: Resultado obtenido con escáner EinScan Pro 2X

4 EXPERIMENTACIÓN, RESULTADOS Y DISCUSIÓN

En este apartado se documentan aquellas pruebas realizadas sobre la aplicación.

4.1 Experimentaciones y pruebas

En el apartado 3.5 (Iteración 4) se utilizaron modelos sintéticos para dotar de mayor contenido la aplicación. Para la carga de los modelos se ha seleccionado una capa arbitraria y se han ido colocando elementos a la vez que se tomaban mediciones del tiempo de carga de la misma.

El objetivo de estas pruebas es ver cuántos objetos pueden cargarse en un tiempo razonable. Para ello vamos a eliminar las mejoras introducidas en el apartado 3.4 (Iteración 3), de manera que se simula de forma más realista una carga de una capa real en un yacimiento.

El hardware utilizado para las pruebas, tiene las siguientes características:

- Intel Core i7-4790 3.80GHz.
- 16 GB RAM DDR3-2400
- NVIDIA GeForce GTX970 4GB VRAM
- Seagate HDD 3.5 1TB 600 MB/s

Los resultados se pueden ver en la Tabla 17. En este caso se utilizan los modelos de alta resolución.

Cantidad de modelos	Tiempo de carga (segundos)	Geometría cargada (nº caras)
5	13.23	224 mil
10	26.72	448 mil
20	53.35	897 mil
50	130.18	2.244 millones
100	333.34	4.488 millones

Tabla 17: Tiempos de carga de los modelos enriquecidos

Se han hecho las mismas pruebas con los modelos simplificados. En la Tabla 18 se muestran los resultados obtenidos

Cantidad de modelos	Tiempo de carga (segundos)	Geometría cargada (nº caras)
5	3.09	12 mil
10	5.90	24 mil
20	12.03	48 mil
50	50.74	121 mil
100	90.423	242 mil

Tabla 18: Tiempos de carga de los modelos simplificados

4.2 Resultados y discusión

Esperar de 1 a 2 minutos para que una capa que contiene entre 20 y 50 elementos distintos esté completamente cargada utilizando los modelos con alta resolución puede ser desesperante para el usuario final. Por este motivo se recomienda utilizar los modelos simplificados, ya que el tiempo de espera es muchísimo menor. Además, como se ha mostrado en el apartado 3.4 (Iteración 3), los modelos simplificados no difieren mucho en cuanto a geometría de los modelos detallados, por lo que visualmente no se aprecian demasiadas diferencias.

Por otro lado, una vez que los modelos se cargan, se pueden manipular para hacerlos encajar en su respectivo lugar con ayuda de las diferentes guías introducidas en la aplicación. La aplicación está preparada para utilizar como guía el modelo 3D del terreno. Tanto el tiempo empleado como la precisión obtenida dependen del usuario experto que maneje en ese momento la aplicación. La Ilustración 111 muestra el resultado del encaje propuesto por el autor de este documento. Cabe mencionar que no se tienen los conocimientos necesarios en arqueología, pero gracias a la utilización de la guía, se consiguen resultados bastantes razonables.

