



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Facultat d'Informàtica de Barcelona



FACULTAD DE INFORMÁTICA DE BARCELONA

GRADO EN INGENIERÍA INFORMÁTICA

ESPECIALIDAD DE COMPUTACIÓN

Amueblado de propiedades en 3D

TRABAJO DE FIN DE GRADO

Joan Manuel Ramos Refusta

Director: Sergi Jiménez

Ponente: Antonio Chica

Tutor GEP: Fernando Marfil Sanchez

20 de junio de 2021

Agradecimientos

Principalmente me gustaría agradecer todo el apoyo ofrecido por mi familia a lo largo de la carrera por estar siempre ahí de forma incondicional.

También me gustaría agradecer a aquellos profesores con los que he coincidido y han aportado sus conocimientos de forma vocacional, aportando motivación a los estudiantes por aprender.

Aprovecho para agradecer a todos mis compañeros de carrera y amigos personales que he me han acompañado durante la carrera y me han apoyado en todo lo posible, tanto fuera como dentro de la universidad.

Y finalmente me gustaría agradecer a Floorfy por la oportunidad laboral, por darme un proyecto acorde a mis necesidades y a los compañeros de trabajo por acogerme tan bien.

Resumen

Floorfy es una de las empresas que se dedica a realizar renders tridimensionales de hogares para inmobiliarias. Estos renders se les llama *tours virtuales*.

Este servicio tiene muchas ventajas y facilitaciones para el usuario. Floorfy cuenta conmigo para realizar un nuevo aplicativo dentro de estos *tours virtuales* que se basa en amueblar un inmueble mediante modelos tridimensionales, a demás de la posibilidad de cambiar los colores y texturas de los objetos, suelos y paredes que hay en el inmueble.

Para ello, también será necesario añadir un catálogo con los modelos y texturas disponibles, a demás de añadir unos controles para que el usuario se pueda manejar bien sobre la nueva herramienta.

Esta nueva funcionalidad que ofrecerá Floorfy permite al usuario previsualizar un inmueble amueblado a su gusto antes de una posible adquisición, con la finalidad de hacerse una idea de cómo quedaría con una decoración concreta.

Resum

Floorfy és una de les empreses que es dedica a realitzar renders tridimensionals de llars per immobiliàries. Aquests renders se'ls anomena *tours virtuals*.

Aquest servei té molts avantatges i facilitacions per a l'usuari. Floorfy comparteix amb mi per fer un nou aplicatiu dins d'aquests *tours virtuals* que es basa en moblar un immoble mitjançant models tridimensionals, a més de la possibilitat de canviar els colors i textures dels objectes, terres i parets que hi ha al immoble.

Per a això, també caldrà afegir un catàleg amb els models i textures disponibles, a més d'afegir uns controls perquè l'usuari es pugui gestionar bé sobre la nova eina.

Aquesta nova funcionalitat que oferirà Floorfy permet a l'usuari previsualitzar un immoble moblat al seu gust abans d'una possible adquisició, amb la finalitat de fer-se una idea de com quedaria amb una decoració concreta.

Abstract

Floorfy is one of the companies that is dedicated to making three-dimensional renderings of homes for real estate. These renders are called virtual tours.

This service has many advantages and facilities for the user. Floorfy is counting on me to create a new application within these virtual tours that is based on furnishing a property using three-dimensional models, in addition to the possibility of changing the colors and textures of the objects, floors and walls that are in it.

To do this, it will also be necessary to add a catalog with the available models and textures, in addition to adding some controls so that the user can handle the new tool well.

This new functionality that Floorfy will offer allows the user to preview a property furnished to their liking before a possible acquisition, in order to get an idea of how it would look with a specific decoration.

Índice

1. Introducción y contextualización	14
1.1. Contexto	14
1.2. Conceptos	15
1.2.1. <i>tour virtual</i>	15
1.2.2. Modelo 3D	16
1.2.3. Materiales y texturas	16
1.3. Identificación del problema	17
1.4. Agentes implicados	17
1.4.1. A quién va dirigido	17
1.4.2. Floorfy	18
1.4.3. Director y ponente	18
1.4.4. Yo	18
1.5. Integración del conocimiento	18
1.6. Identificación de leyes y regulaciones	19
2. Justificación	20
2.1. Competencia	20
2.2. Proyecto en Floorfy	21
3. Alcance	22
3.1. Objetivos y sub-objetivos	22
3.2. Requerimientos	23
3.2.1. Requerimientos funcionales	23
3.2.2. Requerimientos no funcionales	23
3.3. Obstáculos y riesgos	24

4. Metodología	24
4.1. Herramientas	25
5. Planificación temporal	25
5.1. Descripción de tareas	26
5.2. Recursos	31
5.2.1. Recursos humanos	31
5.2.2. Recursos materiales	32
5.3. Gestión del riesgo	33
5.4. Cambios en la planificación inicial	35
6. Gestión económica	36
6.1. Presupuesto	36
6.1.1. Coste del personal	36
6.1.2. Costes genéricos	37
6.1.3. Coste de contingencia	38
6.1.4. Coste de imprevistos	38
6.1.5. Coste total	40
6.2. Control de gestión	40
6.3. Cambios en el presupuesto inicial	41
7. Sostenibilidad	41
7.1. Autoevaluación	41
7.2. Dimensión económica	42
7.3. Dimensión ambiental	42
7.4. Dimensión social	42

8. Herramientas	43
8.1. <i>Three js</i>	43
8.2. HTML y CSS	44
8.3. Obtención de recursos	44
8.4. Herramientas de trabajo	45
8.4.1. Jira	45
8.4.2. BitBucket	46
9. Implementación	47
9.1. Catálogo	47
9.1.1. Muebles	47
9.1.1.1. Object options	49
9.1.2. Pintar paredes y suelos	52
9.1.2.1. Paint options	54
9.2. Amueblado	54
9.2.1. Formato de archivos 3D	54
9.2.1.1. Formato OBJ	55
9.2.1.2. Formato MTL	56
9.2.2. Configuración previa de los objetos	60
9.2.2.1. Tamaño	60
9.2.2.2. Layers	61
9.2.2.3. <i>Pivot point</i>	61
9.2.2.4. Tipo de textura	62
9.2.2.5. Brillo	63
9.2.3. Categorización de objetos	64
9.2.3.1. Categorización por tipo de habitación	64

9.2.3.2. Categorización por tipo de posicionamiento	65
9.2.4. Iluminación	68
9.2.4.1. Ambient Light	68
9.2.4.2. Directional Light	69
9.2.5. <i>Raycasting</i>	70
9.2.5.1. Posicionar un objeto seleccionado	71
9.2.5.2. Seleccionar un objeto de la escena	74
9.2.6. Posición valida	75
9.2.6.1. <i>Bounding Box</i>	76
9.2.6.2. Límites de objetos <i>floor</i>	78
9.2.6.3. Límites de objetos <i>wall</i> y <i>wallFloor</i>	82
9.2.6.4. Límites de objetos <i>item</i>	86
9.2.6.5. Colisión con otros objetos	88
9.2.7. Controles	93
9.2.7.1. Controles básicos mediante ratón	95
9.2.7.2. <i>Slider</i>	96
9.2.7.3. Color	97
9.2.7.4. Textura	99
9.2.7.5. Reset	101
9.2.7.6. Tour Controls	102
9.2.7.7. Clonar	103
9.2.7.8. Lock in place	104
9.2.7.9. Eliminar	104
9.2.7.10. Rotación	105
9.3. Paredes y suelos	106

9.3.1.	Selección de paredes y suelos	106
9.3.2.	Paint Options	108
9.3.2.1.	Textura de pared y suelo	108
9.3.2.2.	Color	110
9.3.2.3.	Unselect	111
9.3.2.4.	Reset	111
10. Conclusiones finales		111
10.1.	Futuro del proyecto	112
10.2.	Conclusiones	112
10.3.	Competencias técnicas	115
10.3.1.	CCO1.1	116
10.3.2.	CCO2.3	116
10.3.3.	CCO2.6	116
10.3.4.	CCO3.1	117

Índice de figuras

1.	Perspectivas en los tours virtuales de Floorfy, fuente: [1]	16
2.	Ejemplo de modelo tridimensional con y sin texturas, fuente: [4]	17
3.	Diagrama de Gantt, fuente: Elaboración propia.	34
4.	Diagrama de Gantt modificado. Elaboración propia.	35
5.	Captura del <i>backlog</i> de mi <i>Jira</i> personal con tareas realizadas. Elaboración propia.	46
6.	Captura del historial de <i>commits</i> y código actualizado en el proyecto. Elaboración propia.	46
7.	Captura del ícono que abre el catálogo en el proyecto. Elaboración propia.	47
8.	Catálogo de muebles. Elaboración propia	48
9.	Ejemplo de dos de las miniaturas de objetos que componen el catálogo. Están elaboradas a partir de la aplicación online 3dviewer. Fuente: [30]	49
10.	Captura catálogo en la sección de Object Options. Elaboración propia.	50
11.	Captura del catálogo en la sección de Object Options en la opción de Textures. Elaboración propia.	51
12.	Captura catálogo en la sección de Object Options en la opción de Textures. Elaboración propia.	51
13.	Catálogo de pintar paredes. Elaboración propia	53
14.	Efectos de aplicar los valores de Ka, Kd y Ks explicados anteriormente, fuente: [34]	58
15.	Representación de los vectores que afectan en la reflexión del modelo de Phong, fuente: [35]	59
16.	Ejemplo del objeto “Taburete” donde sólo el subobjeto que forma parte del asiento es el layer. Se puede observar que únicamente cambia de aspecto esta parte y no el resto del objeto. Elaboración propia.	61

17.	Ejemplo de modificación del <i>Pivot point</i> en <i>Blender</i> y su representación en la escena. Elaboración propia.	62
18.	Catálogo de ambos tipos de texturas. Elaboración propia.	63
19.	Ejemplo al modificar el valor de <i>shininess</i> de un objeto. Elaboración propia.	64
20.	Ejemplo del <i>Pivot point</i> de un objeto de tipo <i>floor</i> . Elaboración propia.	65
21.	Ejemplo del <i>Pivot point</i> de un objeto de tipo <i>wall</i> . Elaboración propia.	66
22.	Ejemplo del <i>Pivot point</i> de un objeto de tipo <i>wallFloor</i> . Elaboración propia.	67
23.	Ejemplo del <i>Pivot point</i> de un objeto de tipo <i>item</i> . Elaboración propia.	67
24.	Comparativa de una escena sin luz y con luz ambiental. Elaboración propia.	68
25.	Comparativa de una escena con luz ambiental y con luz ambiental + luz direccional. Elaboración propia.	69
26.	Representación de la interacción del <i>Raycasting</i> mediante la cámara, la posición del ratón en la pantalla y la escena 3D. Fuente: [37]	70
27.	Ejemplo de posicionamiento con un objeto de tipo <i>wall</i> . Elaboración propia.	72
28.	Representación de la solución al problema de normales invertidas en paredes. Elaboración propia.	73
29.	Representación de la diferencia de opacidad al interseccionar un objeto. Elaboración propia.	74
30.	Captura del objeto "Armario oscuro" seleccionado. Elaboración propia.	75
31.	Captura del objeto "Silla normal" en una posición no válida. Elaboración propia.	76
32.	Dos modelos, uno con <i>bounding box</i> final (anaranjado) y otro con AABB por defecto (en amarillo). Elaboración propia.	77

33.	Representación de los <i>pointsObject</i> (en rojo) y los <i>pointsFloor</i> (en azul) en un objeto de tipo <i>floor</i> . Elaboración propia.	79
34.	Representación de la comprobación que hace falta para que el algoritmo sea robusto en todos los posibles casos. Elaboración propia.	81
35.	Representación de los <i>pointsObject</i> (en rojo) y los <i>pointsWall</i> (en azul) en un objeto de tipo <i>wall</i> . Elaboración propia.	83
36.	Representación de la normal (línea azul) y segmento perpendicular a la normal (línea roja) que se ha utilizado como dirección para hacer el segmento desde p_i hacia <i>aux</i> . Elaboración propia. . .	84
37.	Ejemplo del objeto de tipo <i>wallFloor</i> “Cocina completa 1”, en el que se ve que la parte inferior del objeto no cabe en el suelo. Elaboración propia.	86
38.	Representación de una colisión entre dos objetos. Elaboración propia.	89
39.	Ejemplo de un objeto colisionando en una escena del <i>tour virtual</i> . Elaboración propia.	93
40.	Representación de la modificación del tamaño mediante el uso del <i>slider</i> . Elaboración propia.	97
41.	Objeto “Comoda” con diferentes colores. Elaboración propia. . .	98
42.	Objeto “Comoda” con diferente color y textura. Elaboración propia. .	99
43.	Objeto “Cocina completa 2” diferentes texturas. Elaboración propia.	101
44.	Ejemplo del uso del botón de reset en un objeto. Elaboración propia.	102
45.	Ejemplo del uso del botón clone con varios taburetes con el mismo color y textura. Elaboración propia	103
46.	Ejemplo del uso del control de rotación con un objeto. Elaboración propia.	104
47.	Ejemplo oficial de <i>three js</i> del TransformControl de rotación por defecto. Fuente: [14].	105
48.	Ejemplo del uso del control de rotación con un objeto. Elaboración propia.	106

49.	Ejemplo del uso del selector de suelos o paredes. Elaboración propia.	107
50.	Solución a los hotspots con renderOrder. Elaboración propia. . .	109
51.	Comparación de una habitación del un <i>tour virtual</i> sin texturizar y con las paredes y suelos texturizados mediante el aplicativo. Elaboración propia.	110
52.	Comparación de una habitación del un <i>tour virtual</i> texturizada y texturizada con color. Elaboración propia.	111
53.	Habitaciones modificadas tanto con modelos 3D como con texturas. Elaboración propia.	115

Índice de tablas

1.	Tabla con distintas características de otras empresas similares a Floorfy, fuentes: [7, 8, 9, 10, 11, 12]	20
2.	Tabla de tareas con la duración, dependencias, recursos y agentes implicados, DP: Director del proyecto, I: Investigador, P: Programador, fuente: Elaboración propia	32
3.	Tabla con los costes de personal a partir de glassdoor y neuvoo, fuente: Elaboración propia	36
4.	Tabla con la estimación del coste de personal, fuente: Elaboración propia	37
5.	Tabla de contingencia del 7 % del proyecto, fuente: Elaboración propia	38
6.	Tabla del coste de imprevistos, fuente: Elaboración propia	39
7.	Tabla del presupuesto total del proyecto, fuente: Elaboración propia	40

1. Introducción y contextualización

En los últimos años, a causa del avance tecnológico, hemos visto varios cambios en el modo de uso de algunos servicios tradicionales. Como por ejemplo, el correo electrónico como alternativa a la carta clásica; la mayor parte de periódicos y revistas se han visto forzadas a distribuirse de forma digital; ahora disfrutamos de series y películas a través de una plataforma en línea en vez de consumir un estreno por la televisión tradicional o en el cine físicamente, etcétera.

También ocurre lo mismo en el sector inmobiliario, donde surge, como alternativa a la visita tradicional de un inmueble, el *tour virtual*. La diferencia con su antecesor clásico es que éste se puede realizar a partir de un ordenador o dispositivo móvil, sin necesidad de desplazarse al hogar. Diferentes empresas se dedican a realizar este servicio, con distintos puntos de vista y diferentes herramientas dentro de ellas.

