



UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

Desarrollo de un gemelo digital para la plantación solar UJA

Alumno: Juan Carlos Fernández Pérez

Tutor: Juan Manuel Jurado Rodríguez

Juan Roberto Jiménez Pérez

Dpto: Informática



UNIVERSIDAD DE JAÉN

D./D^a Juan Manuel Jurado Rodríguez y D./D^a Juan Roberto Jiménez Pérez, tutor(es) del Trabajo Fin de Grado titulado: **Desarrollo de un gemelo digital para la plantación solar UJA**, que presenta Juan Carlos Fernández Pérez, autoriza(n) su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, junio de 2022

El estudiante

Los tutores

FERNANDEZ
PEREZ JUAN
CARLOS -
77688092A

Digitally signed
by FERNANDEZ
PEREZ JUAN
CARLOS -
77688092A
Date: 2022.09.09
18:57:05 +02'00'

Juan Carlos Fernández Pérez

JIMENEZ
PEREZ
JUAN
ROBERTO -
44259364G

Firmado
digitalmente por
JIMENEZ PEREZ
JUAN ROBERTO -
44259364G
Fecha:
2022.09.09
18:39:30 +02'00'

Juan Manuel Jurado
Rodríguez

Juan Roberto Jiménez
Pérez

Agradecimientos

Estas palabras van dirigidas a las personas que me han acompañado durante el transcurso del proyecto, aportando ánimos y esperanza en la finalización de mis estudios de grado en la Universidad de Jaén.

En especial, quiero agradecer a mi madre y mi hermana por convivir junto a mí durante todos esos momentos complicados, y a mis compañeros de estudios José Antonio Collado Araque y Manuel Germán Morales por dar todo su apoyo y ánimos siempre que se les necesita.

Agradecer con creces también la paciencia y la labor de Juan Manuel Jurado Rodríguez y Juan Roberto Jiménez Pérezal aportar sus conocimientos y experiencia, que me han permitido poner punto y final a este trabajo y a este capítulo de mi vida.

Finalmente, dar las gracias a todos aquellos familiares y amigos que también han aportado su confianza y consejos.

Tabla de contenidos

1. INTRODUCCIÓN	1
1.1. Motivación	1
1.2. Propósito del proyecto	2
1.2.1. Objetivos	2
1.3. Estructuración del documento	2
2. ANTECEDENTES	5
2.1. Gemelo digital en la actualidad	7
2.2. Tecnologías	10
2.3. Herramientas	13
2.4. Ramas de investigación abiertas	14
2.5. Gemelos digitales para plantaciones solares	18
2.6. Gráficos por ordenador	20
3. ESPECIFICACIÓN DEL TRABAJO	23
3.1. Situación de partida	23
3.2. Alternativas propuestas y viabilidad	24
3.3. Descripción de la solución propuesta	27
3.3.1. Análisis de alternativas	28

3.4. Tecnologías utilizadas	29
3.4.1. Desarrollo web	30
3.4.2. Desarrollo para escritorio	31
3.4.3. Herramientas adicionales	32
3.5. Metodología de desarrollo software	33
3.6. Estimación del proyecto	34
3.6.1. Tamaño y esfuerzo	34
3.6.2. Planificación temporal	36
3.6.3. Presupuesto	36
4. DISEÑO INICIAL	39
4.1. Especificación de requerimientos	39
4.1.1. Requerimientos funcionales	40
4.1.2. Requerimientos no funcionales	40
4.2. Análisis del sistema	41
4.2.1. Perfiles de usuario	41
4.2.2. Casos de uso	42
4.2.3. Modelo de dominio	47
4.3. Diseño del sistema	48
4.3.1. Diagrama de clases de diseño	48
4.3.2. Diagramas de secuencia	49
4.4. Diseño de interfaz e interacción	52
4.4.1. Interacción	52
4.4.2. Interfaz gráfica de usuario	53

4.4.3. Experiencia de usuario (UX)	53
4.4.4. Prototipos de interfaz gráfica	54
5. DESARROLLO	57
5.1. Incremento 0: Desarrollo web	57
5.1.1. Especificación del incremento	58
5.1.2. Implementación de la funcionalidad	58
5.1.3. Pruebas de funcionamiento	63
5.1.4. Conclusión	71
5.2. Incremento 1: Área de dibujado	72
5.2.1. Especificación del incremento	72
5.2.2. Implementación de la funcionalidad	74
5.2.3. Pruebas y consideraciones	77
5.3. Incremento 2: Visualización de modelos	79
5.3.1. Especificación del incremento	79
5.3.2. Implementación de la funcionalidad	80
5.3.3. Pruebas y consideraciones	81
5.4. Incremento 3: Jerarquía de información	83
5.4.1. Especificación del incremento	83
5.4.2. Implementación de la funcionalidad	83
5.4.3. Pruebas y consideraciones	85
5.5. Incremento 4: Carga de nubes de puntos	87
5.5.1. Especificación del incremento	88
5.5.2. Implementación de la funcionalidad	88

5.5.3. Pruebas y consideraciones	93
6. CONCLUSIONES	97
6.1. Objetivos de trabajo futuro	98
7. APÉNDICES	99
7.1. Instalación y compilación del proyecto	99
Bibliografía	III

Lista de figuras

2.1. Esquema básico del concepto de gemelo digital. Adaptación de la figura original de Grieves [2014] “Figure 1”	6
2.2. Esquema de conceptos confundibles al gemelo digital.	8
2.3. Asociación de tecnologías (a) y casos de uso (b) en el ciclo del gemelo digital. Extraído de Jones et al. [2020], Fig.9.	9
2.4. Esquema de las tecnologías facilitadoras del gemelo digital.	13
2.5. Evolución de la industria con la inserción de nuevas tecnologías	15
3.1. Esquema de la metodología de desarrollo incremental	33
3.2. Diagrama de Gantt	37
4.1. Ejemplo de diagrama de frontera	43
4.2. Diagrama de casos de uso “Generación de la escena 3D”.	45
4.3. Diagrama de casos de uso “Virtualización de una instalación adicional”.	45
4.4. Diagrama de casos de uso “Carga de datos”.	46
4.5. Diagrama de casos de uso “Consulta de información”.	46
4.6. Diagrama <i>UML</i> del modelo de dominio	47
4.7. Diseño inicial de clases del sistema	49
4.8. Diagrama de secuencia del caso de uso “Generación de la escena 3D”	50

4.9. Diagrama de secuencia del caso de uso “Virtualización de una instalación adicional”	51
4.10. Diagrama de secuencia del caso de uso “Carga de datos”.	51
4.11. Diagrama de secuencia del caso de uso “Consulta de información”. . .	52
4.12. <i>sketch</i> de la ventana principal de la aplicación	54
4.13. <i>sketch</i> del diálogo para añadir una región	55
4.14. <i>sketch</i> del diálogo para añadir un inversor	55
4.15. <i>sketch</i> del diálogo para añadir un panel a un inversor	56
4.16. <i>sketch</i> del diálogo para cargar una nube de puntos	56
5.1. Aplicación web en funcionamiento	62
5.2. Diagrama de clases del Incremento 0: Desarrollo web	63
5.3. Escena empleada para las pruebas web	64
5.4. Métricas de rendimiento en Opera GX	65
5.5. Estadísticas de rendimiento de <i>BabylonJS</i> en Opera GX	66
5.6. Métricas de rendimiento en Firefox	67
5.7. Estadísticas de rendimiento de <i>BabylonJS</i> en Firefox	68
5.8. Métricas de rendimiento en Microsoft Edge	69
5.9. Estadísticas de rendimiento de <i>BabylonJS</i> en Microsoft Edge	70
5.10. Diagrama de clases inicial del incremento 1	73
5.11. Primitivas de dibujo de triángulos en <i>OpenGL</i>	76
5.12. Aplicación en funcionamiento tras el incremento 1	76
5.13. Diagrama de clases final del incremento 1	78
5.14. Caso de visualización incorrecta de figura triangular	79
5.15. Aplicación en funcionamiento tras el incremento 2	81

5.16. Diagrama de clases final del incremento 2	82
5.17. Esquema de la jerarquía de información	84
5.18. Aplicación en funcionamiento tras el incremento 3	85
5.19. Diagrama de clases final del incremento 3	86
5.20. Ejemplo de generación de rectángulos mediante el vector perpendicular	89
5.21. Ejemplo de la estructura de datos espacial <i>kd-tree</i> en dos dimensiones	91
5.22. Visualización de nube de puntos tras el incremento 4	92
5.23. Consulta de información cargada tras el incremento 4	92
5.24. Diagrama de clases final del incremento 4	94
5.25. Comparación del rendimiento tras cargar dos nubes de puntos. Izquier- da: 4 millones de puntos. Derecha: 30 millones de puntos.	95

Capítulo 1

INTRODUCCIÓN

El término “Gemelo Digital”, al ser relativamente reciente, se encuentra en su fase de evolución y expansión, captando cada vez más el oído de empresas e investigadores de diversos sectores. Esta etapa continúa avanzando hoy en día a un ritmo muy acelerado gracias a la gran ventaja que ofrece aplicar el concepto a los desarrollos de sistemas complejos, de ahí que sea compatible con sectores variados, aunque no se relacionen de ninguna manera. A pesar de la existencia de multitud de definiciones del concepto según el área concreta de aplicación, una forma de concretar, a nivel más o menos general, en qué consiste un gemelo digital podría ser: la combinación de un sistema físico, existente en el mundo real, y un sistema virtual con la mayor fidelidad posible generado a partir de ese sistema real, añadiendo además una interconexión entre ambos que los mantenga sincronizados.

1.1. Motivación

Una de las grandes motivaciones del trabajo viene dada por el interés global sobre el aprovechamiento de energías renovables, siendo cada vez una alternativa más viable frente a las fuentes clásicas mucho más perjudiciales para el medio ambiente. Cada vez son más los recintos públicos y privados que optan por instalar plantaciones basadas en generadores eólicos, solares o hidráulicos, dependiendo de la geografía de la región. El ámbito de este trabajo se reserva exclusivamente a la producción solar, ya que recientemente la Universidad de Jaén tomó como objetivo cubrir, al menos, un 16 % del consumo energético mediante generación solar fotovoltaica¹.

¹<https://www.ujaen.es/servicios/utecnica/proyecto-de-instalacion-para-la-produccion-de-energia-solar-fotovoltaica-2020>

Considerando también el interés actual sobre la investigación y desarrollo del concepto de gemelo digital, la segunda motivación se basa en aprovechar las ventajas que ofrece construyendo un modelo virtual de la instalación solar de la Universidad de Jaén, aportando mayor visibilidad al uso de un gemelo digital.

1.2. Propósito del proyecto

El objetivo final del trabajo será, por tanto, el diseño e implementación de un gemelo digital basado en la plantación solar de la Universidad de Jaén, capaz de recibir un conjunto de datos de entrada relativos al funcionamiento del sistema real. El modelo virtual se construirá en un entorno tridimensional mediante una aplicación gráfica interactiva, habilitando la navegación de la escena recreada. Como punto diferenciador sobre otros enfoques del gemelo digital, los esfuerzos se centrarán en el desarrollo de la aplicación gráfica en lugar de la simulación del comportamiento del sistema virtualizado, aprovechando técnicas de visualización, estructuras de datos espaciales y algoritmos geométricos con el fin de maximizar la fidelidad del modelo sin comprometer su rendimiento y usabilidad.

1.2.1. Objetivos

- Investigación y revisión bibliográfica.
- Análisis, diseño y desarrollo de una aplicación software para la generación de modelos virtuales para la simulación.
- Generación y ejemplo de uso de la aplicación sobre la instalación fotovoltaica de la Universidad de Jaén.
- Uso de la aplicación para generar un modelo virtual sobre la instalación fotovoltaica de la Universidad de Jaén.
- Redacción de la memoria del proyecto.

1.3. Estructuración del documento

A continuación se detallan los diferentes capítulos que forman el presente documento mediante una breve descripción de sus contenidos:

1. INTRODUCCIÓN

Presentación y descripción básica del trabajo fin de grado, donde se justifica el desarrollo, motivación y objetivos a alcanzar del mismo.

2. ANTECEDENTES

Visión general sobre el estado actual del concepto de gemelo digital, indagando en los múltiples avances tecnológicos, metodologías, alternativas... que han surgido a lo largo de los últimos años.

3. ESPECIFICACIÓN DEL TRABAJO

Descripción del trabajo a realizar evaluando la situación actual, las alternativas posibles y la presentación de la solución propuesta, así como las herramientas y metodología para el desarrollo del mismo.

4. DISEÑO INICIAL

Especificación de aspectos relacionados con la arquitectura del sistema software planteado y los modelos de diseño para las diversas partes que lo componen.

5. DESARROLLO

Especificación del desarrollo del trabajo y descripción de métodos y técnicas de interés, así como la documentación de las pruebas realizadas.

6. CONCLUSIONES

Recopilación de opiniones y análisis del trabajo tras su finalización.

7. APÉNDICES

Documentación y guías adicionales de utilidad.

8. Bibliografía

Especificación de la bibliografía empleada a lo largo del desarrollo del trabajo.

Capítulo 2

ANTECEDENTES

Desde hace años, son innumerables los avances y mejoras sobre diversas tareas, tanto de la vida cotidiana como de sectores más específicos que la tecnología propone cada poco tiempo; cada vez más rápidos, pues grandes beneficios implican mayores inversiones en investigación y desarrollo. Entre todos esos avances se distingue un subconjunto más enfocado a la industria, facilitando el diseño y producción de bienes y recursos, además de reducir los costes y aumentando la fiabilidad del producto final. El mayor problema que presenta este subconjunto también es común en otros enfoques: el entorno en que se realizan es altamente complejo y requiere gran capacidad por parte de los administradores para formar un grupo de trabajo capaz de afrontar eficientemente la producción.

Es ante este problema donde aparece la idea de recrear ese entorno de forma que se facilite la navegación y control sobre el mismo, para que posteriormente puedan aplicarse mejoras al original sin necesidad de tomar riesgos excesivos. [Grieves \[2014\]](#) presenta una reseña sobre el concepto de gemelo digital y sus diversos usos a lo largo de su evolución. De primeras, la idea se implementó duplicando el entorno: en concreto, la NASA preparó una réplica del vehículo espacial empleado en su programa *Apollo* con el fin de que recrease el entorno que suponía el vehículo original una vez en el espacio y poder actuar de acuerdo a su estado.

Poco después, comenzaron las propuestas hacia una digitalización de ese gemelo físico, pues los avances en computación se vieron catapultados, permitiendo réplicas virtuales de alta fidelidad con un coste mucho menor que la contraparte física.

El concepto que hoy en día conocemos de gemelo digital fue también presentado por Grieves en 2003, siguiendo el esquema de la figura [2.1](#), aunque en esta época no se disponía de suficiente investigación y desarrollo en cuanto a la producción de soft-

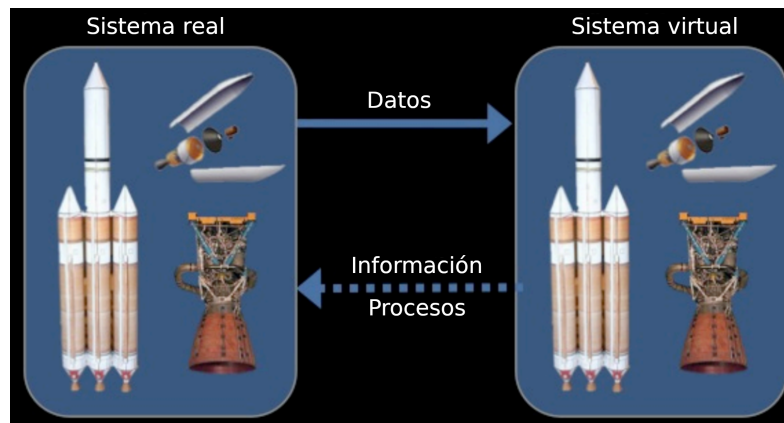


Figura 2.1: Esquema básico del concepto de gemelo digital. Adaptación de la figura original de Grieves [2014] “Figure 1”.

ware para la virtualización y recolección de datos del entorno real. A pesar de esto, tras varios años de inversión, las posibilidades fueron en aumento, y comenzaron a surgir artículos centrados en destacar este concepto de gemelo digital y su evolución. La revisión de Grieves [2014] al transcurrir una década desde que presentó el concepto, extiende su definición a más casos de uso y define precisamente los requisitos de su desarrollo.

Indagando un poco en esta definición original, un gemelo digital queda dividido en tres partes: un sistema real o físico, seguido del sistema virtual (el sistema real virtualizado) y, finalmente, una interconexión entre ambos. En primer lugar, el **sistema real** corresponde al entorno o sistema a digitalizar, permitiendo aplicar modificaciones antes de alterarlo o realizar pruebas y estimaciones sobre su funcionamiento. El objetivo de esto es reducir los costes de implantar cambios que requieran construcción previa, evitar modificaciones del funcionamiento que empeoren el rendimiento, o controlar y facilitar el mantenimiento del sistema.

Junto al sistema real, se desarrolla un **sistema virtual** a modo de réplica con la mayor fidelidad posible, que permita además realizar las modificaciones y consultas necesarias para la gestión de ambos sistemas. Es en este punto donde entra la investigación de las ramas de informática gráfica, redes de ordenadores y más concretamente dispositivos de Internet de las Cosas, *IoT*. Por último, debe prepararse un canal de transmisión entre los dos sistemas que permita la **comunicación bidireccional** entre ambos puntos, recolectando la información del sistema real para poblar el virtual, y enviando señales y procesos de control desde el virtual al real con los cambios.

2.1. Gemelo digital en la actualidad

Gracias a lo prometedor que resultó ser el concepto de Grieves, poco a poco la investigación sobre el término de gemelo digital creció y sigue creciendo a un ritmo exponencial. Por esto, también han surgido muchas revisiones del estado del arte que buscan orientar los próximos puntos de investigación, aprovechando múltiples bases de datos con las publicaciones más recientes sobre las que realizan análisis sistemáticos en busca de los conceptos más interesantes y aquellos en que la investigación debe fijarse en mayor medida.

Para empezar, se destaca el trabajo de [Fuller et al. \[2020\]](#) al revisar y agrupar las tecnologías e investigación realizada sobre el concepto de gemelo digital, pues junto a esta revisión, presenta diferentes definiciones de mayor interés del concepto, seguido de una clarificación de conceptos fácilmente confundibles debido a la similitud que presentan, nombrados a continuación y representados en la figura [2.2](#):

- **Modelo digital:** La comunicación entre ambos modelos no es automática; los cambios aplicados en un extremo no se aplican en el contrario hasta que no actúe algún intermediario adicional.
- **Sombra digital:** La comunicación es direccional, desde el modelo físico hasta el virtual; los cambios del sistema real provocan que la virtualización se actualice para corresponder con la realidad.
- **Gemelo digital:** La comunicación es bidireccional entre los extremos; un cambio en un punto altera el sistema opuesto.

Adicionalmente, esta revisión presenta algunas de las áreas donde el gemelo digital está resultando ser de gran utilidad al complementarlo con los avances en *IoT* e inteligencia artificial (*IA*): para **ciudades inteligentes** se aprovecha el alto grado de conectividad que las redes y dispositivos *IoT* facilitan cada vez más, lo cual implica mayor fidelidad del gemelo desarrollado; en el ámbito de la **fabricación**, se consigue un seguimiento más preciso de los procesos que, al complementarse con mejores algoritmos de *IA*, simplifican la detección de problemas y facilitan predicciones y medidas de rendimiento del sistema en su totalidad; finalmente, en el campo de la **salud** todas las mejoras de la tecnología permiten que se hagan realidad ideas que antiguamente parecían imposibles (“...*the once impossible is becoming possible*.[...]”), con simulaciones y seguimientos que, además, puedan predecir la evolución del paciente a corto y largo plazo, contando adicionalmente otros escenarios que, a pesar de no estar directamente relacionados con el paciente, sigan aportando a su bienestar.

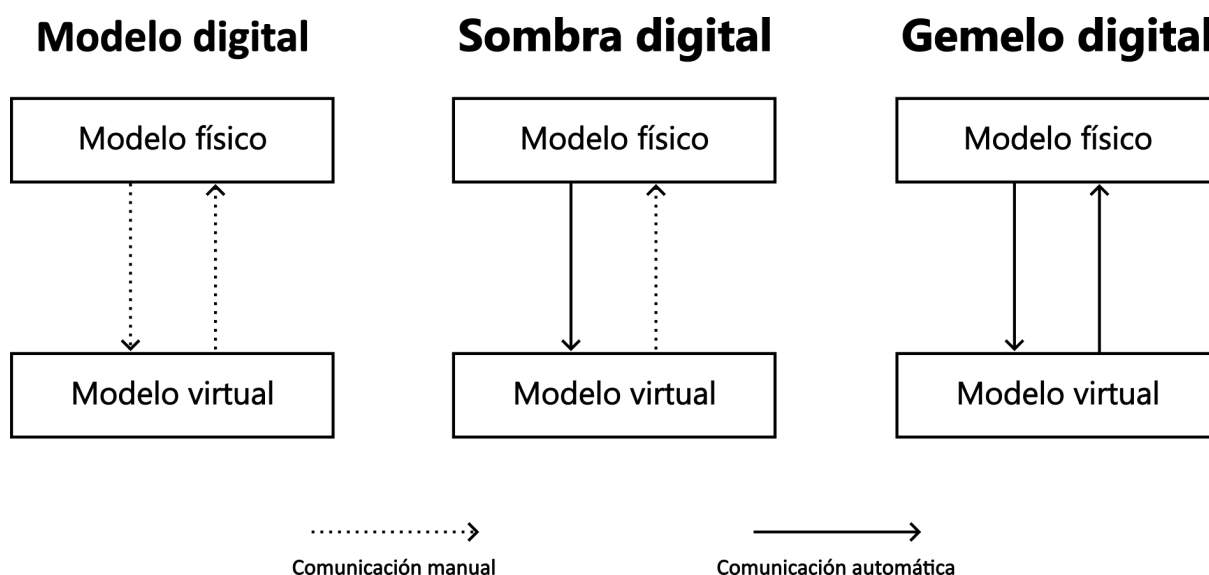


Figura 2.2: Esquema de conceptos confundibles al gemelo digital.

Continuando con otra exploración del gemelo digital, [Jones et al. \[2020\]](#) realiza una revisión sistemática de la literatura relacionada con el concepto, identificando las definiciones por las que ha evolucionado y dando como resultado una caracterización del gemelo digital. Entre todos los temas que encuentra relacionados con el concepto, un subconjunto representa las características básicas que definen al gemelo digital: la parte **física** se compone de una **entidad** (el propio elemento real que quiere virtualizarse), un **entorno** (donde reside la entidad física) y una serie de **procesos** (que realiza la entidad situada en el entorno); la parte **virtual** divide de igual manera en **entidad**, **entorno** y **procesos**; las conexiones **físico-virtual** y **virtual-físico** para el intercambio de información; el concepto de **fidelidad** como la cantidad de parámetros y su precisión en el intercambio; un **estado** correspondiente al valor de los **parámetros** de las dos partes en un instante concreto; y, por último, un proceso de **sincronización** entre ambos extremos que denomina *twinning*.

Partiendo de esta división en características, [Jones et al. \[2020\]](#) presenta también una asociación entre el esquema resultante para el gemelo digital (combinando los elementos anteriormente mencionados) y distintas tecnologías y casos de uso, colocados espacialmente según el ámbito en que se vean más involucrados. La figura 2.3 muestra esta asociación entre el gemelo digital y diversas tecnologías (a) y casos de uso (b). Cabe destacar el posicionamiento de los diferentes elementos; por ejemplo, la simulación o el manejo de datos se encuentran totalmente dentro de la parte virtual, mientras que las aplicaciones como fábricas inteligentes se posicionan completamente en la parte física, a la vez que otras utilidades quedan más centradas entre ambos.

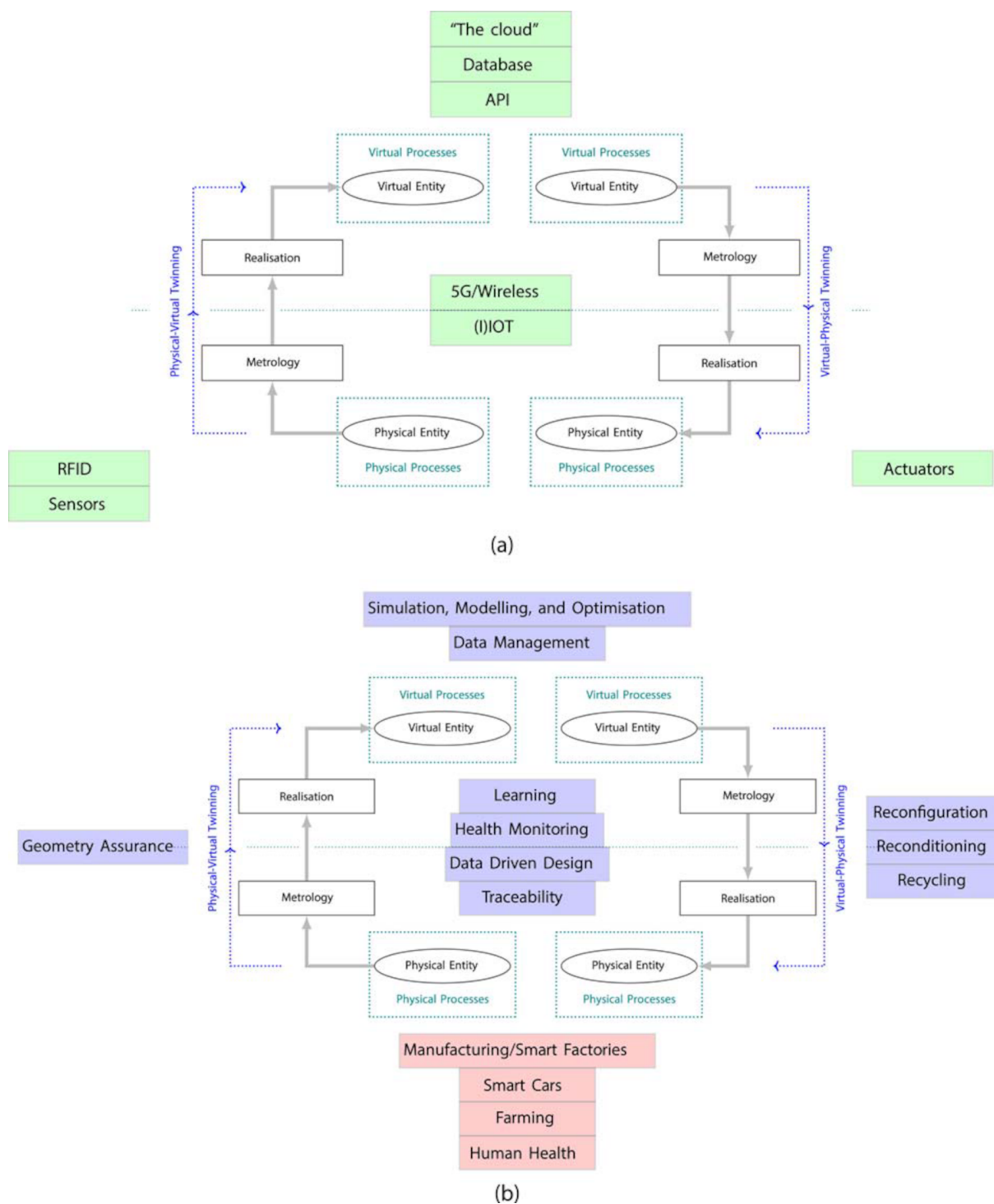


Figura 2.3: Asociación de tecnologías (a) y casos de uso (b) en el ciclo del gemelo digital. Extraído de Jones et al. [2020], Fig.9.

Siguiendo la línea de revisiones sistemáticas, [Semeraro et al. \[2021\]](#) aplica un análisis sobre las publicaciones de varias bases de datos, agrupando las definiciones, usos y tecnologías en los que el gemelo digital se ha visto involucrado los últimos años. Respecto a definiciones, logra formar cinco grupos suficientemente diferenciados según las palabras clave que aparecen en cada definición. Por esto, se reafirma que el concepto de gemelo digital aún se encuentra en fases tempranas, pues no existe una definición específica válida para las distintas aplicaciones que pueda tener; cada implementación hace uso del concepto según le sea necesario para resolver el problema concreto, desviándose en ocasiones de la idea general.

La visión global del estado del arte del gemelo digital muestra que, a pesar de ser un concepto aún en crecimiento y desarrollo, presenta un potencial sobre gran cantidad de ámbitos por el que muchos proyectos deciden adaptarlo de una forma u otra. Si bien es suficientemente amplio para poder implementarse en diversos proyectos, existen varias ramas que se han visto bastante más beneficiadas por su uso: el ámbito industrial quizá sea donde más destaca el gemelo digital, ya que la evolución de *IoT*, mejoras de algoritmos de *IA* y mejor gestión/análisis de información facilitan la labor de abaratar costes de producción al máximo.

Sería posible seguir analizando estudios y revisiones de la literatura que busquen especificar el concepto de gemelo digital (trabajos como por ejemplo los de [Liu et al. \[2021\]](#) o [Enders \[2019\]](#)), pero todos ellos llegan a la misma conclusión que el resto sin dar una definición específica y válida del concepto para las diferentes ramas de investigación abiertas.

2.2. Tecnologías

Tras revisar las publicaciones mencionadas y otras no mencionadas, el futuro general del concepto de gemelo digital es incierto, siendo necesario especificar un ámbito sobre el que analizar el desarrollo que ha experimentado durante los años. A pesar de esta separación, se hace posible generalizar un aspecto de cualquier gemelo digital: las tecnologías que hacen posible su implementación, destacando principalmente **IoT** y técnicas para el **análisis de datos**.

Internet de las Cosas

El concepto de Internet de las Cosas está cada vez más estandarizado y presente en el día a día de todo el mundo; ha supuesto grandes simplificaciones de tareas cotidianas y facilidades o comodidades al alcance de cualquiera. El objetivo del *IoT* es el de añadir un mecanismo de comunicación a todos los dispositivos comunes, presentes en cualquier espacio público o privado, e interconectarlos en una red gigante donde la información fluya de un punto a otro entre dispositivos.

De la misma forma que una red de ordenadores convencional, esto presenta claros problemas de seguridad y estabilidad. En términos de estabilidad, disponer de un gran número de dispositivos transmitiendo y recibiendo datos puede saturar el ancho de banda y provocar pérdidas de información esencial o retardos en datos temporales, pero, afortunadamente, nuevas versiones de protocolos o tecnologías, como recientemente la red inalámbrica **5G**, amplían enormemente los límites de las transmisiones. Por otro lado, la protección de los datos es un tema muy estudiado para las redes convencionales, y es posible aplicar técnicas de encriptado y protocolos de comunicación seguros como **HTTP** o **MQTT** con la protección añadida de **TLS**¹.

Al encontrarse solventados en gran medida estos problemas básicos, el uso del *IoT* sigue en un aumento cada vez mayor, y se pronosticaba que el número de dispositivos conectados se duplique en el próximo lustro, aunque debido a la situación de pandemia global provocada por la Covid-19, un alto porcentaje de empresas decidieron dejar de lado la inversión en tecnologías *IoT*; a pesar de ello, la mayoría continúan apostando por adoptar e instalar plataformas *IoT* (Lasse [2020], De Saulles [2021]).

De forma adicional, el éxito rotundo del *IoT* ha provocado especializaciones del concepto para entornos más específicos. El caso de mayor interés es el de *IoT* industrial, *IIoT*, donde la maquinaria de fabricación se interconecta para simplificar labores como la sincronización o el mantenimiento mediante acceso remoto, siempre con el objetivo central de maximizar la productividad y, con ello, los beneficios.

La utilidad que proporcionan los avances en *IoT* para el gemelo digital es trivial: a mayor capacidad de transmisión, mayor fidelidad podrá lograrse, resultando en mejores simulaciones y predicciones del comportamiento del sistema. También cabe mencionar la utilidad que ofrecen mejores dispositivos, pues si en lugar de simplemente sensores para recolección de datos se realiza una instalación controlable, se facilita la comunicación bidireccional entre ambos modelos que define a un gemelo digital.

¹ *TLS*, *Transport Layer Security*, se encarga de añadir una capa adicional en la transmisión de los datos aplicando algoritmos de cifrado al contenido para que únicamente el destinatario intencionado pueda acceder al mismo.

Análisis de datos

La segunda tecnología, aunque más bien un conjunto de varios métodos y técnicas, consiste en aprovechar toda la información obtenida mediante esos dispositivos *IoT* ya mencionados. En este ámbito surgen multitud de ramificaciones según el tratamiento que se realice y el objetivo final de la información, pero antes de proceder a aplicar alguna metodología sobre los datos, es necesario un proceso previo de preparación; ningún sensor garantiza precisión perfecta, ni las redes de dispositivos aseguran la ausencia de problemas. Al recopilar un conjunto de datos a analizar, debe realizarse una limpieza para eliminar información errónea, redundante o simplemente sin interés, que se haya podido generar durante la lectura o la transmisión. A continuación, los datos tendrán que ser preparados para el análisis según la metodología a utilizar, con el fin de facilitar esta fase.

Volviendo al trabajo de [Fuller et al. \[2020\]](#), menciona también varios campos dentro del análisis de datos que resultan apropiados para el gemelo digital. Tras indicar la ya mencionada **limpieza y filtrado** de la información recolectada, el siguiente punto que presenta es la **estadística**, que permite describir observaciones a partir del conjunto de datos disponible (a modo de ejemplo, sobre un gemelo digital de una línea de fabricación se podría extraer la proporción de productos en buen estado del total fabricado en un día, mes o año). Acto seguido, nombra la **inteligencia artificial** y, más concretamente, el ***Machine Learning***.

El *Machine Learning* tiene como objetivo la automatización de tareas mediante un algoritmo sin haber programado esa tarea. De esta forma, para el análisis de datos, una aplicación es capaz de procesar automáticamente un conjunto de entrada y devolver múltiples datos adicionales que lo caracterizan: desde predicciones sobre su contenido o lo que representan, hasta agrupaciones según diversos rasgos determinados por el usuario. En un nivel de complejidad mayor se encuentra el **Deep Learning**, capaz de analizar conjuntos mucho más complejos, incluso sin preparación previa, extrayendo rasgos comunes al conjunto de entrada haciendo uso de redes neuronales de múltiples niveles cada vez más complejas.

Finalmente, [Fuller et al. \[2020\]](#) nombra la **visualización de los datos** resultantes, pues la mayoría de ocasiones resulta complicado para el usuario comprender información relevante si se da en el formato inadecuado. Empleando gráficas, historiales, agrupaciones, o cualquier otro tipo de herramienta (tanto bidimensional como tridimensional), la comprensión de un enorme conjunto de resultados queda reducida a un simple vistazo, ante lo que el usuario podrá reaccionar instantáneamente frente a alguna incongruencia en los datos.

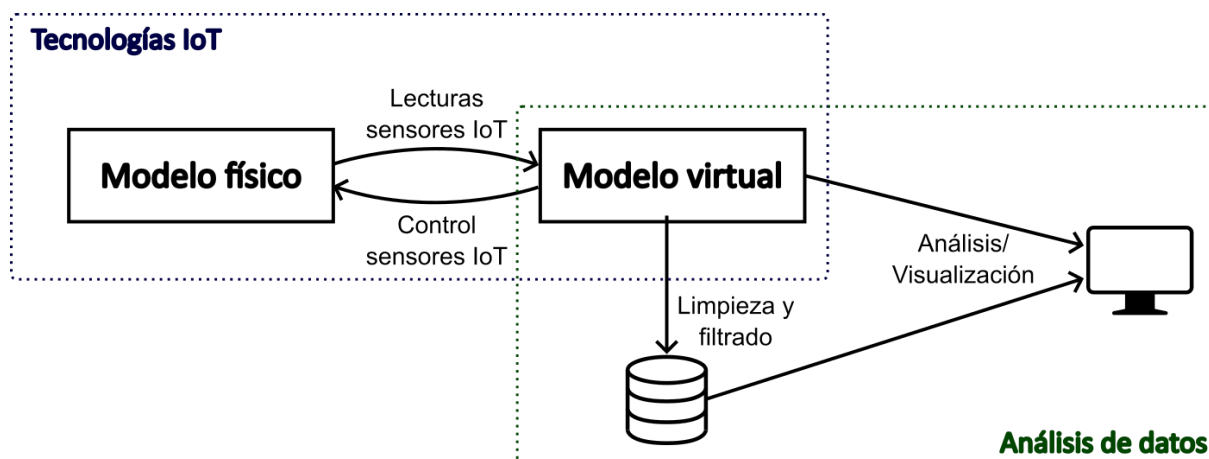


Figura 2.4: Esquema de las tecnologías facilitadoras del gemelo digital.