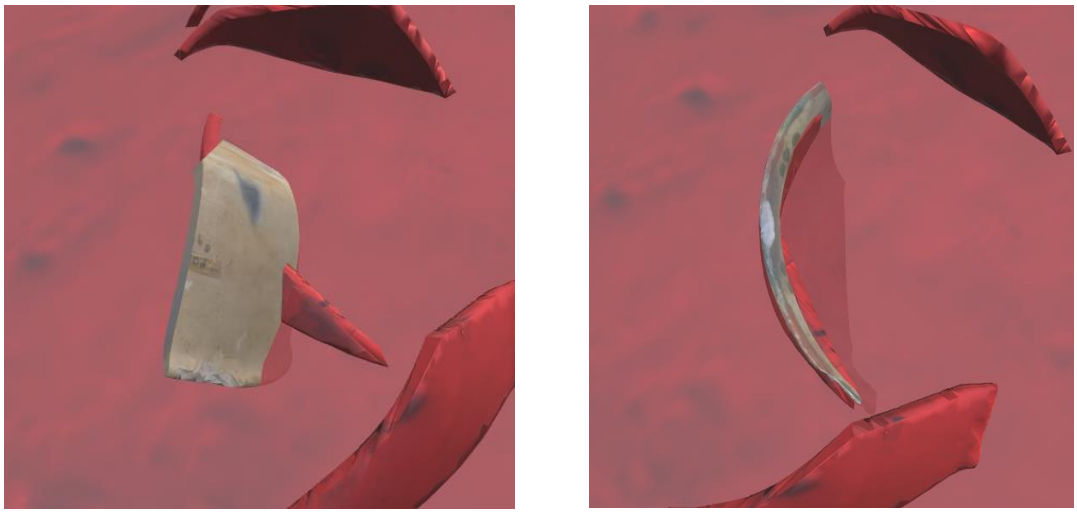


Ilustración 111: Resultados de la recolocación de un elemento. Coloreado de rojo el modelo de referencia para hacer la distinción

Por último, para medir la experiencia de usuario y la usabilidad, distintos usuarios con distintos niveles de conocimiento han ejecutado la aplicación.

Un perfil de usuario más acostumbrado al control por teclado y al uso de manipuladores 3D obtiene una buena sensación de movimiento mientras que un usuario con poca habilidad en este ámbito no necesita demasiado tiempo para adaptarse al control.

Para ambos perfiles la interfaz resulta clara e intuitiva, entendiendo el funcionamiento de la misma sin complicaciones. Además, el grado de detalle de los modelos es suficiente pese a utilizar los modelos simplificados.

Ambos perfiles concuerdan en la cantidad excesiva de teclas necesarias para utilizar al completo la aplicación. No obstante, se incluye un manual de usuario para no tener que memorizar todas ellas.

5 CONCLUSIONES Y TRABAJOS FUTUROS

En este apartado vamos a comentar las conclusiones obtenidas tras la realización de este proyecto y los posibles trabajos futuros derivados del mismo.

5.1 Conclusiones

Se ha desarrollado un sistema cliente-servidor que da soporte a la reproducción virtual de un yacimiento arqueológico.

En el lado del servidor disponemos de una base de datos espacial donde no solo se almacena información textual relevante de un yacimiento arqueológico, sino que también se almacena información espacial, como puede ser el contorno geométrico de cada una de las zonas de excavación, o las matrices de transformación para posicionar correctamente un hallazgo.

En el lado del cliente disponemos de una aplicación gráfica desarrollada en Unity, lo que permite representar una escena virtual con todo detalle aprovechándose de las características del motor de videojuegos. En ella se nos permite manipular la posición y orientación de los modelos mediante manipuladores 3D bien conocidos. Para ayudar en esta manipulación se le proporciona al usuario la posibilidad de utilizar como referencia imágenes o un modelo 3D del terreno en el que se encontraron las piezas. Estas ayudas se utilizan como guía, por lo que el proceso de edición es fácil de realizar y no necesita experiencia. Por otro lado, cualquier experto situado en cualquier parte del mundo puede utilizar la aplicación para visitar la reconstrucción virtual del sitio arqueológico.

En cuanto a la forma de trabajar, se comprende de una serie de pasos que se resumen a continuación. Se utiliza un dispositivo móvil con el que poder escanear de una manera rápida y sencilla cualquier terreno donde se realice una excavación antes de que las piezas se retiren de la misma. El escaneado se lleva a cabo utilizando la tecnología ToF o utilizando fotogrametría con los dispositivos móviles. Posteriormente, los hallazgos más significativos pueden ser escaneados de manera individual utilizando un escáner más preciso, siendo esta la parte más lenta de todo el proceso. Todo este proceso va alimentando la base de datos al mismo tiempo que se introduce información textual como el material, la posición GPS, o la datación histórica de la pieza.