Este proyecto trata sobre una nueva herramienta dentro del *tour virtual* de Floorfy, la cual se basa en poder realizar cambios en los colores y materiales de las paredes, suelos y techos, añadiendo la posibilidad de introducir mobiliario al gusto.

1.1. Contexto

En el Grado de Ingeniería Informática impartido por la Facultad de Informática de Barcelona existen diferentes especialidades a escoger por el alumnado. En este caso, este Trabajo de Fin de Grado pertenece a la rama de Computación, concretamente, al campo de gráficos por computador. Uno de los objetivos principales de este proyecto es poner en práctica el conocimiento adquirido sobre este ámbito durante la carrera en un caso real, como es la elaboración de esta herramienta para Floorfy.

Floorfy es una plataforma fundada en 2016 como una *start-up* que permite a profesionales inmobiliarios virtualizar propiedades a partir de capturas fotográficas hechas con cámaras 360°. Se genera un “tour o visita” virtual interactiva del inmueble, planos 3D, planos de planta acotados, llamada virtual integrada, vídeo comercial (simulando el recorrido de la visita), fotografías 2D en HD, y más. Gracias a sus tours virtuales, se facilita la visualización del inmueble mediante un modelo en 3D.

La plataforma permite al inquilino o comprador potencial visitar el inmueble sin desplazarse, visitando el propio inmueble de forma interactiva y virtual, permitiendo al usuario que se desplace por los pasillos y habitaciones simulando una visita física. Este servicio se puede utilizar en dispositivos móviles, a partir

de un ordenador o incluso utilizando gafas de Realidad Virtual, para así obtener una experiencia más inmersiva.

Actualmente Floorfy cuenta con más de 50.000 tours activos (inmuebles virtualizados) y más de 3.000.000 de visitas virtuales al mes, operando en más de 15 países diferentes [1].

1.2. Conceptos

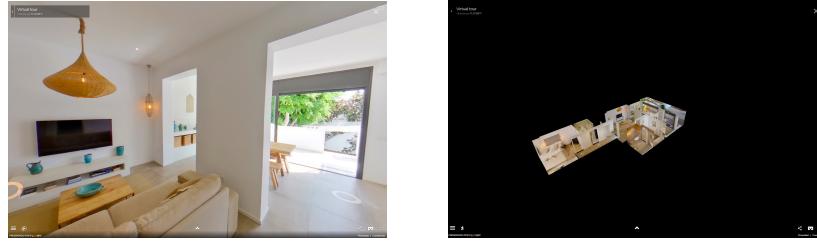
A continuación se definen varios conceptos clave sobre este proyecto.

1.2.1. *tour virtual*

Para hablar de este proyecto, se ha de definir el concepto de *tour virtual*, dado que será el espacio donde se trabaje esa nueva implementación. Un *tour virtual* es una simulación en 3D de una visita de un inmueble. Dentro del tour, hay dos tipos de perspectivas posibles. En la primera nos encontramos dentro de la propiedad, en primera persona y podemos viajar dentro de cada una de las habitaciones a partir de *hotspots*. Estos *hotspots* no son más que los puntos donde se ha realizado una fotografía en 360º. Desde uno de estos puntos podemos visualizar la escena en primera persona desde la posición donde se ha colocado el trípode para hacer esa fotografía. En esta perspectiva hay tantos puntos de vista como fotografías 360º se hayan realizado por este motivo.

También está el punto de vista aéreo. Desde este punto podemos ver toda la casa en tercera persona, visualizando todo el inmueble al completo, pudiendo interactuar para ir observando el inmueble con más detalle. Esta alternativa permite al usuario ver con exactitud toda la distribución de la casa. Se pueden observar ambos puntos de vista en el ejemplo de la figura 1.

Será posible utilizar la nueva funcionalidad desarrollada en este proyecto a partir de cualquiera de los dos puntos de vista descritos.



(a) Vista en primera persona

(b) Vista en tercera persona

Figura 1: Perspectivas en los tours virtuales de Floorfy, fuente: [1]

1.2.2. Modelo 3D

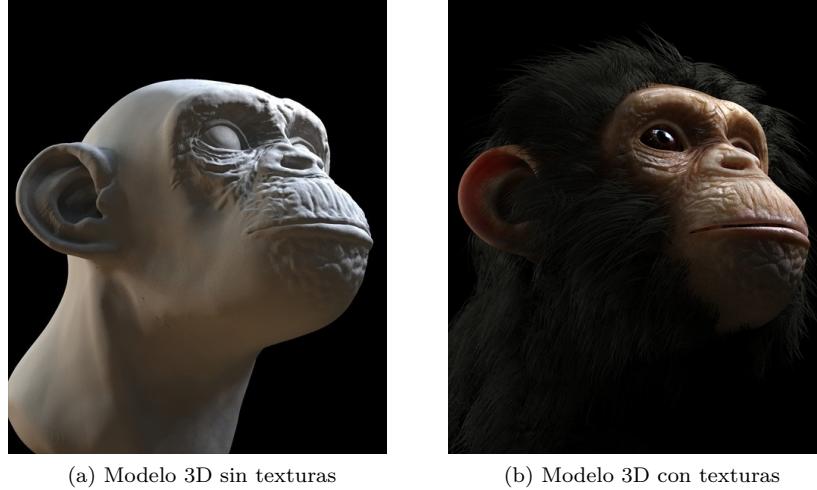
Al proceso de modelado 3D se le define como el proceso de desarrollo de una representación matemática de cualquier objeto tridimensional a través de un software especializado. Al producto generado, se le denomina modelo 3D. Se puede visualizar como una imagen en dos dimensiones aplicando un proceso llamado renderizado 3D [2]. Podemos observar un ejemplo en la primera imagen de la figura 2, en la cual aparece la información geométrica de la cara de un mono a través de su modelo tridimensional.

Por el momento, en los tours virtuales, solo existe un único modelo 3D, una sola forma geométrica: el de la propiedad. A partir de esta nueva herramienta, podremos añadir a la escena mobiliario en forma de modelados 3D.

1.2.3. Materiales y texturas

La palabra ‘material’ etimológicamente quiere decir ‘relativo a la materia’, por lo tanto es una característica de los objetos. En el campo del modelado 3D el material funciona de una manera similar, es un recubrimiento a modo de capa que se le aplica a un objeto modelado, con el fin de darle un estilo determinado.

Un material define las propiedades ópticas de un objeto: su color y si es opaco o brillante. Una textura es un patrón que rompe la apariencia uniforme del material. Muy pocos objetos en el mundo real tienen superficies completamente uniformes. En cambio, la mayoría de ellos tienen patrones o variaciones de color [3]. Un ejemplo de un modelo 3D texturizado es la segunda imagen de la figura 2, en la que aparece el mismo modelo 3D, pero ahora, añadiendo la información que aportan las texturas.



(a) Modelo 3D sin texturas

(b) Modelo 3D con texturas

Figura 2: Ejemplo de modelo tridimensional con y sin texturas, fuente: [4]

1.3. Identificación del problema

La empresa en la que se desarrolla el proyecto está interesada en implementar una nueva herramienta para tener variedad en sus tours virtuales. La herramienta permite al usuario poner diferentes tipos de objetos, modificarles el tamaño, posición y ángulo y permitiendo cambiar el material con el objetivo de que pueda realizar una previsualización de cómo quedaría el inmueble con un mobiliario determinado. Mediante esta nueva herramienta, también será posible modificar el color y material de las paredes, suelos y techos preestablecidos.

La empresa tiene pensado implementar esta nueva funcionalidad de forma automática en un futuro, simplemente indicando el tipo de decoración, se podrían visualizar las habitaciones amuebladas con un determinado estilo. Pero en este proyecto no hablaremos sobre este avance, si no, en una primera etapa de este proyecto, el cual tiene largo recorrido.

1.4. Agentes implicados

1.4.1. A quién va dirigido

Hay dos tipos de potenciales consumidores para la funcionalidad descrita previamente. Un primer tipo son los *editores*, que son trabajadores de la empresa Floorfy, los cuales se dedican a recrear el inmueble en 3D a partir de fotografías

360°. Estos empleados tienen el rol de conectar toda la información transmitida por las imágenes con el *tour virtual*. Esta herramienta les será de gran utilidad para cuando deseen desarrollar nuevos estilos decorativos dentro de un inmueble.

El otro tipo de usuario que utilizará esta nueva herramienta son las personas interesadas en un comprar/alquilar un inmueble con un *tour virtual* en Floorfy. Este tipo de potencial consumidor tiene la finalidad de visualizar cómo quedaría una habitación con una decoración determinada. La herramienta ha de ser intuitiva y fácil de usar para todo tipo de personas, esto quiere decir que cualquier tipo de persona, familiarizada o no con la tecnología, debería ser capaz de utilizar este tipo de herramienta. Por lo tanto, y teniendo en cuenta el público al que va orientado, se deben buscar métodos para facilitar su uso de forma amigable.

1.4.2. Floorfy

Como principal agente de este proyecto, la figura de Floorfy está muy interesada en el desarrollo y éxito de esta nueva funcionalidad en sus tours virtuales. La organización será el máximo beneficiario directo del producto y el que tiene más interés en el correcto desarrollo de la herramienta.

1.4.3. Director y ponente

Tanto el director de este proyecto, el cual figura como CEO de la empresa Floorfy, como el ponente, profesor de la Facultad de Informática de Barcelona, están interesados en el éxito de la ejecución de este proyecto de manera indirecta.

1.4.4. Yo

Soy también un agente implicado directamente en este proyecto. Si se logra ejecutar exitosamente, acabaré el grado en Ingeniería Informática. También considero que el proyecto que Floorfy me ha sugerido es una muy buena oportunidad para aprender y desarrollar mis conocimientos en el campo de la computación gráfica.

1.5. Integración del conocimiento

A continuación se pasa a enumerar las distintas asignaturas de las que he adquirido suficiente conocimiento como para realizar el proyecto en este TFG:

1. PRO1, PRO2. Las primeras asignaturas relacionadas con la programación me ayudaron a asentar las bases de la programación y a empezar a entender cómo funciona un programa en C++.
2. EDA. En EDA adquirí los conocimientos suficientes para entender la importancia de la eficiencia de un programa informático.
3. IDI. Con IDI empecé a tener conciencia sobre la importancia del diseño y la interacción entre el programa y el usuario. A demás de empezar a tener nociones de objetos 3D, texturas, colores e iluminación de entornos mediante OpenGL.
4. PROP. En PROP aprendí a gestionar un proyecto en grupo desde el principio de su desarrollo hasta el final. Es un contacto más directo con lo que se puede hacer en el mundo laboral.
5. EEE. Con EEE aprendí la importancia que tiene la gestión económica durante el desarrollo de un proyecto como este, y lo he podido relacionar con algunos aspectos de GEP.
6. G. En gráficos he conseguido ampliar los conocimientos que empecé a tener en IDI sobre OpenGL.
7. A. En Algoritmia aprendí a resolver problemas con más conciencia sobre la eficiencia y el coste temporal y coste en memoria de las soluciones propuestas.
8. LP. EN LP aprendí la importancia de la forma en la que se elige un lenguaje de programación y lo que implica esta decisión en el programa.
9. VJ. Gracias a la asignatura de VJ aprendía a desarrollar un proyecto relacionado con gráficos de principio a fin. Específicamente relacionado con cámaras, iluminación, sistema de detección de colisiones e información de los objetos 3D.
10. GEOC. En GEOC se adquieren conocimientos sobre geometría que se han aplicado en el proyecto, como intersecciones entre planos y segmentos que han ayudado a realizar las colisiones de los objetos.

1.6. Identificación de leyes y regulaciones

Dado que en la empresa trabajamos con datos relacionados con los clientes como imágenes equirectangulares (imágenes 360º) de sus propiedades, estos datos deben tener una protección para que únicamente pertenezcan a nuestro *dataset* y permitir una seguridad para el cliente. Por tal de cumplir con este tipo de regulación, se trabaja mediante la Ley Orgánica 15/1999 o Ley de protección de datos (LOPD). La LOPD es la ley que se aplica a los datos personales en

España y tiene como finalidad garantizar y proteger las libertades públicas y derechos fundamentales de las personas.

Por otra parte, también podemos hablar sobre la Ley de propiedad intelectual, dado que este TFG se ha realizado en una empresa y se presenta y expone ante la universidad (UPC). La Ley de Propiedad Intelectual recoge una serie de derechos que un autor adquiere en el mismo momento en que cree una obra, independientemente de si la lleva al Registro de Propiedad Intelectual o no, según el Real Decreto Legislativo 1/1996.

2. Justificación

Este tipo de herramienta se ha visto aplicada en varios servicios con objetivos distintos. Por ejemplo, en motores gráficos como *Unity 3D* [5] o en programas informáticos como *Blender* [6], donde podemos añadir modelos en 3D, cambiar su escala, posición y ángulo de rotación, aplicarle materiales y texturas distintas.

Pero lo que buscamos, en este caso, es aplicarlo a la plataforma de Floorfy, teniendo en cuenta que las personas que van a utilizar dicha plataforma no tienen la necesidad de tener conocimientos técnicos sobre este tipo de herramientas. El objetivo es aplicar esta nueva funcionalidad para los inmuebles, y que el usuario sea capaz de realizarlo sin ningún tipo de conocimiento previo sobre el tema y tenga una forma de uso sencilla y amena para cualquier persona.

2.1. Competencia

Como ya he mencionado previamente, hay distintas empresas que han realizado su propio servicio sobre la virtualización de la visita de un inmueble. En la siguiente tabla se puede observar las diferencias en esta funcionalidad de amueblado entre algunas de las plataformas similares a Floorfy.

Amueblado de propiedades 3D en otras empresas			
Empresa	Inmueble	Muebles	Materiales
Matterport 3D	Fotogrametría	Fotogrametría y modelos 3D	Si
Floored	Modelo 3D	Modelos 3D	Si
Shapespark	Modelo 3D	Modelos 3D	Si
Roomle	Modelo 3D	Modelos 3D	Si
Roomy	Modelo 3D	Modelos 3D	No
realisti.co	Fotografías 360º	No	No

Tabla 1: Tabla con distintas características de otras empresas similares a Floorfy, fuentes: [7, 8, 9, 10, 11, 12]

La columna de ‘Inmueble’ hace referencia a cómo se ha generado el habitáculo. En el caso de *Matterport*, utiliza métodos de fotogrametría para conseguir su virtualización del hogar. Esta técnica se basa en capturar varias imágenes alrededor del inmueble y reconstruir el espacio en tres dimensiones calculando la distancia entre objetos [13]. De esta forma, la empresa *Matterport* es capaz de distinguir entre el modelo 3D del habitáculo con los propios muebles, y genera todo el entorno con la información geométrica calculada y la información de colores que reciben las imágenes.

Los muebles, en el caso de *Matterport*, ya están generados mediante fotogrametría, pero también tiene la posibilidad de añadir muebles en 3D.

Es bastante diferente a lo que se utiliza en *Floorfy*, la cual solamente utiliza imágenes en 360º y a partir de éstas, se generan los puntos de perspectiva en primera persona. Para el modelado 3D, se encapsula la información de cada imagen 360º por cada habitación. A este método se le denomina *skybox*.

Como *Floorfy*, *realisti.co* es una compañía que utiliza este tipo de tecnología mediante imágenes 360º. Como vemos en la tabla, esta compañía no ofrece ningún tipo de herramienta para los modelos 3D ni para añadir materiales distintos, de hecho, solo hacen uso de la vista en primera persona por sus tours virtuales.