Todas estas técnicas son de interés para el gemelo digital, pues recolectará grandes cantidades de información según el grado de fidelidad intencionado. El análisis de datos suele parecer la parte más importante, ya que el objetivo de la mayoría de gemelos digitales es el simular o predecir el comportamiento de un sistema, para lo cual se requiere un análisis lo más preciso posible de la información procedente del sistema físico o real.

A modo de resumen, la figura 2.4 expone el esquema de un gemelo digital y las tecnologías *IoT* y de análisis de datos en el flujo de la información. Mediante los sensores, se extrae un conjunto de datos que permiten ajustar el modelo virtual a la realidad, pudiendo además controlar dichos sensores en el caso de que se requiera algún cambio (realizado primero en el modelo virtual). Tras esto, la información se prepara para ser almacenada en una base de datos donde finalmente el usuario puede acceder con el fin de visualizarla tras un análisis previo.

2.3. Herramientas

Siguiendo los requisitos de un gemelo digital en cuanto a tecnologías necesarias para un funcionamiento esencial y óptimo, el siguiente paso natural consiste en descubrir herramientas que habiliten el uso de estas tecnologías, idóneamente orientado a las necesidades adicionales de un gemelo digital.

Debido a ser un concepto basado principalmente en la simulación de una entidad real, la mayoría de herramientas de desarrollo se centran en la recolección y análisis de datos, mediante plataformas *IoT*, ya sea de forma local o en la nube. Por ejemplo, la plataforma **Microsoft Azure** pone a disposición del cliente un producto específico

para el desarrollo de gemelos digitales, basado en *IoT*, con la capacidad de comunicarse con el resto del ecosistema Azure, que habilita el almacenamiento, análisis y visualización de los datos resultantes.

A pesar de que predominan los enfoques orientados a *IoT*, otras herramientas buscan un soporte más híbrido con las capacidades de información espacial. Generalmente, las herramientas de este conjunto suelen ser motores gráficos o librerías sobre las cuales, tras generar el modelo tridimensional, se añade la conexión con otras dedicadas al análisis y recolección de datos.

Un ejemplo reciente de herramienta híbrida para el desarrollo de gemelos digitales es el motor gráfico **Unity 3D**, pues ha comenzado a dar soporte para las necesidades básicas de un gemelo digital oficialmente². De esta forma, se facilita la creación, visualización e interacción 3D en tiempo real (al ser uno de los motores gráficos más usados, principalmente para videojuegos, y de uso gratuito en proyectos con pocos ingresos anuales, consta de muchos recursos y foros de la comunidad con multitud de problemas comunes resueltos), permitiendo además enlazar la aplicación con otras herramientas especializadas o entornos como **Azure**, **Google Cloud** o **Amazon Web Services** para manejar los datos con los que se trate.

2.4. Ramas de investigación abiertas

En las secciones anteriores se exploró el concepto de gemelo digital, llegando a la conclusión de que el concepto se encuentra actualmente en evolución, motivo por el cual no se ha alcanzado una definición suficientemente genérica que abarque las necesidades de los diferentes campos donde puede aplicarse. Esta sección se destina a explorar por encima las ramas en las que existe mayor interés por los gemelos digitales, destacando los motivos que incentivan la investigación y desarrollo del concepto de cara a los años venideros.

Entre dichas ramas, se dedica la sección posterior 2.5 a los avances y la actualidad del concepto relacionados con la gestión de plantaciones solares, pues será el enfoque principal del proyecto a desarrollar.

²<https://unity.com/solutions/digital-twins>, comprobado en 2022-08-27.

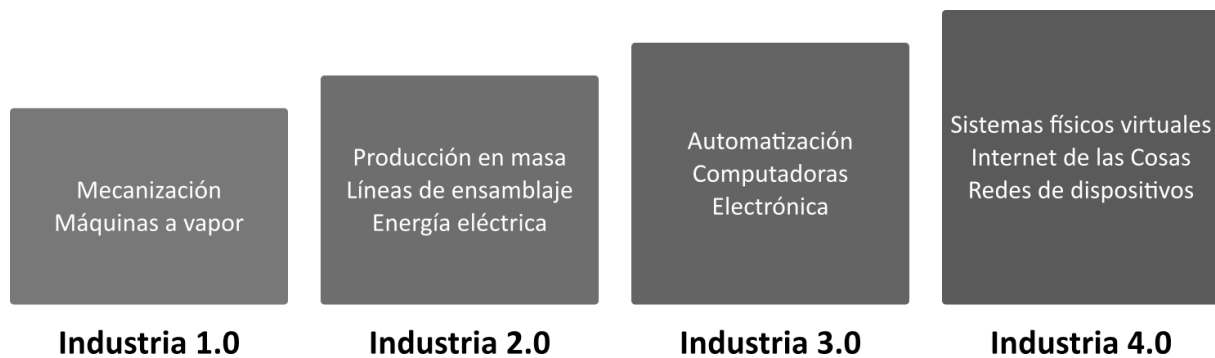


Figura 2.5: Evolución de la industria con la inserción de nuevas tecnologías

Industria 4.0

Anteriormente, en la sección 2.2, se ha explorado la actualidad de la tecnología *IoT* y el análisis de datos como habilitadoras del gemelo digital, pero ambas han sido de mayor utilidad en otros campos, como es el de la industria. El concepto de **Industria 4.0** se ha estandarizado en los últimos años al ser una clara mejora en cuanto al rendimiento para la producción, siendo así la cuarta revolución industrial dispuesta a cambiar en gran medida el paradigma tal como se conocía anteriormente (orientándolo a un entorno digital y lo más autónomo posible). La figura 2.5 muestra las mejoras tecnológicas que implicaron un cambio significativo en industria.

Lasi et al. [2014] describe los cambios que consideraba necesarios en la industria para dar el salto a la cuarta revolución desde dos puntos de vista: considerando los cambios constantes en el marco de trabajo y los avances tecnológicos que propician mejoras en toda la cadena de producción. A continuación, lista una serie de conceptos esenciales que definirán la Industria 4.0, principalmente los indicados en la figura 2.5, junto a otros más centrados en términos de organización o empresa. Finalmente, detalla el interés que perciben los negocios mediante esas mejoras (sobre todo integración con otros sectores, ramas, industrias...) y algunos campos donde podrá aplicarse a modo de ejemplo.

Comparando el concepto de gemelo digital con el de Industria 4.0, gran parte de las características de ambos son o bien similares, o directamente idénticas. Uno de los objetivos de la Industria 4.0 es el de formar un sistema virtual basado en el sistema real físico, el mismo objetivo que tiene un gemelo digital, empleando para ello dispositivos *IoT* (sensores, actuadores, controladores en general) interconectados en una red que permita el control de todo el entorno modelado. Es por esto que la aplicación del gemelo digital a una línea de producción no requiere grandes costes o riesgos; el desarrollo principal queda centrado en los aspectos de simulación con el fin de optimizar la producción.

Con el tiempo han surgido ejemplos prácticos de aplicación del gemelo digital a casos concretos dentro de la Industria 4.0 (como ejemplo, se expone el trabajo de [Vachálek et al. \[2017\]](#)), así como propuestas que buscan generalizar aún más el concepto para que continúe evolucionando (por ejemplo, [Aheleroff et al. \[2021\]](#) introduce un modelo sobre el cual habilitar el gemelo digital como servicio).

Ciudades inteligentes

De igual manera que ocurre en la industria, las ciudades inteligentes se benefician enormemente de los avances tecnológicos que facilitan la recolección y análisis de datos. Estas ciudades inteligentes son centros urbanos dotados de tecnología capaz de recolectar y comunicar una serie de datos específicos, permitiendo digitalizar en cierta medida una ciudad; de esta forma, se agrupa el acceso a información que pueda ser de utilidad para varios fines, relacionados o independientes en cierto grado, en un mismo punto (por ejemplo, datos de producción o consumo energético, circulación del tráfico en rangos horarios diversos, o seguimiento de remodelaciones del entorno, entre otras muchas utilidades).

Añadiendo las características del gemelo digital a una ciudad inteligente, buena parte de la interacción queda automatizada gracias a la propiedad de que tanto el modelo virtual como el real queden entrelazados. Adicionalmente, los costes de desarrollo de un gemelo digital no serían elevados, pues la ciudad inteligente se obtiene mediante una infraestructura de dispositivos *IoT* dispuestos en red, cuyas lecturas se transmiten para ser agregados y analizados, generalmente en la nube, donde se habilitan diversas herramientas para las operaciones necesarias.

Las principales ventajas que ofrece incorporar un gemelo digital se encuentran tras el análisis de la información recolectada, dando lugar a unos cambios tales que optimicen el sistema concreto con el fin de mejorar su rendimiento (mayor producción y menor consumo energético, descongestión de tráfico...). [Ruohomäki et al. \[2018\]](#) presenta una infraestructura con la cual genera un gemelo digital de una ciudad inteligente, simplificando labores en su mantenimiento energético y planificación urbana.

Por otro lado, [White et al. \[2021\]](#) muestra la utilidad de un gemelo digital sobre una ciudad inteligente mediante el uso de simulaciones que faciliten el desarrollo de planes de acción ante eventos fortuitos (en este ejemplo, simula inundaciones de la ciudad, permitiendo observar las zonas de mayor prioridad para evacuación) o de cara al futuro de la urbanización (de nuevo en este caso, simulando aglomeraciones de ciudadanos con la finalidad de observar posibles reestructuraciones).

Sector sanitario

El último aspecto detallado que guarde interés con el concepto de gemelo digital se reserva al sector sanitario, que, a pesar de contener gran cantidad de ramificaciones y enfoques, solo algunos se benefician en mayor medida al compaginarlos con un gemelo digital. La mayoría del contenido de este apartado se basa en el trabajo de [Liu et al. \[2019\]](#), pues expone con claridad los problemas más comunes que busca solucionar, además de presentar el concepto de “*Digital Twin Healthcare*” (*DTH*) con el cual solventar total o parcialmente muchos de esos problemas.

Aunque la exploración e identificación de problemas se especifica, según el objetivo de su trabajo, en la tercera edad, casi todo lo mencionado se aplica también a nivel general. El que probablemente sea el mayor factor decisivo en el tratamiento es la falta de consultas habituales con profesionales del sector debido a las largas colas de espera en centros públicos y el alto coste que supone para muchos la sanidad privada³. Además, el reparto de recursos no es un problema con solución trivial: considerando casos emergentes de alta gravedad, investigación para futuras mejoras en efectividad y el creciente número de población mundial, resulta cada vez más complicado satisfacer las necesidades de todos por igual.

Como ya se ha mencionado, el concepto de *DTH* busca dar solución a los problemas haciendo uso de gemelos digitales. De forma resumida, el *DTH* no es más que un gemelo digital tal como propuso [Grieves \[2014\]](#) en su día, pero haciendo el enfoque principal la información relacionada con centros sanitarios o directamente el paciente; ambos quedarían como el sistema real que busca simularse, generando posteriormente su sistema virtual equivalente donde aplicar optimizaciones y cambios, habilitando la comunicación entre los dos con dispositivos de seguimiento *IoT* que permitan validar y calibrar la simulación.

Una vez detallado el *DTH*, [Liu et al. \[2019\]](#) presenta dos campos donde puede aplicarse en el sector sanitario. Primero, propone su uso para la gestión y diseño de hospitales, pues el gemelo digital ayudaría a aplicar planificaciones óptimas en el uso de salas y camillas o para organización de los empleados, y pueden realizarse simulaciones sobre medicamentos y su efecto tras ser aplicados. Por otro lado, la virtualización podría aplicarse directamente con el paciente, con el fin de realizar seguimientos de su estado en todo momento y analizar el historial en busca de enfermedades antes de que evolucionen a estados de mayor gravedad, o bien preparar y estudiar un plan de medicación diferente con la simulación del efecto sobre el paciente.

³Esto se ha visto intensificado al extremo en los últimos años debido a la Covid-19, donde se han sufrido casos de saturación de hospitales y centros públicos.

Si bien existen otros muchos trabajos y estudios de aplicaciones del gemelo digital sobre el sector sanitario, el concepto de *DTH* resume suficientemente la intención general a la hora de implementarlo en casos prácticos, ya sea más orientado a la simulación (efectos de nuevos medicamentos, recetas, cirugía...) o al análisis de datos (relevantes al historial de un paciente, o a la gestión de un centro médico).

2.5. Gemelos digitales para plantaciones solares

El último apartado a tratar en cuanto a ramas de investigación sobre el gemelo digital consiste en el aprovechamiento de energías renovables, centrado principalmente en la energía solar. Cada vez es posible encontrar un mayor número de recintos, tanto públicos como privados, que optan por instalar un sistema de producción mediante generadores fotovoltaicos (existen otros métodos de aprovechar este recurso, pero no son tan comunes a nivel general), pues siguen evolucionando con métodos y tecnologías cada vez más eficientes; añadiendo a esto las mejoras ya mencionadas en *IoT*, se facilita adicionalmente el mantenimiento de estas instalaciones.

Intentando aplicar el concepto de gemelo digital, aparecen multitud de casos y enfoques concretos suficientemente independientes como para centrarse en partes específicas del concepto. A continuación se listan principalmente dos de estos casos, con el enfoque sobre la **fabricación de los generadores** y sobre el **análisis de un sistema fotovoltaico** en funcionamiento, aportando también ejemplos prácticos donde se aplican a modo de ejemplo y justificación.

En primer lugar, se expone el uso del gemelo digital en la fase de producción de generadores fotovoltaicos, con el objetivo de mejorar su rendimiento energético a la vez que se abarate el coste de producción. [Ebrahimi \[2019\]](#) realiza un estudio sobre las dificultades al desarrollar un gemelo digital para la simulación de generadores de energía renovable, debiendo tener en cuenta la compleja tecnología interna de los mismos, los métodos en que aprovechan el recurso renovable para la generación, y estado del componente para la detección de problemas. Con todo esto, presenta una metodología de modelado con la que desarrollar un gemelo digital capaz de simular el funcionamiento a gran escala de generadores eólicos e hidráulicos, permitiendo además ser enlazado al generador real y habilitar la interconexión entre ambos extremos.

Por otro lado, [Pei et al. \[2021\]](#) centra el objetivo del gemelo digital al control de calidad de una línea de fabricación de células fotovoltaicas. Virtualizando toda la cadena, el modelo simulado recibe información del funcionamiento de cada una de las

partes que influyen durante la producción mediante sensores *IoT*, los que, añadidos a los datos del producto final fabricado, habilitan un análisis partiendo de la extracción de características relevantes basado en *machine learning* supervisado, dando lugar a una configuración de la maquinaria que mejore el rendimiento global.

Como último ejemplo de este enfoque, [Kunze et al. \[2022\]](#) propone el uso de un gemelo digital empírico de células fotovoltaicas extraído mediante el análisis de imágenes con información de su estado físico. Gracias a que cada gemelo dispone de las propiedades concretas de la célula, se identifican los posibles defectos físicos que sufra de tal forma que el conjunto estudiado pueda agruparse y analizarse con métodos más avanzados posteriormente. Este ejemplo muestra un uso del gemelo digital a un nivel inferior: en lugar de abarcar todo un sistema compuesto por múltiples componentes, se virtualiza el producto resultante para ser clasificado automáticamente.

El segundo caso a ejemplificar consiste en analizar el funcionamiento de una instalación fotovoltaica mediante su gemelo digital. [Jain et al. \[2019\]](#) presenta un método de detección de defectos en sistemas fotovoltaicos distribuidos mediante la virtualización del sistema en un gemelo digital. Con la información recibida de los sensores, propone dos metodologías para esa detección que, una vez incorporados al aspecto de simulación del gemelo digital, resultará una clasificación del sistema según su estado en cada instante, tras lo cual podrá adaptarse siguiendo una configuración específica para solventar el problema detectado.

Continuando con el trabajo de [Zohdi \[2021\]](#), surge otra aplicación del gemelo digital sobre un sistema de generación fotovoltaico; en esta ocasión se extiende a un sistema de *agrivoltaica*, donde agricultura y generación fotovoltaica quedan combinadas en una misma instalación⁴. El gemelo digital que propone para este sistema tiene como objetivo la optimización de los parámetros del generador, con las condiciones añadidas que supone la presencia de suelo agrícola, por lo que las simulaciones requieren análisis de la luz incidente sobre los paneles parametrizados del sistema real. La ventaja que induce la implementación de un gemelo digital es, de nuevo, la posibilidad de enlazar ambos extremos y que el ajuste sea automático en mayor medida.

Como punto final, se expone el estudio de [Scharl and Praktijnjo \[2019\]](#) sobre la utilidad del concepto de Industria 4.0 sobre el aprovechamiento de energías renovables, concluyendo, tras analizar la opinión de múltiples expertos, que el gemelo digital aportará mayor transparencia y control sobre grandes bancos de información y sistemas en funcionamiento, simplificando buena parte de las necesidades industriales.

⁴El concepto de *agrivoltaica* fue introducido por [Goetzberger and Zastrow \[1982\]](#), buscando la unión de agricultura y generación solar en un mismo recinto para aprovechamiento del suelo agrícola.

2.6. Gráficos por ordenador

En la mayoría de los casos prácticos ejemplificados en secciones anteriores, el desarrollo del gemelo digital se centra sobre todo en el aspecto de simulación en el modelo virtual o el análisis de los datos recolectados en la interconexión de ambas partes. Otro punto interesante del uso de gemelos digitales reside en la visualización del modelo virtual mediante gráficos por ordenador, pues facilitan la comprensión del problema que resuelven y la interacción con el sistema para el usuario, aligerando las tareas que planea realizar.

Actualmente, es muy común el concepto de generar imágenes de entornos tridimensionales para distintos fines (simuladores, videojuegos, realidad aumentada, realidad virtual...), haciendo uso de tarjetas gráficas, *Graphics Processing Unit* o *GPU*, instaladas en los equipos, que se encuentran especialmente preparadas para cálculos destinados a esta tarea. Si bien una escena virtual puede sacar máximo provecho de la *GPU*, la aparición de nuevas técnicas y métodos para el análisis de datos y/o simulación también pueden requerir su uso intensivo con el fin de minimizar los tiempos de ejecución, debiendo sacrificar para ello la fidelidad visual.

A pesar de esto último, suele ser más común encontrarse ante problemas que no requieran metodologías tan complejas, pudiendo reservar el procesador principal del sistema (u otro equipo remoto) a las tareas de análisis o simulación. Es por esto que surge el interés de construir modelos virtuales con la mayor fidelidad visual posible respecto al sistema real, donde poder aprovechar no solo las ventajas en cuanto a interfaz y facilidad de uso ya mencionadas, sino también ofrecer funcionalidad adicional.

Una de las opciones que presenta mayor utilidad general consiste en la escala del modelo virtual, en cuanto al contenido del sistema virtualizado: en casos donde la simulación y/o análisis sean sencillos computacionalmente, es posible dotar a la aplicación de un mayor número de componentes para que sean también considerados y gestionados digitalmente (por ejemplo, en una instalación fotovoltaica, la cantidad de generadores simultáneos en funcionamiento, o en una línea de fabricación la multitud de maquinaria que actúa en todo el proceso). Por otra parte, muchos procesos se benefician en gran medida de las operaciones realizadas en *GPU*, destacando por ejemplo operaciones con rayos (puestas aún más de moda gracias al enfoque de empresas como Nvidia por el trazado de rayos), triángulos (desde la simple construcción de modelos 3D, hasta optimización y detallado de los mismos con técnicas más avanzadas), vectores y matrices (útiles sobre todo al manejar geometría, pero extrapolables a otros campos como simulación o análisis de datos).

Según el tipo de gemelo digital que se busque implementar, a la hora de considerar el apartado de visualización existen multitud de herramientas y librerías que facilitan la gestión de escenas y la interacción. En la sección 2.3 se mencionó como el motor gráfico **Unity 3D** ofrece soporte al desarrollo de gemelos digitales, aportando solución inmediata al manejo de las escenas y modelos tridimensionales⁵. Al trabajar a un nivel superior con una base preestablecida, el esfuerzo puede destinarse a otros aspectos del gemelo digital, manteniendo un mínimo de fidelidad visual respecto al sistema real.

Si el objetivo del proyecto abarca un alcance mucho mayor del que propone esa base empleando un motor gráfico, también existe la posibilidad de emplear librerías de más bajo nivel para la generación de gráficos 3D. Aunque son más complejos de desarrollar, los sistemas implementados “desde cero” habilitan un alcance mucho más amplio al disponer de libertad total sobre el funcionamiento interno del mismo; incluyendo librerías y recursos adicionales, las posibilidades aumentan en gran medida. Hoy en día se ha popularizado el uso de librerías como **OpenGL** y **DirectX**, pero entre el amplio repertorio de alternativas, algunas como **Vulkan** o diversas variantes de *OpenGL* como **OpenGL ES** y **WebGL** demuestran gran utilidad en sus áreas específicas y son más comunes cada día.

Resumiendo, gran parte de los casos de uso prácticos del gemelo digital refuerzan en mayor medida los aspectos de simulación del modelo virtual y el análisis de datos, pues su objetivo principal suele ser monitorización del sistema o la optimización de sus parámetros. A pesar de esto, la visualización sigue siendo un factor a considerar durante el planteamiento de utilizar un gemelo digital por los múltiples beneficios que conlleva en muchos casos, sobre todo en cuanto al uso de la aplicación, ya que existen múltiples herramientas que facilitan la implementación de aplicaciones gráficas y habilitan la extensión a diversos campos que puedan ser de utilidad.

⁵El objetivo de un motor gráfico es el de encapsular la compleja gestión de la información a bajo nivel requerida para el dibujado de una escena, alejando del usuario el manejo de *shaders* o de datos sobre geometría, topología, color... pero sin impedir que otro usuario experto pueda realizar las modificaciones que considere oportunas sobre esta base.

Capítulo 3

ESPECIFICACIÓN DEL TRABAJO

Una vez explorada la actualidad del concepto del gemelo digital y los usos que recibe en diferentes ámbitos, este tercer capítulo se centrará en definir los objetivos generales del proyecto a desarrollar, considerando las posibilidades y recursos disponibles, de forma que la aplicación final quede correctamente implementada.

3.1. Situación de partida

La intención del trabajo es el diseño y desarrollo de un gemelo digital de la plantación solar instalada en las zonas de aparcamiento del Campus Las Lagunillas de la Universidad de Jaén, con el fin de reducir el esfuerzo necesario en tareas de optimización y monitorización del rendimiento. Como punto de partida, se hará uso de lenguajes y librerías a bajo nivel para mantener un amplio abanico de posibilidades a la hora de añadir funcionalidad y debido también a la experiencia académica del equipo de desarrollo en sistemas gráficos. Adicionalmente, se dispondrá de recursos digitales obtenidos de la instalación física, principalmente imágenes satélite y escaneos tanto de color como térmicos para poblar el modelo virtual.

Cabe destacar la ausencia de interconexión entre los sistemas real y virtual, debido a que la instalación de la plantación solar continúa en marcha durante el desarrollo del trabajo, y, por tanto, no se dispone de los recursos necesarios (en esencia, una red de dispositivos *IoT* con la cual obtener información del funcionamiento y modificar parámetros del sistema) para establecer ese vínculo que caracteriza al gemelo digital.

3.2. Alternativas propuestas y viabilidad

A la hora de desarrollar un software desde cero, son muchas las alternativas entre las que debatir respecto metodologías de diseño, plataforma o herramientas de desarrollo, dispositivos y audiencia objetivo... El principal objetivo del proyecto es el desarrollo de una utilidad que facilite la gestión virtual de una instalación solar, por tanto, se considera como audiencia objetivo aquellos usuarios con suficiente conocimiento en este ámbito, al igual que cierta experiencia previa en el uso de aplicaciones gráficas con el fin de no entorpecer su experiencia por problemas de interacción o incongruencias con el modelo virtual.

Otra decisión principal para desarrollar el trabajo reside en la plataforma donde se desplegará la aplicación final, pues también influye sobre otros aspectos como el lenguaje de programación empleado, requisitos hardware mínimos para el uso de la aplicación o la forma en que se gestiona la información:

- **Aplicación web:** Cada vez son más las aplicaciones que apuestan por un enfoque en la nube, permitiendo a los usuarios acceder desde cualquier lugar y dispositivo, siempre que se encuentre conectado a la red.

Con el desarrollo de un gemelo digital en mente, el enfoque a una aplicación web parece prometedor, pues permite acceder en todo momento a la simulación que se esté realizando, ya sea para consultar resultados o el propio estado del sistema físico recreado. Además, el simple hecho de partir de un sistema conectado a la red permite añadir mayor complejidad a la aplicación aprovechando la conexión entre dispositivos y usuarios (por ejemplo, se podría construir una jerarquía de usuarios dividiendo usuarios comunes de administradores o desarrolladores, que acceden a la información y gestionan el modelo virtual respectivamente).

A pesar de presentar bastantes ventajas interesantes, el desarrollo web también implica una gran cantidad de problemas a tener en cuenta que deben solventarse para mantener una buena experiencia de usuario. Aunque muchos de los mencionados problemas tienen una solución genérica que puede adaptarse a situaciones concretas gracias al amplio uso de esta rama de desarrollo, existen otros detalles que igualmente influyen en gran medida. El principal factor limitante es respecto la parte relacionada con informática gráfica; cada vez se busca mayor fidelidad en los modelos resultantes, pero esto implica un incremento en los requisitos hardware para mantener una correcta experiencia de usuario.

Si bien es posible relevar el trabajo de mantener la escena (geometría, materiales, propiedades de dibujo...) al equipo servidor, el cual normalmente consta

de una alta capacidad de cómputo, el problema pasaría ahora a la transmisión de datos sobre la red: cada usuario dispone de un equipo y calidad de conexión diferente, afectando al rendimiento de la aplicación ya sea por retardos durante el renderizado o pérdidas de datos durante la transmisión.

Esto último es especialmente importante, y más aún en el contexto de un gemelo digital, pues no podemos permitir que se pierda información de forma que el sistema virtual quede modificado o incompleto (ya sea visualmente, dando lugar a discordancias al compararlo al sistema real, o lógicamente, surgiendo errores en los datos de entrada o los resultados de la simulación).

Indagando más en el aspecto del desarrollo de aplicaciones gráficas para web, la alternativa principal que se presenta es el uso de *WebGL*, una librería gráfica basada en *OpenGL ES* (una interfaz de programación formada como un subconjunto de la versión de escritorio de *OpenGL*). A la vez, existen gran cantidad de librerías que encapsulan *WebGL* para facilitar el desarrollo, pero actualmente las que más suelen nombrarse son *Babylon.js* y *Three.js*. El uso de una u otra, como es de esperar, depende del contexto y deberán analizarse los alcances de ambas y la forma en que orientan su implementación, aunque como punto a favor de *Babylon.js* cabe mencionar la disponibilidad de un editor en línea reducido, el cual permite realizar pequeñas pruebas más concretas.

- **Aplicación móvil:** De la misma forma que en el desarrollo web, las aplicaciones móviles también se encuentran a la orden del día debido a la tendencia actual del uso de dispositivos móviles; la inmensa mayoría de personas hacen uso casi constante de su teléfono inteligente o similar.

La principal ventaja que ofrece desarrollar una aplicación para dispositivos móviles es el permitir a los usuarios acceder a ella en cualquier momento, y potencialmente sin necesidad de requerir conexión a una red, ejecutando la aplicación en su totalidad desde el propio dispositivo. Además, aprovechando el hecho de que la sociedad actual está mayoritariamente acostumbrada al uso de estos dispositivos, gran parte del desarrollo de la interacción se resuelve de forma natural (principalmente los gestos y control táctil).

Se ha mencionado la posibilidad de requerir conexión a la red, lo que a su vez implica comprender la capacidad de los dispositivos móviles y las tecnologías actuales de conexión inalámbrica, debido al requisito de una transmisión de datos suficiente para evitar pérdidas o retardos en exceso que entorpezcan la experiencia de usuario. Afortunadamente, cada vez se proponen tecnologías de conexión inalámbrica con mayor calidad, siendo el mejor ejemplo actual la nueva versión del estándar para redes móvil de banda ancha 5G.

Pasando ahora a las desventajas, el uso de dispositivos de tamaño reducido es realmente un arma de doble filo: es cierto que menor tamaño y peso habilitan mayor accesibilidad para los usuarios, pero a cambio se sacrifica la potencia del hardware que incorpora el dispositivo final (procesadores y memorias más pequeñas, añadiendo además una batería para alimentar los componentes). En el caso de un gemelo digital podría no ser tan necesario mucha capacidad de cómputo, ya sea por una simplificación de la geometría o por requerir una simulación liviana, pero si la aplicación busca cierto nivel de fidelidad habrá que buscar formas de optimizar la ejecución (considerando además la amplia variedad hardware y software que incluyen).

En cuanto al desarrollo de estas aplicaciones, las herramientas se amplían en gran medida siguiendo, como se acaba de mencionar, la existencia de no solo diferentes modelos de dispositivos, sino también los múltiples sistemas operativos que puedan incluir. La principal diferenciación a la hora de centrar el desarrollo abarca, por un lado, los dispositivos de *Apple*, que al ser un entorno cerrado requiere desarrollar principalmente con las herramientas que ofrecen, o para dispositivos *Android*, mucho más genéricos y en un entorno abierto, pero añadiendo la problemática de considerar un amplio rango de hardware distinto.

- **Aplicación de escritorio:** La siguiente opción a estudiar se basa en el desarrollo para equipos de escritorio, el enfoque clásico que ha ido evolucionando y adaptándose a las mejoras en la tecnología hardware. Aunque no presente ventajas tan claras como los enfoques anteriores, al ser la metodología clásica de desarrollo es fácilmente adaptable a la mayoría de problemas.

Al trabajar sobre equipos de uso personal, dejando de lado problemas adicionales como la portabilidad (y la pérdida de rendimiento que involucra) o la comunicación y transmisión de datos sobre una red, permite centrar la aplicación totalmente en la funcionalidad que ofrece; en el caso de un gemelo digital, se favorece en gran medida la calidad gráfica del modelo tridimensional así como la eficiencia y/o eficacia de la simulación.

Como ya se ha mencionado, el desarrollo para escritorio presenta como inconveniente la falta de aquellas utilidades que caracterizan a las alternativas previas. Al emplear dispositivos principalmente estáticos y de mayor tamaño, la movilidad se reduce en gran medida ¹.

Adicionalmente, el habilitar la comunicación sobre una red de ordenadores requiere un esfuerzo adicional equiparable al que presentaría una aplicación web.

¹Actualmente, gracias a la evolución y mejora de las tecnologías para la fabricación de equipos portátiles y tabletas, se habilita cada vez más cierto grado de movilidad, dando a cambio buena parte del rendimiento del equipo

En esta ocasión se ignora la parte de transmitir directamente la geometría del modelo virtual, pues cada equipo ejecuta su propia versión, reduciendo la problemática a la transmisión de información y control.

Respecto a las herramientas de desarrollo para aplicaciones de escritorio, nuevamente se presenta un abanico muy amplio de alternativas, cada una con sus ventajas e inconvenientes, considerando además el uso de interfaces de programación u otras herramientas adicionales para la parte gráfica y la de simulación.

Comenzando por la simulación, se hace necesario gestionar gran cantidad de datos y realizar operaciones sobre los mismos; es por esto que lenguajes como *Python* o *R* destacan ante el resto (el primero, al ser un lenguaje interpretado y de alto nivel, permite el uso de gran cantidad de librerías con problemas muy complejos resueltos, mientras que el segundo se desarrolló principalmente como herramienta para estadística).

Por otro lado, tenemos la parte de informática gráfica, también con distintas alternativas. Sobre los lenguajes de programación disponibles, aquellos más conocidos y de uso frecuente disponen de compatibilidad con alguna librería gráfica: principalmente *C/C++* con *OpenGL* y *Java* de forma nativa o mediante *swing*. También cabe considerar la posibilidad de hacer uso de *DirectX* mediante la infraestructura de Microsoft *.Net*.

- **Enfoque híbrido/multiplataforma:** Como alternativa final se propone el desarrollo de una aplicación para diversas plataformas simultáneamente. De esta forma, es posible aprovechar las ventajas que proponen el resto de alternativas mencionadas, aunque como es de esperar, esto incluye un gran esfuerzo adicional por cada plataforma objetivo.

El mayor problema que presenta esta alternativa reside en el mantenimiento de la aplicación para todas las plataformas en que se despliega: en el fondo equivaldría a mantener aplicaciones diferentes de forma separada, simplificando ligeramente el trabajo gracias a que todas las versiones se basan en la misma situación de partida o versión genérica.

3.3. Descripción de la solución propuesta

Tras estudiar las alternativas resumidas anteriormente, el trabajo se desarrollará mediante una **aplicación de escritorio** destinada a equipos con el sistema operativo *Windows*. En primer lugar, se optó por el desarrollo web, pero las dificultades que añade no permitirían ampliar lo suficiente los requisitos principales de la aplicación. A

pesar de que el enfoque seleccionado reduzca considerablemente el rango de audiencia, permite ofrecer a cambio una aplicación más especializada.

Otro de los motivos por el que se valora esta solución reside en el hecho de no requerir conexión a una red para el funcionamiento básico de la aplicación ², evitando tener que considerar los problemas adicionales que involucra la transmisión de datos.

A continuación se enumeran las razones principales que caracterizan la solución seleccionada frente al resto de alternativas propuestas:

- **Enfoque clásico de desarrollo de software**, lo que facilita la resolución de problemas genéricos.
- **Entorno reducido de dispositivos**. La inmensa mayoría de ordenadores de escritorio constan de prestaciones y hardware similar, facilitando el adaptar la aplicación a cada sistema.
- **Simulación local**, permitiendo dar los resultados directamente en la propia aplicación sin requerir aplicaciones adicionales o conexión a una red.
- **Programación de aplicaciones gráficas**. Aunque actualmente se encuentran en auge las aplicaciones gráficas en entornos móviles y web, los principales avances en la tecnología relacionada se aplican en primer lugar sobre las herramientas especializadas en desarrollo para escritorio ³.
- **Experiencia previa** en desarrollo de aplicaciones gráficas para escritorio, principalmente en el lenguaje *C/C++* con la librería *OpenGL*.

3.3.1. Análisis de alternativas

La selección de la alternativa mencionada anteriormente no solo se debe a las ventajas que ofrece, también listadas, también se debe al descarte del resto de opciones debido a los problemas que presentan de cara a suplir las necesidades del proyecto para implementar la funcionalidad requerida.

²A pesar de no ser necesario el acceso a una red, hacer uso de la misma permitiría funcionalidad adicional, la cual podría ser de gran interés como se mencionó en la sección anterior, añadiendo por ejemplo una jerarquía de usuarios o simplemente permitiendo la transmisión de los resultados obtenidos por la simulación.

³A modo de ejemplo se mencionan las recientes tecnologías promocionadas por Nvidia de trazado de rayos, **RTX**, y reescalado mediante *deep learning*, **DLSS**.

Entre el resto de alternativas, la más prometedora es el desarrollo de la aplicación para web, pues sus ventajas (mencionadas en la sección 3.2) beneficiarían mucho a la experiencia final de usuario. Es por esto por lo que el comienzo del proyecto se destina a analizar esta alternativa con el fin de determinar su viabilidad. Los detalles concretos del desarrollo web se abarcarán en el comienzo del capítulo 5; en la sección actual se mencionan los principales motivos del descarte de la alternativa.