5.2 Trabajo futuro

Algunas de las funcionalidades o tareas que pueden dejarse para trabajo futuro son:

- Terminar de implementar algunas de las características que hagan uso total de la base de datos, como puede ser: la implementación de un sistema gestor de usuarios, así como la autentificación para acceder a toda la información u obtener la información relativa a la datación de los descubrimientos.
- Utilizar información real del yacimiento de Cástulo. Es cierto que obtenemos la forma geométrica de las áreas de ellos, pero toda la información restante ha sido generada de manera procedural. Además, estamos utilizando modelos sintéticos y que no pertenecen a descubrimientos reales.
- Continuar con la implementación de la aplicación gráfica de manera que podamos ofrecer más contenido o pulir el comportamiento existente.
- Utilizar la extensión SFCGAL de PostGIS, la cual ofrece funciones 3D avanzadas que proporcionan mejores características de análisis espacial.
- Montar un servidor FTP para la lectura de ficheros a través de Internet.

6 APÉNDICES

En este apartado se incluye información adicional que puede ser relevante para documentar el presente trabajo.

6.1 Guía original del Trabajo Fin de Título

La guía original[37] de este Trabajo Fin de Master fue publicada en la página de la Escuela Politécnica Superior de Jaén en el primer cuatrimestre del curso 2020-2021. Desde entonces no ha sufrido ninguna modificación.

6.2 Manuales de usuario

A continuación, se presenta una lista con todos los controles disponibles en la aplicación.

Para el movimiento de cámara:

- **Tecla W**: movimiento hacia delante.
- **Tecla S**: movimiento hacia atrás.
- **Tecla A**: movimiento hacia la izquierda.
- **Tecla D**: movimiento hacia la derecha.
- **Tecla E**: movimiento hacia arriba.
- **Tecla Q**: movimiento hacia abajo.
- **Botón derecho del ratón + movimiento de este**: modifica el punto hacia el que mira la cámara, haciendo que esta rote sobre su posición.
- **Botón central del ratón + movimiento de este**: modifica la posición de la cámara en la dirección del movimiento.
- **Tecla CTRL + botón derecho del ratón**: movimiento orbital alrededor del escenario.

[37] http://eps-anterior.ujaen.es/TFMtemporal/mostrarTFM.php?id=467

Para acciones sobre los conceptos (áreas, volúmenes, capas y elementos):

- **Tecla TAB + botón izquierdo del ratón**: movimiento de la cámara hacia el concepto seleccionado.
- **Tecla SHIFT + botón izquierdo del ratón**: muestra un panel con toda la información del concepto seleccionado.
- **Tecla CTRL: + botón izquierdo del ratón**: visualización de los conceptos incluidos dentro del concepto seleccionado. También mueve la cámara hacia el concepto seleccionado.
- **Movimiento del ratón**: creación de un pop-up que indica a qué concepto se está apuntando siempre y cuando esto sea posible.

Para la manipulación de los elementos:

- **Botón izquierdo en un elemento:** activa los gizmos para poder ser manipulados.
- **Botón izquierdo fuera del elemento**: desactiva los gizmos que permitían manipular la posición.
- **Tecla 1**: selecciona el gizmo que permite trasladar el elemento.
- **Tecla 2**: selecciona el gizmo que permite rotar el elemento.

Controles adicionales (visualizar líneas, topología, objetos, terreno, etc.):

- **Tecla R**: activa o desactiva la visualización de las líneas que indican las relaciones topológicas.
- **Tecla H**: activa o desactiva la visualización de las líneas que indican la altura de los elementos, respecto a la base de la capa.
- **Tecla G**: activa o desactiva la posibilidad de utilizar gizmos.
- **Tecla I**: activa o desactiva la visualización del conjunto de imágenes.
- **Tecla T**: activa o desactiva la visualización del modelo 3D del terreno, en caso de que exista.
- **Tecla O**: activa o desactiva la visualización de todos los modelos 3D de los elementos.