Las empresas de *Floored*, *Shapespark*, *Roomle* y *Roomy* no se basan en recrear tours virtuales, como *Matterport*, *realisti.co* y *Floorfy*, pero las he considerado importantes dado que utilizan la funcionalidad que se desarrolla en este proyecto. Estas empresas tienen una plataforma donde se crea un habitáculo en 3D, sin ningún tipo de referencia de la realidad. Ofrecen la posibilidad de añadir muebles, modificarlos, y añadir materiales, exceptuando el caso de *Roomy*, en el cual no es posible modificar el material de los objetos añadidos.

2.2. Proyecto en *Floorfy*

Este proyecto, dentro de la empresa *Floorfy*, juntará la vista realista obtenida a partir de las imágenes 360º con modelos 3D. Esta herramienta distinta a las que existen en la competencia dado que el entorno es distinto a las mencionadas anteriormente. Estos modelos en 3D se aplicarán en la escena en ambas perspectivas, facilitando la observación del producto en el inmueble.

También será posible cambiar los materiales de estos objetos, así como el de las paredes, suelos y techos de la vivienda. De esta forma, la nueva funcionalidad tendrá un nivel más alto en cuestión de personalización y el usuario podrá recrear más fácilmente lo que tiene en mente.

Todo el proyecto será creado mediante *three.js* [14], una biblioteca liviana escrita en *JavaScript* para crear y mostrar gráficos por computador en 3D en un navegador web, en conjunción con WebGL [15], la cual es una especificación estándar que define una API implementada en *JavaScript* para la renderización de gráficos en 3D dentro de cualquier navegador web. Tiene la ventaja de que no precisa del uso de *plug-ins* adicionales en cualquier plataforma que soporte OpenGL 2.0 u OpenGL ES 2.0. WebGL está integrada completamente en todos los estándares web del navegador, permitiendo la aceleración hardware (física) de la GPU y el procesamiento de imágenes y efectos como parte del lienzo o *canvas* de la página web.

La empresa ha decidido desarrollar todo su entorno relacionado con el *tour virtual* utilizando esta librería dado que su uso se centra en navegadores web y es la manera más accesible para desarrollar aplicaciones gráficas en este tipo de plataformas.

3. Alcance

A continuación se comentará el alcance de este proyecto de fin de grado, teniendo en cuenta que disponemos de un tiempo de desarrollo limitado y es necesario determinar límites en los objetivos y definir los requerimientos y obstáculos de la nueva funcionalidad.

3.1. Objetivos y sub-objetivos

El objetivo principal de este trabajo es el desarrollo e implementación de la nueva herramienta, la cual permitirá al usuario añadir mobiliario en 3D en un *tour virtual*, a demás de modificar los materiales de los modelados, las paredes, techos y suelos del mobiliario. Seguidamente, se definen los sub-objetivos:

- Implementar un catálogo con los modelados de los muebles 3D a aplicar. Este catálogo estará disponible en ambas perspectivas del tour y el usuario será capaz de utilizar de forma intuitiva esta herramienta y de poner los objetos por el inmueble.
- Implementar la opción de cambiar los materiales de los objetos una vez añadidos al *tour virtual*, tanto en textura como en color. También podemos utilizar una herramienta similar para las paredes, techos y suelos del mobiliario.

3.2. Requerimientos

3.2.1. Requerimientos funcionales

A continuación se definen los requisitos necesarios para el funcionamiento de la nueva aplicación en el *tour virtual*.

- Compatibilidad de navegadores y dispositivos. Dado que la plataforma funciona sobre navegadores web, se plantea que el sistema a desarrollar en este proyecto sea totalmente compatible con los navegadores más populares, para que de esta forma, se reduzca el número de casos de error en los usuarios. También hay que tener en cuenta que se ejecutará en los navegadores de teléfonos móviles y será posible utilizar gafas de Realidad Virtual para su funcionamiento.
- Integración en el código fuente. Dado que esta herramienta se adapta a lo preestablecido en los tours virtuales, hay que realizar su implementación teniendo en cuenta el contexto donde se está haciendo y el objetivo es que el nuevo desarrollo sea compatible con el trabajo desarrollado por Floorfy previamente.

3.2.2. Requerimientos no funcionales

A continuación se definen los requisitos no funcionales del sistema, es decir, todos los requisitos que no describen información a guardar, ni funciones a realizar, sino características de funcionamiento:

- Funcionalidad intuitiva. Tenemos en cuenta que el usuario no tiene porqué tener grandes capacidades tecnológicas, y, por lo tanto, se hace un esfuerzo para que sea intuitiva y fácil de usar.
- Eficiencia. La carga de objetos dentro del inmueble puede dar problemas de eficiencia al interactuar con el tour. Este es uno de los posibles obstáculos que nos podremos encontrar en su desarrollo, y hay que tener en cuenta que la aplicación debe de funcionar de forma fluida y amena para el usuario.
- Adaptable a distintas condiciones. Para que esta aplicación logre tener cabida en el servicio de Floorfy, debe de ser capaz de poder integrarse en cualquiera de sus tours virtuales, independientemente de la forma del modelo 3D del hogar, el tipo de estilo que tenga, etcétera.

3.3. Obstáculos y riesgos

Como todo proyecto, nos encontraremos con diferentes obstáculos y riesgos en el desarrollo del mismo. A continuación trataré de identificarlos anticipadamente para minimizar su impacto e ir precavido a la hora de realizar la implementación.

- Calidad y eficiencia. Teniendo en cuenta que la aplicación se ejecuta en un navegador web, hay que buscar el punto en el que se haga la carga de los modelos 3D en buena calidad y sea suficientemente estable como para sostener todo el aplicativo en curso sin ningún tipo de consecuencia en la eficiencia del *tour virtual*.
- Estética. Debido a que la nueva funcionalidad, en el caso de Floorfy, ha de mezclar el entorno del mobiliario realizado a partir de fotografías reales con el mobiliario en 3D, hay que encontrar un punto en común donde no llegue a desentonar la estética del resultado final.
- Adaptable a distintas condiciones. Para que esta aplicación logre tener cabida en el servicio de Floorfy, debe de ser capaz de poder integrarse en cualquiera de sus tours virtuales.

4. Metodología

El desarrollo de este Trabajo de Fin de Grado está realizado junto con un convenio de prácticas en empresa en la compañía Floorfy. En este caso, se trabajará de forma mixta, tanto en casa como en la oficina, debido a las medidas establecidas por el COVID-19.

Se seguirá una metodología ágil con *sprints* de 3 semanas aproximadamente. Se realizarán reuniones con el director e ingeniero 3D del proyecto a final y principio de cada uno de los *sprints*, para así poder comprobar los avances del proyecto, el estado del mismo, las funcionalidades y objetivos que se han logrado hasta el momento y decidir si habrá cambios de cara al siguiente *sprint*.

Esta metodología se reparte en cuatro *sprints*:

- Durante el primer *sprint* se realizan la definición del alcance y planificación del proyecto. Esta fase está relacionada con lo que se hace durante el curso de GEP.
- En el segundo *sprint* se basa en todo lo relacionado con los modelos 3D a añadir al *tour virtual*. Se comenzará a implementar el catálogo de los

modelos 3D, se buscarán los posibles modelos 3D y se añadirán al seleccionar dentro del catálogo. También se implementarán los controles sobre el objeto en la escena para poder colocarlos donde se desee y se pondrán límites suficientes dentro del inmueble. Paralelamente, se redactará en la memoria final todo este proceso.

- Durante el tercer y último *sprint*, se realizará la implementación sobre las texturas y colores a añadir en los objetos y paredes del *tour virtual*. Primero, se creará el catálogo de textura y color. Después se seleccionarán las texturas adecuadas por cada objeto y finalmente se aplicarán sobre los objetos seleccionados. Este proceso se irá describiendo para formar parte de la memoria final del proyecto.

4.1. Herramientas

Para facilitar el seguimiento del proyecto y tener conciencia del trabajo realizado, se utilizará *Jira* [16], un software de gestión de proyectos que se utiliza en ámbitos laborales. Mediante esta herramienta, podré gestionar las tareas a realizar, indicando el inicio y el final de la misma, y pudiendo añadir varias subtareas dentro de cada sprint.

El repositorio donde se ubica el proyecto de la empresa está en la plataforma *BitBucket* [17], que nos permite tener un seguimiento de todos los cambios del proyecto y sirve como copia de seguridad en caso de fallo. Cada vez que se logre alcanzar el éxito de una de las funcionalidades, se subirá al repositorio en la nube, de esta forma será sencillo hacer un seguimiento de los cambios que se van desarrollando durante el proyecto.

Finalmente, se utilizará la herramienta *GanttProject* [18] para generar un diagrama de *Gantt* y ser capaz de gestionar el proyecto durante el tiempo que se plantea.

5. Planificación temporal

Ahora que tenemos conciencia de los objetivos principales del proyecto, pasamos a evaluar la planificación temporal de los mismos con la finalidad de acabar este proyecto en la fecha estimada y planificar semanalmente las tareas a realizar desde el principio hasta el final del proyecto.

El proyecto empieza el 3 de marzo de 2021 y tiene una finalización prevista para mediados de junio de 2021, antes de la lectura de Trabajo de Fin de Grado. La dedicación al trabajo en la empresa es de 27.5h semanales, y la dedicación diaria será de 5.5h, de lunes a viernes, en la oficina de Floofy o en casa,

dependiendo del día dado las condiciones de COVID-19 que hay actualmente.

El equipo necesario para desarrollar el proyecto es el mismo, un portátil proporcionado por la empresa, por lo tanto, las condiciones de trabajo serán las mismas en casa como en la oficina y se podrá desarrollar correctamente desde casa.

También se empleará el debido tiempo a realizar la documentación del proyecto paralelamente.

El resumen con las tareas se puede observar en la tabla 2 y el respectivo diagrama de *Gantt* en la figura 3.

5.1. Descripción de tareas

Paso a detallar individualmente las tareas a realizar en este proyecto. Estas tareas se agrupan en distintos bloques en función de la fase del proyecto en la que se encuentran. Posteriormente, se pueden ver listadas en la tabla 2 todas las tareas con su respectiva duración, dependencias, recursos utilizados y agentes implicados.

GP - Gestión del proyecto

Para planificar, definir y documentar el Trabajo de Fin de Grado que realizo en este proyecto, es fundamental gestionar correctamente el proyecto. En este apartado corresponde toda la información relacionada con la documentación previa al Trabajo de Fin de Grado en la parte académica, gestionado por la asignatura de Gestión de Proyectos. Se estima que en global el grupo de gestión tendrá una duración de 75 horas.

GP.1 - Contextualización y alcance

Para poder tener una visión clara de los objetivos a realizar y ponerle límites a la estructura del proyecto, es necesario acotar el desarrollo y definir los objetivos principales. El tiempo que se ha empleado en esta tarea ha realizado al principio del desarrollo, para así tener claro inicialmente los puntos a desarrollar. La duración ha sido de 25 horas.

GP.2 - Planificación temporal

Con la finalidad de cumplir con todos los objetivos propuestos en la definición del alcance del proyecto, se realiza la planificación temporal, junto con sus recursos y requerimientos asociados a cada una de las tareas. A demás, se exponen posibles riesgos y obstáculos, así como posibles soluciones para solventarlos en el caso de que suceda algún inconveniente en el desarrollo del proyecto. Se estima una duración de 15 horas.

GP.3 - Presupuesto y sostenibilidad

Se estimará el presupuesto para cuantificar el coste del proyecto, para ello, se realizarán estudios sobre el coste de personal en la empresa, además, se cuantificarán los costes genéricos y los gastos debidos a posibles imprevistos. En esta fase también se analizará el impacto medioambiental, económico y social del proyecto realizado en este Trabajo de Fin de Grado. Se tienen en cuenta las fases de planificación y desarrollo. Se estima una dedicación de 15 horas.

GP.4 - Documento final

Finalmente, se agrupan los puntos descritos previamente en un documento final a entregar. En esta tarea, se utiliza el *feedback* recibido por parte del profesor asignado de Gestión del Proyecto y se refuerzan los puntos más débiles. Su duración es de 20 horas.

TP - Trabajo previo

A continuación se especificarán las tareas previas a realizar en este proyecto, es decir, las tareas que se centran en la preparación y estudio previo del proyecto. Dado que esta fase transcurre antes de el desarrollo del proyecto, se puede realizar de forma paralela a la mayoría de tareas de gestión del proyecto. Se estima una duración total de 95 horas

TP.1 - Reuniones

Antes de realizar el proyecto, se han realizado varias reuniones junto con el director del proyecto y el ingeniero 3D de la empresa para consolidar las bases del proyecto y proponer los objetivos apropiados para este Trabajo de Fin de Grado. También se acumulan todas las reuniones que se tienen en cuenta para

la comprobación de que se están cumpliendo los objetivos a lo largo de este proyecto. Esta fase tiene una duración de 40 horas aproximadamente.

TP.2 - Análisis de la competencia

Dado que existen varias empresas que se dedican a lo mismo que Floorfy, y algunas de ellas utilizan funcionalidades parecidas a la desarrollada en este proyecto, considero importante estudiarlas previamente con el fin de analizar lo que ya está hecho y determinar en qué se puede innovar, conectando lo que quiere Floorfy con lo que es posible presentar en este proyecto como TFG. Este proceso tiene una duración de 20 horas.

TP.3 - Preparación del entorno de trabajo

Se ha tenido que realizar el *onboarding* a la empresa junto con la adaptación del software que utiliza Floorfy y todo su código fuente. Se ha utilizado el portátil proporcionado por la compañía como equipo para desarrollar esta nueva herramienta. Este proceso ha sido necesario para poder desarrollar la nueva funcionalidad en su entorno, y tiene una duración aproximada de 15 horas.

TP.4 - Aprendizaje

Dado que nunca antes había utilizado las herramientas que utiliza la empresa para desarrollar este tipo de software específico, también ha habido un desarrollo de aprendizaje que ha transcurrido a partir de la fase de *onboarding* de este proyecto en concreto. De hecho, siempre se está en constante aprendizaje durante el desarrollo de esta funcionalidad, pero a esta fase específica se refiere al aprendizaje en la parte de trabajo previo. Dura unas 20 horas.

M3D - Modelo 3D

En esta primera parte del proyecto, se desarrolla todo lo relacionado con la herramienta que añade modelos 3D a la escena. Se estima una duración de 200 horas, y se define en 5 subtareas:

M3D.1 - Desarrollo del catálogo

Para facilitar el uso de la herramienta, se desplegará un catálogo con distintas secciones con las respectivas previsualizaciones de los objetos 3D a añadir a la

escena. Esta tarea tiene una duración estimada de 50 horas.

M3D.2 - Búsqueda y selección de modelos 3D

Se tendrán que buscar los modelos adecuados (a partir de algunos proporcionados por la empresa) que encajen con la estética y sean capaces de funcionar en el entorno del *tour virtual*. Dura unas 15 horas.

M3D.3 - Añadir objetos 3D al escenario

Una vez tengamos el catálogo implementado se tendrán que usar los objetos 3D seleccionados para que el usuario pueda añadirlos al escenario 3D. En esta fase se implementa la funcionalidad de añadir el objeto en cuestión al *tour virtual*. Esta tarea dura 30 horas.