Partiendo de la naturaleza de un sistema web, el desarrollo recibe mayor complejidad al tener que implementar tanto el *frontend* (la parte visible por el usuario) como el **backend** (la parte del servidor), además de su mantenimiento. Esta complejidad añade también problemas en cuanto a informática gráfica, ya que deben considerarse tiempos de respuesta mucho más estrictos a la vez que el mínimo uso de ancho de banda posible durante las transmisiones, algo que se complica si quiere trabajarse con instalaciones o nubes de puntos de gran tamaño. Otro punto a considerar es el rango de dispositivos que permite una aplicación web: si bien son muy amplias las posibilidades, también lo son la variabilidad de su capacidad de ejecución, para lo cual habría que adaptar la aplicación a niveles de detalle variables.

Adicionalmente, un desarrollo web debe seguir diversas reglas sobre aspectos como accesibilidad, internacionalización o seguridad, entre otros, definidos por el *World Wide Web Consortium (W3C)*, aumentando aún más el tiempo de desarrollo para suplir estos requisitos. Es especialmente importante considerar aspectos como la seguridad en la web, conociendo la facilidad con que un atacante podría aprovechar cualquier vulnerabilidad para acceder a datos privados o implantar software malicioso.

Ante todos estos problemas, se descarta la alternativa en favor de un desarrollo para escritorio con el fin de reducir gran parte de la complejidad en favor de un menor tiempo de desarrollo. Sobre el resto de alternativas, el desarrollo móvil implica dispositivos de pocas prestaciones, por lo que no es óptimo de cara al objetivo del proyecto. Por otro lado, un desarrollo multiplataforma cae nuevamente en el problema de añadir una complejidad excesiva.

3.4. Tecnologías utilizadas

En esta sección se resumirán las diferentes herramientas, lenguajes de programación, librerías, etc. empleadas en el desarrollo del proyecto.

3.4.1. Desarrollo web

En primer lugar, se mencionan las utilidades con las que se realiza una primera prueba para desarrollar la aplicación destinada a la red y comprobar así su viabilidad.

- **HTML, CSS y JavaScript** forman la base de casi cualquier aplicación web que requiera un mínimo de complejidad en la interacción de usuario. En concreto, estos lenguajes se encargan de estructurar, estilizar y programar respectivamente enfocado en la parte del cliente: la página a la que accede desde un navegador.
- **Babylon.js y TypeScript** dan lugar a la parte de informática gráfica. Se agrupan ambos porque *Babylon.js* hace uso de *TypeScript*, una alternativa a *JavaScript* fuertemente tipificada ⁴. En cuanto a *Babylon.js*, encapsula la interfaz gráfica *WebGL* simplificando el uso de la misma (por ejemplo, se encarga de gestionar la estructura de datos de la escena internamente, tanto geometría como materiales y propiedades de los objetos), ofreciendo directamente funcionalidades básicas como controles de interacción básicos, interfaz gráfica de usuario o propiedades de dibujo al programador sin que deba gestionar su estado interno.

Una de las mayores ventajas que ofrece *Babylon.js* frente al resto de alternativas es la disposición de varios entornos simplificados en línea, accesible en cualquier momento para la realización de pequeñas pruebas con el motor gráfico (*Playground*), un visualizador de escenas y objetos (*Sandbox*) y un editor de materiales/shaders (*Shader creation tool*), todos ellos accesibles desde su página oficial o el repositorio *Git* del proyecto en [Catuhe \[2013\]](#).

- **Node.js** forma la parte de servidor de la aplicación, permitiendo gestionar todo el código, paquetes y recursos adicionales solicitados por los clientes. Adicionalmente, se incluyen paquetes como **Webpack**, encargado de agrupar los recursos fuente (código, imágenes y ficheros de datos, entre otros) aplicando técnicas de compresión para minimizar la carga durante la transmisión y reducir lo máximo posible los errores que puede acarrear.
- **Visual Studio Code** es el entorno principal de desarrollo por excelencia en el ámbito web gracias a la facilidad con que permite ajustar su comportamiento frente a distintos lenguajes en cuanto a depuración, comprobación de sintaxis y todas las funcionalidades esenciales de cualquier editor de textos o entorno de desarrollo integrado (IDE).

⁴ *TypeScript* se implementa como un superconjunto de *JavaScript*, añadiendo sobre este último la tipificación estricta, pero permitiendo simultáneamente el uso de sus variables y funciones originales.

- **Docker** es un conjunto de productos ofrecidos como servicio (corresponde dentro del grupo **plataforma como servicio**, *Platform as a Service* o *PaaS*) basados en virtualización de sistemas operativos en forma de contenedores, con la posibilidad de migrar entre distintas máquinas siempre que sean compatibles con la plataforma. Se emplea en este caso para generar un contenedor que albergue el servidor de la aplicación, pudiendo ponerlo en marcha o gestionarlo en cualquier momento partiendo directamente del contenedor.

3.4.2. Desarrollo para escritorio

Pasando ahora a la aplicación final a desarrollar, se exponen a continuación las herramientas y tecnologías empleadas:

- **C++** como lenguaje principal de programación con el fin de aprovechar la gestión manual de la memoria del sistema en lugar de relevar la tarea a un recolector de basura empleado por otros lenguajes como *Java*. Se hace importante controlar correctamente la memoria, especialmente en una aplicación gráfica, pues deben almacenarse tanto la geometría como las propiedades y materiales de cada modelo a dibujar. Con un correcto control de punteros y referencias, se facilita la gestión de múltiples modelos que compartan propiedades durante el dibujado, además de aligerar la carga durante la creación y destrucción de objetos en los casos que sea necesario.
- La librería gráfica seleccionada es **OpenGL**, principalmente gracias a la experiencia previa, tal como se menciona en la sección anterior [3.3 Descripción de la solución propuesta](#). Aunque estudios como [Shiraef \[2016\]](#) muestran como otras *API* ofrecen ciertas mejoras respecto a la selección, realizar la transición a una plataforma diferente conlleva bastantes problemas adicionales, además de requerir la adaptación al nuevo entorno.

Junto a *OpenGL* se incluyen como librerías adicionales las siguientes: **GLEW** ⁵ encargada de la comprobación de compatibilidad entre extensiones de *OpenGL* y el hardware donde se ejecutará la aplicación; **GLFW** ⁶ destinada a la gestión de la interfaz de usuario, es decir, entrada y salida entre los periféricos conectados al equipo y la aplicación; **GLM** ⁷ centrada en habilitar el uso de gran cantidad de operaciones matemáticas basadas en *GLSL*, incluyendo tanto vectores como matrices de diversas dimensiones.

⁵<https://github.com/nigels-com/glew>. Último acceso en 2022-06-25.

⁶<https://www.glfw.org>. Último acceso en 2022-06-25.

⁷<https://github.com/g-truc/glm>. Último acceso en 2022-06-25.

- **LodePNG** se trata de una librería de *C* especializada en tratar con ficheros en formato *PNG* como su propio nombre indica. Se utilizará en la aplicación para cargar imágenes empleadas para las diversas texturas asignadas a los modelos de la escena. La ventaja que presenta esta librería es el reducido tamaño de los ficheros fuente, que permite incluirla y utilizar su funcionalidad en la mayoría de proyectos con un esfuerzo mínimo.
- Por último, toda la parte de interfaz gráfica de usuario se releva a **Dear ImGui**. Los distintos elementos que emplea requieren de una base funcional capaz de soportar todo el ciclo de renderizado y gestionar la entrada/salida del usuario para construir la interfaz gráfica junto a la interacción necesaria. Aunque su uso se hace algo complejo de primeras, el hecho de implementarse sobre una base estable facilita enormemente enlazar la librería a casi cualquier aplicación desarrollada con *C++*.

3.4.3. Herramientas adicionales

- **Blender** es una aplicación gráfica que agrupa gran cantidad de herramientas y utilidades para la creación y edición de modelos 3D, efectos visuales, animaciones... Su utilidad en el proyecto reside en la gestión de los diversos modelos empleados para la escena virtual, tanto geometría como texturas, y adaptarlos como sea necesario, facilitando su utilización.
- *GNU Image Manipulation Program*, más conocido como **GIMP**, es un editor de imágenes ráster ⁸ multiplataforma y de código abierto. La edición de imágenes se hace necesaria para generar o modificar las texturas (y posteriormente exportarlas al formato requerido) que usarán los modelos a dibujar de la escena virtual.
- **Inkscape** es un editor de imágenes vectoriales ⁹ multiplataforma y de código abierto. Su uso consiste en, igual que *GIMP*, la gestión de texturas, además de la generación de algunas figuras y diagramas empleados en este documento.

⁸Imágenes discretas compuestas por una matriz de píxeles cuadrados, dando la información del color en cada uno de ellos.

⁹Imágenes formadas mediante primitivas en lugar de directamente los píxeles discretos. Esto permite mayor nivel de detalle en muchas situaciones (por ejemplo, no pierden calidad al ampliar la imagen).

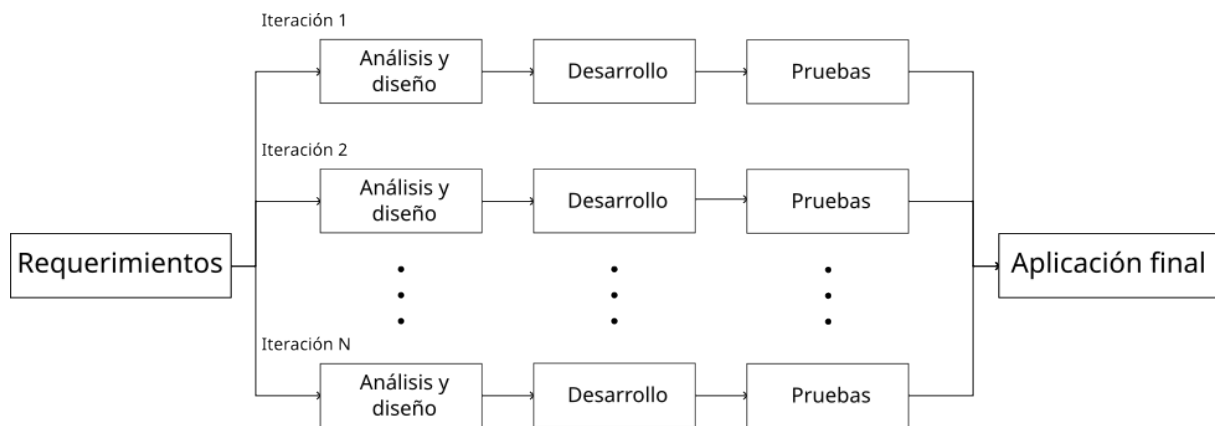


Figura 3.1: Esquema de la metodología de desarrollo incremental

3.5. Metodología de desarrollo software

El siguiente punto del proyecto a definir consiste en la metodología a seguir a la hora desarrollar la aplicación. La metodología seleccionada será la **clásica incremental**, donde a partir de los requisitos analizados se generan diversas versiones de la aplicación, centrando cada una de ellas en una o varias funcionalidades para disponer de un incremento donde pueda comprobarse dicha funcionalidad. La figura 3.1 presenta el esquema básico que siguen los desarrollos con esta metodología.

Esta metodología es ideal cuando existen requisitos mucho más prioritarios, si el equipo de desarrollo posee poca experiencia, o si el cliente busca disponer de un producto rápidamente (que se podrá expandir poco a poco más tarde). Como ya se ha mencionado, el proceso de desarrollo se divide en incrementos que se tratarán como si fuese una aplicación ajena, definiendo en cada una el diseño que seguirá, siguiendo con el desarrollo y una fase de pruebas, terminando con el despliegue del incremento en la aplicación final.

Los problemas que presenta la metodología incremental consisten en una mayor necesidad de planificación y análisis (pues deben prepararse los incrementos a desarrollar y la forma en que se combinarán a la aplicación), se incrementa el coste al reiterar en fases como la de pruebas o si se presenta algún problema en el incremento, además de requerir un conocimiento preciso de la aplicación y poder dividir correctamente la funcionalidad de cada versión.

Aunque presenta varios problemas muy considerables, el proyecto evita alguno de ellos ¹⁰ directamente y permite aprovechar las ventajas de la metodología. Entre estas

¹⁰Al conocer claramente el concepto de gemelo digital, se conoce con totalidad el objetivo de la aplicación y podrá dividirse la funcionalidad sin problema.

ventajas se destacan la facilidad con que puede comprobarse el funcionamiento de cada incremento y evitar errores grandes gracias a estar centrado en una pequeña parte de la funcionalidad, así como el hecho de obtener una aplicación reducida, funcional en etapas tempranas del desarrollo.

3.6. Estimación del proyecto

La última sección de este capítulo se dedicará a definir ciertos criterios estimados del proyecto, principalmente el tamaño del desarrollo, su planificación temporal y el presupuesto del proyecto en su totalidad.

3.6.1. Tamaño y esfuerzo

Para aproximar el esfuerzo requerido durante el desarrollo del proyecto, se recurre al modelo de coste constructivo, generalmente conocido como el modelo **COCOMO**, propuesto por [Boehm \[1981\]](#) como un modelo de aproximación de costes para software. Siguiendo las categorías propuestas por el modelo, el proyecto podría encajar en la categoría *orgánica*, pues es un proyecto simple y desarrollado por un equipo pequeño y con experiencia previa. Adicionalmente, es posible detallar los resultados del modelo mediante tres fases: básico, intermedio y detallado.

En esta ocasión se recurre al modelo de *COCOMO* intermedio, ya que el detallado requiere un análisis mucho más exhaustivo y el básico mide el esfuerzo únicamente mediante el número de líneas de código y diversas constantes según el nivel de detalle. Las variables requeridas por el modelo intermedio orgánico son el número de líneas de código estimadas y un factor de ajuste del esfuerzo calculado a partir de un análisis de diversos aspectos del proyecto y el equipo de desarrollo:

- Líneas de código: Ya que el proyecto se divide en dos partes principales (desarrollo web, seguido por un desarrollo para escritorio), el total de líneas usado para la aproximación será la suma de ambas partes. Respecto a la aplicación web, se estima que con unas 800 líneas de código a escribir permitirían poder comprobar la viabilidad del sistema (construcción de un modelo tridimensional, establecer una simulación simple a modo de ejemplo y preparar una imagen con la aplicación para el servidor mediante *Docker*).

Factor	Valoración	Valor
Garantía de funcionamiento (<i>RELY</i>)	Nominal	1,0
Tamaño de la base de datos (<i>DATA</i>)	Bajo	0,94
Complejidad del producto (<i>CPLX</i>)	Nominal	1,0
Restricciones en el tiempo de ejecución (<i>TIME</i>)	Alto	1,11
Restricciones de memoria principal (<i>STOR</i>)	Nominal	1,0
Volatilidad de la máquina virtual (<i>VIRT</i>)	Alto	1,15
Tiempo de respuesta (<i>TURN</i>)	Bajo	0,87
Capacidad del analista (<i>ACAP</i>)	Nominal	1,0
Experiencia en la aplicación (<i>AEXP</i>)	Nominal	1,0
Capacidad del programador (<i>PCAP</i>)	Nominal	1,0
Experiencia en la máquina virtual (<i>VEXP</i>)	Alto	0,9
Experiencia en el lenguaje de programación (<i>LEXP</i>)	Alto	0,95
Uso de métodos de ingeniería del software (<i>MODP</i>)	Nominal	1,0
Uso de herramientas software (<i>TOOL</i>)	Muy alto	0,83
Plan de desarrollo requerido (<i>SCED</i>)	Alto	1,04

Tabla. 3.1: Factores para el ajuste del modelo *COCOMO* intermedio

En cuanto al desarrollo para escritorio, el total estimado aumentaría hasta las 4000 líneas, considerando: adaptación de una aplicación gráfica básica, incorporación de la librería *Dear ImGui* y diseño de la interfaz gráfica de usuario, así como incorporación de la simulación. El gran incremento en el número de líneas se debe a que, al trabajar a bajo nivel, deben implementarse manualmente muchos aspectos que librerías de alto nivel ya incorporan de forma nativa.

- Factor de ajuste del esfuerzo: Siguiendo el modelo intermedio de *COCOMO* y los 15 factores que el propio Boehm propone para este valor, la tabla 3.1 muestra, para cada uno, la influencia que presenta sobre el proyecto, así como el valor que aporta al factor final de ajuste. El factor de ajuste final equivale a: $eof = 0,77045$.

Una vez conocemos los valores de las variables requeridas y tomando las constantes $a = 3,2$, $b = 1,05$ y $c = 0,38$ (también definidas por Boehm), aplicamos la ecuación del modelo para obtener, en primer lugar, el esfuerzo dado en *personas mes*. A continuación, se podrá calcular el tiempo de desarrollo en meses y, por último, con ambos valores estimados se puede dar una aproximación del tamaño más eficaz del equipo. Todos estos pasos quedan definidos en las ecuaciones 3.1, 3.2 y 3.3 respectivamente.

$$E = a * kloc^b * eof = 3,2 * (0,7 + 4)^{1,05} * 0,77045 = 12,52 \text{ personas mes} \quad (3.1)$$

$$D = 2,5 * E^c = 2,5 * 12,52^{0,38} = 6,532 \text{ meses} \quad (3.2)$$

$$P = \frac{E}{D} = \frac{12,52}{6,532} = 1,917 \approx 2 \text{ personas} \quad (3.3)$$

3.6.2. Planificación temporal

La planificación temporal del proyecto se presenta mediante un diagrama de Gantt donde se muestran, de forma general, las partes del trabajo y su duración estimada. La figura 3.2 muestra dicho diagrama, mostrando la planificación por meses de las tareas generales, además de considerar también los periodos de vacaciones y exámenes del curso 21-22 mediante espacios donde no se avanzaría el desarrollo.

El proyecto se divide en dos partes principales: en primer lugar, el desarrollo de una aplicación web para comprobar la viabilidad de la alternativa (según se mencionó en la sección 3.4 *Tecnologías utilizadas*), y acto seguido se releva el desarrollo a una aplicación de escritorio, dividido en diferentes incrementos detallados en el capítulo 5, y la redacción de la documentación del proyecto.

3.6.3. Presupuesto

Finalmente, se aproxima el coste del proyecto según la planificación temporal estimada en la sección anterior. Siguiendo el diagrama de Gantt de la figura 3.2, el proyecto se desarrolla a lo largo de, aproximadamente, 36 semanas (cada subdivisión vertical equivale a una semana). Considerando una media de 10 horas semanales para el desarrollo, se tendría una extensión total de **360 horas**.

Siguiendo el convenio colectivo establecido en BOE [2018], el salario total anual a partir del año 2020 se establece en 24.640,37 euros para la categoría de analista programador (grupo C, nivel I); la jornada laboral que establece el mismo boletín corresponde a 1.800 horas anuales. Con esto se puede aproximar el salario por cada hora en **13,6891 euros/hora**.

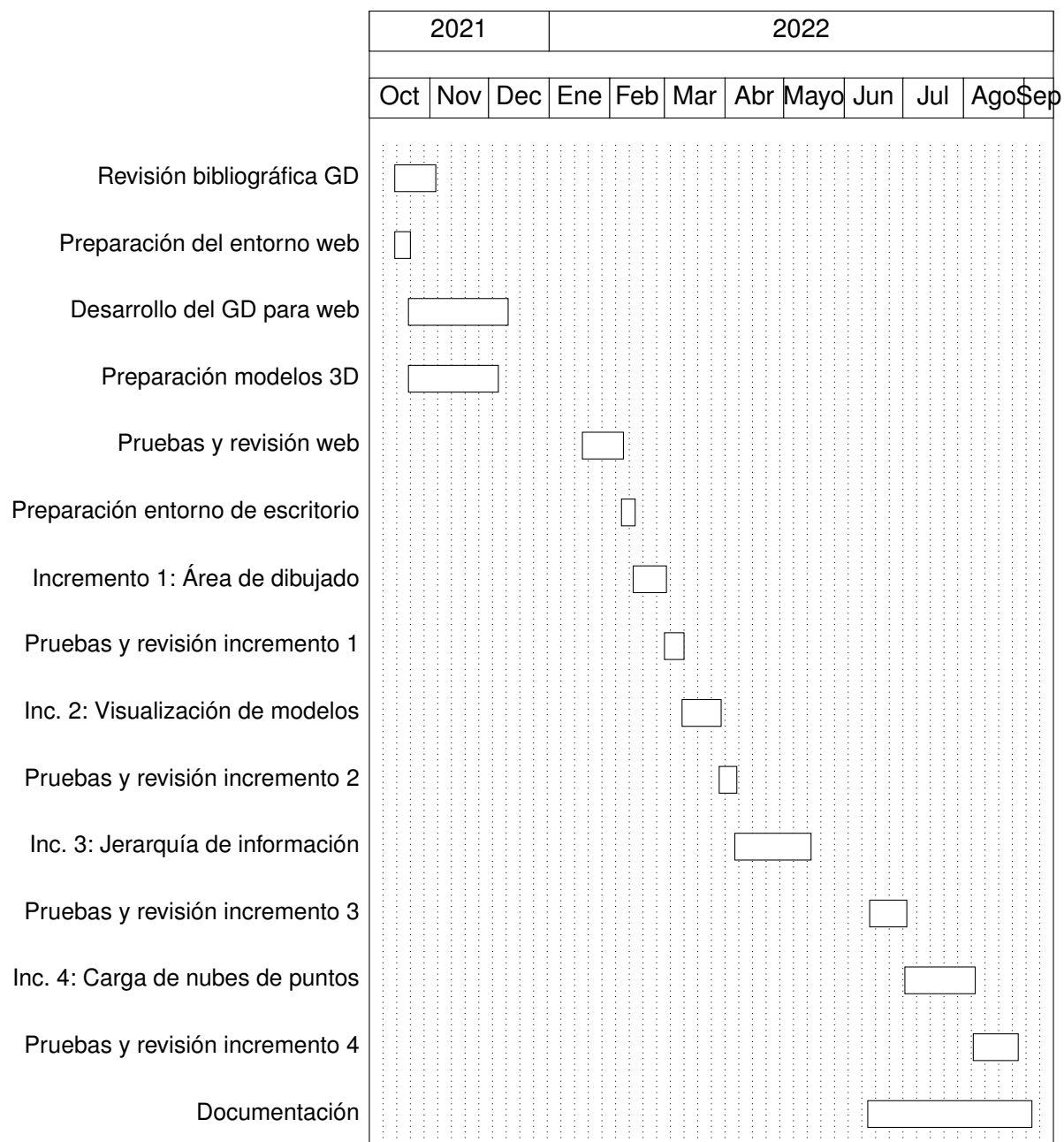


Figura 3.2: Diagrama de Gantt

El equipo empleado para el desarrollo de software cuenta con las características hardware y el precio estimado al comienzo del proyecto¹¹ del siguiente listado:

- CPU: **Intel Core i7-2600**. Precio aproximado: 150 €.
- GPU: **Nvidia GeForce GTX 1060 6 GB**. Precio aproximado: 250 €.
- Memoria RAM: **1x8 GB DDR3** Precio aproximado: 45 €.
- Placa base: *Socket LGA1155, PCIe x16, 2 ranuras RAM DIMM*. Precio aproximado: 65 €.
- Almacenamiento: **HDD 258 GB, SSD 512 GB**. Precio aproximado: 75 €.
- Periféricos genéricos: Teclado, ratón, monitor *1280x720*, monitor *1920x1080*. Precio aproximado: 180 €.

Por otro lado, el software adicional destinado a facilitar ciertas partes del proyecto consta de aplicaciones de uso libre que no requieren la obtención de una licencia previamente. Aunque no añaden coste adicional directo, su uso afecta al tiempo de desarrollo, pues estas herramientas requieren un aprendizaje previo para su uso eficaz. Esto se incluye implícitamente en la planificación temporal descrita en la sección anterior [3.6.2](#).

Adicionalmente, la obtención de información térmica y espacial en forma de nube de puntos requiere sistemas y dispositivos de altas prestaciones, que fácilmente alcanzan elevados precios de compra hasta casi los cientos de miles de euros. Por esto, suele optarse por alquilar el servicio durante el tiempo suficiente, reduciendo el coste a aproximadamente los **1.500 euros** por día de uso.

El proyecto en su totalidad implicaría un coste de **4.928,076 euros** en salario para la profesión de analista programador, más **765 euros** del equipo hardware de trabajo, más unos **1.500 euros** destinados a la obtención de información térmica y espacial mediante nubes de puntos durante un único día. El coste final del proyecto se aproximaría a los **7.193,076 euros**.

¹¹El precio se aproxima tomando en cuenta el precio recomendado de los componentes conocidos o con características similares revisados en septiembre de 2021 y julio de 2022. La diferencia temporal entre revisiones se debe a la inestabilidad provocada en el mercado por la *COVID-19*, que podría provocar cambios en la estimación del coste según la fecha de compra de los componentes.

Capítulo 4

DISEÑO INICIAL

Tras definir en el capítulo anterior aspectos más específicos del proyecto a desarrollar, este siguiente capítulo se dedica a la parte esencial de ingeniería del software. Una de las características del concepto de ingeniería del software consiste en que, debido a ser un campo en constante evolución, su definición cambia y se adapta a las necesidades según sea necesario. [Pressman \[2010\]](#) menciona al inicio de su primer capítulo a modo de introducción esta necesidad de adaptarse a los nuevos enfoques del software, y procede a continuación a comentar las distintas partes que componen la ingeniería del software.

En este proyecto, al hacer uso de una metodología incremental, el diseño quedará separado en los conceptos generales, comunes a toda la aplicación, seguido posteriormente del desarrollo de los incrementos desde su diseño hasta la fase de pruebas. De esta forma se definirán con claridad los requisitos y consideraciones (en cuanto a funcionalidad esencial o sobre la interfaz de usuario) que cada incremento debe respetar durante su desarrollo.

4.1. Especificación de requerimientos

El primer paso del proceso de ingeniería del software consiste en definir los requisitos de la aplicación a desarrollar para que alcance el propósito del proyecto.

4.1.1. Requerimientos funcionales

Un requisito funcional de un sistema software es aquel que define su comportamiento sobre la funcionalidad esencial que ofrece al usuario. Estos requisitos abarcan todo lo que la aplicación debe permitir según lo requiera el usuario.

Los requisitos funcionales que debe abarcar el sistema desarrollado se enumeran a continuación:

- **Generación del modelo 3D de la instalación solar (RF01):** En primer lugar, se indica dónde quedarán colocados tanto los paneles como los diferentes modelos decorativos de la escena, empleando un lienzo donde se visualice la imagen satélite asociada a la instalación real, sobre el cual podrán definirse los distintos elementos. Una vez definido el posicionamiento, se genera la escena permitiendo interactuar con los paneles generados y consultar la información asociada al panel seleccionado.
- **Gestión de múltiples instalaciones (RF02):** Se generan varias escenas tridimensionales representando diferentes instalaciones solares, permitiendo cambiar en cualquier momento entre ellas.
- **Carga de datos (RF03):** Los paneles generados en una escena se poblarán de datos mediante la información recolectada por los inversores de la instalación real, o bien a través de una nube de puntos térmica.
- **Consulta de información (RF04):** Mediante la interacción con la escena se consultará la información de la instalación con respecto a panel, inversor o de la instalación en su totalidad. Se habilita la búsqueda para acceder fácilmente a los datos correspondientes a un elemento concreto, por ejemplo: un inversor a partir de su nombre o el número de paneles que agrupa.

El enfoque principal del proyecto será sobre los dos primeros requisitos, pues se relacionan mucho más con la informática gráfica; aun así, la aplicación dispondrá también de la funcionalidad esencial para los requisitos *RF03* y *RF04*, aunque de forma básica que permitan su extensión en el futuro.

4.1.2. Requerimientos no funcionales

Los requisitos no funcionales detallan el funcionamiento del sistema, indicando principalmente restricciones y comportamientos específicos sobre las diferentes ope-

raciones. Son más complicados de definir frente a los requisitos funcionales porque caracterizan y detallan la aplicación, mientras que los funcionales se centran en definirla de forma más genérica.

Los requisitos no funcionales de la aplicación a desarrollar serían:

- Rendimiento máximo del sistema, minimizando el uso de **memoria principal** y el **tiempo de respuesta**.
- Capacidad de cargar y gestionar modelos 3D y nubes de puntos de **gran tamaño** en un tiempo aceptable (millones de caras/puntos).
- **Requisitos de interfaz de usuario**, que abarcan los requisitos relacionados con la usabilidad del sistema:
 - **Facilidad de uso**: simpleza en los controles e interacción, permitiendo acceder fácilmente a toda la funcionalidad.
 - **Flexibilidad**: utilización de diversas maneras de intercambio de información con el sistema implementado.
 - **Robustez**: resistencia o tolerancia a errores.

4.2. Análisis del sistema

La siguiente fase se centra en analizar y crear un modelo para el desarrollo de la aplicación, de forma que abarque todos los requerimientos a la vez que mantenga cierta robustez y permita la ampliación del sistema.

4.2.1. Perfiles de usuario

Durante el diseño de cualquier sistema software deben considerarse los distintos tipos de usuario que harán uso de la aplicación, y analizar acordemente el sistema para satisfacer los requisitos de usabilidad que requieran. En la definición de perfiles de usuario se omite al administrador del sistema, encargado del mantenimiento y ampliación de la aplicación, para indagar más en el resto de usuarios finales que interactuarán con el sistema.

Para el sistema a desarrollar se define principalmente un perfil de **usuario intermedio**, que posee conocimientos de informática para manejar una aplicación gráfica

interactiva eficazmente, además de conocer las características básicas de una instalación solar que le permita usar correctamente el sistema para alcanzar sus objetivos.

Generalmente, suelen definirse diversos perfiles en proyectos software para abarcar toda la audiencia objetivo; en esta ocasión, se reduce a un perfil que aúna las dos cualidades principales requeridas para la utilización del sistema. Suponiendo que un usuario novel quiera hacer uso de la aplicación, es muy posible que su experiencia se vea obstaculizada al no disponer de conocimientos previos para el control e interacción. En cuanto a la generación del modelo virtual de la instalación solar, la falta de conocimiento puede provocar incongruencias con la equivalencia al modelo físico real. Por estos motivos, la propuesta de perfiles de usuario se reduce a un usuario con experiencia previa básica, siendo además extensible a usuarios expertos.

4.2.2. Casos de uso

Los casos de uso definen situaciones de utilización del sistema y la interacción del mismo con el usuario. Permiten detallar diversas clases y sus conexiones necesarias para dar solución al problema que buscan resolver, abarcando adicionalmente los actores (entidades que interactúan con el sistema), el flujo de eventos (acciones a realizar para solucionar el problema), y diversas condiciones tanto de entrada como de salida.

La manera en que comúnmente se representa un caso de uso es mediante un diagrama de frontera, representando mediante un rectángulo la parte interna del sistema, dejando fuera del mismo las partes externas de este. La figura 4.1 muestra un ejemplo de diagrama de frontera.

Partiendo de los requisitos funcionales del sistema, analizados en la sección anterior 4.1 [Especificación de requerimientos](#), es posible obtener los principales casos de uso, listados y detallados a continuación:

1. Generación de la escena 3D

Actor primario: Usuario

Condiciones de entrada: Ninguna

Flujo de eventos:

1. Seleccionar el tipo de objeto a posicionar.
2. Definir el posicionamiento de los objetos mediante interacción con el lienzo.

Visual Paradigm Standard(Juan Carlos Fernández Pérez(Universidad de Jaén))

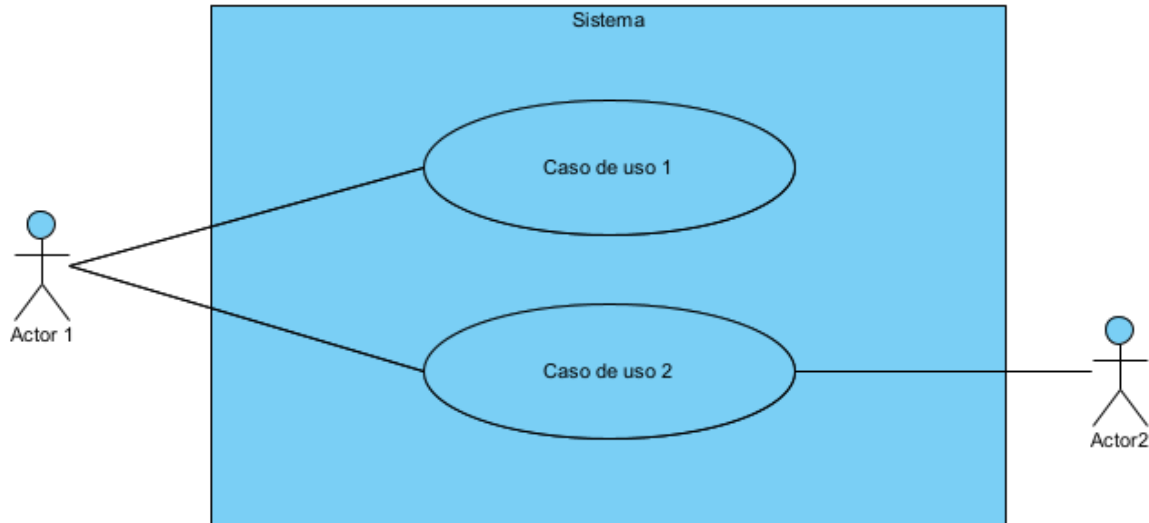


Figura 4.1: Ejemplo de diagrama de frontera

3. Solicitar construcción de la geometría.
4. Visualizar la escena en el entorno tridimensional.

Condiciones de salida: El sistema genera la escena a partir de los datos de posicionamiento dados por el usuario.

2. Virtualización de una instalación adicional

Actor primario: Usuario

Condiciones de entrada: Ninguna

Flujo de eventos:

1. Seleccionar opción de añadir una nueva región para la instalación solar.
2. Parametrizar la nueva región indicando nombre e imagen satélite.
 - 2.A Si los parámetros son correctos, generar la nueva región y ponerla a disposición del usuario.
 - 2.B Si no, indicar los parámetros erróneos.

Condiciones de salida: El sistema genera y mantiene la nueva escena almacenada para su acceso en cualquier momento.

3. Carga de datos

Actor primario: Usuario

Condiciones de entrada: El usuario ha generado el modelo de la instalación solar virtual.

Flujo de eventos:

1. Seleccionar la región en que se encuentra la instalación deseada.
2. Solicitar la carga de datos para la instalación solar.
 - 2.a. Si los datos proceden de un fichero de texto o una base de datos, leer la información relevante.
 - 2.b. Si los datos proceden de una nube de puntos térmica:
 - 2.b.1 Cargar y visualizar la nube de puntos.
 - 2.b.2 Posicionar la nube sobre el modelo virtual.
3. Asignar la información a los distintos elementos del modelo virtual.
 - 3.1 Si se produce algún error o existen incongruencias, notificar al usuario.

Condiciones de salida: El sistema añade la información a cada panel del modelo virtual.

4. Consulta de información

Actor primario: Usuario

Condiciones de entrada: El usuario ha generado y poblado la instalación solar virtual.

Flujo de eventos:

1. Seleccionar la región en que se encuentra la instalación deseada.
2. Interaccionar con el elemento del que obtener información.
3. Recopilar información del objeto y agregar si fuese necesario.
4. Mostrar la información recopilada.

Condiciones de salida: El sistema recopila y muestra la información al usuario.

Diagramas de casos de uso

Para los distintos casos de uso definidos, se muestran los diagramas correspondientes a cada uno mediante las figuras [4.2](#), [4.3](#), [4.4](#) y [4.5](#).