- **Tecla C**: activa o desactiva la visualización del elemento seleccionado previamente

6.3 Tablas de referencia para el modelo COCOMO

En la Tabla 19 se presentan las constantes asociadas a cada atributo utilizado para el cálculo del multiplicador en el modelo COCOMO, mientras que la Tabla 20 muestra los valores asociados al tipo de proyecto.

Atributos	Muy bajo	Bajo	Nominal	Alto	Muy alto	Extra alto
Fiabilidad	0,75	0,88	1,00	1,15	1,40	-
Tamaño de la base de datos	-	0,94	1,00	1,08	1,16	-
Complejidad	0,70	0,85	1,00	1,15	1,30	1,65
Restricciones de tiempo de ejecución	-	-	1,00	1,11	1,30	1,66
Restricciones de memoria virtual	-	-	1,00	1,06	1,21	1,56
Volatilidad de la máquina virtual	-	0,87	1,00	1,15	1,30	-
Tiempo de respuesta	-	0,87	1,00	1,07	1,15	-
Capacidad de análisis	1,46	1,19	1,00	0,86	0,71	-
Experiencia en la aplicación	1,29	1,13	1,00	0,91	0,82	-
Calidad de los programadores	1,42	1,17	1,00	0,86	0,70	-
Experiencia en la máquina virtual	1,21	1,10	1,00	0,90	-	-

Experiencia en el lenguaje	1,14	1,07	1,00	0,95	-	-
Técnicas actualizadas de programación	1,24	1,10	1,00	0,91	0,82	-
Utilización de herramientas de software	1,24	1,10	1,00	0,91	0,83	-
Restricciones de tiempo de desarrollo	1,22	1,08	1,00	1,04	1,10	-

Tabla 19: Valores de los quince modificadores del modelo COCOMO

Tipo de proyecto	a	b	c	d
Orgánico	3,20	1,05	2,5	0,38
Semilibre	3	1,12	2,5	0,35
Empotrado	2,80	1,20	2,5	0,33

Tabla 20: Constantes asociadas al tipo de proyecto en el modelo COCOMO

7 DEFINICIONES Y ABREVIATURAS

En este apartado se describe la terminología empleada a lo largo del presente documento con el fin de aclarar algunos conceptos y evitar confusiones.

7.1 Abreviaturas

3D	Tres dimensiones
2D	Dos dimensiones
LiDAR	Light Detection and Ranging
API	Application Programming Interface
REST	Representational State Transfer
RV / VR	Realidad Virtual
RA / AR	Realidad Aumentada
SIG	Sistema de Información Geográfica
C#	C Sharp
IDE	Integrated Development Environment
XP	eXtreme Programming
COCOMO	COnstructive COst Model
JSON	JavaScripts Object Notation
OGC	Open Geospatial Consortium
DE-9IM	Dimensionally Extended 9-Intersection Model
UTM	Universal Transverse Mercator
URL	Uniform Resource Locator
QR	Quick Response
GPS	Global Positioning System
WGS84	World Geodetic System 1984
PDO	PHP Data Objects

DAO	Data Access Object
DEM	Digital Elevation Model
CSV	Comma Separated values
LIFO	Last In, First Out
LWRP	Lightweight Render Pipeline
HDRP	High Definition Render Pipeline
WKT	Well Know Text
CGAL	Computational Geometry Algorithms Library
ToF	Time of Flight
FPS	Frames Per Second

7.2 Definiciones

Fotogrametría	Conjunto de técnicas que, mediante la toma de imágenes, estudia la forma, dimensión y posición de un objeto.
Realidad Virtual	Representación de escenas o imágenes de objetos producidos por un sistema informático.
Realidad Aumentada	Inclusión de elementos virtuales sobre una representación de la realidad física.
Script	Pequeño fichero que implementa un comportamiento escrito en un lenguaje de programación.
Asset	Cualquier elemento que se puede utilizar en un proyecto de Unity: modelo 3D, imagen, sonido, etc.
Conceptos principales	Los conceptos principales del yacimiento arqueológico son: áreas, volúmenes, capas y elementos.
Prefab	Objeto reutilizable creados con una serie de características que puede ser instanciado en el proyecto tantas veces como sea oportuno.