M3D.4 - Controles de transformaciones geométricas

Una vez añadido el objeto al escenario 3D, el usuario podrá realizar cambios en el modelo, concretamente podrá modificar su escala, posición y ángulo dentro del *tour virtual*, para así darle la posibilidad de poner el objeto donde y como desee. El desarrollo de esta mecánica durará 55 horas.

M3D.5 - Definir los límites en la propiedad

También hay que definir los límites para que no sea posible poner un objeto fuera de la casa, o entre dos pisos distintos. Para ello, necesitamos observar las distintas posibilidades y no permitir que se hagan posibles. La implementación de este desarrollo durará 50 horas.

MAT - Materiales

El siguiente objetivo a implementar es la posibilidad de que el usuario pueda cambiar el color y textura de los objetos añadidos, así como las paredes, techos y suelos del inmueble. Este proceso durará 115 horas aproximadamente y se divide en las siguientes 4 fases:

MAT.1 - Desarrollo del catálogo

Así como en la opción de los objetos teníamos un catalogo, aquí también. Habrá un catálogo específico por cada objeto, así como uno para suelos, otro para paredes y techos. En el catálogo se podrá previsualizar la textura y los colores a aplicar. Esta fase dura 50 horas.

MAT.2 - Búsqueda y selección de texturas

Así como los objetos 3D, también hay que hacer una búsqueda y selección de las texturas a aplicar en los objetos como en las paredes y techos. Este proceso durará 15 horas aproximadamente.

MAT.3 - Texturas

Como ya he comentado, en el catálogo se podrán escoger las texturas y los colores. En esta fase, se implementará la selección entre unas cuantas texturas y se aplicará al objeto o paredes de la propiedad, en función del objeto seleccionado. La tarea dura 20 horas.

MAT.4 - Color

La siguiente parte a desarrollar es el color. Una vez se haya escogido la textura, se le podrá cambiar el color mediante una paleta de colores. En esta fase se crea la paleta de colores y se implementa la funcionalidad para que se refleje en el objeto seleccionado. Se estima una duración de 30 horas.

FIN - Finalización

Para finalizar el proyecto, se autoevalúa el proyecto que se ha realizado y se decide si hay posibles mejoras, a demás de redactar la memoria para tener listo el Trabajo de Fin de Grado, y se enfoca de cara a la presentación del proyecto en frente del tribunal. El proceso tiene una durabilidad estimada de 80 horas y se definen las siguientes tres fases:

FIN.2 - Retoques finales

En esta fase se autoevaluará el proyecto realizado, tanto por mí mismo como por parte de la empresa, y se decidirá si es apropiado realizar unos últimos retoques al resultado final del proyecto. La duración estimada es de 20 horas.

FIN.3 - Documentación final

A la hora de presentar este proyecto como Trabajo de Fin de Grado, se recurre a un documento final, por lo tanto, a lo largo del transcurso del desarrollo de este proyecto se irá documentando la memoria por de las distintas fases. La documentación final se realizará una vez acabado el proyecto en la empresa y contendrá todas las fases descritas previamente. La duración estimada es de 50 horas.

FIN.4 - Presentación

Se presentará este Trabajo de Fin de Grado mediante una presentación, una vez finalizada la documentación. De esta forma, el tribunal podrá evaluar el trabajo realizado en este proyecto junto con la memoria. Se preparará el material de soporte para la presentación, así como el guión, y se realizarán ensayos para poder estar preparado a la hora de realizar la lectura. La duración de este proceso se estima que dure unas 20 horas.

5.2. Recursos

5.2.1. Recursos humanos

En esta sección se encuentran los distintos recursos humanos en el desarrollo. A continuación se enumeran los 3 distintos roles, se pueden encontrar en la tabla 2 en el apartado “Roles”.

- Director del proyecto. Se encarga de la planificación del proyecto, tiene que decidir entre las diversas posibles opciones que puedan surgir durante el desarrollo del proyecto y guiar a los empleados para que se pueda realizar de forma correcta.
- Investigador. El investigador se dedica a buscar y encontrar diversos recursos para que el programador pueda llevar a cabo su desarrollo.
- Programador. Se dedica a implementar el sistema de la nueva funcionalidad mediante código de programación

5.2.2. Recursos materiales

A continuación se exponen los recursos materiales necesarios para la realización de este proyecto. En la tabla 2 se hace referencia en la columna “Recursos”.

- Portátil. Ordenador portátil proporcionado por la empresa. Es la herramienta principal de trabajo en este proyecto, con él se realizan las reuniones, se programa y se documenta todo el proyecto.
- overleaf [19]. Procesador de textos online para crear la documentación necesaria en L^AT_EX[20].
- PhpStorm [21]. Entorno de trabajo en el que se desarrolla toda la aplicación utilizando el lenguaje de programación *JavaScript*.
- GanttProject [18]. Herramienta para la creación de diagramas de Gantt.
- Fuentes de internet. Para encontrar distintos modelos 3D o texturas, se han utilizado diversas fuentes de internet en distintos portales web.

ID	Tarea	Duración (horas)	Dependencias	Recursos	Roles
GP	Gestión del proyecto	75			
GP.1	Contextualización y alcance	25	TP.2	Portátil, overleaf	DP
GP.2	Planificación temporal	15	GP.1	Portátil, overleaf, GanttProject	DP
GP.3	Presupuesto y sostenibilidad	15	GP.2	Portátil, overleaf	DP
GP.4	Documento final	20	GP.1-3	Portátil, overleaf	DP
TP	Trabajo previo	95			
TP.1	Reuniones	40		Portátil	DP, I, P
TP.2	Análisis de la competencia	20		Portátil	I
TP.3	Preparación del entorno de trabajo	15		Portátil, PhpStorm	I, P
TP.4	Aprendizaje	20	TP.3	Portátil, PhpStorm	I, P
M3D	Modelo 3D	200			
M3D.1	Desarrollo del catálogo	50	TP.4	Portátil, PhpStorm	P
M3D.2	Búsqueda y selección de modelos 3D	15	M3D.1	Portátil, fuentes de internet	I
M3D.3	Añadir objetos 3D al escenario	30	M3D.2	Portátil, PhpStorm, modelos 3D	P
M3D.4	Controles de transformaciones geométricas	55	M3D.3	Portátil, PhpStorm	P
M3D.5	Definir los límites en la propiedad	50	M3D.4	Portátil, PhpStorm	P
MAT	Materiales	115			
MAT.1	Desarrollo del catálogo	50		Portátil, PhpStorm	P
MAT.2	Búsqueda y selección de texturas	15	MAT.1	Portátil, fuentes de internet	I
MAT.3	Texturas	20	MAT.2	Portátil, PhpStorm, texturas	P
MAT.4	Color	30	MAT.3	Portátil, PhpStorm	P
FIN	Finalización	100			
FIN.1	Retoques finales	30		Portátil, PhpStorm	DP, I, P
FIN.2	Documentación final	50	FIN.1	Portátil, overleaf	DP
FIN.3	Presentación	20	FIN.2	Portátil	DP
	Total	585			

Tabla 2: Tabla de tareas con la duración, dependencias, recursos y agentes implicados, DP: Director del proyecto, I: Investigador, P: Programador, fuente: Elaboración propia

5.3. Gestión del riesgo

Es importante la prevención de los riesgos y obstáculos que pueden surgir a lo largo del desarrollo, para así evitar que su impacto sea mayor. Durante este proyecto, es probable que se tengan que cambiar algunos de los objetivos iniciales con tal de obtener un resultado mejor que lo inicialmente planteado. A continuación se describen los posibles obstáculos y riesgos ya descritos en el apartado de Obstáculos y riesgos y los planes alternativos para solventarlos.

- Dificultades diversas. Dado que el proyecto tiene varias facetas, puedo verme estancado en alguna de ellas y así ralentizar el transcurso del proyecto. Es por este motivo, se ha decidido dejar un espacio de “Retoques finales” para lograr todos los objetivos planteados inicialmente. No debería afectar a la duración del proyecto debido a que precisamente se ha dejado este margen con tiempo suficiente.
- Lentitud en la carga. Teniendo en cuenta que el proyecto tiene como base el *tour virtual* previamente establecido, quizás surgen algunos problemas de rendimiento a la hora de cargar, a demás de la propiedad entera, los diversos objetos tridimensionales añadidos a la escena. Es por esto que, para este problema se sugiere cargar inicialmente los objetos a añadir por el usuario en una calidad inferior, para así acelerar el proceso de carga, y una vez esté establecido, cargar el mismo objeto en una calidad superior. De esta forma la carga inicial será mucho más rápida. Este obstáculo podría añadir una duración adicional al proyecto de 20 horas.
- Modelos 3D o texturas insuficientes. Puede que no se obtengan con facilidad los modelos 3D o las texturas a aplicar. De ser así, tendríamos que desarrollar e implementar toda esta nueva funcionalidad pero con menos variedad en los objetos y en las texturas. Este riesgo tendrá un impacto mínimo en la duración del proyecto.

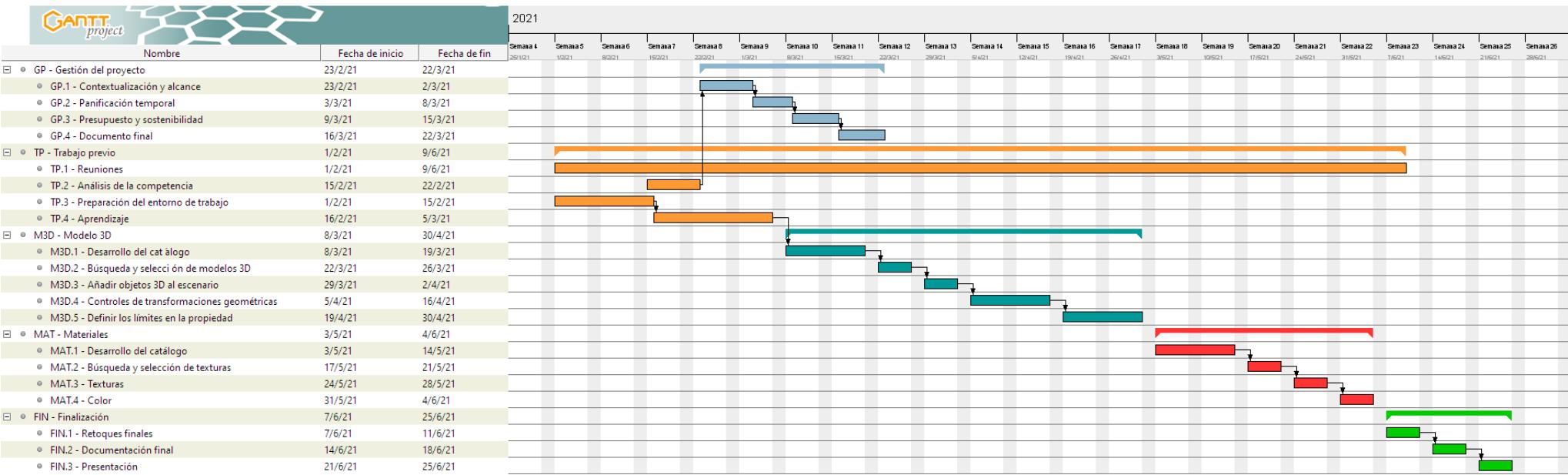


Figura 3: Diagrama de Gantt, fuente: Elaboración propia.

5.4. Cambios en la planificación inicial

Durante el proyecto, se ha tratado de realizar la planificación inicial de forma estricta. Pero han surgido algunos inconvenientes en algunas de las tareas que han hecho modificar esta planificación. Los más contundentes han sido:

- Debido a los cambios en las ideas sobre el desarrollo del proyecto, se han ido añadiendo objetos y modificándolos mediante herramientas de modificación de objetos (como *Blender*) para que encaje perfectamente en el catálogo. La tarea de seleccionar modelos 3D se ha ido haciendo durante prácticamente toda la planificación del proyecto. Esta modificación en la planificación inicial se ha hecho por dos razones: por el hecho de añadir una nueva sección *Item* que son objetos que se pueden poner encima de otros objetos (como ordenadores, cerámica, pequeños electrodomésticos, etc.) y por añadir la categoría de objetos que se ponen en pared y suelo a la vez (objetos *wallFloor*).
- Definir los límites dentro de la propiedad se ha elaborado antes de lo previsto, por el motivo de que se necesita saber donde se puede colocar el objeto una vez cargado.
- Dado que las tareas de Textura y Color se han realizado en menos tiempo del esperado, tengo más tiempo para retoques finales.

La planificación y su Gantt queda, por lo tanto, de la siguiente forma:

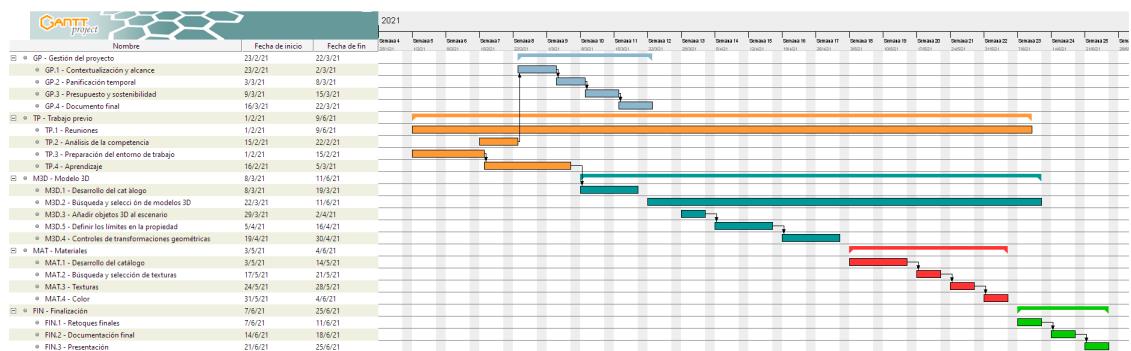


Figura 4: Diagrama de Gantt modificado. Elaboración propia.

6. Gestión económica

Dado que ahora sabemos las tareas a realizar y su planificación temporal en el proyecto, pasamos a estimar los costes económicos que supone el desarrollo de esta nueva herramienta. Se dividen los diferentes tipos de costes en personal, espacio de trabajo y herramientas utilizadas. Además, se estima el coste de los posibles imprevistos que puedan surgir durante el desarrollo, realizando un plan de contingencia, una partida de imprevistos y cómo solventarlos.

6.1. Presupuesto

6.1.1. Coste del personal

En el anterior apartado sobre la planificación temporal se ha especificado el rol que se asume en cada una de las tareas del proyecto. En la tabla 3 se puede observar el coste por hora de cada uno de los 4 roles implicados en este desarrollo: Director del proyecto, investigador y programador, mencionados previamente en el punto de Recursos humanos. Los datos han sido obtenidos en la página *glassdoor* [22], y se ha utilizado la herramienta de *neuvoo* [23] para obtener el salario por hora y así facilitar el cálculo.

En la tabla 4 se puede observar las partidas por tarea del proyecto a partir de los costes de personal de la tabla 3. Se añade el coste que supone la seguridad social mediante la aproximación de multiplicar por 1,3 cada coste, esto dará un resultado más realista a la hora de estimar el coste total del personal. En el cómputo global, la estimación del coste de personal en este proyecto es de 15464€.