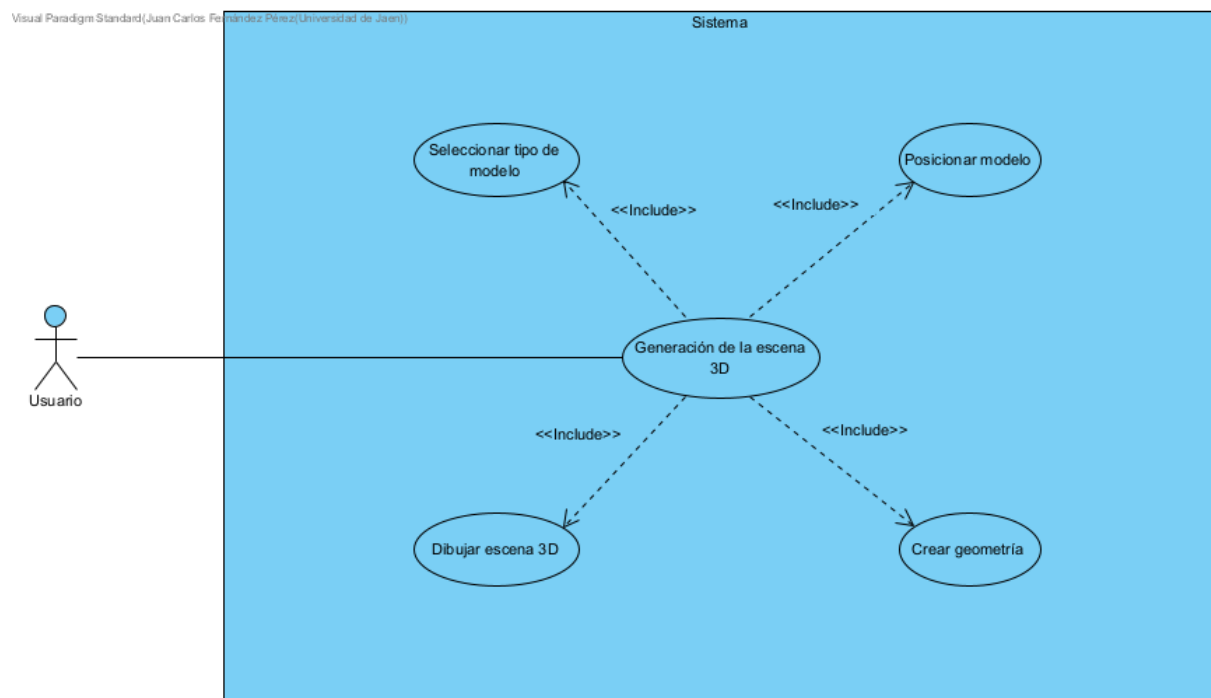


Figura 4.2: Diagrama de casos de uso “Generación de la escena 3D”.

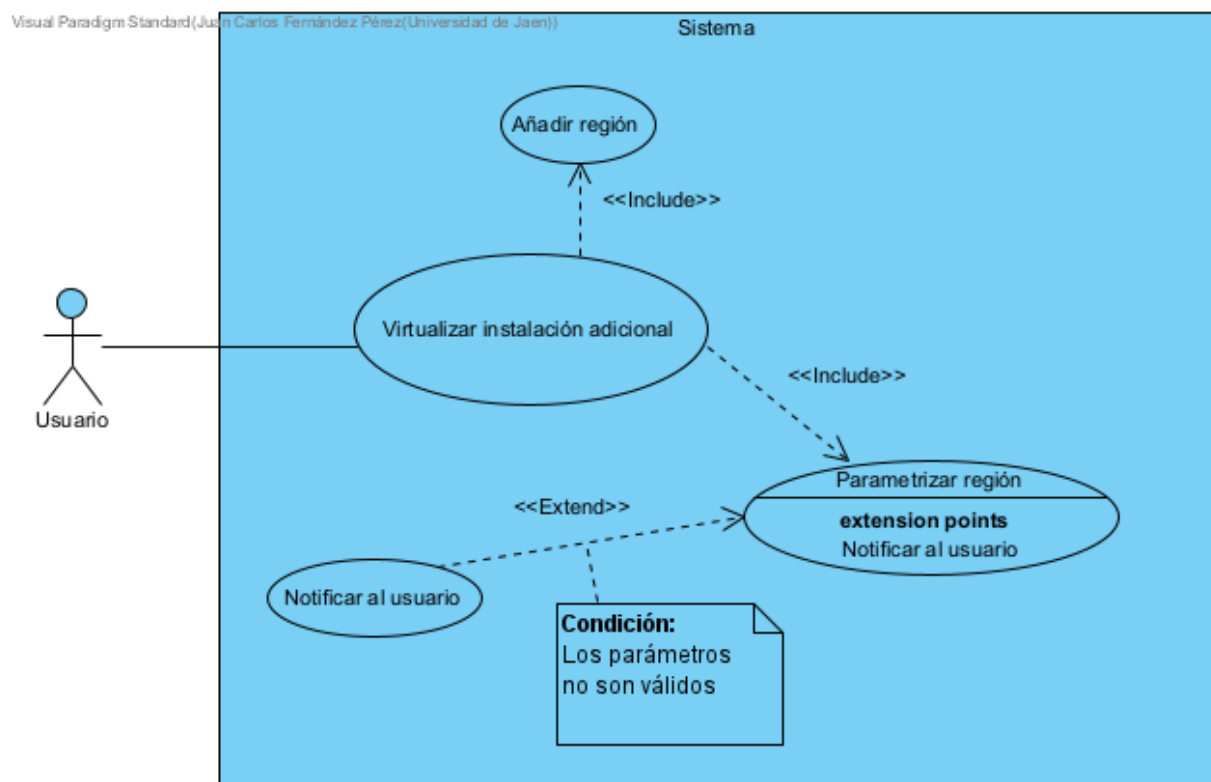


Figura 4.3: Diagrama de casos de uso “Virtualización de una instalación adicional”.

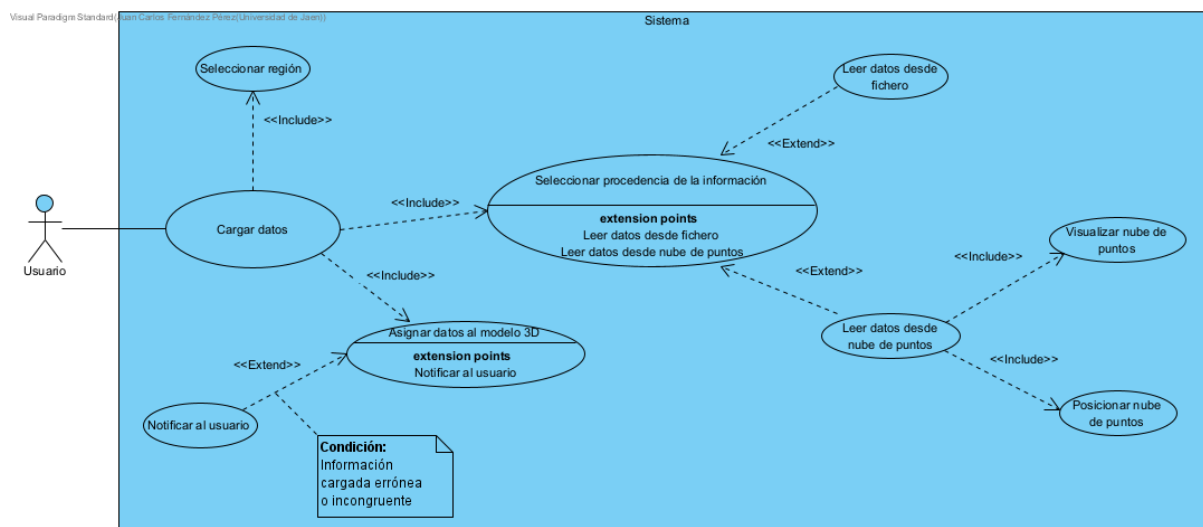


Figura 4.4: Diagrama de casos de uso "Carga de datos".

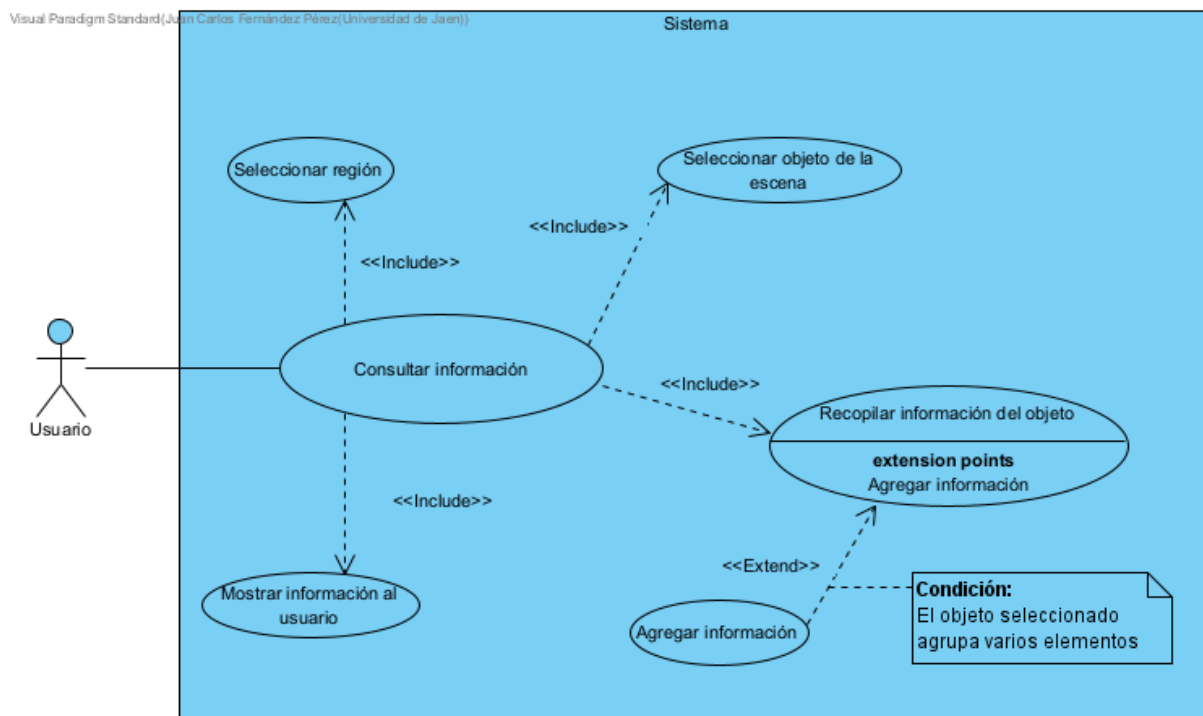
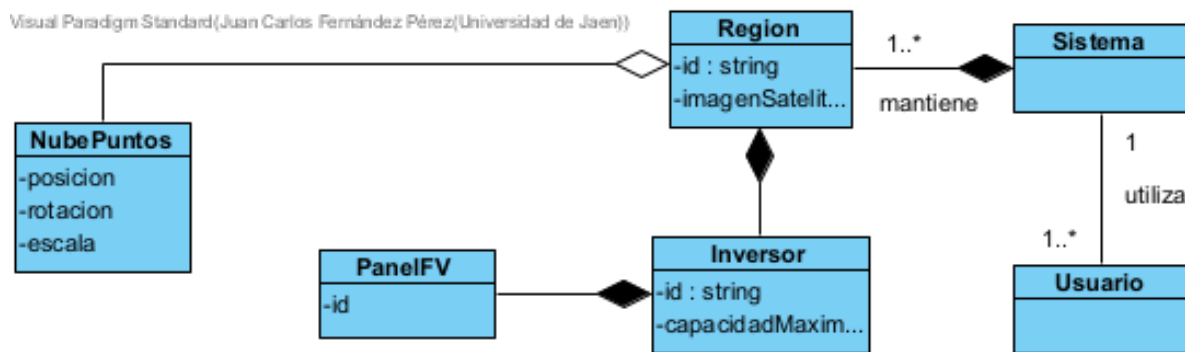


Figura 4.5: Diagrama de casos de uso "Consulta de información".

Figura 4.6: Diagrama *UML* del modelo de dominio

4.2.3. Modelo de dominio

El modelo de dominio, según indica [Craig \[2004\]](#), representa de forma conceptual las diversas clases significativas y la relación entre ellas mediante un diagrama *UML*, facilitando el trabajo a los desarrolladores, especialmente en cuanto a la programación orientada a objetos. Siguiendo el diagrama de la figura 4.6, se describen ahora las distintas “clases” que aparecen en el mismo:

- **Usuario**: Representa a los usuarios que hacen uso e interactúan con el sistema.
- **Sistema**: Representa la aplicación en su totalidad, agrupando los distintos elementos virtualizados y sus parámetros.
- **Region**: Representa cada instalación solar gestionada, empleando la imagen satélite de la región donde se encuentra (o se plantea montar) la instalación real como base de la visualización y del lienzo sobre el que posicionar el resto de objetos de la escena.
- **NubePuntos**: De forma opcional, una región puede incluir una nube de puntos para la visualización. Requiere las transformaciones básicas para posición, rotación y escalado en la escena tridimensional.
- **Inversor**: Representa un conjunto de paneles solares en una instalación solar, agregando la información de los mismos.
- **PanelFV**: Representa un panel generador de la instalación, asociado a un único inversor de la región en la que se encuentra.

4.3. Diseño del sistema

Una vez planteado el análisis del sistema respecto a la funcionalidad y requerimientos necesarios, el siguiente paso consiste en plantear la forma en que se implementará la aplicación de tal manera que esa fase de desarrollo sufra los mínimos problemas inesperados posibles, pues retardarían la finalización del desarrollo e incrementan el coste del proyecto.

Esta tarea no es nada simple; son muchas las condiciones, limitaciones y objetivos a tener en cuenta durante el desarrollo de un producto software que pueden dar lugar a problemas que no solo provoquen los ya mencionados retardos y costos adicionales, también pueden involucrar un cambio en el planteamiento de la solución propuesta.

El objetivo de esta sección consiste en presentar el diseño ideado para el desarrollo de la aplicación, haciendo uso de diversos diagramas que muestren los objetos que interfieren durante la ejecución, sus relaciones, así como el flujo de comunicación entre las diferentes partes del sistema.

4.3.1. Diagrama de clases de diseño

En primer lugar, se prepara un diagrama de clases de diseño de *UML*, centrado en representar las clases (objetos) que aparecen en el sistema junto a las relaciones entre ellas. Adicionalmente, se incluyen los atributos o propiedades y las operaciones básicas de esas clases que permitan controlar su estado y acceder a la funcionalidad que encapsulan respectivamente. La figura 4.7 presenta un diseño de clases inicial sobre el que construir la aplicación gráfica, compuesto principalmente de los siguientes elementos comunes a cualquier diagrama de clases:

- **Clases:** Representan una “plantilla” con la que crear objetos que interfieren en el sistema, indicando sus atributos (estado) y funciones (comportamiento). Es importante no confundir una clase con el objeto en sí, pues pueden parecer idénticos: el objeto es una instancia concreta de una clase.
- **Relaciones:** Una relación entre varias clases indica que una o varias de ellas requiere de las relacionadas para su funcionamiento o creación. Algunas de estos tipos de relación se presentan a continuación:
 - *Asociación:* Es el tipo más básico de relación, existente cuando se produce cualquier tipo de conexión entre las clases.

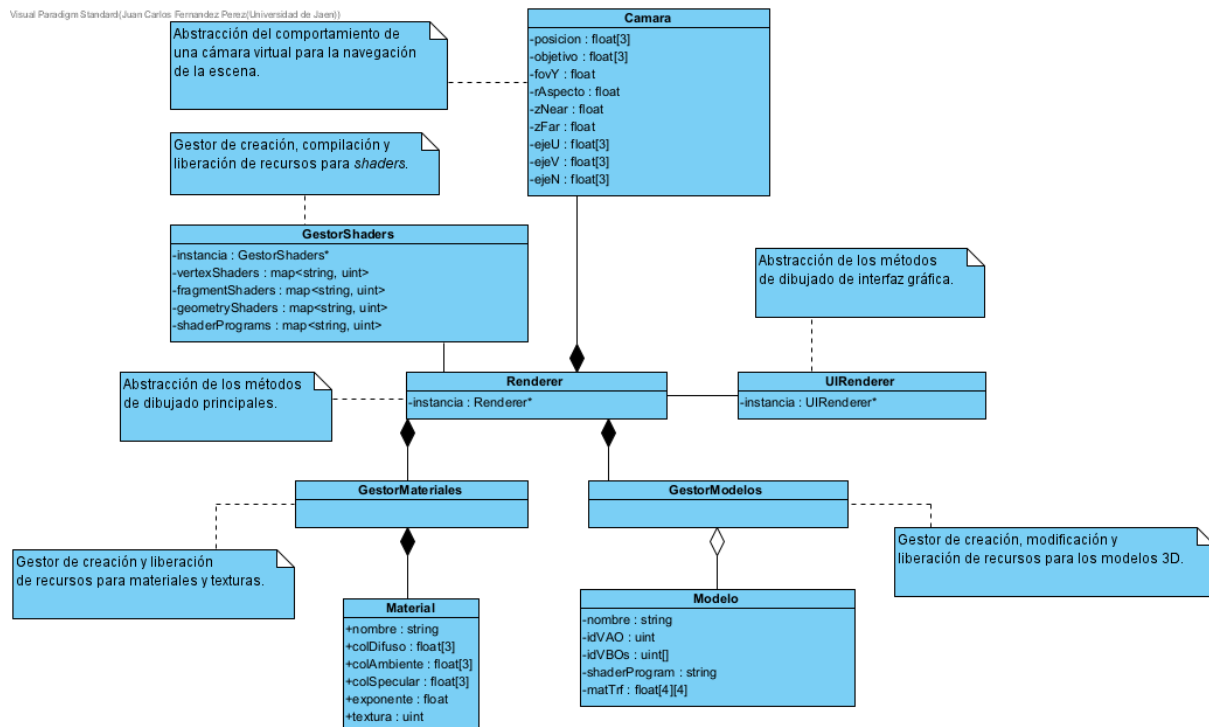


Figura 4.7: Diseño inicial de clases del sistema

- **Agregación:** Indica que una clase está formada por un conjunto de objetos de otra clase o clases. Un ejemplo simple es el de una librería, donde la clase *Librería* estaría agregada por la clase *Libro*; un libro no requiere de una librería para existir, pero una librería sí necesita los libros para ser instanciada.
- **Composición:** De forma similar a la agregación, la composición indica que una clase se compone de un conjunto de objetos de otras clases, pero en esta ocasión esos objetos están ligados al objeto compuesto (si este desaparece, todos los objetos que lo componen también).
- **Generalización:** Representa que una clase es “hija” de otra, es decir, comparte unos atributos y comportamientos básicos con la posibilidad de extenderlos. Un ejemplo sencillo de esta relación sería el de una clase *Animal* correspondiente a la generalización de las clases *Perro* o *Gato*.

4.3.2. Diagramas de secuencia

Otra herramienta muy útil durante el diseño de un sistema son los diagramas de secuencia, también de *UML*, donde se abarca el flujo de la comunicación a lo largo del sistema según transcurre el tiempo, indicando además las clases y actores relevantes,

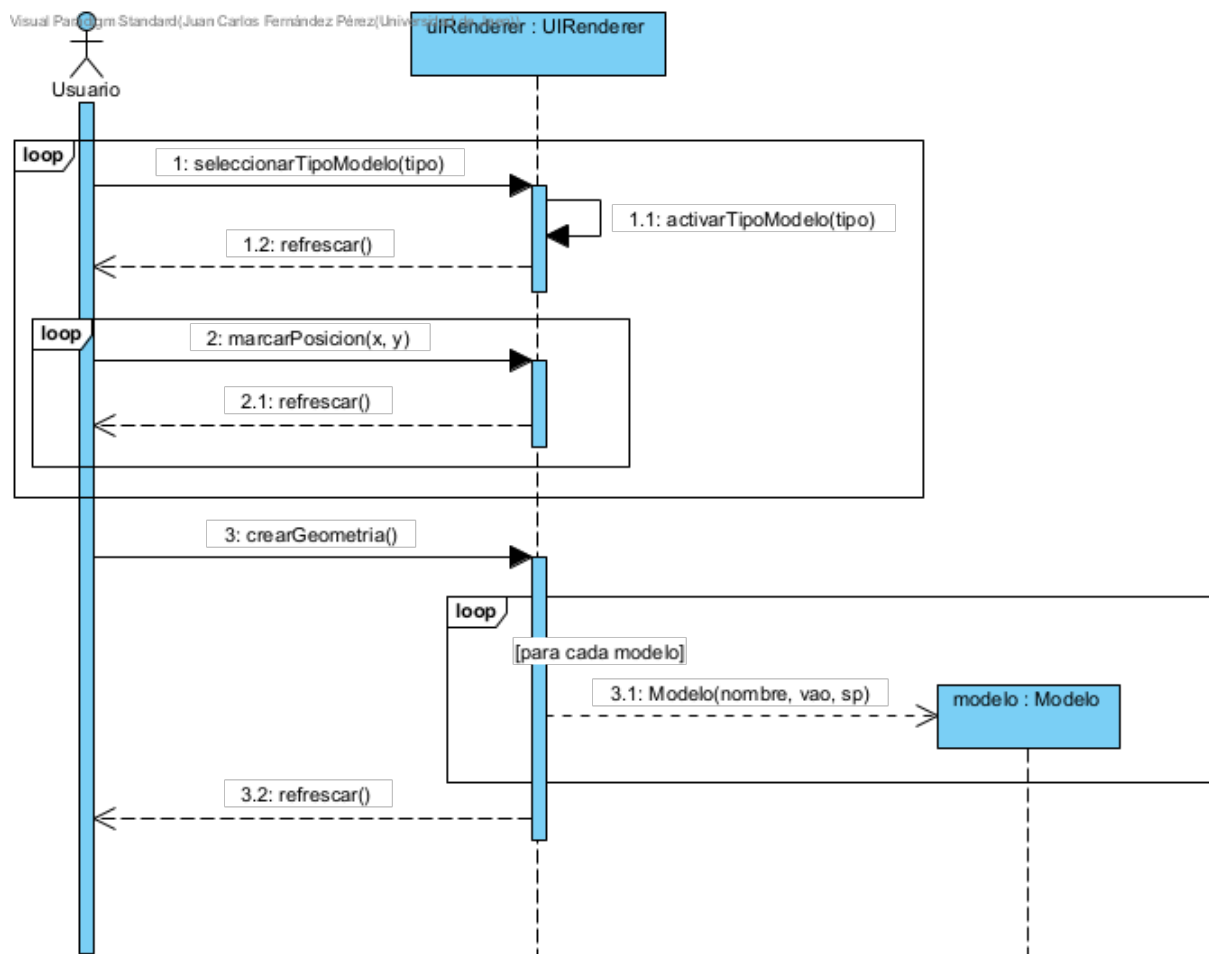


Figura 4.8: Diagrama de secuencia del caso de uso “Generación de la escena 3D”

así como su línea de vida para aquellos objetos que se crean/destruyen durante la transmisión. Las figuras 4.8, 4.9, 4.10 y 4.11 presentan los diagramas de secuencia propios al diseño de los casos de uso indicados. Estos diagramas permiten representar diversos tipos de mensajes según su origen, destino y contenido:

- Los mensajes **síncronos** implican que el emisor ha enviado la carga y esperará a una respuesta antes de reanudar sus tareas.
- De forma similar a los mensajes síncronos, un mensaje **asíncrono** es aquel donde el emisor no se detiene hasta recibir respuesta. Aun así, puede seguir existiendo una respuesta; la única diferencia reside en cuándo el emisor espera a ella o sigue sus tareas.
- La emisión de un mensaje de **retorno** implica que el destinatario ha recibido y tratado el mensaje original, devolviendo al emisor una respuesta que le devuelve el control en caso de ser originalmente un mensaje síncrono.

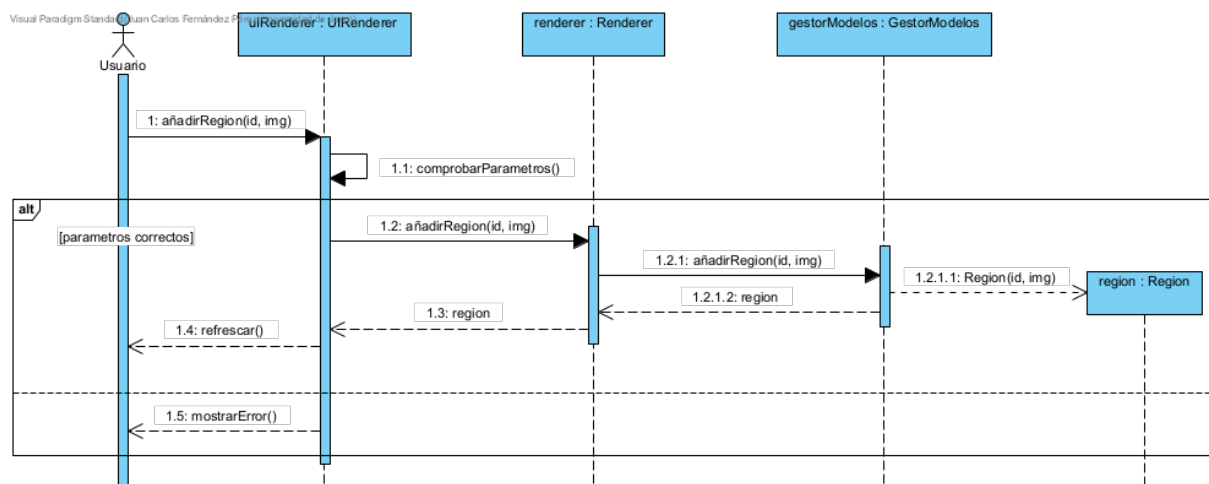


Figura 4.9: Diagrama de secuencia del caso de uso “Virtualización de una instalación adicional”.

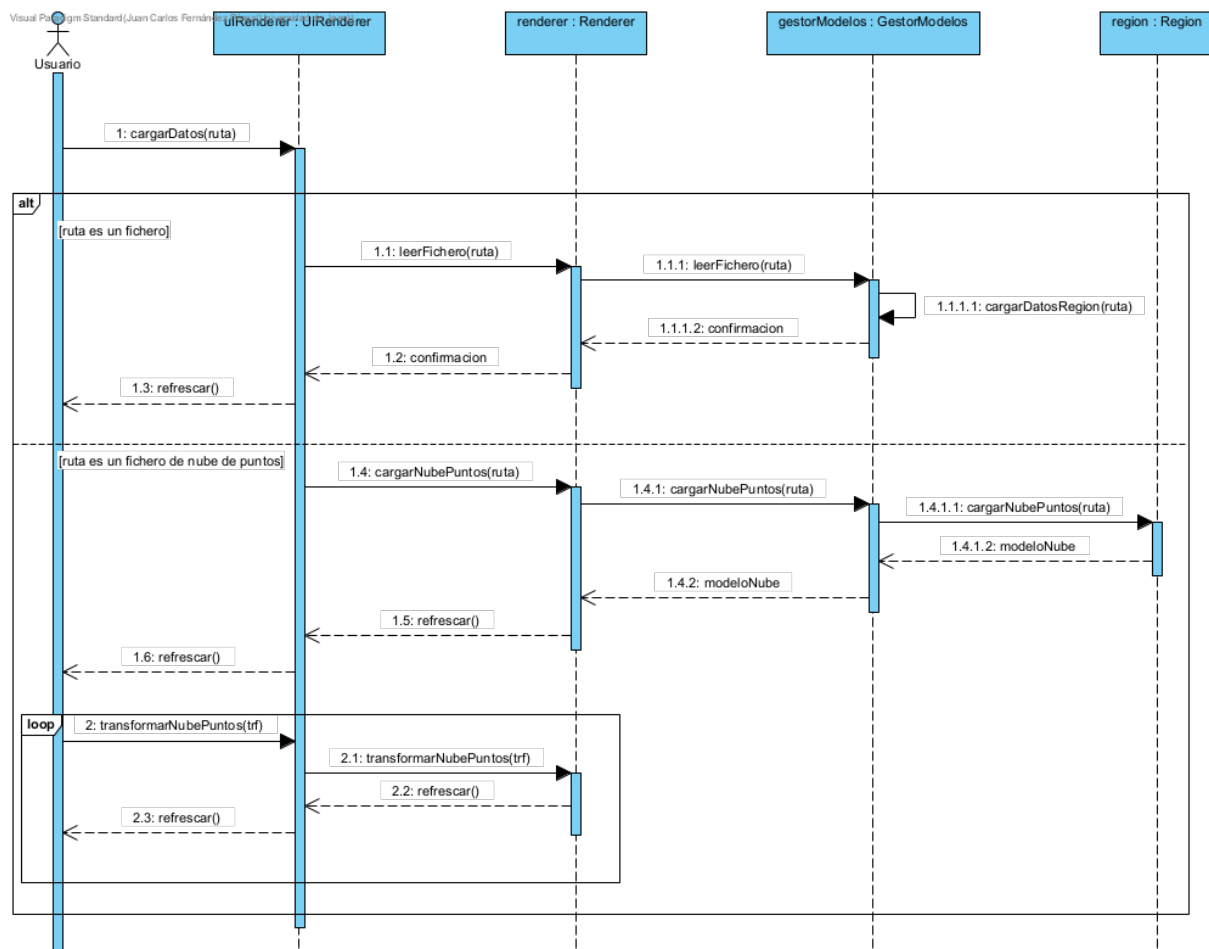


Figura 4.10: Diagrama de secuencia del caso de uso “Carga de datos”.

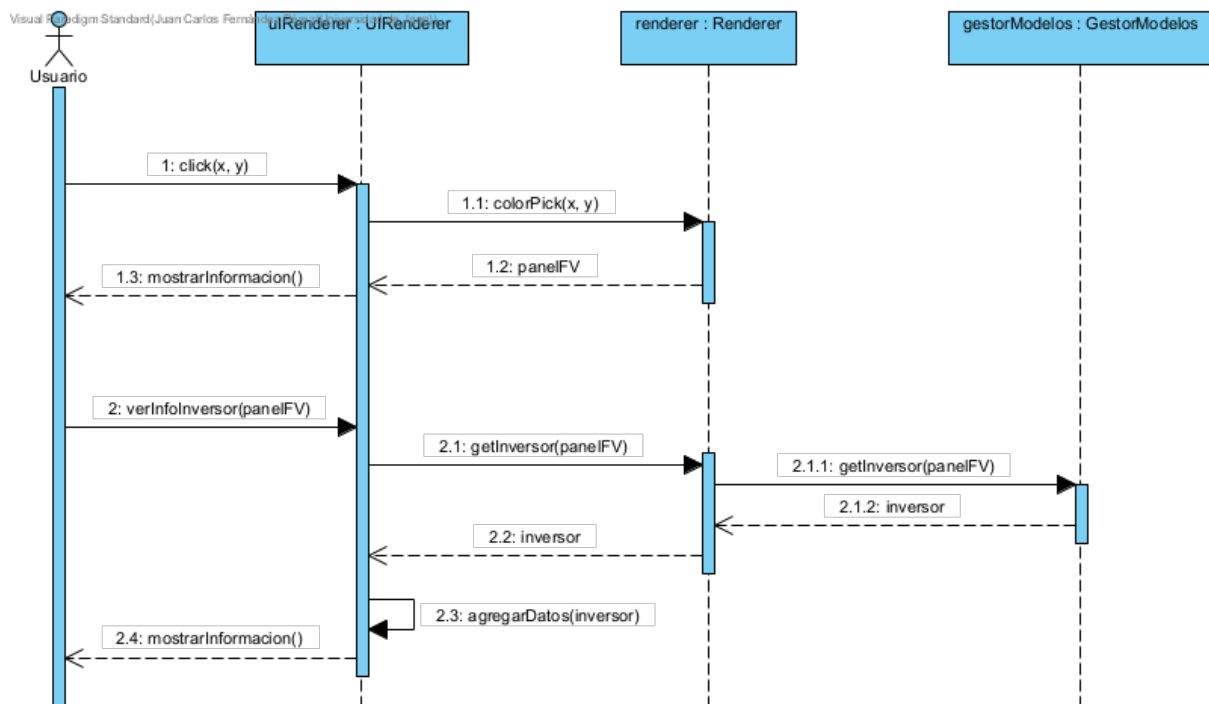


Figura 4.11: Diagrama de secuencia del caso de uso “Consulta de información”.

4.4. Diseño de interfaz e interacción

Finalmente, queda por definir la parte que hará de intermediario entre el usuario y el sistema, formada por las interfaces (principalmente la interfaz gráfica que se presenta al usuario) y la interacción (las acciones que se pueden realizar con las interfaces).

4.4.1. Interacción

Cualquier sistema destinado a ser utilizado por personas requiere tener definido un estilo y unas guías generales para la interacción entre ambos extremos para que la comunicación sea simple y precisa. En una aplicación gráfica, la interacción es esencial, ya que el sistema en su totalidad se basa en dar respuesta a las acciones que realiza el usuario en cualquier momento.

Para el desarrollo del proyecto se utiliza la librería *GLFW* junto a *OpenGL*, que encapsula toda la gestión de entrada y salida (teclado, ratón, ventanas...) y facilita el acceso a tales dispositivos, así como a su estado. De esta forma, se podrá habilitar el uso de los periféricos típicos de entrada y salida sin añadir un esfuerzo excesivo, pues *GLFW* se encargará de mantener el estado de los botones, posiciones de punteros y otro tipo de información adicional relacionada con los mismos en todo momento.

4.4.2. Interfaz gráfica de usuario

Una buena interfaz gráfica requiere un diseño previo que simplifique el trabajo requerido por parte del usuario para navegar por las diferentes ventanas de la aplicación.

Al hacer uso de *Dear ImGui*, parte del desarrollo de la interfaz gráfica queda simplificado, ya que se encarga de solventar varios problemas que acarrea. Una de las mayores ventajas reside en el estilo que siguen los diferentes elementos que forman la interfaz gráfica, pues comparten características sobre el estilo que propone.

Este último detalle sobre *Dear ImGui* influye además durante el diseño de metáforas para la aplicación. Si bien generalmente presentan gran cantidad de beneficios para la experiencia de uso del usuario, diseñar y crear buenas metáforas que adicionalmente se adapten a las guías de estilo de la librería requiere un esfuerzo considerable. Es por esto que la aplicación estará formada principalmente de textos descriptivos de las acciones que realiza cada elemento de la interfaz gráfica.

4.4.3. Experiencia de usuario (UX)

Otra parte vital de cualquier proyecto destinado al uso por personas es el idear un diseño que, además de ser accesible y garantice facilidad de uso, provea una experiencia que atraiga al usuario a continuar utilizando el sistema. Siguiendo la definición que propone [Don Norman \[2016\]](#), donde la experiencia de usuario abarca todos los aspectos en los que se ve involucrado el propio usuario (no solo la interfaz, también aspectos externos a la aplicación, centrados sobre la empresa o la marca tras su desarrollo), es posible intuir la necesidad de invertir sobre este punto sobre todo cuando el proveedor quiere formar un entorno para el usuario.

En este caso concreto se desarrolla una única aplicación independiente, por tanto, el diseño de la experiencia de usuario puede reducirse a las partes ya analizadas de interfaz e interacción, ya que construyen la principal manera en que el usuario interactuará con el entorno en su conjunto. Un motivo adicional por el que esta parte queda simplificada es debido al tipo de aplicación; una aplicación de escritorio interactiva se centra en una ventana principal sobre la cual se accede al resto de funcionalidades, por tanto, el flujo de la interacción será casi lineal, originando poca complejidad durante la navegación.

4.4.4. Prototipos de interfaz gráfica

A continuación se exponen los prototipos, en forma de *sketch*, de las ventanas que visualizará el usuario a la hora de hacer uso de la aplicación (figuras 4.12, 4.13, 4.14, 4.15 y 4.16). Estos *sketch* son unos dibujos generalmente hechos a mano que permiten planificar los elementos que forman la interfaz gráfica y su posición aproximada en las ventanas y diálogos, siendo una herramienta muy útil en las etapas tempranas del diseño.

El paso siguiente debería ser la creación de unos *wireframe* respectivos a cada *sketch* diseñado para definir finalmente la posición y alineación exacta de botones, textos, imágenes... Debido a que las ventanas y diálogos ideados son simples y con un número reducido de elementos, no se hace realmente necesario la creación de *wireframes* al quedar señalada la posición aproximada de cada elemento.