Corrutina	Función que tiene la habilidad de pausar su ejecución y devolver el control a Unity. Comprende una forma de hacer que la lógica suceda con el tiempo.
Shader	Pequeño programa que realiza cálculos gráficos, generalmente para sombrear una escena.
Rendering pipeline	Conjunto de operaciones que permiten obtener una imagen a partir de modelos 2D o 3D.

8 BIBLIOGRAFÍA

Amad-ud-Din, Halin, I. A., & Shafie, S. B. (2009). A review on Solid State time of flight TOF range image sensors. *2009 IEEE Student Conference on Research and Development (SCOReD)*, 246-249. https://doi.org/10.1109/SCORED.2009.5443066

Armstrong, B. J., Blackwood, A. F., Penzo-Kajewski, P., Menter, C. G., & Herries, A. I. R. (2018). Terrestrial laser scanning and photogrammetry techniques for documenting fossil-bearing palaeokarst with an example from the Drimolen Palaeocave System, South Africa. *Archaeological Prospection*, *25*(1), 45-58. https://doi.org/10.1002/arp.1580

Austin, A. (2014). Mobilizing Archaeologists: Increasing the Quantity and Quality of Data Collected in the Field with Mobile Technology. *Advances in Archaeological Practice*, *2*, 13-23. https://doi.org/10.7183/2326-3768.2.1.13

Aziz, A. (2012). *Brief comparison of SDLC models*.

Bekele, M. K., Pierdicca, R., Frontoni, E., Malinverni, E. S., & Gain, J. (2018). A Survey of Augmented, Virtual, and Mixed Reality for Cultural Heritage. *Journal on Computing and Cultural Heritage*, *11*(2), 7:1-7:36. https://doi.org/10.1145/3145534

Benko, H., Ishak, E. W., & Feiner, S. (2004). Collaborative mixed reality visualization of an archaeological excavation. *Third IEEE and ACM International Symposium on Mixed and Augmented Reality*, 132-140. https://doi.org/10.1109/ISMAR.2004.23

Caggiani, M. C., Ciminale, M., Gallo, D., Noviello, M., & Salvemini, F. (2012). Online non destructive archaeology; the archaeological park of Egnazia (Southern Italy) study case. *Journal of Archaeological Science*, *39*(1), 67-75. https://doi.org/10.1016/j.jas.2011.08.017

Carr, M., & Verner, J. (2021). *Prototyping and Software Development Approaches*.

Clementini, E., Di Felice, P., & Oosterom, P. (1993). *A Small Set of Formal Topological Relationships Suitable for End-User Interaction. 692*, 277-295. https://doi.org/10.1007/3-540-56869-7_16

Cosmas, J., Itegaki, T., Green, D., Joseph, N., Gool, L. V., Zalesny, A., Vanrintel, D., Leberl, F., Grabner, M., Schindler, K., Karner, K., Gervautz, M., Hynst, S., Waelkens, M., Vergauwen, M., Pollefeys, M., Cornelis, K., Vereenooghe, T., Sablatnig, R., … Meyns, E. (2003). *Providing Multimedia Tools for Recording, Reconstruction, Visualisation and Database Storage/Access of Archaeological Excavations*. The Eurographics Association. https://doi.org/10.2312/VAST/VAST03/165-174

D'Urso, M. G., Corsi, E., & Corsi, C. (2018). Mapping of archaeological evidences and 3D models for the historical reconstruction of archaeological sites. *2018 Metrology for Archaeology and Cultural Heritage (MetroArchaeo)*, 437-442. https://doi.org/10.1109/MetroArchaeo43810.2018.9089783