Rol	Coste (€/hora)
Director del proyecto	24
Investigador	15
Programador	13

Tabla 3: Tabla con los costes de personal a partir de glassdoor y neuvoo,
fuente: Elaboración propia

ID	Tarea	Duración (horas)	Roles	Coste	Coste SS
GP	Gestión del proyecto	75		1800	2340
TP	Trabajo previo	95		2800	3640
GP.1	Contextualización y alcance	25	DP	600	780
GP.2	Planificación temporal	15	DP	360	468
GP.3	Presupuesto y sostenibilidad	15	DP	360	468
GP.4	Documento final	20	DP	480	624
M	Modelo 3D	200		2630	3419
M3D.1	Desarrollo del catálogo	50	P	650	845
M3D.2	Búsqueda y selección de modelos 3D	15	I	225	292.5
M3D.3	Añadir objetos 3D al escenario	30	P	390	507
M3D.4	Controles de transformaciones geométricas	55	P	715	929.5
M3D.5	Definir los límites en la propiedad	50	P	650	845
MAT	Materiales	115		1425	1852.5
MAT.1	Desarrollo del catálogo	50	P	550	715
MAT.2	Búsqueda y selección de texturas	15	I	225	292.5
MAT.3	Texturas	20	P	260	338
MAT.4	Color	30	P	390	507
FIN	Finalización	100		3240	4212
FIN.1	Retoques finales	30	DP, I, P	1560	2028
FIN.2	Documentación final	50	DP	1200	1560
FIN.3	Presentación	20	DP	480	624
	Total	585		11895	15464

Tabla 4: Tabla con la estimación del coste de personal, fuente: Elaboración propia

6.1.2. Costes genéricos

En el apartado anterior sobre la planificación temporal se ha explicado que el trabajo se desarrolla de lunes a viernes en las oficinas de Floorfy o desde casa. Tendremos en cuenta entonces, el pago del alquiler de las oficinas, incluyendo electricidad, internet y agua, mediante una estimación media sobre el precio de este tipo de espacios de *coworking* con equipamiento estándar (baño, escritorios, ordenador)[24]. Dado que el proyecto se desarrolla durante unos 5 meses aproximadamente, y el coste medio estimado es de unos 250, asumiremos que el precio total del espacio de trabajo es de unos 1250€.

Para realizar el cálculo de los costes genéricos se ha tenido en cuenta la planificación mencionada previamente, juntamente con la planificación estimada mediante el diagrama de *Gantt* 3.

- *overleaf*. Es una herramienta de código abierto, por lo tanto, no tiene coste. [22]

- *GanttProject*. Es una herramienta gratuita, sin coste. [18]
- *PhpStorm*. Tiene un coste de 199€/año. Dado que el proyecto transcurre durante 5 meses, asumir un único pago de este servicio será suficiente. [21]
- Portátil. El portátil que se ha utilizado para desarrollar el proyecto en su totalidad tiene un coste de 1500€.

Dado que el único dispositivo de hardware que se emplea durante el proyecto es el portátil, se debe estimar su amortización. Se estima una vida útil de 4 años para el dispositivo, teniendo en cuenta durante un año hay 220 días hábiles y 8 horas laborables durante una jornada laboral, podemos deducir su amortización empleando la fórmula

$$\frac{\text{Coste}}{\text{Vida} * 220 * 8},$$

donde Coste es el coste del dispositivo y Vida es la vida útil del dispositivo. Teniendo en cuenta que el portátil se utiliza durante las 585 horas del proyecto, obtenemos una amortización de 125€.

6.1.3. Coste de contingencia

Dado que durante el transcurso del proyecto pueden surgir imprevistos, es importante añadir un sobrecoste al presupuesto total del proyecto para cubrir posibles imprevistos y reducir su impacto económico. En este caso, dado que el proyecto a desarrollar no supone demasiado riesgo, se ha decidido fijar un valor de 7 % en el sobrecoste. A continuación se detalla la contingencia total del proyecto reflejada en la tabla 5.

Presupuesto	Coste (€)	Contingencia (€)
Espacio de trabajo	1250	86
Software	199	14
Hardware	125	9
Personal	15464	1082
Total	17038	1193

Tabla 5: Tabla de contingencia del 7 % del proyecto, fuente: Elaboración propia

6.1.4. Coste de imprevistos

Antes de finalizar con la estimación del coste total del proyecto, se realiza el cálculo del coste de los imprevistos que puedan surgir durante el desarrollo, siguiendo los puntos mencionados en el apartado sobre planificación temporal.

En la tabla 6 se pueden observar los costes adicionales de cada imprevisto mencionado. Las revisiones sobre el cálculo de las desviaciones se realizarán una vez acabada cada tarea, en la reunión que se hace al final de cada una de las tareas, y los directores del proyecto decidirán si es viable realizar los cambios oportunos.

- Dificultades diversas. Si se necesitase más tiempo de lo previsto por algún inconveniente durante el desarrollo, se asume que se necesitarían 25 horas a cargo del rol de programador y unas 10 horas por parte del director del proyecto adicionales. Esto supone un coste de 565€. Se estima un riesgo en este imprevisto de 15 % debido a que las tareas del proyecto no son demasiado complicadas individualmente, pero se puede complicar al juntar varias fases del desarrollo.
- Lentitud en la carga. Asumiendo la posibilidad de que este imprevisto ocurra, se ha trazado un plan de ruta a seguir. El plan alternativo supondrá unas 15 horas por parte del rol de investigador, otras 20 por parte del programador y 10 horas del director del proyecto, con lo cual, un coste adicional de 500€. Este caso se considera que tiene un riesgo del 20 % dado que puede suponer un problema encontrar modelos 3D tan específicos para nuestras tareas.
- Recursos insuficientes. En este imprevisto se asimila la posibilidad de no tener demasiado éxito en la investigación de recursos. En este caso el único rol que participa es el de investigador, suponiendo unas 20 horas adicionales, que suponen 300€ más al presupuesto del proyecto. Este inconveniente tiene un riesgo del 15 % por el mismo motivo que el imprevisto anterior.
- Fallo en el dispositivo. También se considera la posibilidad de que nuestra herramienta de trabajo principal falle, en ese caso, se deberá reemplazar por otro nuevo. Para el caso del portátil de la empresa, se estima un riesgo de 5 %.

Imprevisto	Coste (€)	Riesgo (%)	Coste total (€)
Dificultades diversas	565	15	85
Lentitud en la carga	500	20	100
Recursos insuficientes	300	15	45
Fallo en el dispositivo	1500	5	75
Total	2865		305

Tabla 6: Tabla del coste de imprevistos, fuente: Elaboración propia

6.1.5. Coste total

Por último, agruparemos todos los costes calculados previamente para así obtener el coste total del proyecto. En la tabla 7 se presenta el cálculo para obtener el coste total.

Tipo	Coste (€)
Espacio de trabajo	1200
Software	199
Hardware	1500
Personal	15464
Contingencia	1193
Imprevistos	305
Total	18661

Tabla 7: Tabla del presupuesto total del proyecto, fuente: Elaboración propia

El presupuesto estimado del proyecto es de 18661€. En el caso de que no se produzcan imprevistos durante el desarrollo o no se utilicen los planes de contingencia descritos, se aprovechará el presupuesto (1498€) para continuar el proyecto de cara al futuro, y acelerar el proceso de automatización de la herramienta.

6.2. Control de gestión

La gran mayoría de proyectos no finalizan con el presupuesto estimado inicialmente, eso sucede por la aparición de nuevos obstáculos y riesgos diferentes que no han sido previstos. Una vez estimado el presupuesto de nuestro proyecto, pasamos a mostrar las fórmulas que nos servirán para calcular las desviaciones de los valores reales con los teóricos previstos.

- Desviación de horas consumidas por tarea.

$$(Horas estimadas - Horas reales) * Coste estimado$$

- Desviación de costes según las horas consumidas por tarea.

$$(Horas estimadas - Horas reales) * Coste real$$

- Desviación de coste en recursos humanos por tarea.

$$(Coste estimado - Coste real) * Horas reales$$

- Desviación total de horas.

$$Horas estimadas - Horas reales$$

- Desviación total de recursos.

$$Coste\ estimado - Coste\ real$$

6.3. Cambios en el presupuesto inicial

De acuerdo con esta modificación en la planificación, se modifica el coste, incrementando el pago en el rol de investigador. Previamente, en esta tarea (Búsqueda y selección de modelos 3D) duraba 15 horas, mientras que ahora se estiman unas 45. Por lo tanto, se añaden 30 horas más (pagadas como 15€/h), y el resultado del supuesto coste final asciende a 15914€ (Antes 15464€).

Las otras modificaciones en la planificación no tienen consecuencias directas con el coste del proyecto.

7. Sostenibilidad

Al realizar un proyecto de esta magnitud, es importante ser conscientes de el impacto en las diversas dimensiones de la sostenibilidad que existen. A continuación se realiza una autoevaluación sobre la sostenibilidad del proyecto, a demás de análisis sobre la sostenibilidad en las siguientes dimensiones: Económica, ambiental y social.

7.1. Autoevaluación

Durante mi paso por el grado en ingeniería informática realizado en la Facultad de Ingeniería de Barcelona, mi conocimiento sobre la sostenibilidad se ha ido enriqueciendo, dado que todos los alumnos hemos sido expuestos a diferentes reuniones y conferencias relacionadas con el tema, a demás de algunos trabajos en determinadas asignaturas implantadas por la universidad. En la realización de este TFG, me he podido dar cuenta de la importancia que tiene ser conscientes de ello, dado que he podido trabajar en un proyecto de una empresa real.

Considero que la dimensión económica es la que siempre se tiene en cuenta a la hora de realizar un proyecto informático, y quizás la social y ambiental no se tienen tan en cuenta, pero son igual o más importantes que la primera. La social y ambiental miran por un bien común, mientras que la económica solo vela por la misma entidad, quien produce el proyecto. Es por esta razón que considero que deberían de ser las tres igual de importantes, y la realización de este proyecto me ha ayudado a tener más conocimiento sobre el tema. Personalmente, este

proyecto de Trabajo de final de Grado me aportará los conocimientos necesarios para ser aún más consciente de el impacto en la sostenibilidad.

7.2. Dimensión económica

Teniendo en cuenta las posibilidades que tiene la herramienta desarrollada durante este proyecto de cara al futuro (un posible amueblado automático en las propiedades de la empresa), considero que el primer paso de este proyecto inicial se ajusta bien a su presupuesto estimado dado el beneficio que supondrá.

A pesar de que las soluciones existentes ya realizan este trabajo, la empresa no tenía una herramienta parecida, y se beneficiará del resultado, para así ser una mejor competencia entre las distintas empresas que ofrecen tours virtuales.

7.3. Dimensión ambiental

El proyecto realizado, por suerte, no tiene un impacto demasiado grave a nivel ambiental. Únicamente tenemos que tener en cuenta la fabricación del hardware utilizado, el cual impacta negativamente en el medio ambiente. Dado que para el proyecto se utiliza un solo ordenador, el impacto no ha sido demasiado elevado.

La empresa en la que se hace el proyecto en sí, también ayuda indirectamente a reducir el impacto medioambiental, dado que se centra en reducir al consumidor los desplazamientos que se suelen hacer a la hora de gestionar la evaluación de una vivienda. En el caso de el proyecto, especular cómo se vería un inmueble con una determinada decoración se puede hacer mucho más rápido y más sencillo, evitando la posible contaminación que supondría realizar los desplazamientos para comprobar el estilo de los distintos hogares.

7.4. Dimensión social

A nivel individual, desde que empecé a estudiar informática he tenido gran interés en los gráficos por computador, y considero que la realización de este proyecto es una excelente oportunidad para introducirme en el ámbito laboral y para profundizar mis conocimientos sobre el campo.

Mirando por el beneficio social, tal y como se explicaba en la dimensión ambiental, esta nueva herramienta permite a los clientes evitar desplazamientos para comprobar diferentes pisos, y así poder amueblarlos al gusto sin tener que moverse, de una forma cómoda, rápida y sencilla. A demás de facilitar a los

editores de la empresa Floorfy a desarrollar su trabajo de una manera más eficaz.

8. Herramientas

A continuación se hará una definición de las herramientas utilizadas que fueron necesarias durante la realización del proyecto.

8.1. *Three js*

Tal y como se ha mencionado en el anterior punto 2.2, *three js* será la principal herramienta utilizada para este proyecto. Esta librería facilita la manipulación del entorno en 3D. Cabe añadir que todo el código desarrollado por la empresa a lo largo de su vida ha sido realizado mediante esta herramienta, y para añadir todo el proyecto al *tour virtual* era lo más viable.

Three js es una librería liviana escrita en *JavaScript* para generar y animar contenido gráfico en 3D en un navegador web. Esta librería dibuja todo el contenido 3D mediante la API *WebGL*.

WebGL es una tecnología diseñada para trabajar directamente con la GPU (unidad de procesamiento gráfico). Esta API suele resultar bastante compleja al uso, es por eso que se han creado otros estándares para facilitar la creación de contenido 3D en navegadores, como es el caso de *three js*, la cual es la librería más popular en su función dentro de la comunidad dado su bajo nivel de complejidad respecto *WebGL*.

WebGL es un sistema de muy bajo nivel que dibuja puntos, líneas y triángulos a través de la GPU basados en el código proporcionado. El programador proporciona ese código mediante dos funciones que se denominan *vertex shader* y *fragment shader*, y cada una de ellas está escrita en un lenguaje tipo C/C++ muy estrictamente tipado ¹ llamado *GLSL* (*OpenGL Shading Language*).

El *vertex shader* se encarga de manipular los atributos de los vértices que forman parte de las esquinas de nuestros polígonos mostrados por pantalla, y *fragment shader*, por otra parte, se encarga de cómo se ven los píxeles entre los vértices. Estos píxeles son interpolados entre los vértices definidos siguiendo unas reglas especificadas.

Durante la explicación del desarrollo se irán explicando los distintos métodos

¹Un lenguaje de programación es fuertemente tipado si no se permiten violaciones de los tipos de datos, es decir, dado el valor de una variable de un tipo concreto, no se puede usar como si fuera de otro tipo distinto a menos que se haga una conversión.

de esta librería que se han empleado en el proyecto.

8.2. HTML y CSS

Para que el usuario pueda interactuar de forma correcta con la nueva aplicación, se utiliza un catálogo hecho mediante HTML y CSS para que se pueda manipular de manera intuitiva y eficiente con las diversas funcionalidades del aplicativo desarrollado en este proyecto.

HTML (HyperText Markup Language, en español “Lenguaje de Marcado de HiperTexto”) es un lenguaje con el que se define el contenido de las páginas web. Básicamente se trata de un conjunto de etiquetas que sirven para definir el texto y otros elementos que compondrán una página web, como imágenes, listas, vídeos, etc. En este caso, se utilizará para la elaboración de los botones con diversas funcionalidades, miniaturas de los objetos y texturas para poder identificarlos, controles, etcétera. Los elementos que forman el catálogo se explicarán de forma más concreta en el punto 9.1.

Para conseguir unos resultados más profesionales y con una mejor presentación se requiere de una capacidad para el diseño y artísticas que podremos moldear mediante CSS (Cascading Style Sheets), en español “Hojas de estilo en cascada”, es un lenguaje de diseño gráfico para definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado. Es muy usado para establecer el diseño visual de los documentos web, e interfaces de usuario escritas en HTML, como es el caso del catálogo desarrollado en el proyecto.