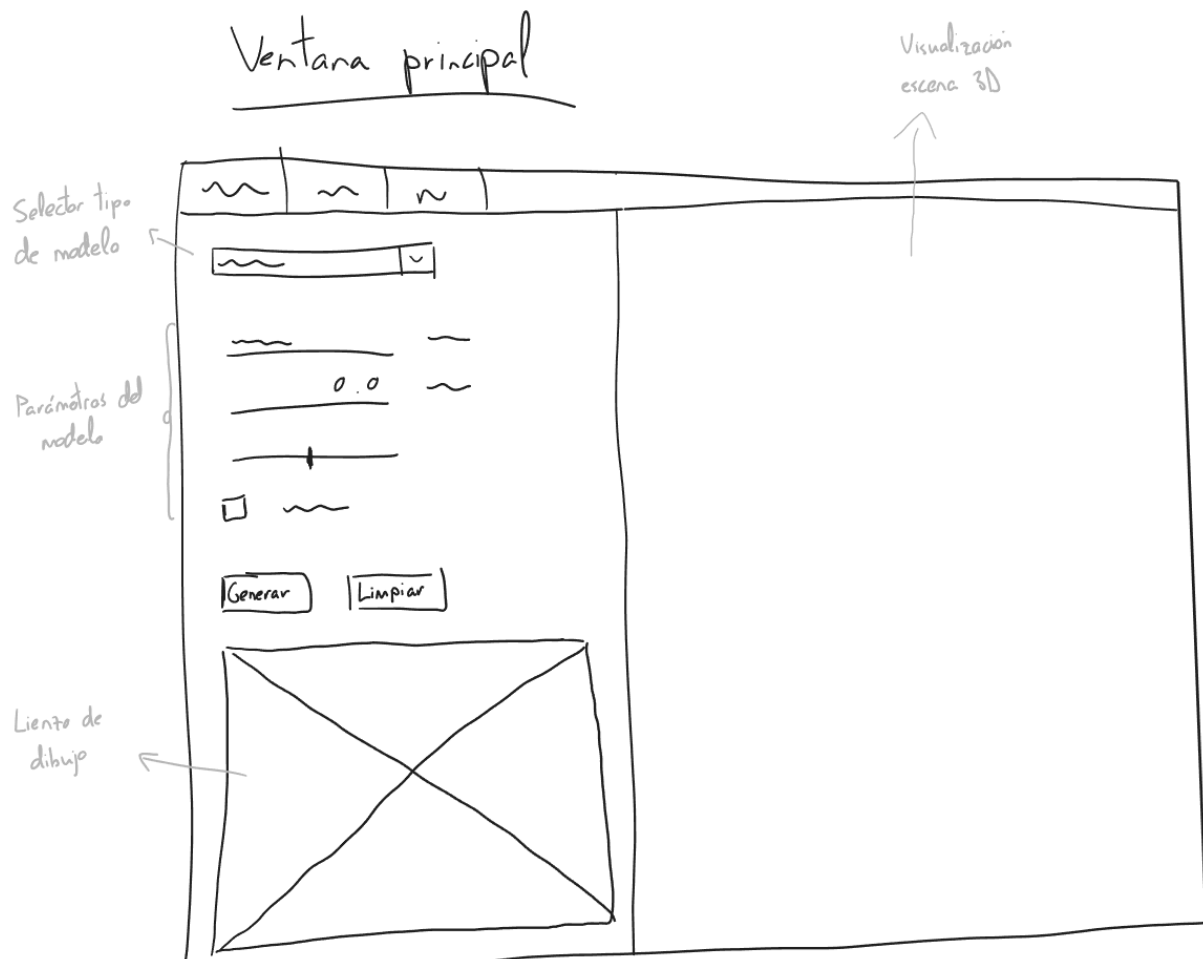
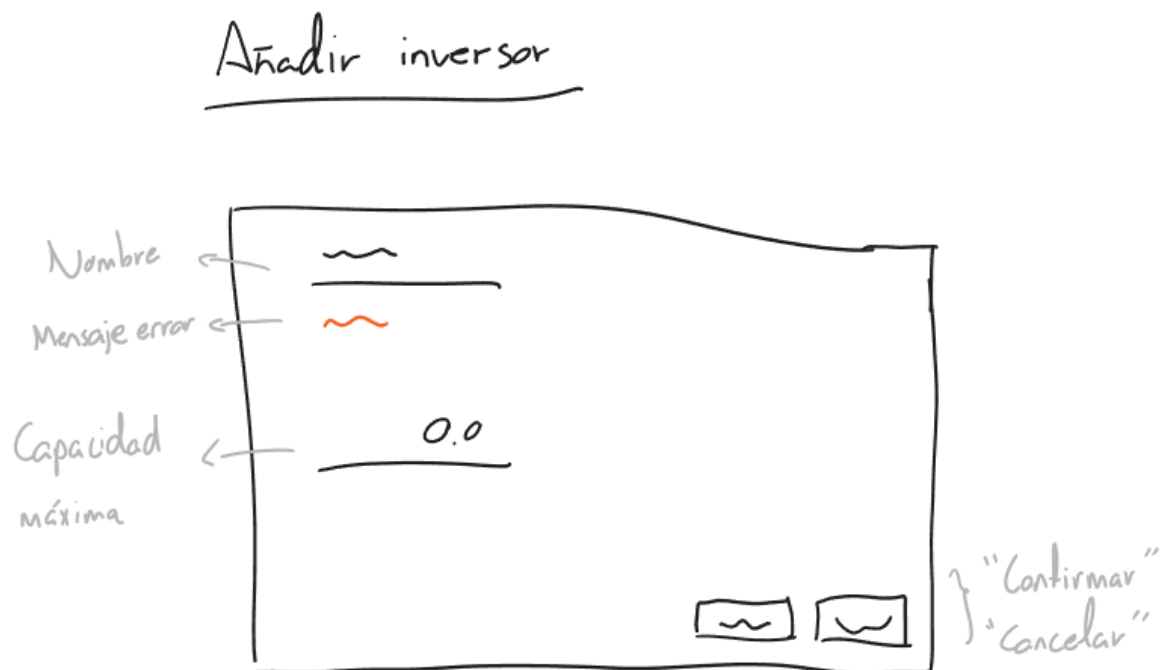


Figura 4.12: *sketch* de la ventana principal de la aplicación

Figura 4.13: *sketch* del diálogo para añadir una regiónFigura 4.14: *sketch* del diálogo para añadir un inversor

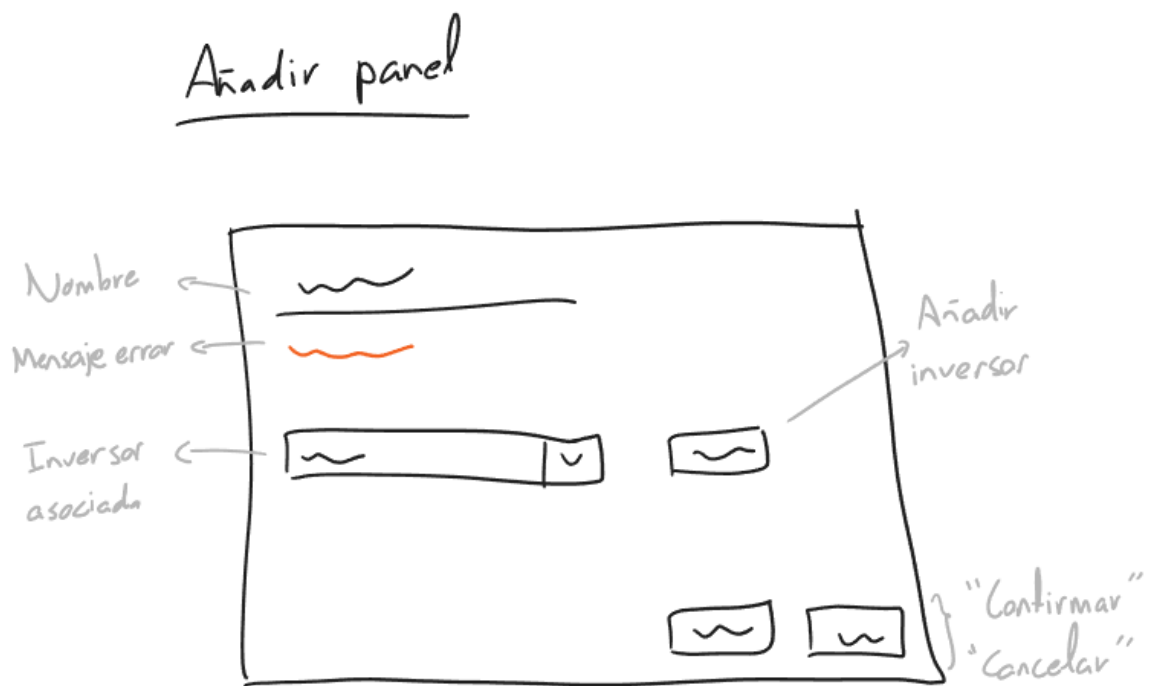


Figura 4.15: *sketch* del diálogo para añadir un panel a un inversor

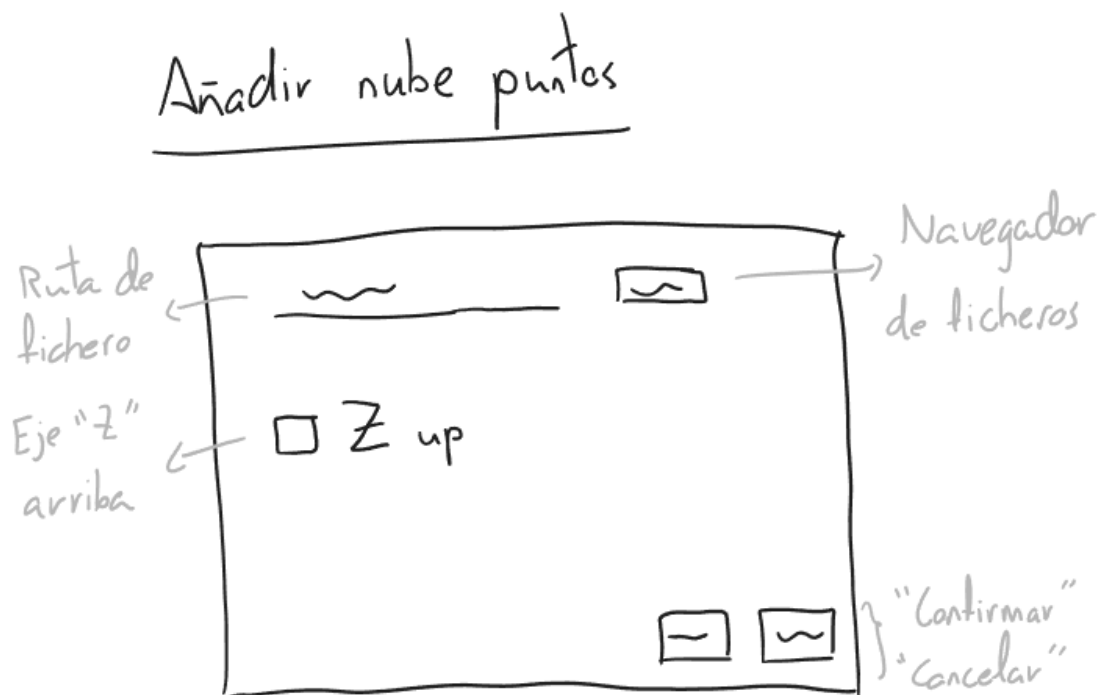


Figura 4.16: *sketch* del diálogo para cargar una nube de puntos

Capítulo 5

DESARROLLO

El siguiente paso tras describir un diseño inicial de la aplicación que abarque los objetivos mínimos que debe implementar es comenzar el desarrollo con la codificación de la funcionalidad propuesta. Este capítulo se dedica a detallar la implementación de los diferentes incrementos por los que la aplicación evoluciona hasta una versión final con la funcionalidad esencial en funcionamiento.

La estructura de la sección para cada incremento consiste en: una primera visión general de la intención del incremento, seguido de subsecciones en las que se especifican aún más los objetivos y la manera en que se implementan, un detalle más amplio de la implementación (junto a diagramas, figuras, pseudocódigo... que faciliten la comprensión del estado del incremento) y finalizando con un conjunto de pruebas con las que verificar el correcto funcionamiento del sistema.

5.1. Incremento 0: Desarrollo web

Tal como se menciona en capítulos anteriores, el primer enfoque del desarrollo se propone como una aplicación web, accesible desde el navegador con la interacción clásica de cualquier aplicación gráfica. El objetivo principal de este incremento consiste en comprobar la viabilidad del enfoque web para la implementación de la funcionalidad requerida, decidiendo finalmente si mantener el esfuerzo sobre este, o dar el salto a otro enfoque con características diferentes o en el que se disponga de mayor experiencia de desarrollo.

5.1.1. Especificación del incremento

Como entorno de desarrollo se escoge **Node.js** para la ejecución de código *JavaScript* en el servidor, ya que junto a su gestor de paquetes, **NPM**, la gestión y utilización de librerías adicionales se reduce a unos pocos comandos desde el terminal.

La implementación de los objetivos se realiza mediante la librería **BabylonJS**, que a su vez utiliza *TypeScript* para una codificación fuertemente tipificada. Con esto, se prepara el entorno tridimensional con posibilidades de interacción y navegación que conste de una recreación del entorno de la instalación, incluyendo el terreno, vegetación y elementos decorativos varios, además de un conjunto de paneles que simulen las placas fotovoltaicas, con diversos niveles de detalle.

Estos niveles de detalle, *Level of Detail* o *LOD*, consisten en subdivisiones del panel hacia elementos del mismo más concretos: el nivel más alto (el más general) será el **generador** en su totalidad, seguido de un nivel intermedio que lo separe en **módulos**, el cual simultáneamente agrupa varios **sub generadores** en el nivel con mayor detalle.

Una vez implementados estos objetivos, se procede a la realización de diversas pruebas para comprobar el rendimiento del sistema y valorar así la viabilidad de esta alternativa de desarrollo de cara a próximos incrementos.

5.1.2. Implementación de la funcionalidad

Entorno web y 3D

En un entorno web cliente-servidor, existen varios instantes que pueden tomarse como punto de entrada: el envío de la petición, la aceptación o rechazo del servidor ante ella, y, en caso de aceptación, la carga del documento principal recibido. Afortunadamente, los navegadores, dispositivos de red y entornos de servidor actuales gestionan automáticamente la gran mayoría de problemas en la comunicación, pudiendo reservar los esfuerzos al documento principal, que será el punto de partida de la aplicación y el entorno tridimensional.

Una vez aceptada la petición y recibidos los recursos del servidor, *JavaScript* abstraer toda la gestión del documento principal y los recursos adicionales a un objeto global, *Document Object Model* o **DOM** para abreviar. La característica principal de este objeto es la posibilidad de buscar los elementos que forman la página web (en

el código *HTML* es idóneo definir identificadores para acceder eficientemente a esos objetos), y la preparación de acciones ante eventos específicos.

BabylonJS debe ejecutarse tras cargar el *DOM*, por tanto, se toma como punto de partida la respuesta al evento “*DOMContentLoaded*”, que ejecutará el contenido una vez se haya cargado el documento principal en su totalidad. Es en este punto donde se busca un elemento de tipo *HTMLCanvasElement* sobre el que se asigna un motor encargado de la visualización 3D en cada fotograma. El listado 5.1 muestra el código que implementa este comportamiento. El resultado de ejecutar la aplicación (acceder al servidor) en este punto resulta en la obtención del documento principal con un entorno tridimensional vacío, sobre el que tampoco se puede interactuar.

```
1 document.addEventListener('DOMContentLoaded', function (event) {  
2     //Obtener el canvas donde renderizar del DOM  
3     const canvas = document.getElementById("renderCanvas") as  
      HTMLCanvasElement;  
4  
5     //Crear un objeto BabylonJS.Engine con ese canvas  
6     const engine = new Engine(canvas);  
7     //Respuesta al evento de redimensionado  
8     window.addEventListener('resize', function () {  
9         engine.resize();  
10    })  
11  
12    //Bucle principal de dibujado  
13    engine.runRenderLoop(() => {  
14        scene.render();  
15    })  
16 })  
17
```

Listado 5.1: Respuesta al evento de carga del *DOM*

LOD de los paneles e interacción

Tras establecer el objeto **Engine** encargado del renderizado, es posible asignarle una escena con modelos y parámetros de visualización, y controlar en general el dibujado de forma previa al bucle principal (en ese punto, recopila sus propiedades para redibujar el siguiente fotograma). Con el fin de simplificar la redacción del código y abstraer funcionalidad, se preparan dos clases: **SceneBuilder.ts**, donde se gestionará todo lo relacionado con la escena y su visualización; y **PanelBuilder.ts**, que encapsula el control de los diversos paneles y sus varios *LOD*.

Comenzando por la clase **PanelBuilder.ts**, se compone principalmente de dos *array* en los que almacenar cada panel construido y el identificador interno que *BabylonJS* le asocia a la malla de triángulos, para que posteriormente pueda realizarse una búsqueda según la selección con el puntero. Cada panel, a su vez, se representa con una clase **SolarPanel.ts** que mantiene parámetros relativos a los diferentes *LOD* y un *array* bidimensional con todos los sub generadores que componen el panel, además de disponer de diversos atributos y métodos para la generación de la malla a dibujar representando ese panel en el espacio tridimensional.

El cúmulo de mallas para los componentes de cada panel generado (el panel completo y las subdivisiones en módulos y sub generadores) se maneja de forma conjunta, mediante un sistema que *BabylonJS* denomina **Solid Particle System, SPS¹**, que trata a todo el conjunto como una única malla a dibujar simultáneamente compartiendo propiedades. Los diferentes objetos y referencias con tipos como *SolidParticle* o *SolidParticleSystem* pertenecen a la librería y encapsulan esta funcionalidad mencionada.

Respecto a los niveles de detalle, *BabylonJS* contempla la posibilidad de añadir *LOD* a una malla con simplemente el método *addLodLevel(number, Mesh)*, cuyos parámetros representan la distancia respecto a la cámara tras la cual se emplea la malla indicada como segundo parámetro. A pesar de que la librería gestiona la visualización de los diversos niveles de forma automática, durante la interacción con el puntero para seleccionar uno de los paneles debe manejarse manualmente, por esto aparecen atributos en *PanelBuilder.ts* referenciando al elemento seleccionado en cada momento ("*selectedParticle*", "*selectedGenerator*", y "*selectedPanel*").

Estos últimos parámetros se actualizan cuando el usuario interacciona con el entorno mediante el clic principal del ratón sobre el *canvas* de *HTML* en que se dibuja la escena. Para ello, se aprovecha la clase **Scene** de *BabylonJS* al presentar diversos métodos y propiedades para controlar la interacción con la escena activa; concretamente, *onPointerDown*, que permite actuar ante el evento de presionar el clic del puntero. Al detectar este evento, se llama al método **PanelBuilder::pickPanel(...)** para comprobar, entre los paneles creados, si alguno ha sido objetivo del puntero.

La búsqueda del panel seleccionado es lineal sobre los *array*: en primer lugar, se recorren los identificadores de las mallas de cada panel, ya que, independientemente del nivel de detalle, todos los módulos y sub generadores comparten el mismo identificador del panel en que se encuentran; si alguno coincide, la siguiente búsqueda se realiza sobre el *LOD* concreto según la distancia a la cámara principal para ob-

¹https://doc.babylonjs.com/divingDeeper/particles/solid_particle_system/sps_intro, comprobado en 2022-09-04

tener qué módulo o sub generador ha sido seleccionado. Finalmente, se actualizan los atributos necesarios para mantener esa selección y poder resaltarla visualmente y actualizar la información mostrada en la interfaz gráfica de usuario.

Decoración del entorno

Concretando ahora la utilidad de la clase **SceneBuilder.ts**, su uso queda destinado tanto a la gestión de los paneles desde *PanelBuilder.ts*, como a la interacción del usuario y creación del entorno (en términos de elementos decorativos). Anteriormente, se ha mencionado el tratamiento de eventos de ratón empleando propiedades de la clase *Scene*; para el teclado también existe esa posibilidad, pero *JavaScript* presenta una solución más directa mediante los métodos de escucha de la ventana del navegador, que devuelve la información del evento, incluyendo la tecla presionada directamente.

Con el fin de facilitar la población de la escena, se emplea la interacción por teclado junto al sistema que ofrece *BabylonJS* de *gizmos*, herramientas que permiten interactuar con el entorno de una forma más directa, sin recurrir exclusivamente al código (por ejemplo, el *gizmo* de traslación permite desplazar un modelo por la escena arrastrando con el puntero en la dirección deseada). La librería también cuenta con un inspector (debe gestionarse su visibilidad manualmente desde el código) con el que modificar diversos aspectos del funcionamiento del motor de visualización, controlar elementos de la escena, monitorizar el uso de recursos, etc.

El siguiente punto es la población de la escena con modelos decorativos y ajustes del entorno. Recurriendo al sistema de animaciones de *BabylonJS*, se dispone una recreación de ciclo día-noche mediante una luz direccional cuyo vector director es rotado hasta dar una vuelta completa, y, por tanto, provocando cambios constantes en la iluminación y sombreado que aparenten el paso del tiempo.

En cuanto a elementos decorativos, se idea e implementa un pequeño sistema con el que añadir modelos entre un conjunto cargado previamente a la escena. Concretamente, se emplean modelos de diversos árboles ya preparados² que se duplican (en términos de instancia, no de geometría, para evitar redundancia de información) y posicionan según corresponda.

Esta última labor de mantener las copias virtuales de los modelos se implementa en la clase **SceneBuilder.js** con un par de colecciones de tipo mapa, donde la clave será un identificador, y el valor será la malla del modelo cargado para uno, dejando

²Casi todos los recursos visuales se obtienen de colecciones de acceso y uso gratuito: <https://www.turbosquid.com/3d-models/free-obj-mode-tree-pack/506851>, comprobado en 2022-09-04

para el segundo el número de veces clonada la malla con misma clave. Con esto, es posible generar multitud de árboles y posicionarlos en la escena a placer, ya que quedan internamente diferenciados con identificadores únicos (por esto se emplea un mapa secundario como contador de clones, si varios objetos comparten identificador, pueden aparecer problemas inesperados en el comportamiento de la librería).

Finalmente, las clases constructoras disponen de métodos para la generación de la escena mediante un fichero que contenga esa escena serializada, para poder así mantener los cambios realizados durante la ejecución. La figura 5.1 presenta una captura de la aplicación ejecutando la escena básica con diversos paneles y elementos decorativos dispuestos sobre el espacio, empleando como guía aproximada una imagen satélite a modo de suelo. Es posible apreciar la falta de generación de sombras y puntos en que el sombreado es demasiado intenso (principalmente en algunos de los árboles, dejando algunas hojas y troncos en completa oscuridad), a pesar de que la aplicación está preparada y dispone de métodos para el cálculo de mapas de sombras, facilitado por *BabylonJS*, por lo que se hace necesario una revisión futura en la cual garantizar su correcto funcionamiento en las distintas plataformas donde se ejecute.

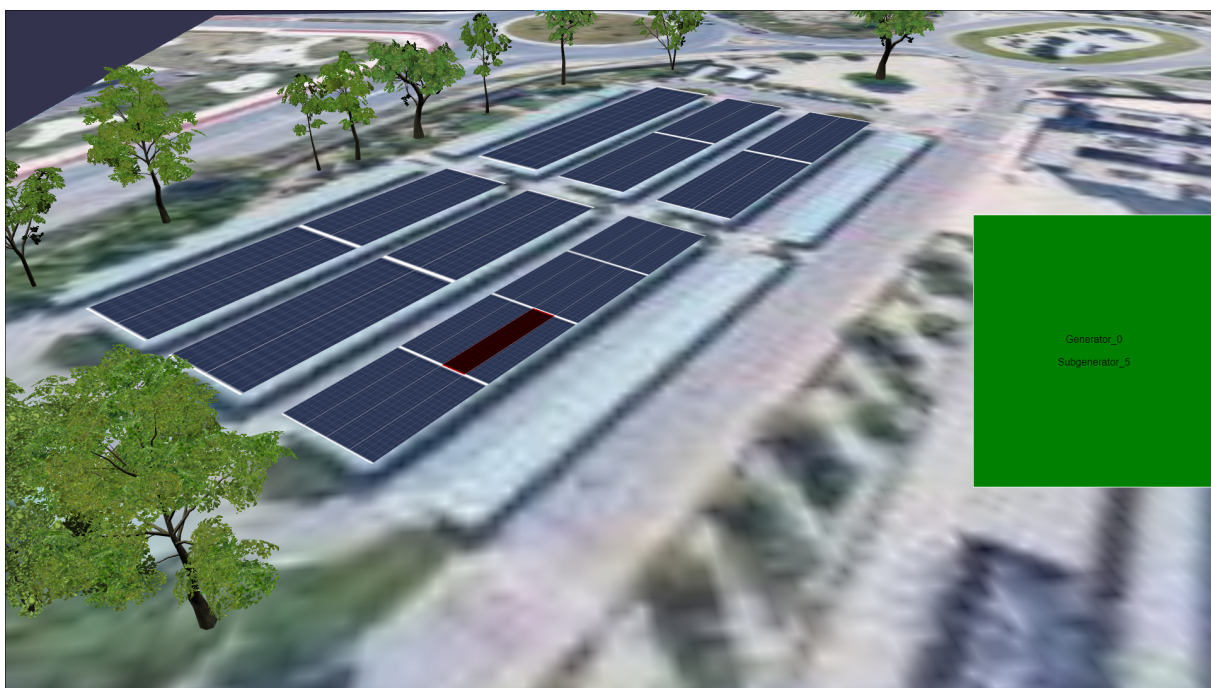


Figura 5.1: Aplicación web en funcionamiento

Diagrama de clases

La figura 5.2 muestra el diagrama de clases implementado para el desarrollo de este incremento, detallando el contenido de cada clase y las relaciones entre ellas.

Visual Paradigm Standard(Juan Carlos Fernandez Perez(Universidad de Jaen))

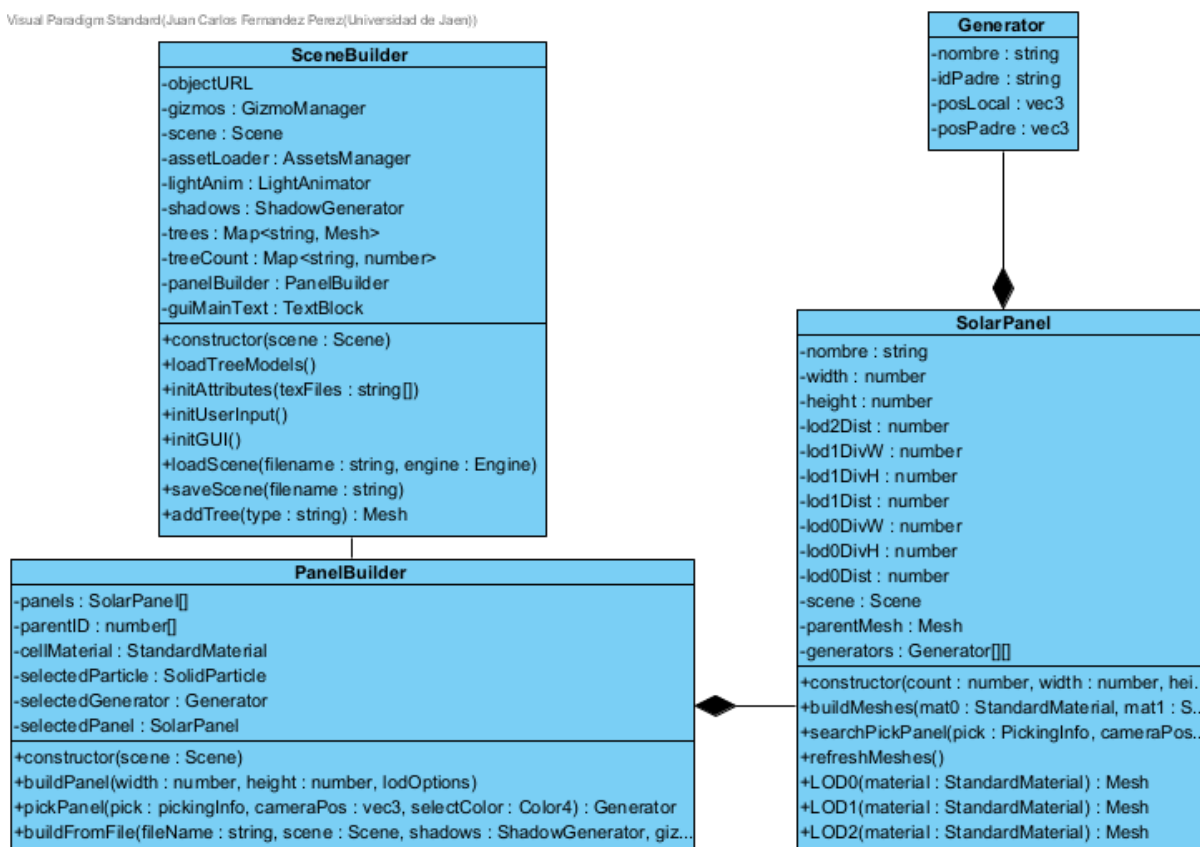


Figura 5.2: Diagrama de clases del Incremento 0: Desarrollo web

5.1.3. Pruebas de funcionamiento

Antes de concluir en la viabilidad de esta alternativa de desarrollo, se han realizado un par de pruebas sencillas para verificar el funcionamiento de la aplicación en diversos navegadores, comprobando si existen problemas para ejecutarla en cuanto a incompatibilidad o rendimiento. Las pruebas de rendimiento se realizan colocando un conjunto de 1500 paneles dispuestos en bloque, con los tres niveles de detalle (panel, módulo y sub generador). El equipo donde se realizan las pruebas se define en el capítulo 3, durante un uso normal (diversas aplicaciones en segundo y varias pestañas adicionales abiertas en el navegador) y con el servidor ejecutándose en la propia máquina. Dichas pruebas consisten en una interacción con la aplicación durante un lapso de cinco minutos, cambiando cada pocos minutos la posición de la cámara para provocar actividad en el proceso de dibujado.

La imagen de la figura 5.3 muestra una captura de la aplicación ejecutando la escena con la configuración de paneles mencionada para la prueba de rendimiento.

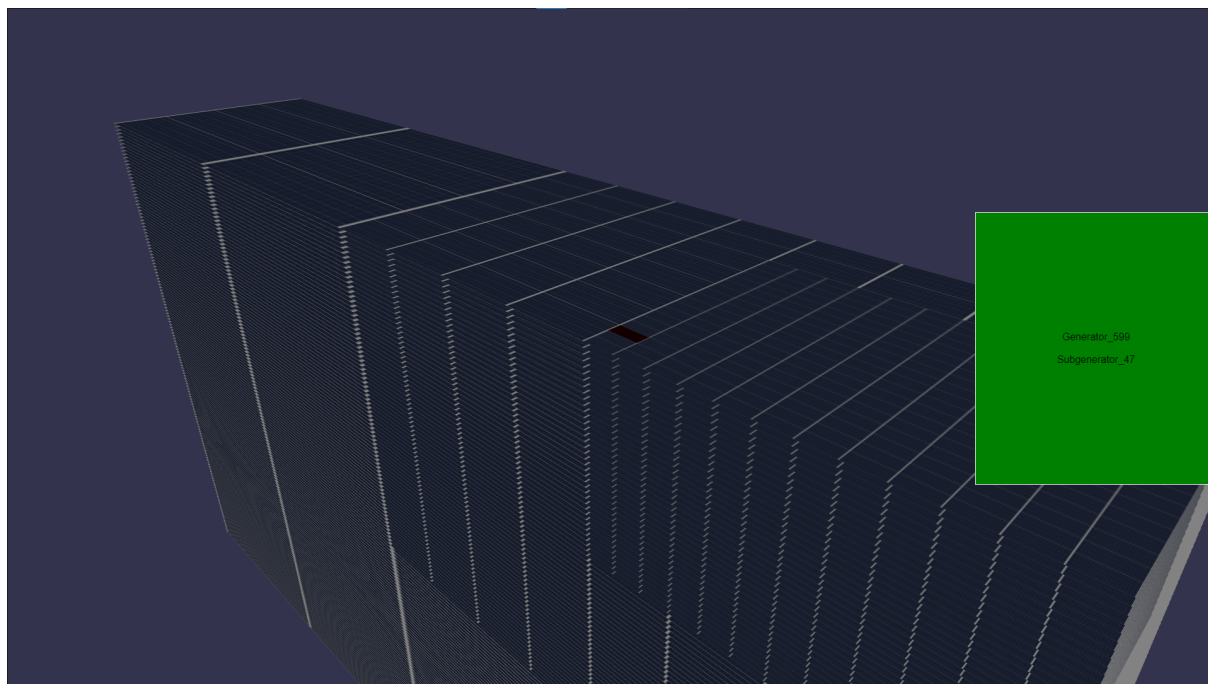


Figura 5.3: Escena empleada para las pruebas web

Opera GX

El primer navegador comprobado es **Opera GX** en su versión **89.0**. La figura 5.4 muestra el rendimiento del sistema durante los minutos de la prueba. Se puede apreciar como, durante la extensión de la prueba, uso de *CPU* se mantiene en el mismo rango, pero con gran cantidad de picos de utilización (da la impresión de ser debido al uso de aplicaciones en segundo plano y el sistema operativo). En cuanto a *GPU*, su porcentaje de uso queda también relativamente constante en los periodos donde no hay interacción, y varía al desplazar la cámara según la cantidad visible de paneles.

Por otro lado, la figura 5.5 muestra el inspector de *BabylonJS* tras la prueba. Dejando para más adelante el conteo de elementos activos en la escena, el apartado de tiempo medio de fotograma en *GPU*, “*GPU Frame time (average)*”, indica una media de 13.42 ms por fotograma; este tiempo supone, en condiciones óptimas, una tasa de fotogramas superior a 60 fotogramas por segundo, *FPS*, de media.

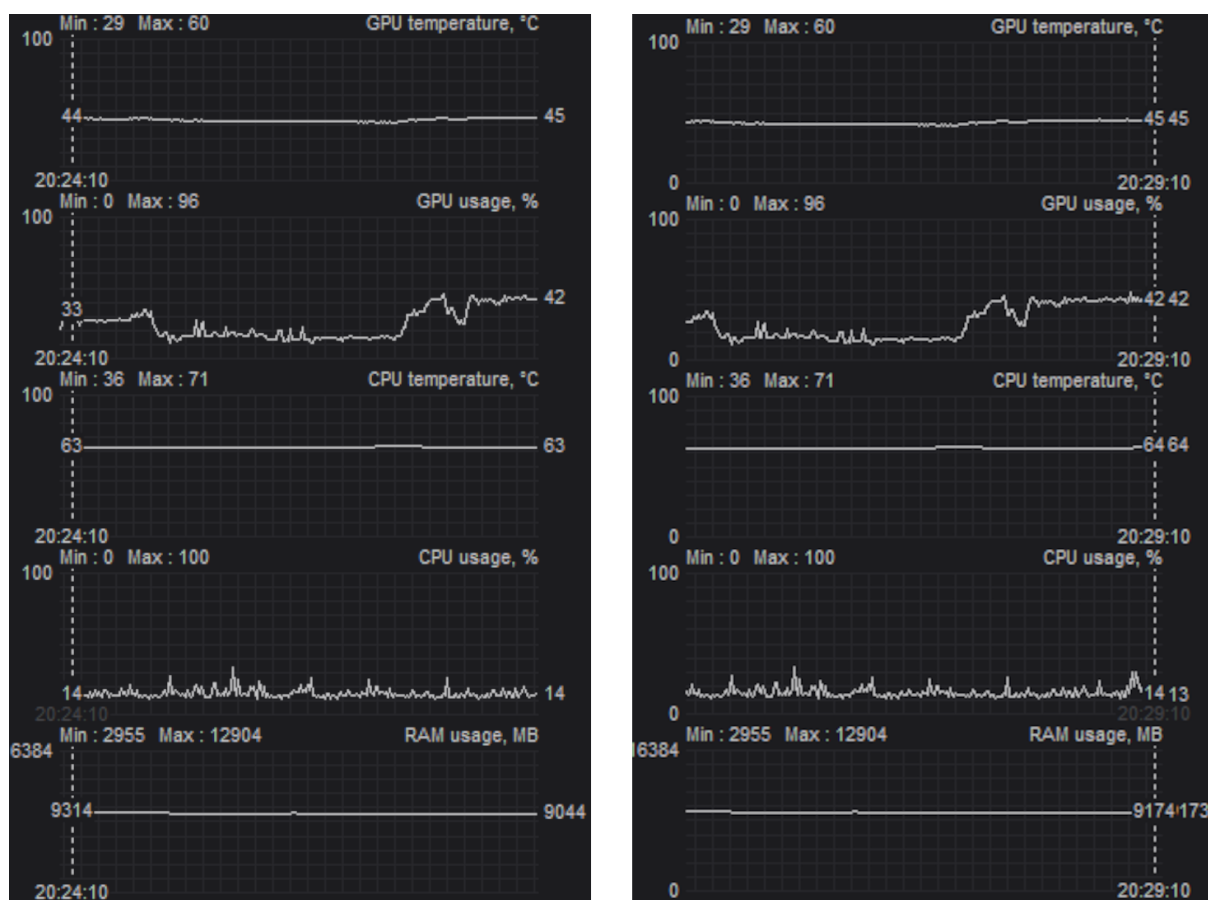


Figura 5.4: Métricas de rendimiento en Opera GX

INSPECTOR	
Version	4.2.0
FPS	33
COUNT	▼
Total meshes	9626
Active meshes	3171
Active indices	282414
Active faces	94138
Active bones	0
Active particles	0
Draw calls	1864
Total lights	2
Total vertices	723886
Total materials	11
Total textures	13
FRAME STEPS DURATION	▼
Absolute FPS	34
Meshes selection	6.93 ms
Render targets	0.00 ms
Particles	0.00 ms
Sprites	0.00 ms
Animations	0.01 ms
Physics	0.00 ms
Render	0.00 ms
Frame total	29.62 ms
Inter-frame	1.55 ms
GPU Frame time	13.01 ms
GPU Frame time (average)	13.42 ms
SYSTEM INFO	▼
Resolution	1620x1080
Hardware scaling level	1
WebGL version	2
Std derivatives	✓
Compressed textures	✓

Figura 5.5: Estadísticas de rendimiento de *BabylonJS* en Opera GX

Firefox

Como segunda alternativa se emplea **Firefox** versión **104.0.1**. Nuevamente, la figura 5.6 presenta las estadísticas de rendimiento durante la prueba. En esta ocasión, el uso de *CPU* sigue manteniéndose relativamente estable, excepto por un pico concreto, pero se encuentra aislado y no hay otros cambios repentinos de utilización, por lo que se deduce que se debe a cuestiones de aplicaciones en segundo plano o el sistema operativo. Explorando las métricas de *GPU*, esta vez la utilización de forma más drástica, pero momentánea, destacando diversos picos de mayor y menor uso, aunque manteniendo una media relativamente estable.