Eltner, A., & Sofia, G. (2020). Chapter 1—Structure from motion photogrammetric technique. En P. Tarolli & S. M. Mudd (Eds.), *Developments in Earth Surface Processes* (Vol. 23, pp. 1-24). Elsevier. https://doi.org/10.1016/B978-0-444-64177-9.00001-1

Ferrari, L., & Prizozzi, E. (2020). *Learn PostgreSQL*. Packt. https://www.packtpub.com/product/learn-postgresql-12/9781838985288

Galeazzi, F. (2016). Towards the definition of best 3D practices in archaeology: Assessing 3D documentation techniques for intra-site data recording. *Journal of Cultural Heritage*, *17*, 159-169. https://doi.org/10.1016/j.culher.2015.07.005

Galeazzi, F., Callieri, M., Dellepiane, M., Charno, M., Richards, J., & Scopigno, R. (2016). Web-based visualization for 3D data in archaeology: The ADS 3D viewer. *Journal of Archaeological Science: Reports*, *9*, 1-11. https://doi.org/10.1016/j.jasrep.2016.06.045

Griffo, M., Cimadomo, P., & Menconero, S. (2019). INTEGRATIVE IRT FOR DOCUMENTATION AND INTERPRETATION OF ARCHAEOLOGICAL STRUCTURES. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *XLII-2-W15*, 533-539. https://doi.org/10.5194/isprs-archives-XLII-2-W15-533-2019

Jensen, P. (2018). Semantically Enhanced 3D: A Web-based Platform for Spatial Integration of Excavation Documentation at Alken Enge, Denmark. *Journal of Field Archaeology*, *43*(sup1), S31-S44. https://doi.org/10.1080/00934690.2018.1510299

Katsianis, M., Kotsakis, K., & Stefanou, F. (2021). Reconfiguring the 3D excavation archive. Technological shift and data remix in the archaeological project of Paliambela Kolindros, Greece. *Journal of Archaeological Science: Reports*, *36*, 102857. https://doi.org/10.1016/j.jasrep.2021.102857

Koller, D., Frischer, B., & Humphreys, G. (2010). Research challenges for digital archives of 3D cultural heritage models. *Journal on Computing and Cultural Heritage*, *2*(3), 7:1-7:17. https://doi.org/10.1145/1658346.1658347

Landeschi, G. (2019). Rethinking GIS, three-dimensionality and space perception in archaeology. *World Archaeology*, *51*(1), 17-32. https://doi.org/10.1080/00438243.2018.1463171

Lucas, G. (2012, febrero). *Understanding the Archaeological Record*. Cambridge Core; Cambridge University Press. https://doi.org/10.1017/CBO9780511845772

Manual Unity. (s. f.-a). *Unity - Manual: Render pipelines*. Recuperado 21 de junio de 2021, de https://docs.unity3d.com/Manual/render-pipelines.html

Manual Unity. (s. f.-b). *Unity - Scripting API: Gizmos*. Recuperado 21 de junio de 2021, de https://docs.unity3d.com/ScriptReference/Gizmos.html

Martínez-Fernández, A., Benito-Calvo, A., Campaña, I., Ortega, A. I., Karampaglidis, T., Bermúdez de Castro, J. M., & Carbonell, E. (2020). 3D monitoring of Paleolithic archaeological excavations using terrestrial laser scanner systems (Sierra de Atapuerca, Railway Trench sites, Burgos, N Spain). *Digital Applications in Archaeology and Cultural Heritage*, *19*, e00156. https://doi.org/10.1016/j.daach.2020.e00156

Meghini, C., Niccolucci, F., Felicetti, A., Ronzino, P., Nurra, F., Papatheodorou, C., Gavrilis, D., Theodoridou, M., Doerr, M., Tudhope, D., Binding, C., Scopigno, R., Vlachidis, A., Richards, J., Wright, H., Geser, G., Cuy, S., Fihn, J., Fanini, B., & Hollander, H. (2017). ARIADNE: A research infrastructure for archaeology. *Journal on Computing and Cultural Heritage*, *10*, 1-27. https://doi.org/10.1145/3064527

Obe, R. O., & Hsu, L. S. (2015). *PostGIS in Action*.