Toda esta parte ha sido diseñada e implementada por mí, pero de cara al futuro, y con la finalidad de dar un aspecto más profesional, la encargada en la empresa de hacer el diseño gráfico modificará mis diseños iniciales para que el proyecto final luzca mejor y con una estética más acorde y profesional dado que carezco de formación suficiente sobre el diseño. Estos cambios son a futuro y no se verán involucrados en el TFG.

8.3. Obtención de recursos

Varios de los objetos y texturas incorporados en este proyecto fueron facilitados por la empresa Floorfy, dado que ya disponían de un conjunto de objetos 3D de un proyecto que no llegó a finalizar hace varios años atrás.

Pero no todos los objetos han sido proporcionados por la empresa. Dada la necesidad de poner más objetos y texturas disponibles en el catálogo y de filtrar los más óptimos, se han incorporado al conjunto de modelos e imágenes varios elementos encontrados en páginas como:

- Polantis [25]. Polantis es una empresa francesa de TI fundada por arquitectos en 2008 con la dedicación de acceder a la creación de contenido 3D y BIM. De aquí se han extraído la mayoría de objetos 3D y algunas texturas.
- Turbosquid [26]. Varios diseñadores de todas partes del mundo comparten sus proyectos en Turbosquid y se pueden adquirir de forma gratuita o de pago.
- Free3D [27]. Es el lugar para compartir y adquirir modelos 3D mediante varios miles de usuarios que utilizan esta herramienta.
- CGTrader [28]. Es un mercado de modelos 3D para proyectos VR / AR y CG, y una comunidad de diseñadores 3D profesionales. De esta página se han substraído sobretodo objetos como muebles de salón e ítems.
- Textures.com [29]. Es un sitio web que ofrece imágenes digitales de todo tipo de materiales. De esta página se han adquirido la mayoría de texturas que hay en el catálogo.

Estas páginas tienen un catálogo de modelos 3D y texturas gratuitos y de pago, de los cuales he tenido que investigar cuáles son más acordes, en función de su estética, número de polígonos (para hacer una carga más eficiente), su disposición en el catálogo (por ejemplo, si hacen falta más variedad de sillas, o hay suficientes sofás en el catálogo...) y el formato en el que viene dicho objeto, ya que en el proyecto se ha decidido utilizar modelos en formatos .OBJ y .MTL. (Se explica qué son este tipo de archivos en el punto 9.2.1).

El proceso de obtención de recursos está relacionado con las tareas de Búsqueda de modelos 3D y Búsqueda de texturas, que se basan en la parte del rol de investigador al buscar modelos y texturas acordes. Están explicadas en los puntos M3D.2 - Búsqueda y selección de modelos 3D y MAT.2 - Búsqueda y selección de texturas.

8.4. Herramientas de trabajo

En esta sección se hará una breve explicación sobre las herramientas que están relacionadas con la empresa Floorfy para le mantenimiento del código y la comprobación de que se están realizando las tareas de forma correcta.

8.4.1. Jira

Para comprobar y asegurar de que todos los puntos del proyecto se cumplen, Floorfy ha facilitado herramientas de trabajo como *Jira*, donde se pueden especificar las tareas a seguir por sprint y facilitar un seguimiento del proyecto.

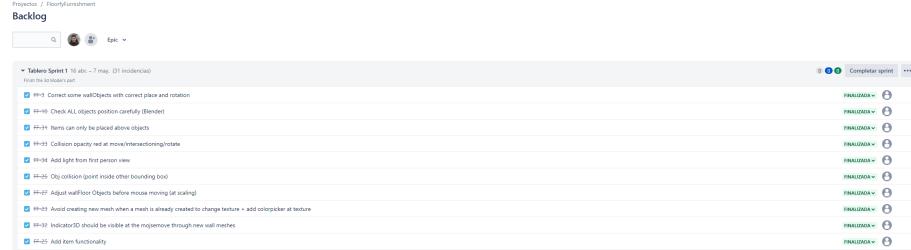


Figura 5: Captura del *backlog* de mi *Jira* personal con tareas realizadas. Elaboración propia.

En esta herramienta, se han tenido que añadir cada una de las tareas en el *backlog* y actualizarlo cada vez que se complete de forma satisfactoria una tarea o bien se ha tenido que cambiar el propósito de una de las tareas asignadas.

A cada tarea asignada al *backlog*, se le ha ido asignando a una función, primero *TO DO*, que representa a aquellas tareas que están por hacer, luego *WORK IN PROGRESS*, las cuales se están ejecutando y por último *DONE*, cuando la tarea ha sido finalizada con éxito.

8.4.2. BitBucket

La empresa proporciona el repositorio donde se encuentran el contenido del proyecto junto con su versión completa, que se encuentra en la plataforma BitBucket. Desde ahí se irá actualizando el código progresivamente.

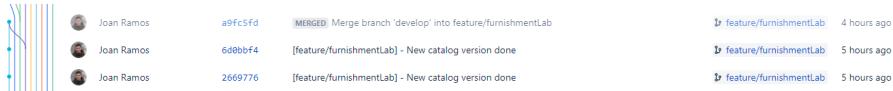


Figura 6: Captura del historial de *commits* y código actualizado en el proyecto. Elaboración propia.

El trabajo que se ha ido añadiendo de otros desarrolladores en el producto final de la empresa también se ha tenido que ir actualizando y mezclando con el proyecto dado que en un futuro se planea unir la parte del proyecto realizado como TFG junto con el producto final del *tour virtual*. BitBucket ha permitido que todo el código de mi rama se mantenga actualizada correctamente de forma segura.

Junto con *Jira*, estas dos aplicaciones se han ido utilizando mediante las reuniones que se han tenido con los directores del proyecto sobre la metodología a aplicar en el desarrollo.

9. Implementación

A continuación se explicarán los diferentes puntos de la implementación que se ha desarrollado en el proyecto, haciendo referencia a las tareas especificadas previamente, explicando aquellos métodos de *three js* que se consideran importantes y describiendo los pasos de los algoritmos que se han implementado.

Se divide en tres piezas fundamentales: el Catálogo, la parte de Amueblado que involucra todo lo relacionado con los modelos 3D y la parte de Pintar paredes, que explica todo el proceso de cambiar la apariencia de las paredes y suelos del inmueble.

9.1. Catálogo

En este apartado se explican las componentes que tiene el catálogo de la herramienta. Se utilizará a modo de “manual de uso” y se expondrá de forma liviana las características que tiene, las cuales se explicarán con más profundidad en los futuros apartados.

Para empezar, se nos abrirá el catálogo haciendo click en el siguiente ícono, ubicado en la parte inferior izquierda de la pantalla junto con otros botones del *tour virtual*.



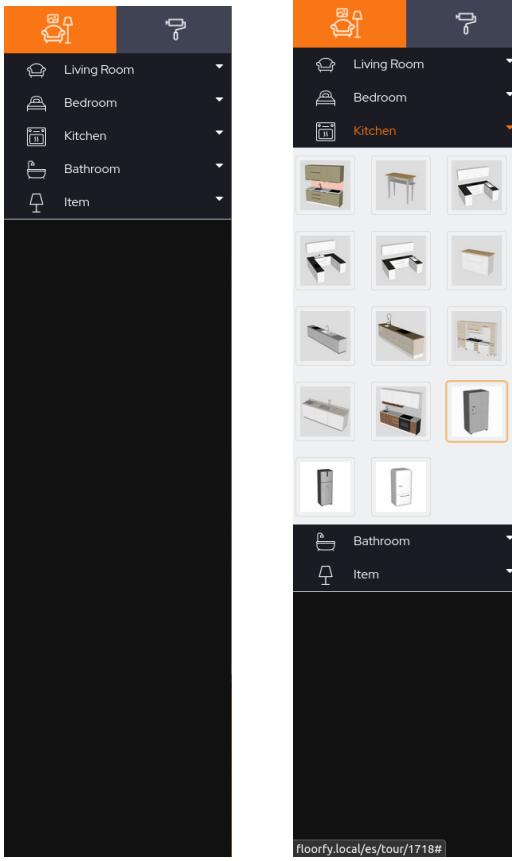
Figura 7: Captura del ícono que abre el catálogo en el proyecto. Elaboración propia.

El catálogo es la herramienta que facilita la interactividad entre el usuario y las funcionalidades del proyecto. Ha sido creado en HTML y CSS y se compone, principalmente, de dos secciones:

9.1.1. Muebles

En esta sección encontraremos miniaturas de los modelos 3D separados en función de su lugar en un habitáculo. Se ha decidido dividir entre 4 grupos: Living room, Bedroom, Kitchen, Bathroom e Item.

La tarea que se relaciona con todo el proceso de creación del catálogo para muebles está especificada en M3D.1 - Desarrollo del catálogo.



(a) Catálogo de muebles.

(b) Catálogo de muebles con una sección abierta.

Figura 8: Catálogo de muebles. Elaboración propia

- Living room. Aquí encontraremos objetos 3D que solemos encontrar en un salón, como por ejemplo, televisores, sofás, mesas, sillas para comer, etcétera.
- Bedroom. Se encuentran objetos tales como camas, escritorios, sillas de oficina y mesillas de noche.
- Kitchen. Típico mobiliario de cocina, como neveras y cocinas enteras.
- Bathroom. Aquí podremos encontrar lavabos, duchas, baños y váteres.
- Item. En esta sección se hallarán los objetos que podemos poner únicamente encima de otros objetos, como jarrones, lámparas, ordenadores, libros, etcétera.

Los objetos se podrán previsualizar mediante una pequeña miniatura. Esta miniatura ha sido generada haciendo una captura de pantalla en el visor online de objetos 3D 3dviewer. En ella se han ido cargando todos los objetos en un fondo totalmente blanco, para que la muestra de la miniatura sea pueda ver el objeto 3D de forma simple y concisa, a continuación se pueden ver un par de ejemplos.



(a) Miniatura del objeto “Comoda”



(b) Miniatura del objeto “Mesilla de noche”

Figura 9: Ejemplo de dos de las miniaturas de objetos que componen el catálogo. Están elaboradas a partir de la aplicación online 3dviewer. Fuente: [30]

Si el usuario hace click en una de las miniaturas que se muestran en el catálogo, directamente se abrirá la sección de “Object options”, que permite manipular el objeto seleccionado una vez llegue a estar dentro de la escena. A demás, se desplazará por pantalla el objeto a medida que se mueve el cursor.

9.1.1.1. Object options

Esta pestaña se abre una vez se selecciona un objeto en la sección del catálogo “Muebles” o bien se hace click en un objeto situado en algún lugar del *tour virtual*. Dentro se encuentran los siguientes componentes (por orden, de arriba a abajo):

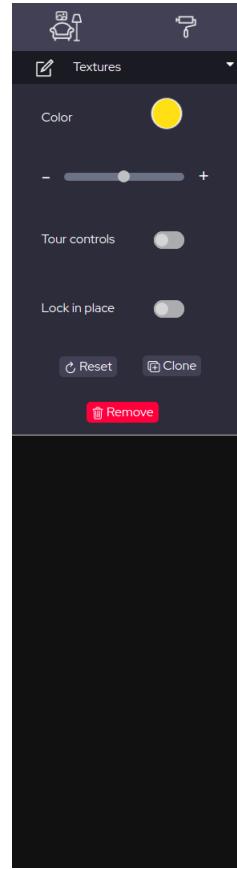


Figura 10: Captura catálogo en la sección de Object Options. Elaboración propia.

- **Textures.** Al hacer click en esta opción, se nos abrirá una sección con diversas texturas en función del objeto. Se previsualizarán con miniaturas de las texturas, y al hacer click en una de ellas, se cambiará la textura de aquellas partes del objetos seleccionadas previamente por defecto, junto con el color, estas dos secciones se dedican a cambiar la apariencia visual de las partes más destacables de los objetos.

En el punto 9.2.2.2 se explica cómo se realiza el proceso de selección de cada uno de las partes más destacables de un objeto (layers), y en el punto 9.2.7.4 se explica cómo se aplica la textura a uno de estos objetos.

La parte del catálogo de muebles relacionada con las texturas es específica de la tarea MAT.1 - Desarrollo del catálogo.

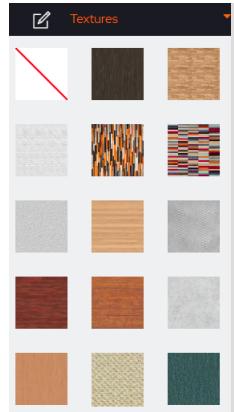


Figura 11: Captura del catálogo en la sección de Object Options en la opción de Textures. Elaboración propia.

- Color. Al seleccionar esta opción se nos abrirá una ventana emergente con un selector de color. Cuando se haga click en un color, el objeto seleccionado cambiará de color aquellas partes del objeto seleccionadas por defecto como layers, el método para añadir color a estos subobjetos se explica en el apartado 9.2.7.3.

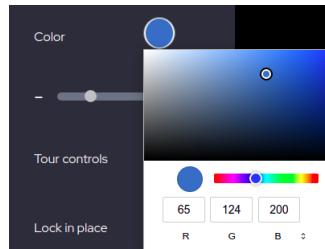


Figura 12: Captura catálogo en la sección de Object Options en la opción de Textures. Elaboración propia.

La tarea relacionada con el método para añadir color está especificado en MAT.4 - Color.

- Scale. Se utiliza un *slider* para esta herramienta. Un *slider* es una barra selectora con un rango entre dos valores. Este *slider* sirve para cambiar el tamaño del objeto una vez aplicado correctamente en la escena. Si el *slider* va hacia la izquierda, el tamaño del modelo es decrementado, y si va hacia la derecha, incrementa. Explicado en el apartado 9.2.7.2.
- Tour controls. Dado que los controles básicos del objeto solapaban con los controles de visualización del entorno del *tour virtual*, se decidió poner

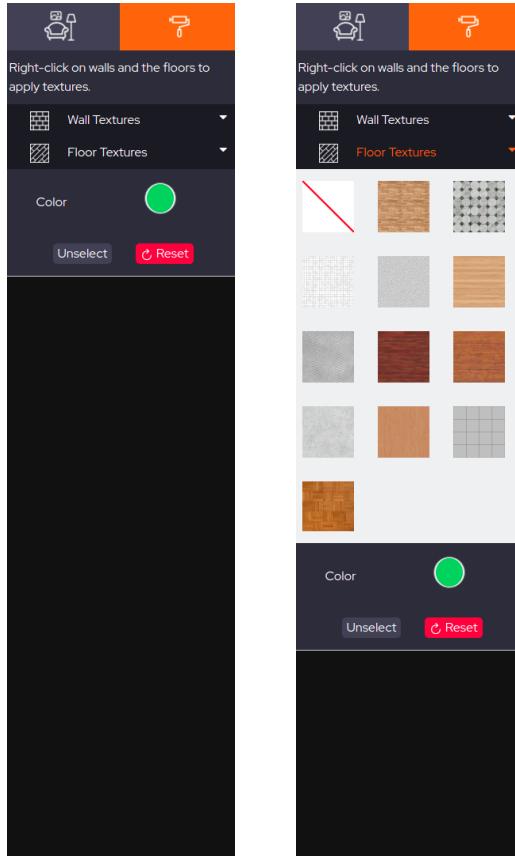
este *switch* para cambiarlo a la hora de agregar el objeto desde el catálogo. De esta forma, si el botón está activado, se utilizarán los controles de modificación del objeto, y si no lo está, se seguirán utilizando los controles clásicos para la visualización del tour que se especificarán en el punto 9.2.7.6.