Basándose en las mismas estadísticas dadas por la figura 5.7, en esta ocasión el sistema parece no haber registrado correctamente el tiempo medio de fotograma, por tanto, la comparación se realiza basándose en la utilización de recursos del sistema contra el resto de alternativas.

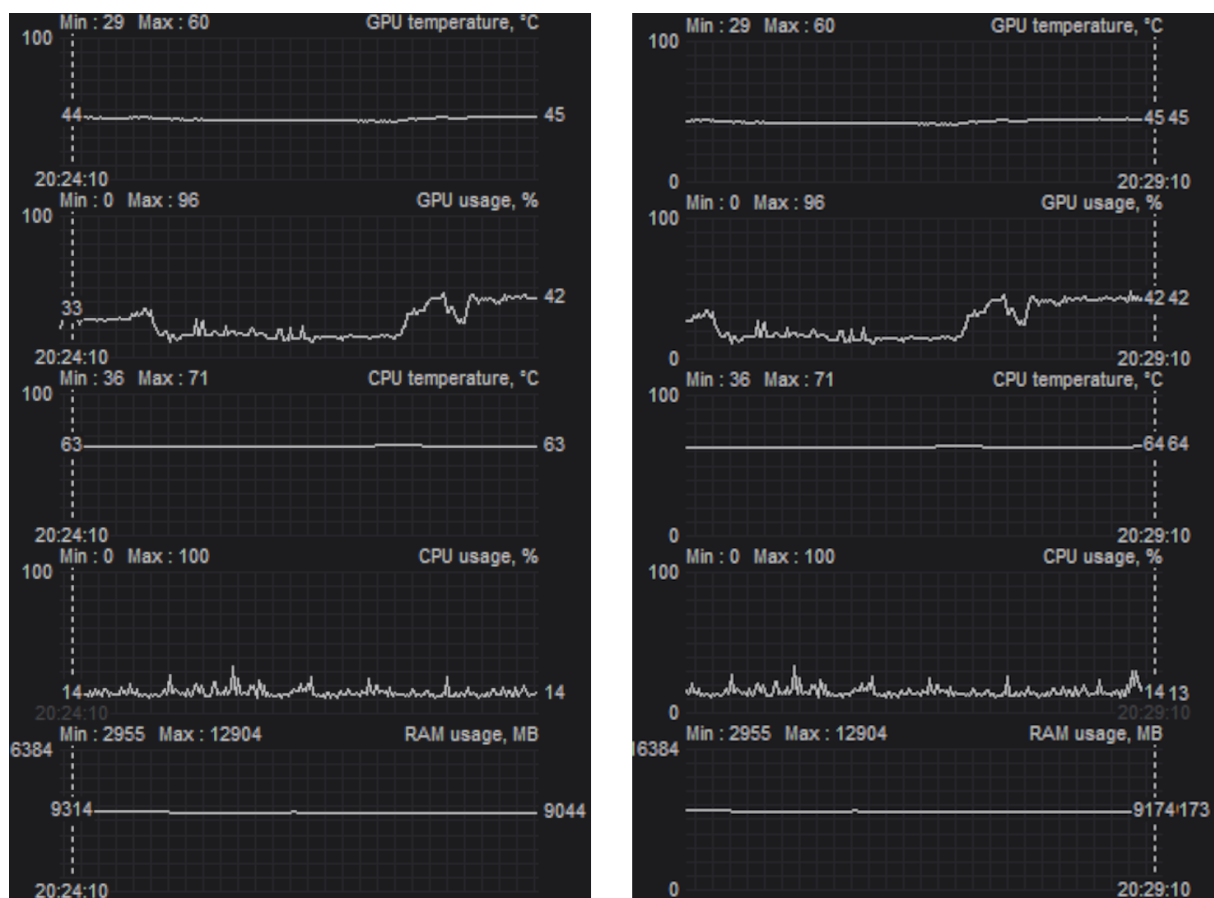
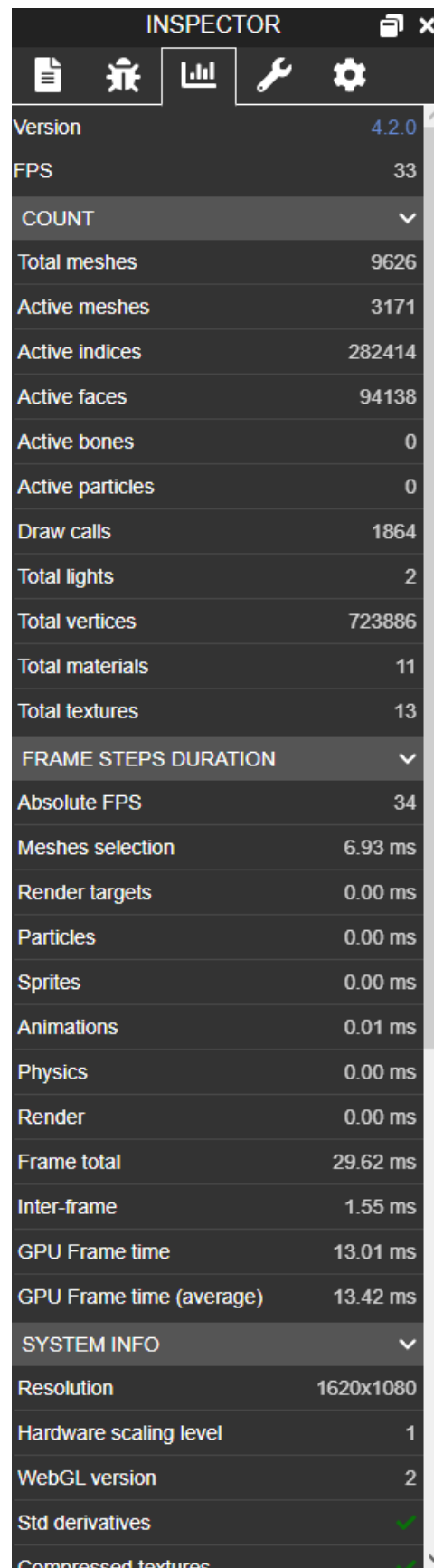


Figura 5.6: Métricas de rendimiento en Firefox



INSPECTOR	
Version	4.2.0
FPS	33
COUNT	
Total meshes	9626
Active meshes	3171
Active indices	282414
Active faces	94138
Active bones	0
Active particles	0
Draw calls	1864
Total lights	2
Total vertices	723886
Total materials	11
Total textures	13
FRAME STEPS DURATION	
Absolute FPS	34
Meshes selection	6.93 ms
Render targets	0.00 ms
Particles	0.00 ms
Sprites	0.00 ms
Animations	0.01 ms
Physics	0.00 ms
Render	0.00 ms
Frame total	29.62 ms
Inter-frame	1.55 ms
GPU Frame time	13.01 ms
GPU Frame time (average)	13.42 ms
SYSTEM INFO	
Resolution	1620x1080
Hardware scaling level	1
WebGL version	2
Std derivatives	✓
Compressed textures	✓

Figura 5.7: Estadísticas de rendimiento de *BabylonJS* en Firefox

Microsoft Edge

Por último, se accede a la aplicación desde **Microsoft Edge** en su versión **105.0**. Analizando la figura 5.8, el porcentaje de uso de *CPU* se mantiene como en las anteriores pruebas, con algún pico aislado, probablemente debido a otras aplicaciones en funcionamiento. El interés reside sobre la utilización de *GPU*, ya que, a rasgos generales, mantiene estable un porcentaje mayor de uso, con múltiples picos decrecientes debidos a los movimientos de cámara.

En esta ocasión, sí se dispone de un registro en las estadísticas de la figura 5.9, indicando una media de 12.69 ms por fotograma, un valor muy próximo al resultado obtenido en **Opera GX**, pero ligeramente superior en cuanto a “calidad” (un menor tiempo de fotograma implica, de media, mayor tasa de fotogramas).

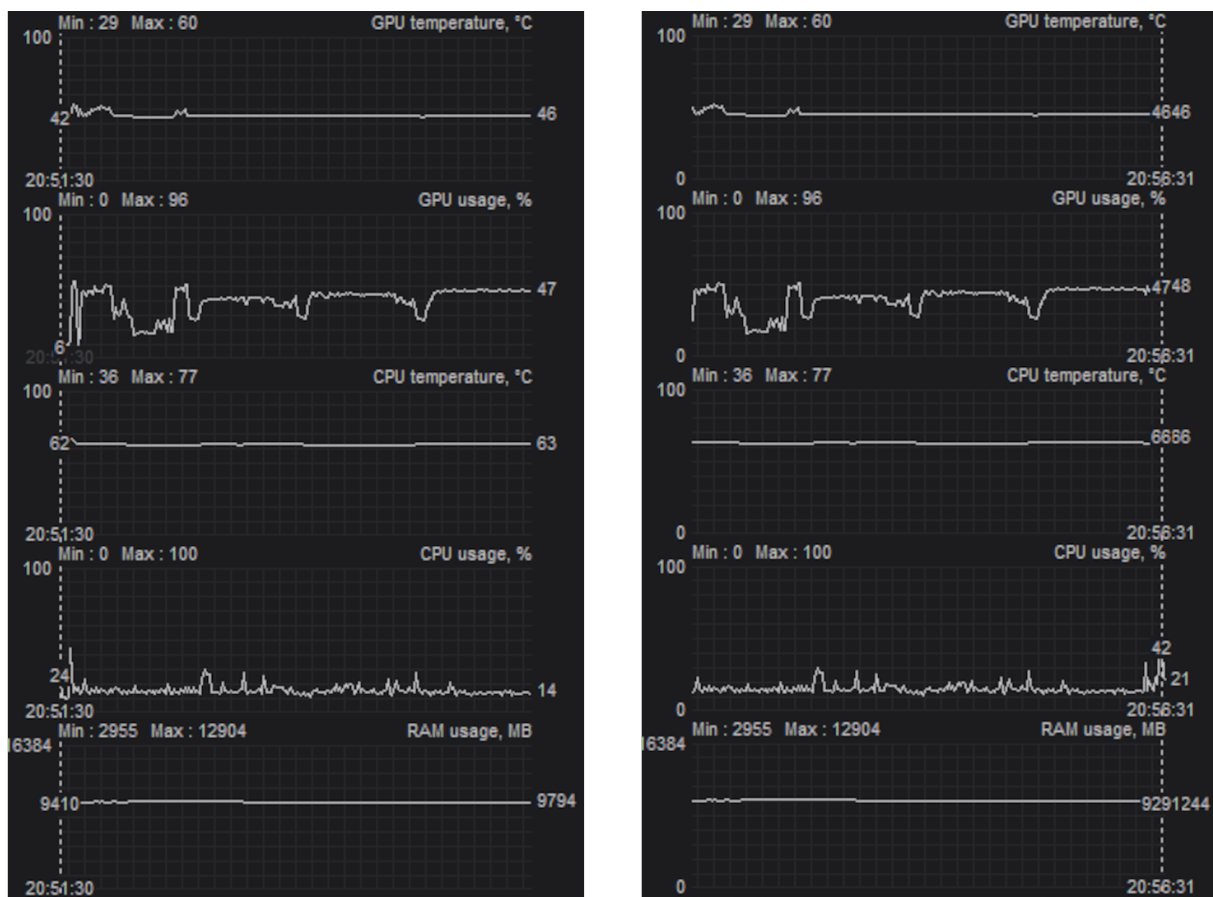
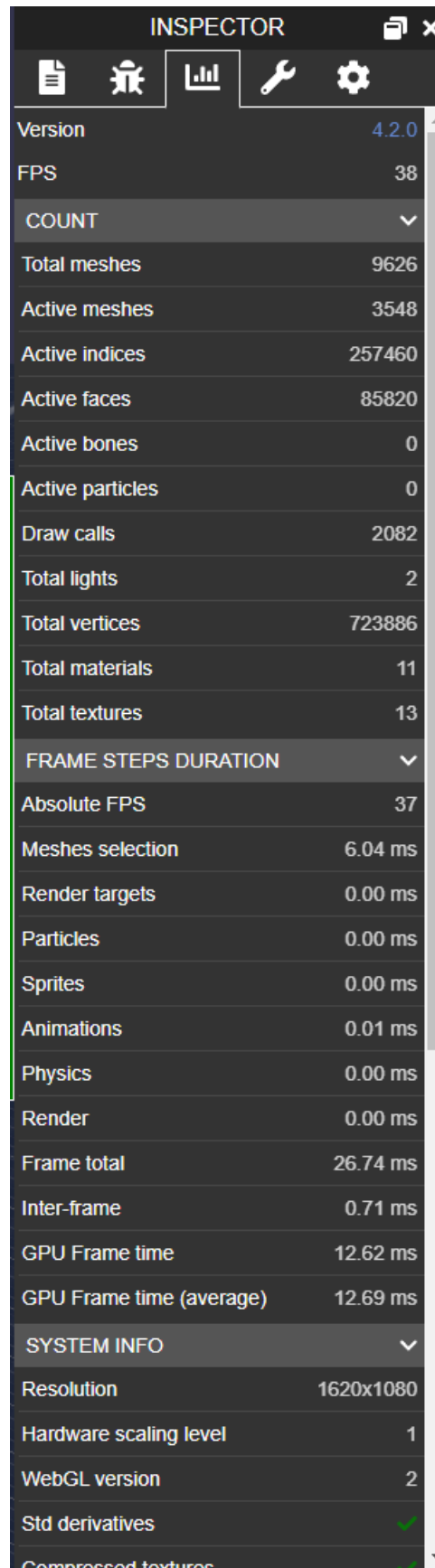


Figura 5.8: Métricas de rendimiento en Microsoft Edge



The image shows the BabylonJS Inspector tool interface. At the top, there's a title bar 'INSPECTOR' with a close button. Below it are five icons: a document, a camera, a bar chart (selected), a wrench, and a gear. The main area is a scrollable list of performance metrics. The 'COUNT' section is expanded, showing various counts. The 'FRAME STEPS DURATION' section is also expanded, showing time durations in milliseconds. The 'SYSTEM INFO' section is partially visible at the bottom.

INSPECTOR	
Version	4.2.0
FPS	38
COUNT	
Total meshes	9626
Active meshes	3548
Active indices	257460
Active faces	85820
Active bones	0
Active particles	0
Draw calls	2082
Total lights	2
Total vertices	723886
Total materials	11
Total textures	13
FRAME STEPS DURATION	
Absolute FPS	37
Meshes selection	6.04 ms
Render targets	0.00 ms
Particles	0.00 ms
Sprites	0.00 ms
Animations	0.01 ms
Physics	0.00 ms
Render	0.00 ms
Frame total	26.74 ms
Inter-frame	0.71 ms
GPU Frame time	12.62 ms
GPU Frame time (average)	12.69 ms
SYSTEM INFO	
Resolution	1620x1080
Hardware scaling level	1
WebGL version	2
Std derivatives	✓
Compressed textures	✓

Figura 5.9: Estadísticas de rendimiento de *BabylonJS* en Microsoft Edge

5.1.4. Conclusión

Comparando los resultados obtenidos del pequeño conjunto de pruebas realizado, se aprecia que no existe mucha diferencia entre navegadores distintos; si hubiese que escoger aquel con mejores estadísticas de rendimiento, probablemente sería **Microsoft Edge** al presentar una utilización más estable de los recursos a la vez que ofrece una supuesta tasa media de fotogramas mayor. En términos de estabilidad, la aplicación se ejecuta sin problemas en los navegadores seleccionados (entre los tres, abarcan los diferentes núcleos que la gran mayoría de otros navegadores también emplean, por lo que no debería afectar en mayor medida al funcionamiento), independientemente del rendimiento que ofrecen. Como información adicional, la aplicación es también accesible desde dispositivos móviles (comprobado con **Android 9**), aunque en ese caso sí hay un notable impacto en el rendimiento, imposibilitando la interacción.

Viendo estos resultados, queda demostrada la viabilidad del enfoque en cuanto a escalabilidad del sistema: en un caso real, el número de elementos visibles y capaces de interacción en la escena será cientos de veces menor que en la prueba realizada. A pesar de que a simple vista resulte de gran interés continuar el desarrollo en un ámbito cliente-servidor, hay diversos problemas que deberían considerarse con anterioridad. El factor que posiblemente sea más preocupante es la transmisión de recursos, algo totalmente dependiente de la calidad de la conexión y el ancho de banda disponible; las pruebas anteriores se realizan con clones de las instancias de un mismo panel, compartiendo geometría y textura, pero en escenas de casos reales persisten multitud de modelos e imágenes de textura de gran tamaño, pudiendo llegar a provocar largos tiempos de espera y problemas durante la carga.

Además de este problema, finalmente se opta por descartar el enfoque web debido sobre todo a la falta de experiencia en este entorno de desarrollo, requiriendo mayores tiempos de implementación y pruebas del software. La alternativa del desarrollo para escritorio resulta más interesante al asegurar cierta fluidez durante el proceso de implementación al disponer no solo de más experiencia, sino también recursos adicionales como librerías y paquetes de software específicos de ese entorno.

5.2. Incremento 1: Área de dibujado

Tras descartar la continuación del desarrollo como aplicación web, el siguiente incremento consiste en la preparación del entorno de escritorio partiendo de una base con la funcionalidad esencial para la visualización de una escena y la gestión de las estructuras y recursos necesarios para ello. Adicionalmente, el enfoque de la aplicación pasa a un nivel más genérico, habilitando que el usuario defina el posicionamiento de cada generador usando una imagen satélite como guía.

5.2.1. Especificación del incremento

El desarrollo se lleva a cabo como aplicación de escritorio codificada en **C/C++**, gestionando el proyecto en su totalidad mediante **Visual Studio 2022** y **vcpkg** como gestor de librerías. La generación del entorno tridimensional se realiza mediante **OpenGL**, con **GLEW** y **GLFW** encargados de controlar la compatibilidad con el sistema y enlazar los dispositivos de entrada/salida respectivamente. Durante el dibujado se hace uso de *shaders*, codificados con el lenguaje **GLSL**, cuyas estructuras de datos vienen recopiladas para uso general en la librería **GLM**, que se usará a lo largo del código cuando sea posible.

En esta ocasión, se sigue el diseño del sistema, clases, interacción e interfaz gráfica propuestos en el capítulo anterior, pues se mantendrán para el resto del desarrollo hasta el final del proyecto. El cambio más relevante respecto al trabajo previo en web, es la intención de generalizar la construcción de la escena, motivo por el cual se facilita, desde la propia interfaz gráfica, un área de dibujado donde el usuario pueda delimitar zonas que correspondan bien a elementos decorativos o a los propios paneles generadores. Todo esto se implementa con la librería **Dear ImGui**, encargada de controlar casi por completo la interfaz gráfica de la aplicación.

A diferencia del incremento anterior, no se desarrolla la aplicación completamente desde cero: aprovechando otra aplicación centrada en el proceso de dibujado (podría compararse a un motor gráfico extremadamente simple a bajo nivel) que solventa problemas varios en cuanto a gestión de *shaders*, materiales, texturas... y los procesos de dibujado, creación de geometría y sombras. Partiendo de esta base, mucha de su funcionalidad quedará reemplazada o excluida según sea necesario en cada incremento para los objetivos que propone. La figura 5.10 presenta el diagrama de clases del punto de partida que implementa la funcionalidad mencionada.

Visual Paradigm Standard(Juan Carlos Fernandez Perez(Universidad de Jaen))

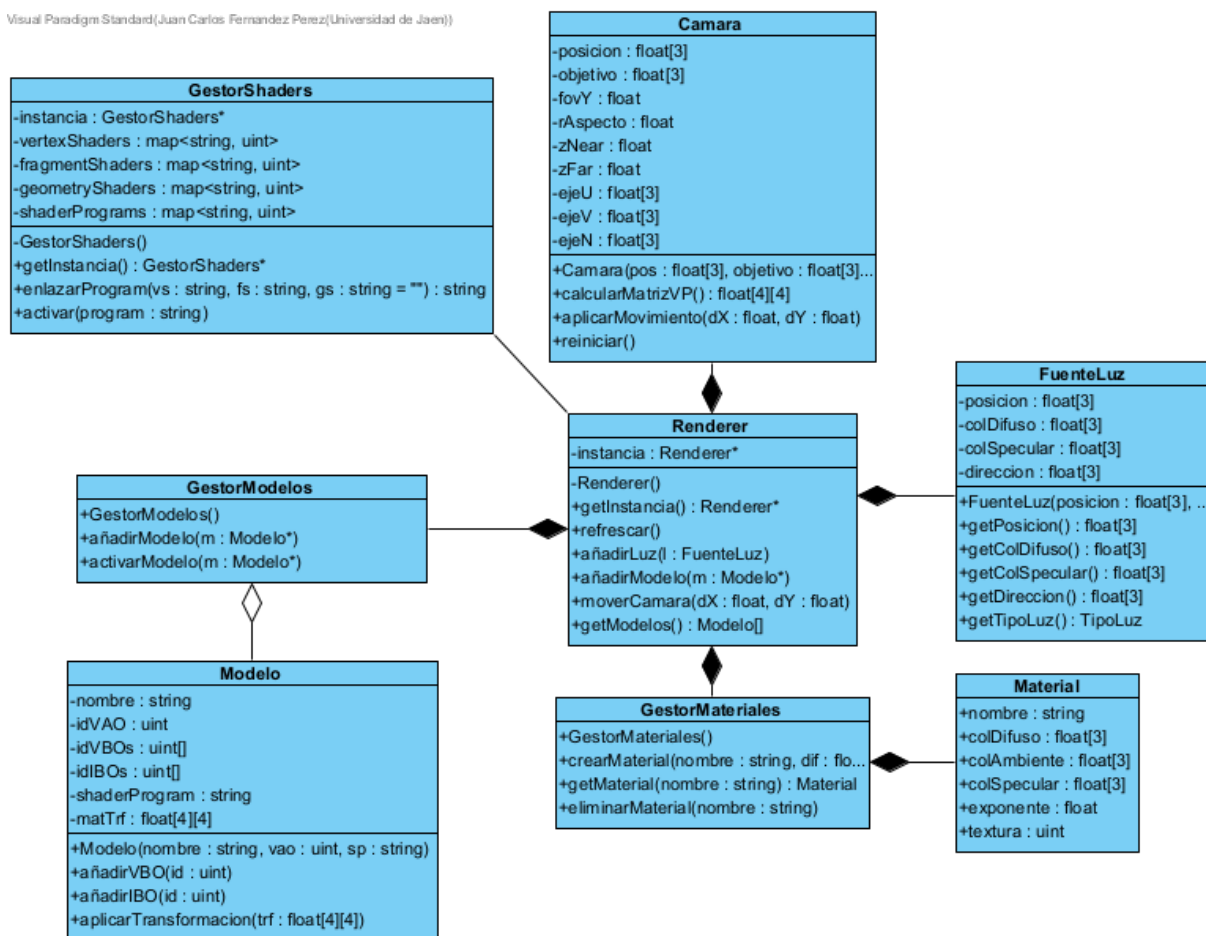


Figura 5.10: Diagrama de clases inicial del incremento 1

5.2.2. Implementación de la funcionalidad

Interfaz gráfica de usuario e interacción

La utilización de *Dear ImGui* requiere diversos cambios en el punto de partida empleado, pues este realiza el dibujado de un fotograma como respuesta a una acción del usuario, pero la librería requiere un bucle principal que refresque la imagen continuamente. Dejando a un lado ese detalle, el resto de requisitos para uso no presentan mayor complejidad, siendo únicamente métodos de inicialización y comprobaciones de compatibilidad. Una vez todo se encuentra funcionando, se genera la interfaz gráfica con el área donde el usuario delimita los diferentes elementos del escenario.

Con el fin de facilitar la usabilidad a la hora de ese posicionamiento, se preparan un conjunto de herramientas que generen figuras geométricas básicas, manteniendo también la posibilidad de representar polígonos de cualquier número de vértices. Es en este punto donde debe respetarse la geometría de una malla, ya que en *OpenGL* se forman las caras al unir los vértices en sentido **antihorario**, para así obtener el vector normal de esa superficie hacia el exterior. Independientemente de la manera en que el usuario defina cada área, la clase **FormaPoligono** define el método **FormaPoligono::getVerts()** que devuelve la colección de vértices ordenados correctamente.

Aunque dicha clase devuelva los vértices listos para la generación de geometría, internamente no almacena toda esta información; según el tipo de figura que representa, requiere de un mínimo distinto de información:

- **FormaPoligono** mantiene todos los vértices.
- **FormaRectangulo** solo requiere dos esquinas opuestas, pues el otro par se obtiene como combinación de sus coordenadas.
- **FormaTriangulo** también requiere los tres vértices que forman la cara.
- **FormaCirculo** hace uso de únicamente un primer punto como centro, y otro punto que indique el radio de la circunferencia, generando posteriormente los vértices que la forman con un simple bucle y el número de subdivisiones. Descrito en el algoritmo 1.
- **FormaModelo** requiere solo un punto, que representa la posición en el espacio del modelo 3D asociado.

Algoritmo 1: Generación de segmentos de una circunferencia**Result:** Vértices de la circunferencia $radI \leftarrow 2\pi / \text{segmentos};$ $r \leftarrow 0;$ **for** $s \leftarrow 0$ **to** segmentos **do** $nVert.x = \cos(r) * \text{radio} + \text{centro.x};$ $nVert.y = \sin(r) * \text{radio} + \text{centro.y};$ $r \leftarrow r + radI;$ **end**

Con estas herramientas, la codificación de la interacción con el área de dibujo se reduce a almacenar la posición del puntero relativa al área cada vez que se presiona sobre ella, comprobando las restricciones de cada figura en cuanto al número de vértices que permite. Al mantener la información relativa en lugar de absoluta, la conversión al espacio tridimensional queda relevada a otra operación posterior, pudiendo aplicar las transformaciones necesarias.

Generación de geometría

El siguiente paso, una vez se dispone de las diferentes áreas delimitadas por el usuario, consiste en generar la información geométrica, topológica y relativa a la visualización de esos elementos en el entorno tridimensional. Uno de los parámetros presentados en la interfaz gráfica es la altura, indicando el tamaño que tendrá la forma al realizar una extrusión vertical; de esta manera, se pueden definir superficies indicando altura cero, o volúmenes con altura independiente.

En los casos donde la altura sea mayor que cero, y, por tanto, se aplique extrusión, los índices a los vértices que se facilita a *OpenGL* para el dibujado se disponen estructurados como una tira de triángulos, “*GL_TRIANGLE_STRIP*”, en vertical. La figura 5.11 presenta las diferentes primitivas disponibles en *OpenGL* para la interpretación de los índices de vértices a la hora de dibujar triángulos. El resto de parámetros del modelo son calculables a partir de la geometría: las normales con la combinación de los vectores normales de las caras que comparte cada vértice, y las coordenadas de textura con un incremento según el número de vértices y de divisiones en altura.

Se muestra un ejemplo de la aplicación en funcionamiento en la figura 5.12, donde se han definido diversas zonas en el área de dibujo y generado la geometría correspondiente, tras asignar a cada elemento un material con color diferente para que distingan con claridad ante el color de fondo.

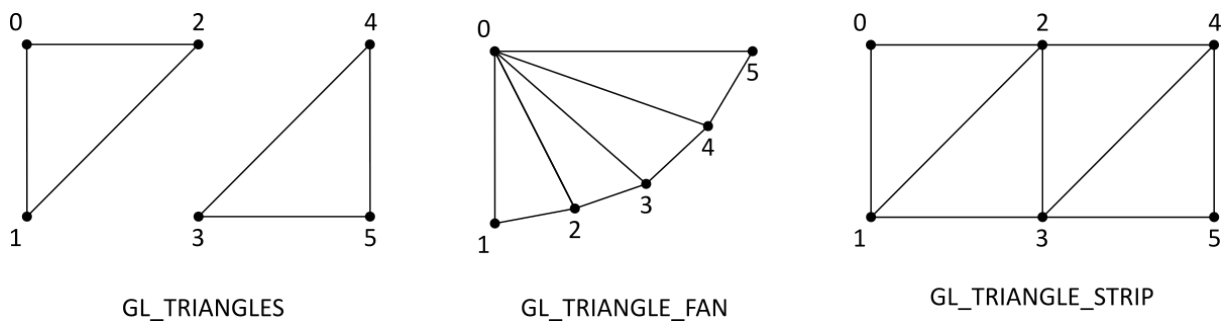


Figura 5.11: Primitivas de dibujo de triángulos en *OpenGL*

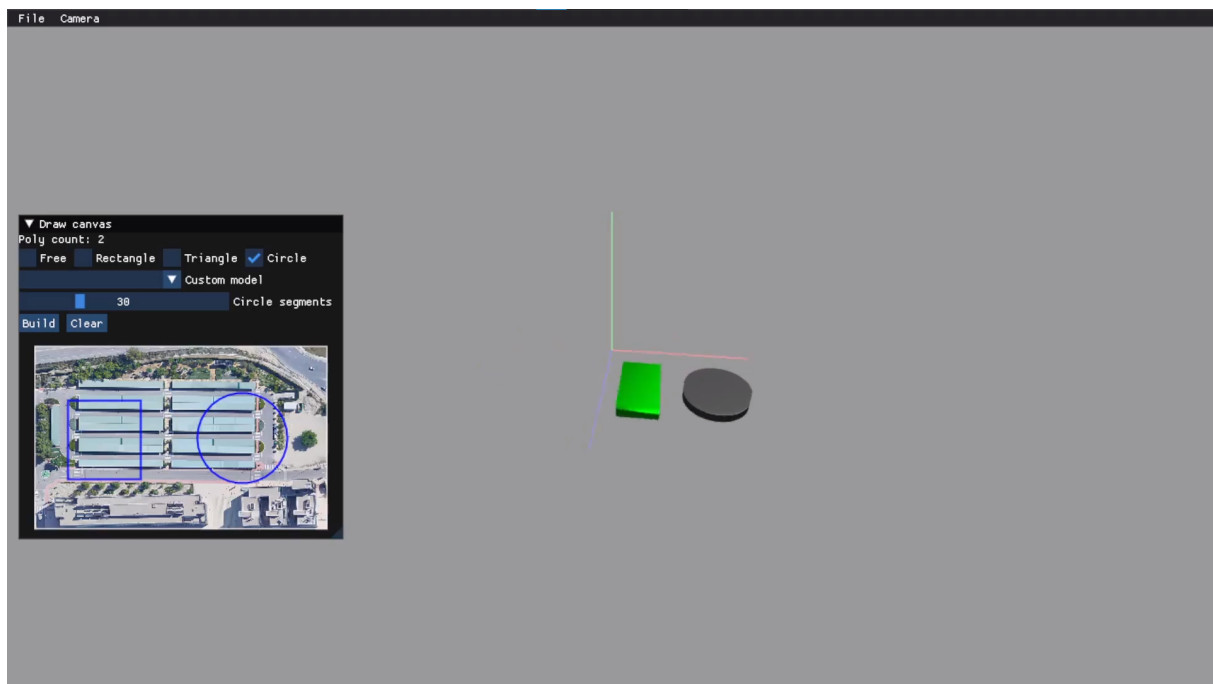


Figura 5.12: Aplicación en funcionamiento tras el incremento 1

Diagrama de clases

Una vez finalizado el incremento, la aplicación sigue el diagrama de clases presentado por la figura 5.13, habiendo añadido al punto de partida el uso de las formas para el dibujado. Por el momento, el código relativo a la interfaz de usuario se encuentra sobre todo en el fichero principal, *main.cpp*, pero se plantea la abstracción a una nueva clase en incrementos futuros.

5.2.3. Pruebas y consideraciones

Entre la funcionalidad implementada en este incremento, el punto desde el cual más problemas pueden surgir consiste en la generación de la geometría, ya que durante el proceso, cualquier error o desliz, por pequeño que sea, puede provocar una visualización incorrecta. Entre las herramientas de dibujado ofrecidas, únicamente la delimitación de **triángulos** presenta un problema apreciable en la figura 5.14.

Analizando esta situación, se puede apreciar que la iluminación sobre el modelo funciona correctamente, pero la forma que se visualiza no coincide con la visualización que se espera. Con esta información, se deduce que el problema viene dado por un orden incorrecto en la **definición de los índices** a los vértices, ya que, aunque es algo complejo de comprender visualmente, se están dibujando las caras internas del modelo, de ahí que solo mantenga el contorno correctamente. Esto se debe a que, en cualquier entorno gráfico, siempre se eliminan del proceso aquellas caras que den la espalda a la cámara para reducir al máximo posible la carga de trabajo. En *OpenGL*, la información geométrica del objeto se cede a los *shaders*; internamente realiza los cálculos necesarios para excluir caras ocultas según el tipo de primitiva, dando explicación a por qué la iluminación sí es correcta.