Ohori, K. A., Ledoux, H., & Stoter, J. (2015). An evaluation and classification of nD topological data structures for the representation of objects in a higher-dimensional GIS. *International Journal of Geographical Information Science*, *29*(5), 825-849. https://doi.org/10.1080/13658816.2014.999683

Okita, A. (2014). *Learning C# Programming with Unity 3D*.

PostgreSQL. (2021, mayo 13). *Inheritance*. PostgreSQL Documentation. https://www.postgresql.org/docs/11/ddl-inherit.html

Power, C., Lewis, A., Petrie, H., Green, K., Richards, J., Eramian, M., Chan, B., Walia, E., Sijaranamual, I., & Rijke, M. D. (2017). Improving Archaeologists’ Online Archive Experiences Through User-Centred Design. *Journal on Computing and Cultural Heritage*, *10*(1), 3:1-3:20. https://doi.org/10.1145/2983917

Pree, W. (1995). Design Patterns for Object-Oriented Software Development. *ACM Press, Addison-Wesley*. http://papers.cumincad.org/cgi-bin/works/paper/990b

Richards-Rissetto, H. (2017). What can GIS + 3D mean for landscape archaeology? *Journal of Archaeological Science*, *84*, 10-21. https://doi.org/10.1016/j.jas.2017.05.005

Ridel, B., Reuter, P., Laviole, J., Mellado, N., Couture, N., & Granier, X. (2014). The Revealing Flashlight: Interactive Spatial Augmented Reality for Detail Exploration of Cultural Heritage Artifacts. *Journal on Computing and Cultural Heritage*, *7*(2), 6:1-6:18. https://doi.org/10.1145/2611376

Saravia, G., & Elizabeth, L. (2012). *Metodologías ágiles y desarrollo basado en conocimiento* [Tesis, Universidad Nacional de La Plata]. http://sedici.unlp.edu.ar/handle/10915/24942

Seifert, C., Bailer, W., Orgel, T., Gantner, L., Kern, R., Ziak, H., Petit, A., Schlötterer, J., Zwicklbauer, S., & Granitzer, M. (2017). Ubiquitous Access to Digital Cultural Heritage. *Journal on Computing and Cultural Heritage*, *10*(1), 4:1-4:27. https://doi.org/10.1145/3012284

Strobl, C. (2016). *Dimensionally Extended Nine-Intersection Model (DE-9IM)* (pp. 1-7). https://doi.org/10.1007/978-3-319-23519-6_298-2

Talbot, J. (2017, julio 9). *What's right with risk matrices?* Juliantalbot. https://www.juliantalbot.com/post/2018/07/31/whats-right-with-risk-matrices

Universidad del Pais Vasco. (2021, junio 10). *El Modelo COCOMO*. http://www.sc.ehu.es/jiwdocoj/mmis/cocomo.htm

Wheatley, D., & Gillings, M. (2013). *Spatial Technology and Archaeology: The Archaeological Applications of GIS*. CRC Press.

Zhou, M., & Guan, Q. (2019). A 25-Intersection Model for Representing Topological Relations between Simple Spatial Objects in 3-D Space. *ISPRS International Journal of Geo-Information*, *8*(4), 182. https://doi.org/10.3390/ijgi8040182

Zlatanova, S., Rahman, A. A., & Shi, W. (2004). Topological models and frameworks for 3D spatial objects. *Computers & Geosciences*, *30*(4), 419-428. https://doi.org/10.1016/j.cageo.2003.06.004