- Lock in place. Al activarlo en un objeto seleccionado, este no se podrá mover por mucho que arrastremos el click encima del objeto. Se utiliza para asegurar que el objeto yace en la posición deseada evitando posibles errores del usuario al trazar un desplazamiento erróneo del ratón cuando su intención es moverse por el *tour virtual*, su funcionalidad se explica en 9.2.7.8.
- Reset. Si el objeto ha tenido modificaciones en su textura o color, al hacer click en este botón volverá a tener los valores por defecto en textura y color. Explicado en 9.2.7.5
- Clone. Este botón clona un objeto seleccionado. El objeto clonado mantendrá las características de rotación, textura, tamaño y color. En el apartado 9.2.7.7 se explica su funcionalidad implementada.
- Remove. La traducción directa sería “eliminar”, y al hacer click en el botón sobre un elemento seleccionado, aparecerá una ventana emergente para confirmar que el objeto seleccionado se eliminará de la escena y al aceptar, desaparecerá. Se explica la funcionalidad en el punto 9.2.7.9.

9.1.2. Pintar paredes y suelos

Al hacer click en esta sección, podremos observar dos secciones, una con las texturas de tipo pared y otras con las texturas de tipo suelo, las miniaturas de las texturas son directamente la imagen de la textura a aplicar.

La tarea que se relaciona con el proceso de creación del catálogo para la sección de pintar paredes y suelos está especificada en MAT.1 - Desarrollo del catálogo.



(a) Catálogo de pintar paredes.

(b) Catálogo de pintar paredes con la sección de Floor Textures abierta.

Figura 13: Catálogo de pintar paredes. Elaboración propia

Dentro de esta sección tendremos la posibilidad de seleccionar y deseleccionar suelos y paredes mediante click derecho, tal y como muestra el texto informativo. Si estamos seleccionando paredes y techos y de repente seleccionamos un suelo, las paredes se deseleccionarán y empezaremos a utilizar el selector para suelos. De esta forma no se puede interferir entre cambiar el color a un suelo y a una pared, dado que utilizan texturas diferentes.

Una pared/suelo se puede seleccionar al hacer click derecho en él, se nos dará un feedback visual iluminando sus bordes con un tono anaranjado. Se puede deseleccionar volviendo a hacer click sobre la pared/suelo, al hacer esto, veremos como las líneas de selección de pared desaparecen.

Una vez tengamos seleccionados un conjunto de paredes/suelos, podremos seleccionar la textura a aplicar haciendo click en la miniatura del catálogo. Si el usuario hace click en una pared con la sección de pintura abierta, directamente se abrirá el catálogo de texturas para paredes, y si hace clic en un suelo, se abrirá en el catálogo la sección de texturas de suelo. También hay unas opciones de pintado que se comentan a continuación.

9.1.2.1. Paint options

En esta sección hay formas de modificar la pintura de paredes y suelos.

- Color. Este selector de color es igual que el que podemos encontrar en los controles de objeto 9.2.7.3. Con su uso, podremos cambiar el color de las paredes seleccionadas si se ha cambiado su textura previamente. Se puede ver en 9.3.2.2
La tarea relacionada con el método para añadir color, como en los modelos, está especificado en MAT.4 - Color.
- Unselect. Sirve para deseleccionar todas las paredes/suelos que tengamos seleccionadas. Explicado en 9.3.2.3
- Reset. Haciendo click en este botón, podremos volver a ver las paredes/suelos seleccionados tal y como estaban antes de cambiar ninguna textura. En el apartado 9.3.2.4 se explica su funcionalidad.

9.2. Amueblado

En esta sección se explicará todo lo relacionado con los modelos 3D que se añaden a la escena y todas sus funcionalidades descritas anteriormente en el catálogo.

9.2.1. Formato de archivos 3D

Para el desarrollo de este proyecto, ha sido necesario trabajar con modelos 3D. Se ha decidido utilizar modelos 3D con el formato .OBJ dado que es uno de los formatos más comunes en el mundo del modelaje tridimensional, por lo tanto, más fácil de encontrar que otro tipo de formatos, a demás de que la empresa ya contaba con varios objetos 3D de en este formato.

9.2.1.1. Formato OBJ

La extensión del archivo OBJ es conocido como Wavefront 3D Object File, es un formato de archivo usado para un objeto tridimensional que contiene las coordenadas 3D (líneas poligonales y puntos), mapas de textura, y otra información de objetos [31].

Dentro de un archivo OBJ es común encontrar que el objeto es subdividido entre varios objetos de menor tamaño y que juntos, forman el objeto en su totalidad. En cada uno de estos subobjetos tenemos la siguiente información, principalmente:

- Vértices geométricos. Un vértice se especifica en el fichero OBJ mediante una línea que comienza con la letra *v*. A esto le siguen sus coordenadas (*x*, *y*, *z*). Estos son números que definen la posición del vértice en tres dimensiones.

v x y z

v 0.123 0.234 0.345

- Vértices normales. En la geometría de los gráficos por computador, un vértice normal en un vértice de un objeto es un vector direccional asociado con un vértice, destinado a reemplazar la verdadera normal geométrica de la superficie. Comúnmente, se calcula como el promedio normalizado de las normales de superficie de las caras que contienen ese vértice. Se puede especificar con una línea que comience con *vn*. Se especifica el vértice normal mediante las coordenadas (i, j, k).

vn i j k

vn 0.6969 0.0000 0.7171

- Vértices de textura. Especifica un vértice de textura y sus coordenadas (*u*, *v*, *w*). Una textura 1D solo requiere coordenadas de textura *u*, una textura 2D requiere coordenadas de textura tanto *u* como *v*, y una textura 3D requiere las tres coordenadas. *u* es el valor de la dirección horizontal de la textura, *v* es el valor de su dirección vertical y *w* es un valor para su profundidad.

vt u v w

vt 0.5 1 0

- Caras de un objeto. Las caras se definen mediante listas de vértices y se pueden representar mediante tres o más vértices. Cada triplete de números especifica un vértice geométrico, vértice de textura, y vértice normal, en ese orden. Los números de referencia deben estar separados por barras (/).

```
f v/vt/vn v/vt/vn v/vt/vn v/vt/vn
f 40/89/15 38/90/15 42/91/17 44/92/17
```

Si solo hay vértices y normales de vértice para un elemento de cara (no vértices de textura), se debe indicar con dos barras (//). Por ejemplo, para especificar sólo el vértice y los números de referencia normales de vértice, sería de la siguiente forma:

```
f v//vn v//vn v//vn v//vn
f 1//7 53//4 53//3 45//44
```

- Materiales de referencia. Los materiales que describen los aspectos visuales de los polígonos se almacenan en archivos .mtl externos. Se puede hacer referencia a más de un archivo de material MTL externo desde el archivo OBJ. El archivo .mtl puede contener una o más definiciones de material con nombre.

```
mtllib [nombre de archivo .mtl externo]
```

- Referenciar material. Esta etiqueta especifica el nombre del material para el elemento que le sigue. El nombre del material coincide con una definición de material con nombre en un archivo .mtl externo.

```
usemtl [nombre del material]
```

9.2.1.2. Formato MTL

El archivo Wavefront Material Template Library (MTL) es un archivo complementario para uno o más archivos Wavefront OBJ. Al igual que el anterior formato OBJ, el formato MTL fue utilizado y documentado por Wavefront Technologies en la década de 1990 [32].

Un archivo MTL contiene una secuencia de definiciones de material, cada una de las cuales comienza con la palabra clave `newmtl` y un nombre para el material. Las declaraciones en una definición de material constan de una palabra clave, seguida de opciones, valores o referencias específicas de palabras clave a archivos complementarios para su uso como mapas de textura. La definición de material termina al final del archivo o en la siguiente instrucción `newmtl`. Las declaraciones que definen las características del material pueden estar en cualquier orden. Un ejemplo de una definición de material es:

```

madera.mtl

newmtl Madera
Ka 1.000000 1.000000 1.000000
Kd 0.640000 0.640000 0.640000
Ks 0.500000 0.500000 0.500000
illum 2
Ns 96.078431
Ni 1.000000
d 1.000000
Tr 0.000000
map_Kd woodtexture.jpg

```

El ejemplo utiliza las siguientes palabras clave:

- Ka. Especifica el color ambiental, para tener en cuenta la luz que se dispersa por toda la escena utilizando valores entre 0 y 1 para los componentes RGB.
- Kd. Especifica el color difuso, que normalmente aporta la mayor parte del color a un objeto. En este ejemplo, Kd representa un color gris, que será modificado por un mapa de textura de color especificado en la declaración map_Kd.
- Ks. Especifica el color especular, el color que se ve donde la superficie es brillante y trata de simular la reflexión a un espejo.
- illum. Especifica el modelo de iluminación utilizado por el material. illum = 1 indica un material plano sin reflejos especulares, por lo que no se utiliza el valor de Ks . illum = 2 denota la presencia de reflejos especulares, por lo que se requiere una especificación para Ks
- Ns. Define el foco de reflejos especulares en el material. Los valores de Ns normalmente oscilan entre 0 y 1000, con un valor alto que da como resultado un brillo compacto y concentrado.
- Ni. Define la densidad óptica (también conocida como índice de refracción) en el material actual. Los valores pueden oscilar entre 0,001 y 10. Un valor de 1,0 significa que la luz no se dobla cuando atraviesa un objeto.
- d. Especifica un factor de disolución (no transparencia), cuánto se disuelve este material en el fondo. Un factor de 1.0 es completamente opaco. Un factor de 0.0 es completamente transparente.
- Tr. Especifica un factor de transparencia, cuánta opacidad tiene este material. Un factor de 0.0 es completamente opaco. Un factor de 1.0 es completamente transparente. Este valor es opuesto a d.

- map_Kd. Especifica un archivo de textura de color (archivo de imagen) que se aplicará a la reflectividad difusa del material. Durante el renderizado, los valores de map_Kd se multiplican por los valores de Kd para derivar los componentes de color RGB. También existe map_Ka y map_Ks, para aplicar el mapa al color ambiental o el color especular.

Juntando los atributos de color ambiental, color difuso y color especular (K_a , K_d y K_s , respectivamente), llegamos al modelo de reflexión de Phong [33], un popular modelo en los gráficos por computador que se basa en ser un modelo de iluminación y sombreado que asigna brillo a los puntos de una superficie modelada.

En la siguiente imagen se puede observar la repercusión de cada uno de los valores descritos, K_a , K_d y K_s .

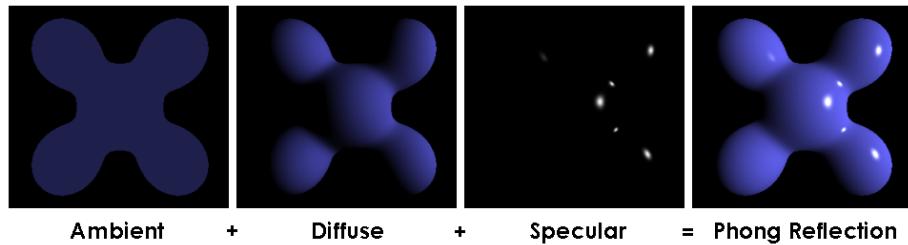


Figura 14: Efectos de aplicar los valores de K_a , K_d y K_s explicados anteriormente, fuente: [34]

El modelo de Phong se basa en el cálculo de vectores unitarios que son afectados por la iluminación del entorno. El vector V es el vector de vista, desde dónde percibe el espectador la escena (cámara), L es el vector de luz incidente, R es el vector de reflexión de la luz incidente (L) respecto la normal de la superficie del modelo (N).

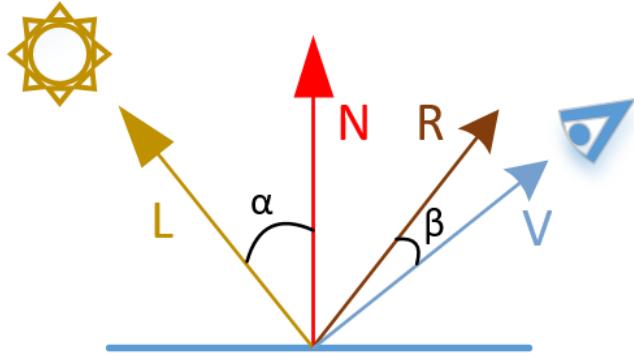


Figura 15: Representación de los vectores que afectan en la reflexión del modelo de Phong, fuente: [35]

Después de encontrar estos vectores, se hace uso de las siguientes fórmulas para calcular la ecuación de iluminación Phong:

Primero, el color difuso viene dada por la ecuación:

$$\text{Color difuso} = Kd \cdot (N \cdot L)$$

Donde Kd es la constante de color difuso. $(N \cdot L)$ es lo mismo que el coseno del ángulo entre N y L , por lo que a medida que el ángulo disminuye, el valor difuso resultante es mayor.

El color especular, viene dado por la siguiente ecuación

$$\text{Color especular} = \text{Color difuso} + Ks \cdot (R \cdot V)^{Ns}$$

Por lo tanto:

$$Kd \cdot (N \cdot L) + Ks \cdot (R \cdot V)^{Ns}$$

Donde Kd es el componente difuso y Ks es el componente especular. Esta es la ecuación de iluminación Phong generalmente aceptada. A medida que disminuye el ángulo entre el vector de vista (V) y la luz reflejada (R), se obtendrá más specularidad.

Mediante la librería *three.js* se ha utilizado los métodos `OBJLoader` y `MTLLoader` para que carguen ambos ficheros por cada uno de los objetos. Únicamente

se necesitan los archivos OBJ y MTL junto con archivos de imagen opcionales para algún tipo de material al que se le tenga que añadir un map. Esta carga se hace una vez el usuario hace click en la miniatura de uno de los objetos del catálogo.

9.2.2. Configuración previa de los objetos

Dado que los objetos recogidos de varias fuentes han sido diseñados con propósitos diferentes entre sí, no todos los objetos han venido con una configuración adecuada para el proyecto, a continuación se explican las diversas modificaciones que se han tenido que hacer en algunos objetos para que luzcan mejor en el *tour virtual*.

Esta sección se considera parte de la tarea M3D.3 - Añadir objetos 3D al escenario, dado que se han realizado estas modificaciones para poder añadir de forma correcta el objeto al escenario 3D.

9.2.2.1. Tamaño

Dado que los objetos que se utilizan han sido extraídos de diversas fuentes, cada uno de ellos tiene un tamaño distinto, hay unos de un tamaño enorme y otros con tamaño exageradamente pequeño. Es por esta razón que se ha ido ajustando manualmente el valor de su escala.

El proceso para hacerlo ha sido introducir el objeto en el *tour virtual* e ir ajustando el parámetro de *scale* del objeto añadido hasta conseguir un tamaño acorde con la realidad. Ese tamaño será el tamaño por defecto que tenga el objeto al instanciarlo por primera vez en la escena.

three.js facilita el escalamiento de un objeto con solo ajustar los valores para el escalado del objeto en cuestión mediante la siguiente función:

```
object.scale.set(x,y,z);
```

Donde *x*, *y* y *z* son los valores de escala que tendrían en cada uno de los tres ejes dimensionales y **object** es el objeto de tipo Object3D que escalaremos. Internamente, estas funciones se realizan mediante matrices de transformaciones geométricas ².

²Una transformación geométrica es una operación o la combinación de varias de ellas, en que se parte de una forma original para generar otra nueva estableciendo una relación biunívoca entre ellas.