Finalmente, la aplicación es ejecutada en distintos equipos de configuraciones variadas con el fin de comprobar el funcionamiento y soporte en un hardware diferente. Además del equipo principal de trabajo, se ha comprobado y verificado el funcionamiento en: un equipo de escritorio con procesador principal *AMD Ryzen 5 2600* y *GPU AMD Radeon RX590*; un equipo portátil con procesador *Intel i5-8300H* y *GPU Nvidia GTX 1050*; un segundo equipo portátil que consta de un procesador *Intel Pentium 4415Y* con gráficos integrados *Intel HD 615*. Todos emplean como sistema operativo *Windows 10* de 64 bits. Por el momento no existen problemas graves en cuanto al funcionamiento y compatibilidad de la aplicación con otros sistemas, pudiendo dar comienzo el siguiente incremento.

```

classDiagram
    class GestorShaders {
        -instancia : GestorShaders*
        -vertexShaders : map<string, uint>
        -fragmentShaders : map<string, uint>
        -geometryShaders : map<string, uint>
        -shaderPrograms : map<string, uint>
        +GestorShaders()
        +getInstancia() : GestorShaders*
        +enlazarProgram(vs : string, fs : string, gs : string = "") : string
        +activar(program : string)
    }
    class GestorModelos {
        +GestorModelos()
        +añadirModelo(m : Modelo*)
        +activarModelo(m : Modelo*)
    }
    class Modelo {
        -nombre : string
        -idVAO : uint
        -idVBOS : uint[]
        -idIBOs : uint[]
        -shaderProgram : string
        -matTrf : float[4][4]
        +Modelo(nombre : string, vao : uint, sp : string)
        +añadirVBO(id : uint)
        +añadirIBO(id : uint)
        +aplicarTransformacion(trf : float[4][4])
    }
    class GestorMateriales {
        +GestorMateriales()
        +crearMaterial(nombre : string, dif : float[3], col : float[3], exp : float, tex : uint)
        +getMaterial(nombre : string) : Material
        +eliminarMaterial(nombre : string)
    }
    class Camara {
        -posicion : float[3]
        -objetivo : float[3]
        -fovY : float
        -rAspecto : float
        -zNear : float
        -zFar : float
        -ejeU : float[3]
        -ejeV : float[3]
        -ejeN : float[3]
        +Camara(pos : float[3], objetivo : float[3], fovY : float, rAspecto : float, zNear : float, zFar : float, ejeU : float[3], ejeV : float[3], ejeN : float[3])
        +calcularMatrizVP() : float[4][4]
        +aplicarMovimiento(dx : float, dy : float)
        +reiniciar()
    }
    class Renderer {
        -instancia : Renderer*
        +Renderer()
        +getInstancia() : Renderer*
        +refrescar()
        +añadirLuz(l : FuenteLuz)
        +añadirModelo(m : Modelo*)
        +moverCamara(dx : float, dy : float)
        +getModelos() : Modelo[]
    }
    class FuenteLuz {
        -posicion : float[3]
        -colDifuso : float[3]
        -colSpecular : float[3]
        -direccion : float[3]
        +FuenteLuz(posicion : float[3], colDifuso : float[3], colSpecular : float[3], direccion : float[3])
        +getPosicion() : float[3]
        +getColDifuso() : float[3]
        +getColSpecular() : float[3]
        +getDireccion() : float[3]
        +getTipoLuz() : TipoLuz
    }
    class Material {
        +nombre : string
        +colDifuso : float[3]
        +colAmbiente : float[3]
        +colSpecular : float[3]
        +exponente : float
        +textura : uint
    }
    class FormaPoligono {
        -vertices : vector<float[2]>
        -grosor : float
        -color : uint
        -material : string
        +FormaPoligono(verts : vector<float[2]>, grosor : float, color : uint, mat : string)
        +dibujarArea()
    }
    class FormaRectangulo {
        -redondEsquinas : float
        +FormaRectangulo(vertA : float[2], vertB : float[2], vertC : float[2], vertD : float[2], grosor : float, color : uint, mat : string)
    }
    class FormaTriangulo {
        +FormaTriangulo(vertA : float[2], vertB : float[2], vertC : float[2], grosor : float, color : uint, mat : string)
    }
    class FormaCirculo {
        -divisiones : uint
        +FormaCirculo(centro : float[2], vert : float[2], grosor : float, color : uint, mat : string)
    }
    class GestorFormas {
        +GestorFormas()
        +crearForma(nombre : string, forma : Forma*)
        +getForma(nombre : string) : Forma*
        +eliminarForma(nombre : string)
    }
    GestorShaders --> Renderer
    GestorModelos --> Renderer
    GestorMateriales --> Renderer
    Camara --> Renderer
    FuenteLuz --> Renderer
    Material --> GestorMateriales
    FormaPoligono --|> FormaRectangulo
    FormaPoligono --|> FormaTriangulo
    FormaPoligono --|> FormaCirculo
    GestorFormas --> GestorModelos
    GestorFormas --> GestorMateriales
    GestorFormas --> GestorShaders
    
```

The diagram illustrates the architecture of the Visual Paradigm Standard, showing the relationships between various classes and their attributes and methods.

GestorShaders

- instancia : GestorShaders*
- vertexShaders : map<string, uint>
- fragmentShaders : map<string, uint>
- geometryShaders : map<string, uint>
- shaderPrograms : map<string, uint>
- +GestorShaders()
- +getInstancia() : GestorShaders*
- +enlazarProgram(vs : string, fs : string, gs : string = "") : string
- +activar(program : string)

GestorModelos

- +GestorModelos()
- +añadirModelo(m : Modelo*)
- +activarModelo(m : Modelo*)

Modelo

- nombre : string
- idVAO : uint
- idVBOS : uint[]
- idIBOs : uint[]
- shaderProgram : string
- matTrf : float[4][4]
- +Modelo(nombre : string, vao : uint, sp : string)
- +añadirVBO(id : uint)
- +añadirIBO(id : uint)
- +aplicarTransformacion(trf : float[4][4])

GestorMateriales

- +GestorMateriales()
- +crearMaterial(nombre : string, dif : float[3], col : float[3], exp : float, tex : uint)
- +getMaterial(nombre : string) : Material
- +eliminarMaterial(nombre : string)

Camara

- posicion : float[3]
- objetivo : float[3]
- fovY : float
- rAspecto : float
- zNear : float
- zFar : float
- ejeU : float[3]
- ejeV : float[3]
- ejeN : float[3]
- +Camara(pos : float[3], objetivo : float[3], fovY : float, rAspecto : float, zNear : float, zFar : float, ejeU : float[3], ejeV : float[3], ejeN : float[3])
- +calcularMatrizVP() : float[4][4]
- +aplicarMovimiento(dx : float, dy : float)
- +reiniciar()

Renderer

- instancia : Renderer*
- +Renderer()
- +getInstancia() : Renderer*
- +refrescar()
- +añadirLuz(l : FuenteLuz)
- +añadirModelo(m : Modelo*)
- +moverCamara(dx : float, dy : float)
- +getModelos() : Modelo[]

FuenteLuz

- posicion : float[3]
- colDifuso : float[3]
- colSpecular : float[3]
- direccion : float[3]
- +FuenteLuz(posicion : float[3], colDifuso : float[3], colSpecular : float[3], direccion : float[3])
- +getPosicion() : float[3]
- +getColDifuso() : float[3]
- +getColSpecular() : float[3]
- +getDireccion() : float[3]
- +getTipoLuz() : TipoLuz

Material

- +nombre : string
- +colDifuso : float[3]
- +colAmbiente : float[3]
- +colSpecular : float[3]
- +exponente : float
- +textura : uint

FormaPoligono

- vertices : vector<float[2]>
- grosor : float
- color : uint
- material : string
- +FormaPoligono(verts : vector<float[2]>, grosor : float, color : uint, mat : string)
- +dibujarArea()

FormaRectangulo

- redondEsquinas : float
- +FormaRectangulo(vertA : float[2], vertB : float[2], vertC : float[2], vertD : float[2], grosor : float, color : uint, mat : string)

FormaTriangulo

- +FormaTriangulo(vertA : float[2], vertB : float[2], vertC : float[2], grosor : float, color : uint, mat : string)

FormaCirculo

- divisiones : uint
- +FormaCirculo(centro : float[2], vert : float[2], grosor : float, color : uint, mat : string)

GestorFormas

- +GestorFormas()
- +crearForma(nombre : string, forma : Forma*)
- +getForma(nombre : string) : Forma*
- +eliminarForma(nombre : string)

Herramienta para delimitación de áreas sobre el lienzo

This tool is used for delimiting areas on the canvas.

78

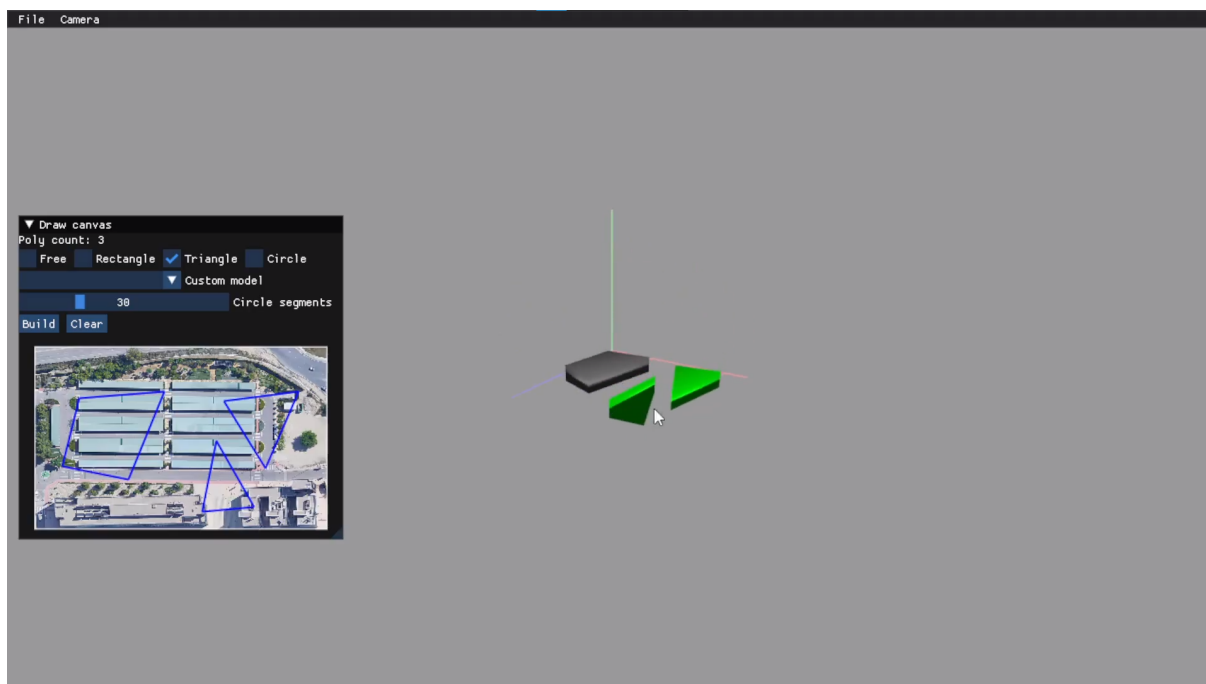


Figura 5.14: Caso de visualización incorrecta de figura triangular

5.3. Incremento 2: Visualización de modelos

Actualmente, es posible delimitar áreas siguiendo varias formas geométricas básicas o marcando polígonos libremente. El siguiente paso natural es habilitar la carga de modelos 3D desde fichero que permitan poblar la escena de elementos decorativos, enlazando también sus materiales y texturas si lo incluyen. Otro aspecto preocupante reside en la organización del código y el flujo de ejecución entre la visualización de la escena y la interfaz gráfica, siendo necesario una refactorización que facilite la inclusión de nueva funcionalidad en el futuro.

5.3.1. Especificación del incremento

El soporte para carga de modelos 3D requiere adaptar la aplicación para ello, pues el punto de partida empleado no permitía cargar modelos de múltiples mallas de triángulos exportados en un fichero siguiendo algún formato estándar. La lectura y extracción de información desde fichero se realiza con la librería **Assimp**, que facilita una estructura interna poblada con los datos del fichero leído: geometría, normales, rutas a las imágenes y coordenadas de textura, animaciones... Del mismo modo, **LodePNG** se encarga de la lectura de imágenes, siendo una librería de cabecera extremadamente ligera, facilitando su instalación y uso en casi cualquier aplicación.

5.3.2. Implementación de la funcionalidad

Modelos 3D compuestos

Aunque la aplicación no soporte la carga de modelos formados por múltiples mallas, texturas y materiales, las estructuras de datos donde se almacenan pueden mantenerse sin modificación. Internamente, los modelos reciben dos estructuras adicionales con las cuales contener una colección de mallas, **Modelo::MeshIndex**, y la información del *Vertex Buffer Object*, o *VBO*, que facilitar a *OpenGL* para su dibujado.

En la carga de un modelo, se recorren las estructuras jerárquicas de *Assimp*, recopilando vértices, índices, normales y coordenadas de textura para cada una de las mallas que forman el objeto cargado, almacenando cada conjunto en el búfer respectivo de *OpenGL* para los modelos. Posteriormente, durante el bucle principal de dibujado, se recopila cada modelo y se itera sobre las mallas que contiene, activando para cada una sus estructuras de datos, activando el material correspondiente y pasando los datos a sus *shaders* asociados, y mandando por último la llamada a dibujar a *OpenGL*. Al ser modelos importados, la primitiva básica siempre será *GL_TRIANGLES*.

Reestructuración de la interfaz gráfica

Otro cambio que incluye este incremento es una remodelación del diseño de clases respecto al anterior. Hasta ahora, la codificación de la interfaz gráfica de usuario se realizaba directamente en el fichero de entrada a la aplicación, añadiendo una complejidad excesiva para mantener el estado entre fotogramas y establecer comunicación con el resto de clases del sistema. Por este motivo, se genera la clase **UIRenderer**, encargada de abstraer todo aquello relacionado con la interfaz gráfica: atributos con el estado y contenido de cada elemento (ventanas, botones, selectores, cuadros de texto...) y métodos que simplifican el código del bucle principal, dedicados a cada parte de la interfaz (ventana principal, barra de tareas, ventanas emergentes...).

Aprovechando esta remodelación, también se ajusta la interfaz al diseño presentado en el capítulo anterior, fijando la ventana principal con el área de dibujo a un lado de la pantalla. La abstracción realizada en este incremento permitirá, en un futuro, añadir soporte para internacionalización y localización de la aplicación de una manera más efectiva, evitando en mayor medida texto en bruto directamente sobre el código. La figura 5.15 presenta la aplicación tras cargar diversos modelos de árboles y arbustos, además del cambio en el posicionamiento de la interfaz gráfica.

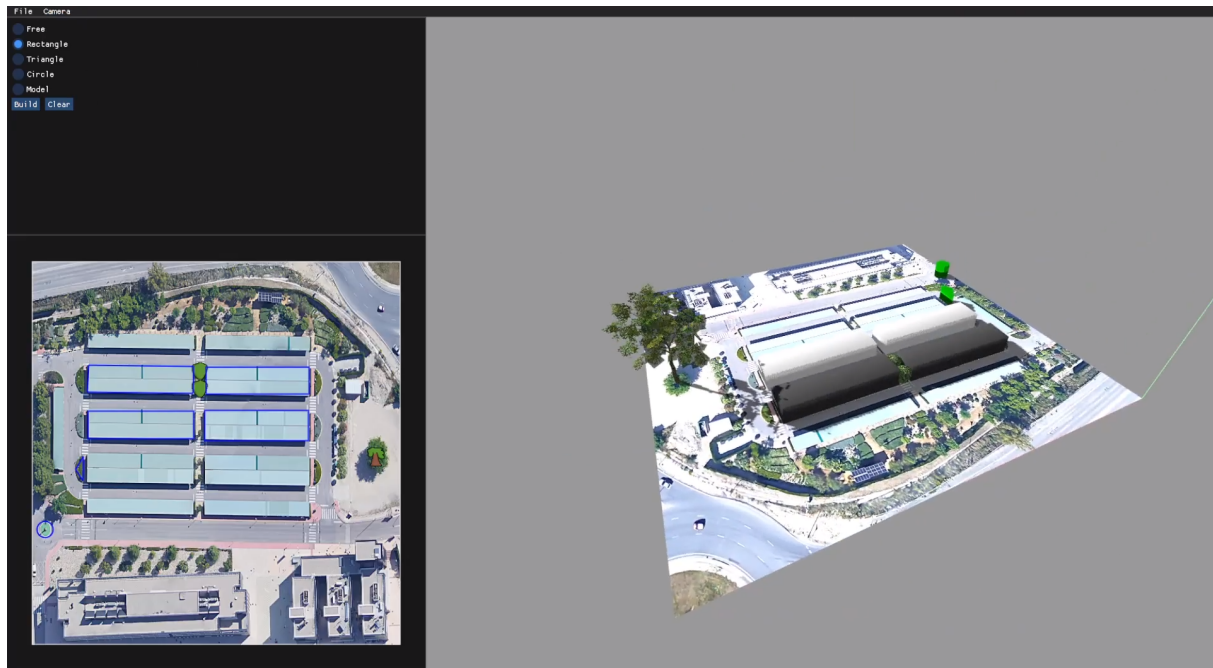


Figura 5.15: Aplicación en funcionamiento tras el incremento 2

Diagrama de clases

El diagrama de clases respectivo a la finalización de este incremento puede consultarse en la figura 5.16 (se omiten las estructuras simples contenidas en otras clases y métodos de menor relevancia como *getters* y *setters*).

5.3.3. Pruebas y consideraciones

Este incremento no dispone de pruebas específicas de compatibilidad o rendimiento en multitud de sistemas con características variadas, debido al resultado positivo del incremento anterior y la escasez de múltiples cambios que requieran una comprobación intensiva. La aplicación en este punto continúa funcionando correctamente en el equipo principal de desarrollo, y permite la inclusión de modelos que cuenten con múltiples mallas de triángulos y texturas, siendo razón suficiente para dar paso al siguiente incremento del proyecto.

Visual Paradigm Standard(Juan Carlos Fernandez Perez(Universidad de Jaen))

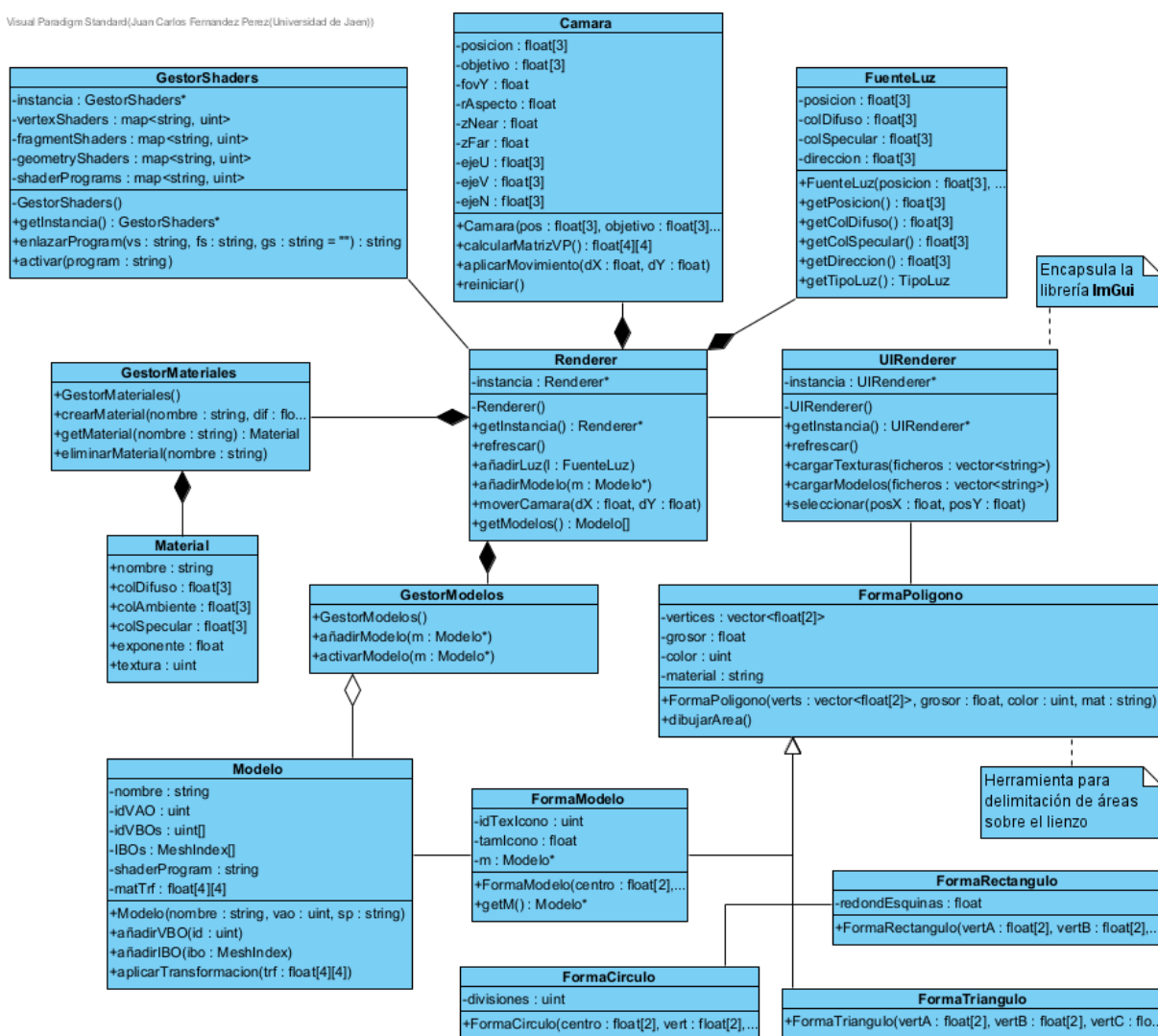


Figura 5.16: Diagrama de clases final del incremento 2

5.4. Incremento 3: Jerarquía de información

Tras establecer una base encargada de la visualización de la escena y modelos importados desde fichero, que además gestione la interacción con la interfaz gráfica y los periféricos de entrada, a continuación es turno de añadir la creación de paneles generadores que puedan mantener la información que, en un futuro, se recolecte de la plantación solar real.

5.4.1. Especificación del incremento

A diferencia del incremento 0, donde se aplicaba una subdivisión de los paneles en módulos y sub generadores, al buscar una generalización de la información, esta vez la jerarquía contará con tres niveles, siendo el **panel generador** el más bajo (y específico). Por encima del panel se encuentran los **inversores**, que convierten la corriente recibida de los paneles a alterna en el sistema real; en la aplicación, serán los encargados de agrupar paneles y dar una visión general del conjunto. Como nivel superior, las **regiones** representan instalaciones en su totalidad, formando un cúmulo de inversores, junto a la información geográfica de la misma, aunque por el momento se limita a una imagen satélite que también se usará como guía para el área de dibujo (en próximas iteraciones se hará uso de coordenadas *UTM*, *Universal Transverse Mercator*, para el posicionamiento de nubes de puntos).

También se gestionará en este incremento la respuesta a la selección de un panel mediante el clic principal del puntero. Si en la posición del puntero en ese instante se encuentra un panel generado previamente, aparecerá una ventana emergente con la información que contiene ese panel. Esta selección se implementa con la técnica de selección por color o *color picking*, que se detallará en el próximo apartado.

5.4.2. Implementación de la funcionalidad

Estructuración de los datos

Partiendo de la clase **GestorModelos**, se genera otra clase **InverterIndexer** donde se controlará todo el acceso a las estructuras de datos de la jerarquía de información. Junto a ella, cada nivel queda representado con las clases **InverterRegion**, **Inverter** y **FVPanel**, mencionadas de mayor a menor generalización. Las colecciones empleadas

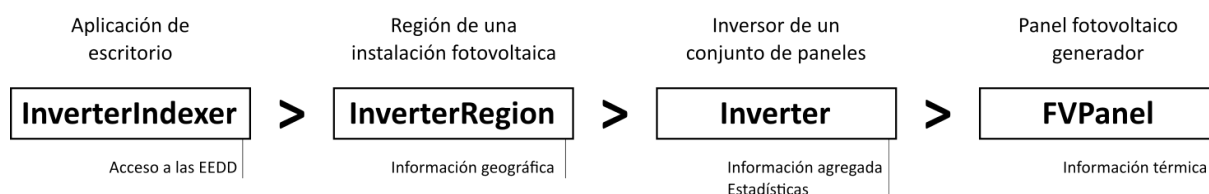


Figura 5.17: Esquema de la jerarquía de información

para mantener los conjuntos de objetos del nivel inferior en cada una son estructuras de tipo **mapa**, garantizando acceder rápidamente a cualquier elemento del conjunto y operar con él. En el nivel inferior, cada panel también dispone de un puntero al modelo 3D que lo representa en el espacio, permitiendo su identificación cuando sea seleccionado. El esquema básico de la jerarquía de la información se presenta en la figura 5.17, indicando, a rasgos generales, qué representa cada uno de los niveles y la función que ejercen en el sistema almacenando los datos.

Durante el uso de la aplicación, se habilita la libre creación y borrado de los diferentes elementos existentes en cada nivel, según la selección activa (si se trabaja sobre, por ejemplo, la región X, todas las modificaciones se aplicarán sobre su contenido, por lo cual, al cambiar a una región Y, no contiene relación alguna sobre la anterior). Cada vez que se modifique la estructura de datos jerárquica, se muestra una ventana emergente solicitando la confirmación del usuario para aplicar esta acción, aportando mayor valor en términos de usabilidad a la aplicación. Si se presenta alguna incongruencia durante la inclusión de un nuevo elemento (por ejemplo, identificadores repetidos), se presentará un mensaje de error con un breve indicativo del problema, imposibilitando la confirmación hasta que sea corregido.

Selección por color

Siguiendo lo mencionado anteriormente, a continuación se detalla el funcionamiento del algoritmo de selección por color empleado. Esta técnica saca provecho de la existencia de múltiples *pipelines* de renderizado en *OpenGL*. Una *pipeline* podría definirse como un ciclo encargado de dibujar en un *buffer*, que posteriormente podrá emplearse según convenga (el ejemplo trivial es el *pipeline* por defecto, cuyo resultado se muestra por pantalla). Dicho *buffer* no es más que una matriz de datos (en esta ocasión de color, aunque hay otros tipos empleados para otras técnicas) que puede leerse en cualquier momento y obtener el valor de una posición dada.

Reservando un *pipeline* secundario, que no se muestre en pantalla, para visualizar todos los elementos con los que pueda interactuarse de la escena con un color úni-

co para cada uno de ellos, la selección se reduce a extraer el color del *buffer* en la posición del puntero y comparar si alguno de los elementos coincide. La figura 5.18 muestra el resultado de seleccionar un panel en el entorno tridimensional con el puntero, apareciendo una ventana emergente con su información, además de presentar la propia escena con diversos elementos decorativos y los paneles almacenados jerárquicamente (uno de los datos que muestra la selección es el inversor asociado).

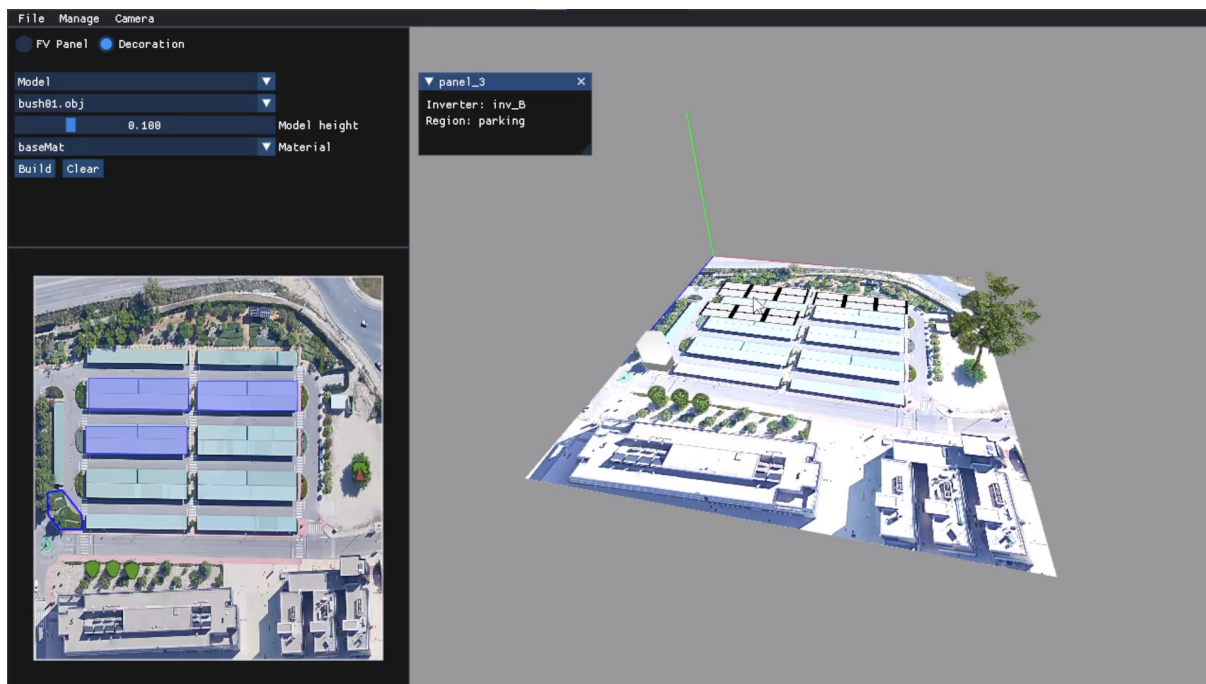


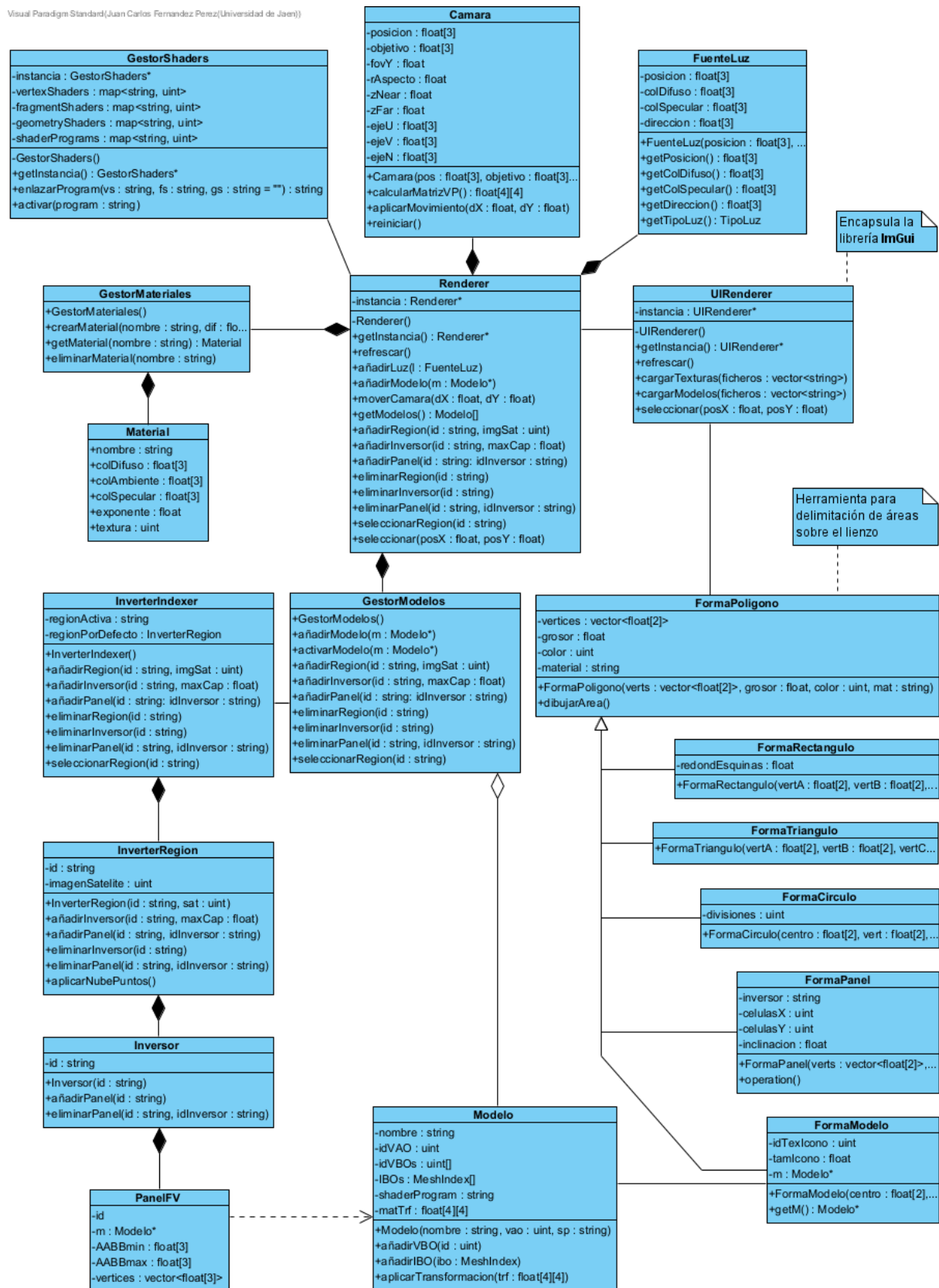
Figura 5.18: Aplicación en funcionamiento tras el incremento 3

Diagrama de clases

Las nuevas clases implementadas y sus relaciones con el resto del sistema quedan representadas en el diagrama de clases de la figura 5.19, donde se omiten estructuras simples contenidas en otras clases y métodos poco relevantes con la finalidad de mejorar la legibilidad del diagrama.

5.4.3. Pruebas y consideraciones

Aunque la aplicación funcione correctamente, tal como muestra la figura 5.18, la técnica de la selección por color presenta dos aspectos negativos que pueden comprometer la robustez del sistema en casos extremos. En primer lugar, existe un límite en la cantidad de elementos que pueden aparecer simultáneamente durante la selección, ya que no disponemos de infinitos colores representables por la máquina. A



pesar de esto, es realmente un problema más bien leve, pues aunque no disponemos de una cantidad ilimitada de colores, siguiendo el esquema *RGB* con valores enteros en el rango $[0, 255]$, el total disponible será el producto $256 * 256 * 256$, gracias a que cada canal de color abarca todo el rango, resultando en más de 16 millones de colores.

Por otro lado, durante la explicación de la técnica se indica el uso de *pipelines* de renderizado para dibujar los elementos seleccionables de la escena. En la mayoría de casos este proceso es inapreciable, pero realmente implica volver a dibujar parte del escenario, y dependiendo de la cantidad de modelos capaces de interacción, la operación de selección puede retardar el refresco de la imagen, resultando en una experiencia de uso negativa. Aun así, la intención de esta aplicación no alcanza estos casos de uso extremos, siendo la implementación de selección por color más que suficiente para los objetivos propuestos.

Respecto a las pruebas de funcionamiento, se analiza la aplicación durante un uso normal, creando una región con su información geográfica correspondiente, y poblando la escena de elementos decorativos, paneles generadores e inversores. En este proceso no han surgido problemas destacables, pero sí existe un detalle a tener en cuenta que debe solucionarse en la siguiente iteración: el área de dibujo no mantiene las proporciones de la imagen satélite, dando lugar a deformaciones respecto al entorno real. Ya que próximamente se hará uso de nubes de puntos escaneadas de la instalación física, esa deformación no permitirá ajustar correctamente la nube sobre la escena generada, de ahí que deba ser solventado cuanto antes.

5.5. Incremento 4: Carga de nubes de puntos

El último incremento implementado dentro de la extensión del proyecto se basa en añadir soporte para nubes de puntos térmicas a la aplicación, pudiendo importar y posicionar una nube en la escena, habilitando las transformaciones geométricas básicas de forma interactiva. La intención de este posicionamiento es el facilitar la carga de información sobre cada uno de los paneles escaneados mediante una proyección de la nube sobre la escena; cada panel, al ser un rectángulo, agrupa en su interior un determinado subconjunto de puntos, que lo dotarán de la información recopilada.

Ese posicionamiento mencionado puede, en muchas ocasiones, ser complicado de ajustar manualmente, por tanto, la generación de regiones se acompaña de la delimitación del área que abarca mediante coordenadas *UTM*. Los escaneos realizados con los que obtener información térmica parten también de un sistema de coordenadas

local relativo a una posición del mundo real, pudiendo unificar en la aplicación los dos espacios de coordenadas y encuadrar con precisión los diversos elementos.

5.5.1. Especificación del incremento

El soporte para nubes de puntos se obtiene enlazando a la aplicación la librería **PCL**, *Point Cloud Library*, encargada de diversos aspectos del procesamiento con nubes de puntos y con una inmensa cantidad de utilidades y funciones. Entre esas utilidades, dispone de herramientas para la lectura/escritura de ficheros con formatos típicos empleados con nubes de puntos, además de estructuras de datos espaciales destinadas específicamente a optimizar el acceso a esa información.

Esto último es especialmente importante en el contexto de este incremento, ya que evaluar la totalidad de la nube contra todos los paneles definidos compromete la usabilidad del sistema: la búsqueda lineal es una operación extremadamente lenta y provoca largos tiempos de espera en un entorno que debe ejecutarse a tiempo real.

5.5.2. Implementación de la funcionalidad

Mejoras y ajustes de interfaz gráfica

Durante la etapa de pruebas del incremento anterior, se detectó un error con la interfaz gráfica donde el área de dibujado deforma la imagen original, causando descuadres respecto al sistema real. Esto se soluciona considerando la razón de aspecto de la imagen satélite facilitada cuando se importa en el sistema, ajustando el área de dibujo a la región disponible de la interfaz gráfica respetando esa misma razón de aspecto. Junto a este cambio, también se ajusta el entorno tridimensional para que el plano sobre el cual se pinta la textura de la imagen satélite quede con la proporción de anchura y altura, además de centrar el origen de coordenadas con el punto central de ese plano.

De manera adicional, se mejora la usabilidad del sistema, añadiendo facilidades en la interacción con el área de dibujado: primero, se incluye la posibilidad de aumentar y disminuir el *zoom* sobre la imagen, mejorando la precisión a la hora de posicionar los elementos; además, el proceso de delimitación de los paneles se modifica hasta permitir definir rectángulos de orientación libre, haciendo uso del vector perpendicular tal y como muestra la figura [5.20](#).

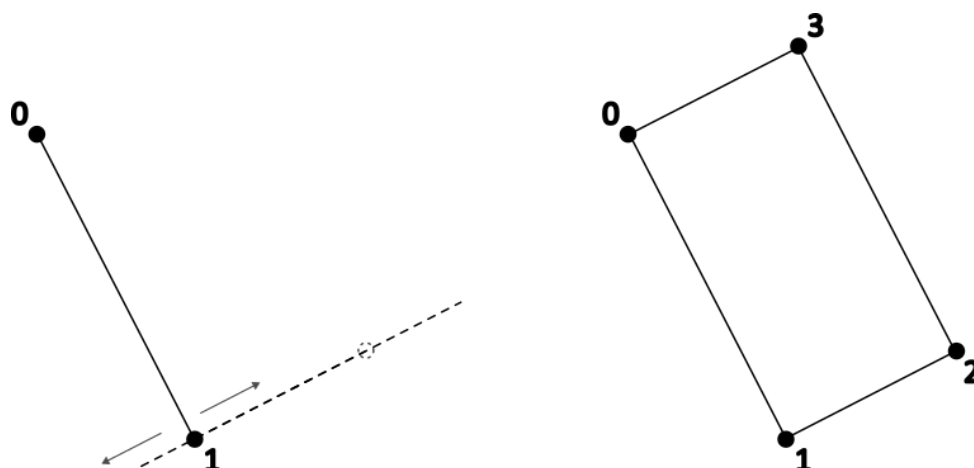


Figura 5.20: Ejemplo de generación de rectángulos mediante el vector perpendicular

Lectura y visualización de ficheros *PLY*

Como ya se ha mencionado anteriormente, la lectura de la nube de puntos queda relevada a *PCL*, que agrupa en una de sus estructuras de datos el contenido del fichero, si no encuentra ningún error durante el proceso. En este caso, cada punto estará formado por sus coordenadas en el espacio, *XYZ*, y el color registrado por el escáner, *RGB*. La colección devuelta por *PCL* que contiene esta información se recorre iterativamente, recopilando el contenido en otras estructuras (una para posición y otra para color) donde serán enviados a *OpenGL* en conjunto, provocando que la nube en su totalidad sea dibujada con una única llamada, empleando la primitiva *GL_POINTS*.

En este punto debe considerarse la definición del espacio de coordenadas utilizado, ya que en muchas ocasiones los ejes *Y* de altura y *Z* de profundidad se intercambian. Por esto, se añade a la interfaz gráfica la posibilidad de indicar el eje de altura al cargar una nube de puntos, realizando la conversión en caso de usar *Z* como eje de altura. Dicha conversión consiste en simplemente intercambiar los ejes *Y* y *Z*, invirtiendo el valor de profundidad: $(x, y, z) \rightarrow (x, z, -y)$.

Posicionamiento automático por coordenadas *UTM*

Durante la creación de una región, se pueden incluir, además de la imagen satélite correspondiente, las coordenadas *UTM* de esa imagen, que delimitan la propia región. Cuando se indican estas coordenadas (siendo únicamente necesarias dos esquinas, pues se consideran solo regiones rectangulares), la aplicación genera su sistema de coordenadas local centrado en esa imagen que, además, mantiene la razón de aspecto para evitar deformaciones, como ocurría en anteriores incrementos.

Igualmente, a la hora de importar una nube de puntos, se puede indicar, junto a la ruta del fichero, si dispone de un sistema local de coordenadas, dando los valores de compensación u *offset* sobre las que se basa ese sistema. De cualquier forma, el resultado tras cargar la nube queda posicionado en la aplicación, y puede transformarse libremente. En caso de indicar ese *offset*, la nube debería quedar posicionada con gran precisión, correspondiendo con la imagen satélite, siempre que no exista algún problema entre los sistemas de coordenadas. La figura 5.22 presenta la aplicación en funcionamiento tras generar una región y cargar una nube de puntos, dando las coordenadas *UTM* y el *offset* relativo de ambas.

Carga de datos desde la nube de puntos

Para terminar, el último punto por implementar se basa en aplicar la información térmica de la nube de puntos a los paneles construidos en la escena, una vez quede correctamente posicionada. Esa operación consiste en proyectar los puntos desde tres dimensiones a un plano en dos dimensiones, siendo el plano objetivo el que forma el “suelo”, es decir, el plano *XZ* (considerando *Z* como eje de profundidad). Una vez proyectados, da comienzo la evaluación de qué puntos pertenecen a qué paneles, momento en que se da paso a dos técnicas de algoritmos geométricos con las cuales maximizar la eficiencia del proceso.

En primer lugar, los paneles rara vez estarán alineados paralelamente al área evaluada (ya que pueden definirse libremente sin restricciones en la dirección del rectángulo). Con el objetivo de evitar comparar todos y cada uno de los puntos, se recurre a una estructura de datos espacial, facilitada por *PCL*, donde se almacena y ordena toda la nube espacialmente: un **kd-tree**, más concretamente un *2d-tree* al trabajar en dos dimensiones. Esta estructura se basa en un árbol, dejando los puntos separados espacialmente en regiones generadas por un eje central. Los nodos intermedios y la raíz del árbol representan esos ejes divisorios, intercalando en cada nivel cada dimensión, dejando en las hojas los puntos del conjunto. A un lado de cada nodo quedarán los elementos cuya coordenada sea menor sobre la dimensión de ese nivel, dejando al otro los que sean mayores (la igualdad puede quedar en cualquiera de los dos lados, el árbol resultante siempre quedará balanceado). La figura 5.21 muestra un ejemplo de *kd-tree* sobre un conjunto de puntos, considerando valores iguales al lado izquierdo.

Gracias a que esta estructura de datos se especializa en búsquedas por rango, se toma provecho de esa propiedad buscando los puntos que queden dentro no del propio panel, sino de la *Axis Aligned Bounding Box*, o **AABB** para abreviar, de cada uno, ya que, como su nombre indica, está alineada con los ejes globales y facilita la

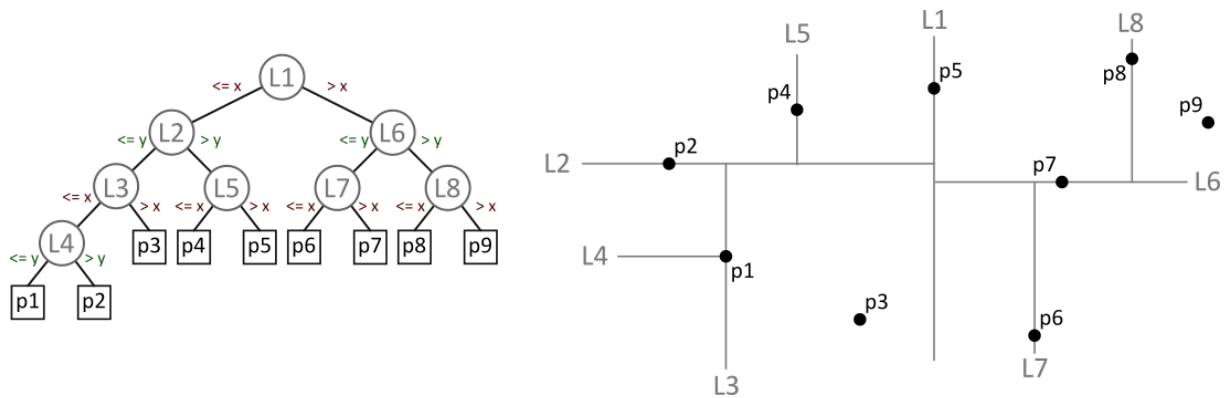


Figura 5.21: Ejemplo de la estructura de datos espacial *kd-tree* en dos dimensiones

operación de búsqueda. Con este nuevo subconjunto mucho más reducido de puntos, es posible realizar una búsqueda lineal comparando uno a uno si se encuentran en el rectángulo del panel sin comprometer en exceso el tiempo de ejecución. La evaluación de un punto dentro de un polígono consiste en comparar cada arista frente al punto en cuestión, calculando si se encuentra al lado izquierdo o derecho (este cálculo se basa en el área del triángulo que forman, descrito en el algoritmo 2). Si todas las aristas, tras evaluadas con el punto, dejan al mismo a su izquierda (considerando ser colineal también a la izquierda), estará contenido, pero si alguna devuelve otro resultado, entonces quedará fuera del polígono.

Con todo esto, cada punto que pase la evaluación de estar contenido añade su información al panel en cuestión, que agregará todos los datos recopilados, dando un promedio de los valores con la media aritmética del conjunto. Una muestra del funcionamiento viene dada en la figura 5.23, donde, tras seleccionar un panel, se muestra el color medio y el conteo de los puntos contenidos.

Algoritmo 2: Clasificación de un punto respecto a un segmento

Result: Clasificación del punto evaluado

$vA \leftarrow \text{arista.fin} - \text{arista.inicio};$

$vB \leftarrow \text{puntoTest} - \text{arista.inicio};$

$\text{area2} \leftarrow vA.x * vB.y - vA.y * vB.x;$

if $\text{area2} > 0$ **then**

return *ClasificacionPunto::IZQ*;

end

if $\text{area2} < 0$ **then**

return *ClasificacionPunto::DER*;

end

return *ClasificacionPunto::COLINEAL*;

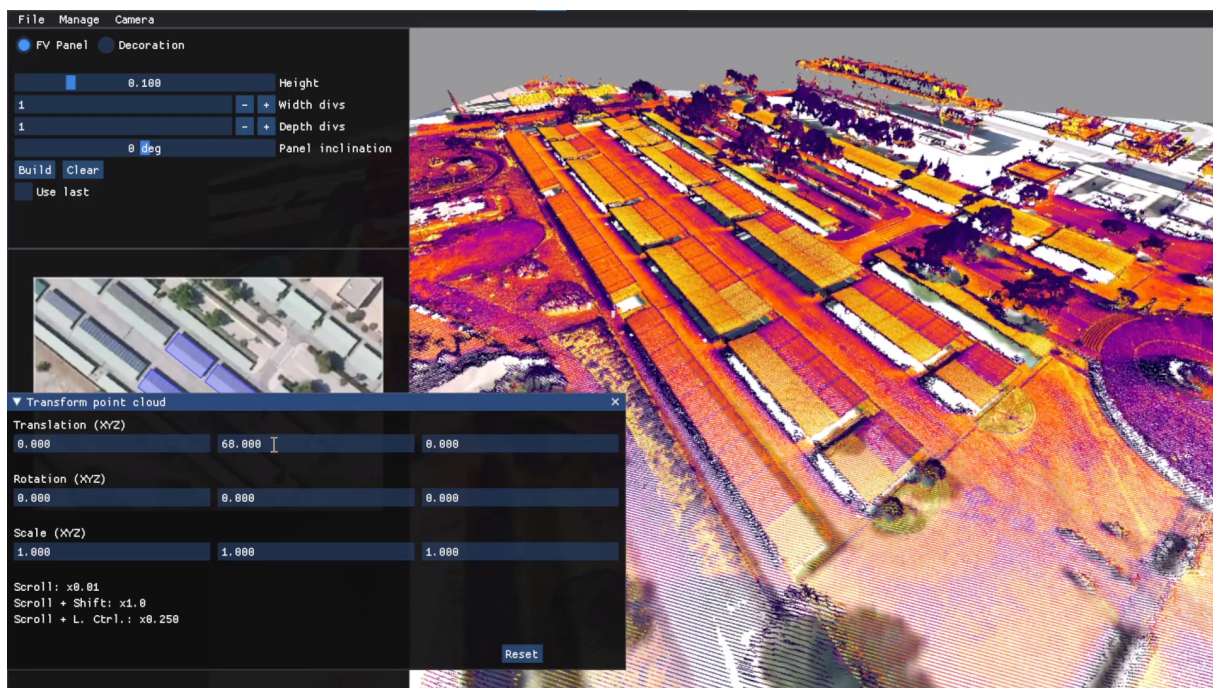


Figura 5.22: Visualización de nube de puntos tras el incremento 4

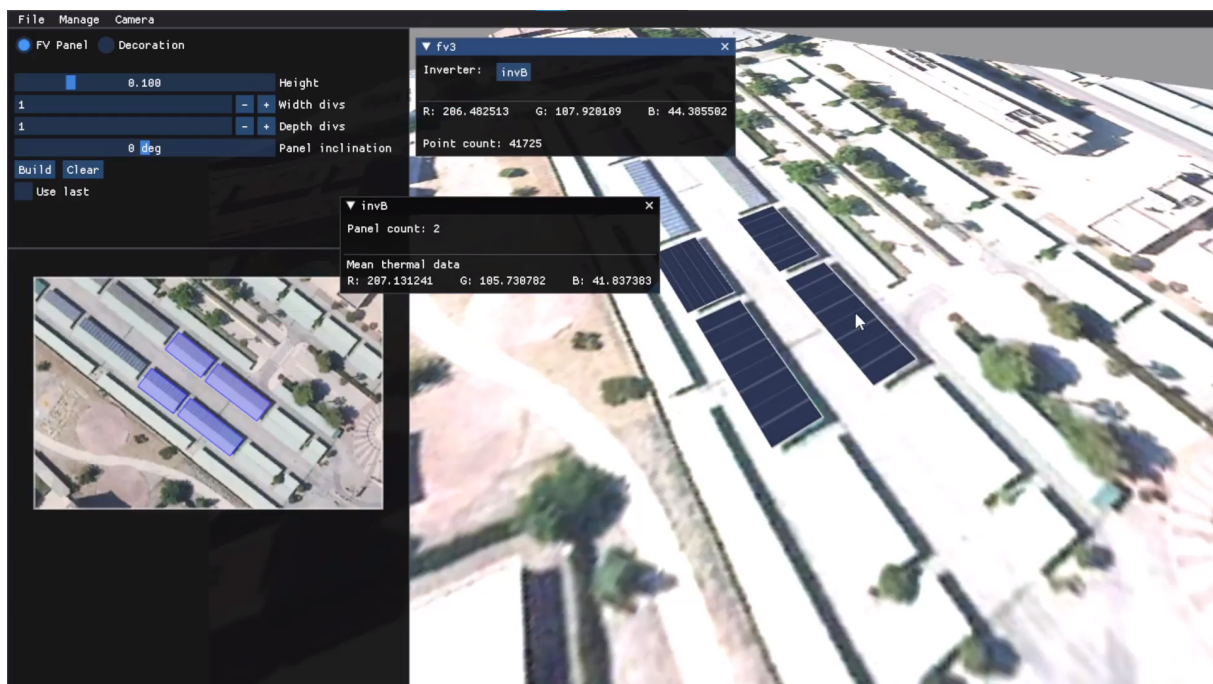


Figura 5.23: Consulta de información cargada tras el incremento 4

Diagrama de clases

La figura 5.24 muestra el diagrama final de clases al acabar el cuarto incremento, siendo además el último estado del proyecto hasta su finalización.

5.5.3. Pruebas y consideraciones

Uno de los puntos esenciales para la validación de este incremento es la robustez de la aplicación a la hora de cargar nubes de puntos; se propone como objetivo encontrar la cantidad de puntos que puede soportar simultáneamente sin que la estabilidad se vea comprometida. En la figura 5.22, la nube mostrada está compuesta por casi 4 millones de puntos y se visualiza correctamente y de forma fluida, alcanzando hasta 60 fotogramas por segundo sin mucho estrés en la máquina principal de desarrollo; al comprobar la carga de una nube mucho mayor de 29 millones de puntos, el proceso de importación se extiende a varios minutos.

Con la figura 5.25 se presentan las métricas de rendimiento del equipo tras importar las dos nubes de puntos. En ambos casos, puede apreciarse ligeramente el instante en que comienza la carga, pues el gráfico de utilización de *GPU* baja hasta el mínimo casi instantáneamente (en la primera gráfica, se encuentra sobre el tercer cuarto del historial, mientras que para la segunda, está situado poco antes de la mitad). Algo que sí es claramente apreciable es el lapso de tiempo desde que comienza hasta que termina la carga y, por tanto, sube repentinamente el uso de *GPU*: para la nube de casi 30 millones de puntos este periodo extiende mucho más en el tiempo. También se percibe un claro impacto en el uso de memoria principal, donde la segunda y mayor nube presenta un pico muy elevado de utilización, y, una vez estabilizado, se mantiene por encima en comparación a la nube de menor tamaño (aproximadamente 1 GB más).

Respecto a la prueba de rendimiento anterior, cabe mencionar que la aplicación no está preparada para el uso de concurrencia, además de encontrarse en el modo de compilación para depuración, lo que afecta en gran medida a su ejecución. Implementando en un futuro una carga que aproveche varios hilos, los tiempos de espera se podrían reducir hasta ser aceptables, incrementando la usabilidad del sistema hasta nubes de varias decenas de millones de puntos.

Siendo este el último incremento implementado en la extensión del proyecto, se vuelven a realizar las pruebas de compatibilidad con otros equipos, verificando que las nuevas librerías y funcionalidades puedan ejecutarse en hardware diferente. Entre los equipos comprobados, los mismos que se indican en la sección de pruebas del

Visual Paradigm Standard(Juan Carlos Fernandez Perez(Universidad de Jaen))

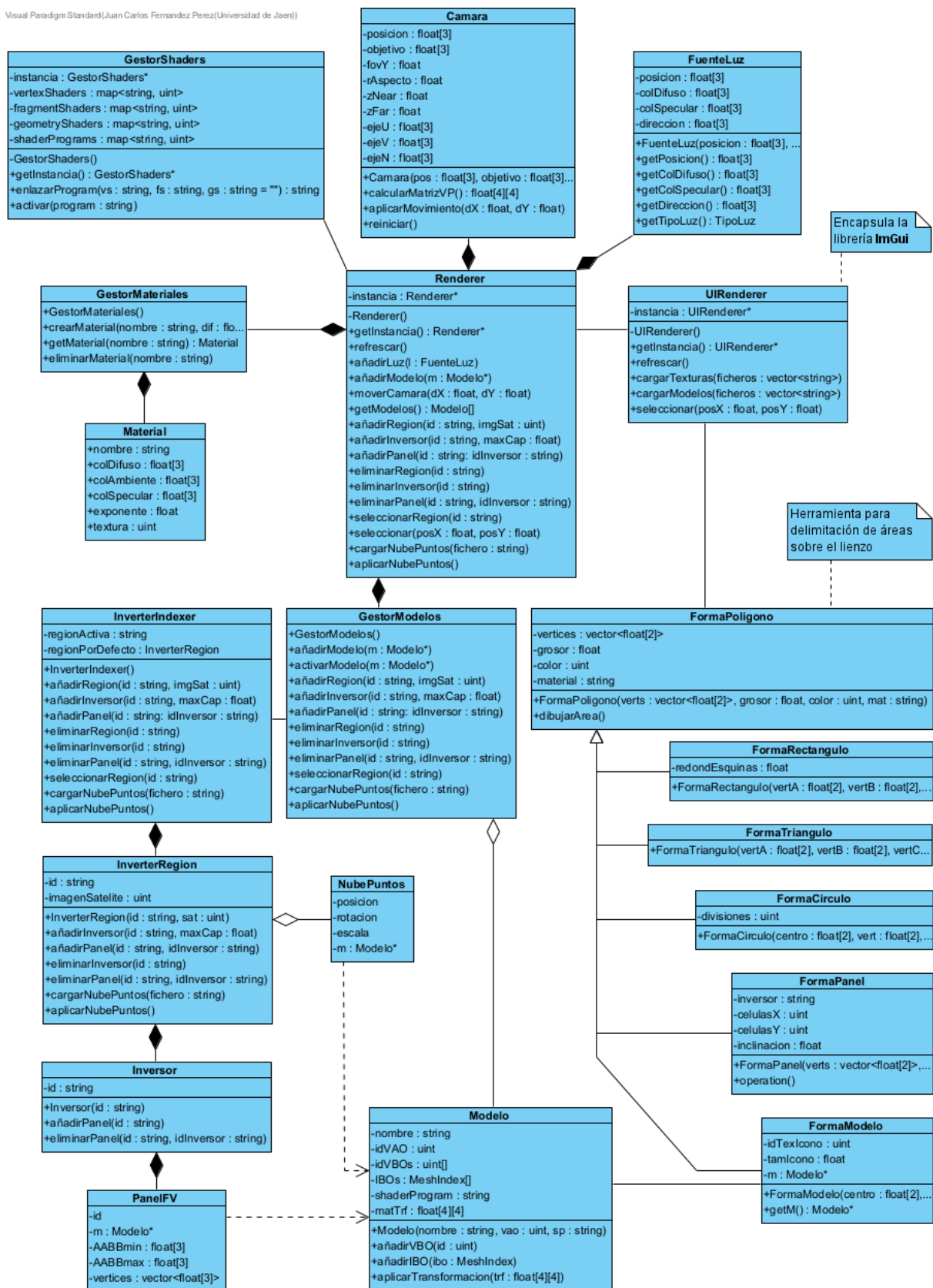


Figura 5.24: Diagrama de clases final del incremento 4

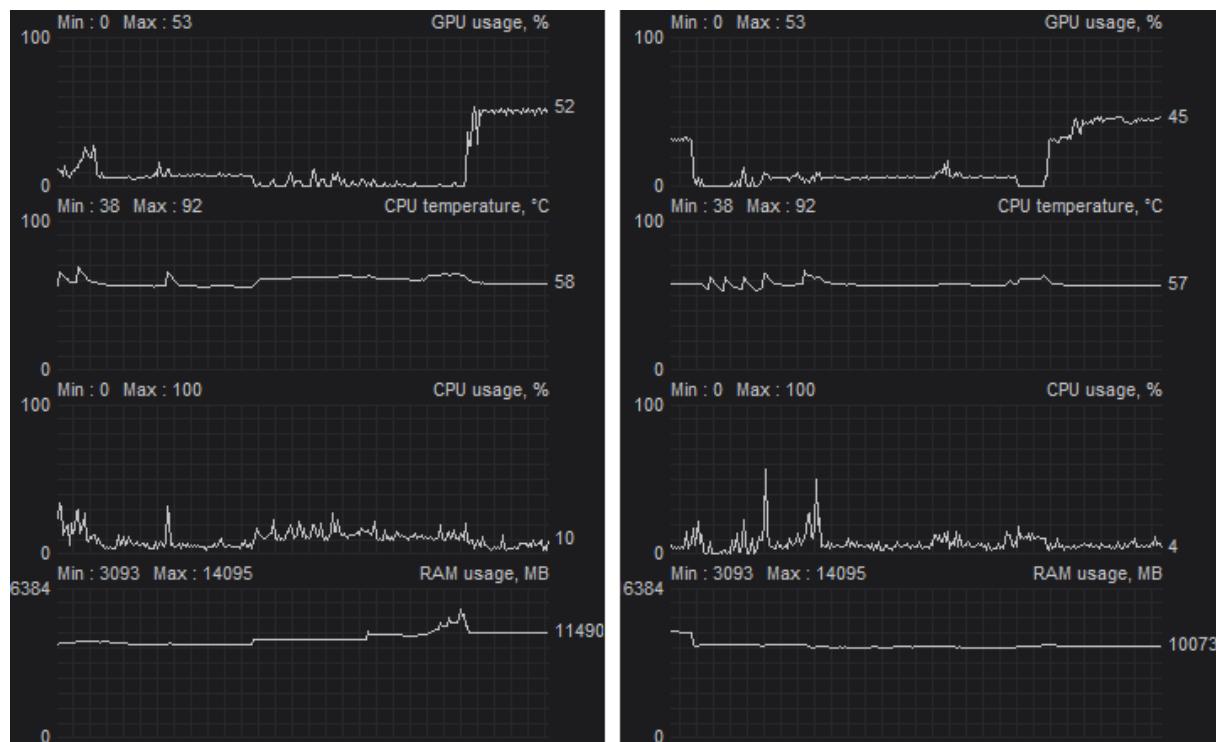


Figura 5.25: Comparación del rendimiento tras cargar dos nubes de puntos. Izquierda: 4 millones de puntos. Derecha: 30 millones de puntos.

incremento 1 (sección 5.2.3), ninguno aparenta presentar problemas durante la utilización básica de la aplicación: crear una región e importar una imagen satélite, definir diversos elementos decorativos y paneles generadores con sus inversores correspondientes, y cargar una nube de puntos aplicando su información al escenario. Lo que sí era de esperar es la obtención mucho más tardía de resultados en los equipos de prestaciones inferiores, aunque de igual manera el sistema sigue siendo estable y proporciona una interacción fluida.

Durante las pruebas anteriores y el uso de la aplicación, surgen dos problemas a considerar para el futuro. Empezando por el de menor relevancia, puesto que es únicamente visual, en la figura 5.23 pueden verse los diferentes paneles generados, pero uno de ellos se encuentra con la textura orientada en una dirección distinta; esto es seguramente debido al orden en que se definen los vértices al delimitar el rectángulo en el área de dibujo, surgiendo desajustes en las coordenadas de textura.

Por otro lado, un problema de mayor seriedad viene dado por la falta de advertencias al cargar una nube de puntos cuyo formato no es compatible (por ejemplo, que no disponga de la información de color). Si esta situación se da, la aplicación colapsa al instante aunque mostrando un mensaje de error. Esto deberá solventarse como prioridad en siguientes incrementos, analizando previamente el formato de la nube y notificando al usuario la incompatibilidad con la aplicación.

Capítulo 6

CONCLUSIONES

Durante el desarrollo del proyecto se ha abarcado la evolución del concepto de gemelo digital, analizando trabajos y estudios de los últimos años que han condicionado nuevas mejoras en su implementación. Esto ha permitido comprender el concepto y lo que aporta a trabajos de diferentes sectores en que puede aplicarse.

Con este trabajo se busca aportar a la extensión de la visibilidad del término, desarrollando para ello una aplicación gráfica partiendo de las bases que presenta cualquier gemelo digital. A pesar de no incluir la totalidad de las funciones que se tenían en mente, el resultado aportado forma una base estable sobre la construir una aplicación con cierta robustez y con un alto rango de posibilidades.

Estas características de la aplicación se alcanzan gracias a la inclusión de técnicas y métodos preparados especialmente para aplicaciones gráficas que garanticen fluidez en la tasa de fotogramas generados. Sin estas mejoras, buena parte de la funcionalidad provocaría inestabilidad en el sistema durante su uso. Al igual que con el concepto de gemelo digital, los métodos mencionados han requerido también un estudio previo y la preparación del algoritmo que lo implementan, pues el uso de determinados métodos podía no ser beneficioso en el ámbito del trabajo; esas alternativas debían conocerse para ser evaluadas antes de implementarse.

6.1. Objetivos de trabajo futuro

Tras el desarrollo de los diversos incrementos de la aplicación, siguen surgiendo motivos que inciten a la continuación del trabajo. De primeras, a la finalización del cuarto y último incremento se detectaron, durante la fase de pruebas, varios **problemas que deberán ser solventados**, idóneamente antes de añadir nueva funcionalidad.

En cuanto a esa nueva funcionalidad, queda pendiente implementar la compatibilidad con maneras más diversas de **captación de datos** del sistema real, tomando como objetivo principal el soporte con una red de dispositivos *IoT* que lo monitoricen constantemente. Habilitando esta monitorización continua, podrán incluirse procesos de **análisis de datos** en busca de problemas con el rendimiento del sistema, de forma que queden notificados al usuario y puedan tratarse según sea conveniente.

Los objetivos propuestos buscan orientar el desarrollo hacia el resto de características propias del gemelo digital pendientes de ser implementadas, tratando de alcanzar finalmente la interconexión entre los sistemas real y virtual que facilite la gestión automática de la instalación solar fotovoltaica modelada.

Capítulo 7

APÉNDICES

7.1. Instalación y compilación del proyecto

El proyecto consta de multitud de librerías y dependencias para su compilación, siendo necesaria su instalación previa. Es recomendable el uso de algún gestor de paquetes; debido a que la aplicación está destinada a sistemas Windows, la opción empleada durante el desarrollo es **vcpkg**¹, encargado de descargar, instalar e integrar las librerías en el entorno de desarrollo (concretamente, **Visual Studio 2022**).

Mediante el comando `vcpkg.exe install <nombrePaquete>` se buscará e instalará el paquete indicado, incluyendo además sus dependencias adicionales. La tabla 7.1 presenta los paquetes que requieren instalación desde *vcpkg*, indicando además la versión en que se utilizan para la compilación del proyecto. Aquellas entradas sin versión hacen referencia a extensiones de la librería principal.

Una vez instaladas, el siguiente paso debe ser integrarlas en el entorno de desarrollo, para lo cual *vcpkg* facilita el siguiente comando: `vcpkg.exe integrate install`.

El último paso consiste en abrir el directorio del proyecto con los ficheros fuente, y generar la caché para *CMake* (Visual Studio se encarga de ello automáticamente al detectar el fichero de configuración). Una vez generada, si no han surgido problemas durante la instalación, el proyecto podrá compilarse desde el entorno, generando un fichero ejecutable con las librerías compartidas enlazadas en el mismo directorio.

¹<https://github.com/microsoft/vcpkg>

Nombre del paquete	Versión
opengl	2022-03-14#1
glew	2.2.0
glfw3	3.3.8
glm	0.9.9.8#1
assimp	5.2.4
imgui	1.88#1
imgui[docking-experimental]	
imgui[glfw-binding]	
imgui[opengl3-binding]	
pcl	1.12.0#6
pcl[opengl]	

Tabla. 7.1: Listado de librerías y dependencias

Bibliografía

- Michael Grieves. Digital twin: manufacturing excellence through virtual factory replication. *White paper*, 1(2014):1–7, 2014.
- David Jones, Chris Snider, Aydin Nassehi, Jason Yon, and Ben Hicks. Characterising the digital twin: A systematic literature review. *CIRP Journal of Manufacturing Science and Technology*, 29:36–52, 2020.
- Aidan Fuller, Zhong Fan, Charles Day, and Chris Barlow. Digital twin: Enabling technologies, challenges and open research. *IEEE access*, 8:108952–108971, 2020.
- Concetta Semeraro, Mario Lezoche, Hervé Panetto, and Michele Dassisti. Digital twin paradigm: A systematic literature review. *Computers in Industry*, 130:103469, 2021.
- Mengnan Liu, Shuiliang Fang, Huiyue Dong, and Cunzhi Xu. Review of digital twin about concepts, technologies, and industrial applications. *Journal of Manufacturing Systems*, 58:346–361, 2021.
- Nadja Enders, Martin Robert y Hoßbach. Dimensions of digital twin applications-a literature review. 2019.
- Knud Lasse. State of the iot 2020: 12 billion iot connections, surpassing non-iot for the first time, 2020. URL <https://iot-analytics.com/author/knud-lasse-lueth/>. comprobado en 2022-08-21.
- Martin De Saulles. Internet of things statistics, 2021. URL <https://informationmatters.net/internet-of-things-statistics/>. comprobado en 2022-08-21.
- Heiner Lasi, Peter Fettke, Hans-Georg Kemper, Thomas Feld, and Michael Hoffmann. Industry 4.0. *Business & information systems engineering*, 6(4):239–242, 2014.
- Ján Vachálek, Lukás Bartalský, Oliver Rovný, Dana Šišmišová, Martin Morháč, and Milan Lokšík. The digital twin of an industrial production line within the industry 4.0 concept. In *2017 21st international conference on process control (PC)*, pages 258–262. IEEE, 2017.

- Shohin Aheleroff, Xun Xu, Ray Y Zhong, and Yuqian Lu. Digital twin as a service (dtaas) in industry 4.0: an architecture reference model. *Advanced Engineering Informatics*, 47:101225, 2021.
- Timo Ruohomäki, Enni Airaksinen, Petteri Huuska, Outi Kesäniemi, Mikko Martikka, and Jarmo Suomisto. Smart city platform enabling digital twin. In *2018 International Conference on Intelligent Systems (IS)*, pages 155–161. IEEE, 2018.
- Gary White, Anna Zink, Lara Codecá, and Siobhán Clarke. A digital twin smart city for citizen feedback. *Cities*, 110:103064, 2021.
- Ying Liu, Lin Zhang, Yuan Yang, Longfei Zhou, Lei Ren, Fei Wang, Rong Liu, Zhibo Pang, and M Jamal Deen. A novel cloud-based framework for the elderly healthcare services using digital twin. *IEEE access*, 7:49088–49101, 2019.
- Amir Ebrahimi. Challenges of developing a digital twin model of renewable energy generators. In *2019 IEEE 28th international symposium on industrial electronics (ISIE)*, pages 1059–1066. IEEE, 2019.
- Feng-Que Pei, Yi-Fei Tong, Ming-Hai Yuan, Kun Ding, and Xi-Hui Chen. The digital twin of the quality monitoring and control in the series solar cell production line. *Journal of Manufacturing Systems*, 59:127–137, 2021.
- Philipp Kunze, Stefan Rein, Marc Hemsendorf, Klaus Ramspeck, and Matthias Demant. Learning an empirical digital twin from measurement images for a comprehensive quality inspection of solar cells. *Solar RRL*, 6(5):2100483, 2022.
- Palak Jain, Jason Poon, Jai Prakash Singh, Costas Spanos, Seth R Sanders, and Sanjib Kumar Panda. A digital twin approach for fault diagnosis in distributed photovoltaic systems. *IEEE Transactions on Power Electronics*, 35(1):940–956, 2019.
- TI Zohdi. A digital-twin and machine-learning framework for the design of multiobjective agrophotovoltaic solar farms. *Computational Mechanics*, 68(2):357–370, 2021.
- Adolf Goetzberger and A Zastrow. On the coexistence of solar-energy conversion and plant cultivation. *International Journal of Solar Energy*, 1(1):55–69, 1982.
- Stefan Scharl and Aaron Praktijnjo. The role of a digital industry 4.0 in a renewable energy system. *International journal of energy research*, 43(8):3891–3904, 2019.
- David Catuhe. Babylon.js, 2013. URL <https://github.com/BabylonJS/Babylon.js#useful-links>. comprobado en 2022-06-23.
- Joseph A Shiraef. An exploratory study of high performance graphics application programming interfaces. 2016.

Barry W. Boehm. *Software engineering economics*. Englewood Cliffs, New Jersey, 1981. ISBN 978-0-13-822122-5.

BOE. Resolución de 22 de febrero de 2018, de la Dirección General de Empleo, por la que se registra y publica el XVII Convenio colectivo estatal de empresas de consultoría y estudios de mercado y de la opinión pública, 2018.

R.S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill higher education. McGraw-Hill Education, 2010. ISBN 9780073375977. URL https://books.google.es/books?id=y4k_AQAAIAAJ.

Larman Craig. *UML y patrones*. Prentice Hall, November 2004.

Jakob Nielsen Don Norman. The definition of user experience (ux), 2016. URL <https://www.nngroup.com/articles/definition-user-experience/>. comprobado en 2022-07-17.