9.2.2.2. Layers

A la hora de cambiar la textura o color de un objeto, en lugar de aplicar este cambio a todo el objeto, se ha decidido seleccionar manualmente aquellos subobjetos del objeto principal que tienen más importancia. Por ejemplo, de un taburete como en la imagen 16, se ha seleccionado únicamente la parte del asiento y no las cuatro patas. De esta forma, a la hora de seleccionar su color o textura, cambiará únicamente el aspecto de la parte del asiento.



Figura 16: Ejemplo del objeto “Taburete” donde sólo el subobjeto que forma parte del asiento es el layer. Se puede observar que únicamente cambia de aspecto esta parte y no el resto del objeto. Elaboración propia.

De cada objeto se han ido seleccionado los subobjetos principales de forma manual, y se han guardado en el atributo que se denomina layers para así tener la facilidad de cambiar esos subobjetos a la hora de seleccionar una nueva textura o un color distinto y que no tenga repercusión en la resta de subobjetos del objeto en cuestión.

9.2.2.3. *Pivot point*

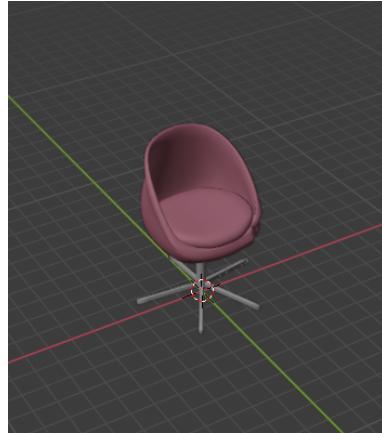
El *Pivot point* es el punto del objeto 3D donde se encuentra el origen. Este punto de origen será el punto de referencia a la hora de girar el objeto o escalarlo o moverlo por la escena.

Algunos de los objetos no venían con un *Pivot point* adecuado y las transformaciones geométricas disponibles (trasladar, rotar y escalar) no se realizaban como se esperaba. A demás, este *Pivot point* se ha tenido que ajustar de manera distinta en función del tipo de objeto, ya sea tipo *floor*, *wall* o *wallFloor*, como se explicará en el punto 9.2.3.2.

Para realizar cambios para posicionar estos *Pivot points* de manera adecuada se ha decidido utilizar *Blender*. *Blender* es una suite de creación de contenido 3D totalmente integrada, que ofrece una amplia gama de herramientas esenciales, incluyendo Modelado, Renderizado, Animación y Rigging, Edición de Vídeo, VFX, Composición, Texturizado, y algunos tipos de Simulaciones [36].

Se ha considerado mejor esta opción que cambiar por código el *Pivot point* de aquellos objetos con un mal posicionamiento ya que cada objeto tiene unas características distintas y habría que hacerlo objeto por objeto y eso ensuciaría el código.

Desde *Blender*, simplemente se ha ajustado el objeto con tal de que tenga su *Pivot point* de forma correcta, como en el siguiente objeto de tipo *floor*, cuyo *Pivot point* yace en la cara inferior del objeto.



(a) Objeto “Sillón oficina” en *Blender*, con su *Pivot point* en la cara inferior del modelo.



(b) Objeto “Sillón oficina” en el *tour virtual*, con su *Pivot point* en la cara inferior del modelo.

Figura 17: Ejemplo de modificación del *Pivot point* en *Blender* y su representación en la escena. Elaboración propia.

9.2.2.4. Tipo de textura

Dado que no todos los objetos tienen sentido que tengan todo tipo de texturas, se ha decidido hacer una división entre objetos con textura de tipo tejido y objetos con textura tipo mueble.

Este proceso se ha realizado de modo manual en función del tipo de objeto, por ejemplo, las estanterías y mesas tienen textura de tipo mueble, como tipos de maderas y mármoles. Y los objetos como sofás o alfombras tienen textura de

tipo tejido, como por ejemplo distintos estampados o cuero.

De cada uno de los objetos se ha ido seleccionando su tipo y se le ha asignado el tipo de texturas que tiene.



Figura 18: Catálogo de ambos tipos de texturas. Elaboración propia.

9.2.2.5. Brillo

Por motivos de iluminación, no todos los objetos tienen configurado un brillo adecuado a la escena. *three js* reconoce el objeto cargado mediante OBJLoader como un objeto Object3D y una de sus características en el material es el *shininess*. Este componente hace referencia a la reflexión especular (*Ns* en el apartado 9.2.1.2).

Para modificar el valor del *shininess* mediante la siguiente línea de código en *three js*:

```
object.children[i].material.shininess = 10;
```

Donde la variable *i* es el índice de cada uno de los hijos del objeto, de esta forma, se pondrán todos los subobjetos de *object* con el valor del *shininess* de su material a 10.

Se han encontrado objetos con un *Ns* muy elevado y otros con un *Ns* demasiado bajo. Es por ello que se ha decidido ajustar un valor por defecto a todos los materiales de objetos (y subobjetos) en el catálogo. Con un valor de 10, se

considera un valor suficientemente realista para el ambiente que hay en los *tour virtuales* que dispone Floorfy.



(a) Objeto “Double Bed” con su configuración de brillo por defecto.



(b) Objeto “Double bed” con *shininess* a 10.

Figura 19: Ejemplo al modificar el valor de *shininess* de un objeto.
Elaboración propia.

Como se puede observar en la figura 19, la imagen sin modificar refleja bastante color especular donde no tiene sentido que se refleje, esto sucede con más de un objeto por cómo se ha decidido hacer el objeto en sí. Es por esto que se ha decidido bajar el *shininess* de todos los objetos a 10, como la imagen a la derecha, que se nota con la especularidad más baja y se mezcla de una manera más realista con el entorno.

9.2.3. Categorización de objetos

A la hora de realizar la implementación del código, se ha decidido establecer dos tipos de categorización, una por el tipo de habitación en el que se puede encontrar un objeto, para facilitar al usuario encontrar el tipo de objetos específico deseado (Living room, Bedroom, Kitchen, Bathroom e Item), y otra por tipo de posicionamiento del objeto dentro del *tour virtual* (*floor*, *wall* y *wallFloor*).

9.2.3.1. Categorización por tipo de habitación

Se ha decidido hacer este tipo de categorización para que el usuario pueda encontrar de forma más cómoda el tipo de objeto que está buscando mediante el uso del catálogo. De esta forma, si el usuario se encuentra en un baño, simplemente accede a la sección de modelos de baño y logrará encontrar de forma

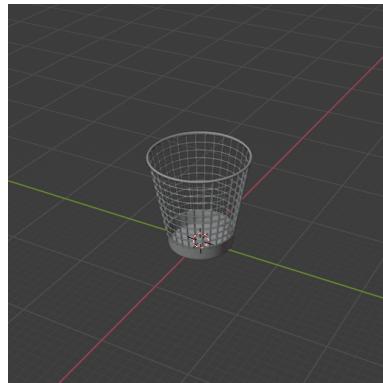
más rápida el objeto que busca. Los tipos son Living room, Bedroom, Kitchen, Bathroom e Item y han sido explicados previamente en el punto 9.1.1.

El proceso de selección de estos modelos ha sido de forma manual, buscando entre los modelos 3D que me facilitó la empresa y buscando en páginas web donde proporcionan objetos 3D de forma gratuita. Se han añadido a sus respectivas secciones una vez decidido su tipo.

9.2.3.2. Categorización por tipo de posicionamiento

Se ha decidido representar el posicionamiento de objetos en los 4 tipos de objetos distintos que se explican a continuación. En el caso de obtener un objeto con un *Pivot point* descentrado acorde a su tipo, se ha utilizado *Blender* para corregir este fallo, tal y como se explica en el anterior punto 9.2.3.2.

- **Floor.** Este tipo de objetos se caracteriza por estar únicamente en el suelo. Su *Pivot point* está centrado en la cara inferior del objeto tal y como se representa en la siguiente imagen.



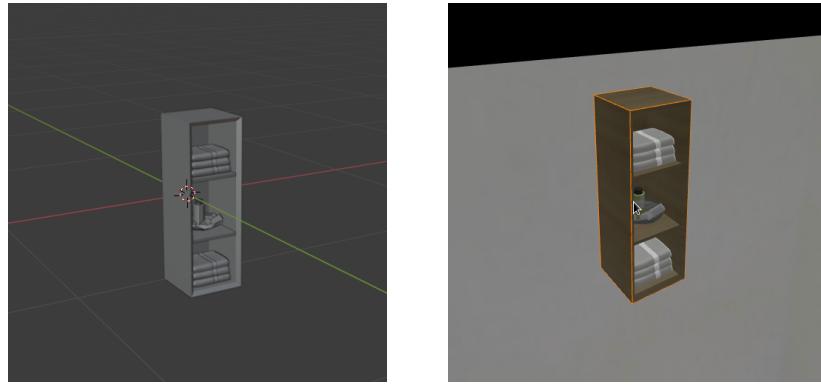
(a) Objeto “Papelera” en *Blender*, se puede ver que el *Pivot point* se encuentra en la conjunción entre los ejes x y y z.



(b) Objeto “Papelera” en un *tour virtual*, se puede observar que su posición está respecto al ratón en el suelo, es decir, en la ubicación del *Pivot point*.

Figura 20: Ejemplo del *Pivot point* de un objeto de tipo *floor*. Elaboración propia.

- **Wall.** Los objetos de tipo *wall* tienen el *Pivot point* en el centro de la cara posterior (la más lejana en el eje Z) como se puede observar a continuación.

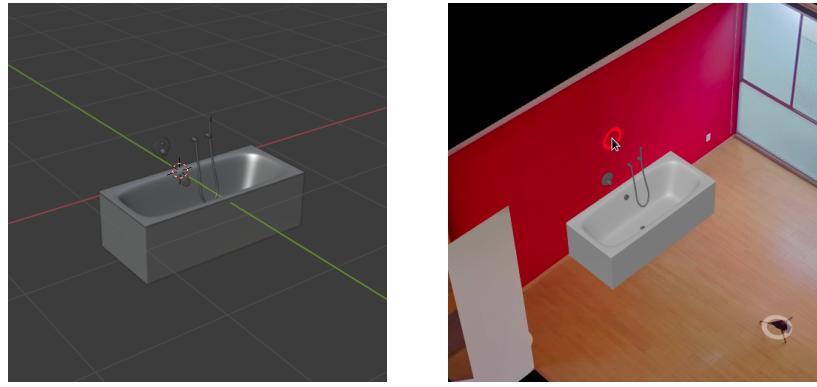


(a) Objeto “Estantería Baño” en *Blender*, se puede ver que el *Pivot point* se encuentra en la conjunción entre los ejes *x* y *y*.

(b) Objeto “Estantería Baño” en un *tour virtual*, se puede observar que su posición está respecto al ratón en la pared, es decir, en la ubicación del *Pivot point*.

Figura 21: Ejemplo del *Pivot point* de un objeto de tipo *wall*. Elaboración propia.

- *WallFloor*. Este tipo de objetos son similares a los objetos de tipo *wall*, con la única diferencia de que la posición del ratón en el eje *y* es indiferente, dado que siempre se actualizará la posición del objeto en el suelo. En el ejemplo siguiente se puede observar.

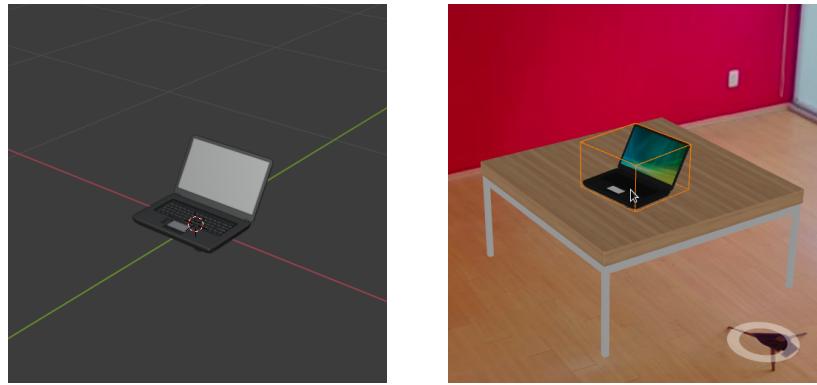


(a) Objeto “Bath” en *Blender*, se puede ver que el *Pivot point* se encuentra en la conjunción entre los ejes *x* *y* *z*.

(b) Objeto “Bath” en un *tour virtual*, se puede observar que su posición está respecto los valores de *x* y *z*, pero no en la posición *y* del ratón situado en la pared, respecto al *Pivot point* asignado.

Figura 22: Ejemplo del *Pivot point* de un objeto de tipo *wallFloor*. Elaboración propia.

- *Item*. Funcionan exactamente igual que los objetos de tipo *floor*, con la particularidad de que éste objeto no mira los límites del suelo, si no los del objeto en el cual se posiciona.



(a) Objeto “Portatil” en *Blender*, se puede ver que el *Pivot point* se encuentra en la conjunción entre los ejes *x* *y* *z*.

(b) Objeto “Portatil” en un *tour virtual*, se puede observar que su posición está respecto al ratón en un objeto, es decir, en la ubicación del *Pivot point*.

Figura 23: Ejemplo del *Pivot point* de un objeto de tipo *item*. Elaboración propia.

Se pueden colocar en cualquier parte de un habitáculo siempre y cuando sea en sus respectivas posiciones y cumplan las condiciones de colisión entre otros objetos y límites dentro de la habitación que se explican a partir de la posición válida en el punto 9.2.6.

Es importante recalcar que este proceso implica que cuando movamos el cursor por el escenario del *tour virtual* al colocar/mover uno de los modelos del catálogo, el objeto centrará el *Pivot point* en el cursor. Esta parte está relacionada con el uso de *Raycasting* (método de *three js*) que se explica en profundidad en el punto 9.2.5.

9.2.4. Iluminación

Para que la escena luzca los elementos de forma correcta, necesitamos añadir luz al escenario. Previamente, por cómo está hecho el render 3D de los *tour virtuales*, no se ha hecho uso de ningún tipo de luz, ya que únicamente dependían de las imágenes en 360º para recrear todo el escenario.

Pero ahora que se han añadido objetos tridimensionales, se han necesitado dos elementos de iluminación para que los objetos se muestren y se vean de la forma más realista posible.

9.2.4.1. Ambient Light

Gracias a la librería *three js*, podemos añadir a la escena luz ambiental. Este método de la librería permite añadir a la escena una luz que ilumina de forma global todos los objetos de la escena independientemente de su posición de forma equitativa y sin ningún tipo de dirección. Su efecto se puede ver a continuación.



(a) Escena con varios objetos sin ningún tipo de luz.

(b) Escena con varios objetos con luz AmbientLight.

Figura 24: Comparativa de una escena sin luz y con luz ambiental.
Elaboración propia.

Como se puede observar, nos resuelve el poder ver los colores de los objetos de forma correcta, ya que antes estábamos a oscuras. Pero de todas formas, únicamente podemos ver el efecto del color ambiental de los materiales de los objetos y no podemos ver el efecto de color espectral ni difuso que se muestra en la imagen 14, dado que no existe el vector que representa la luz (dado que AmbientLight no tiene dirección). Es por eso que se decide añadir otro tipo de luz a la escena, una luz con dirección.

9.2.4.2. Directional Light

Este tipo de luz es una luz direccional que se emite en una dirección específica. Esta luz se comporta como si estuviera infinitamente lejos y los rayos producidos a partir de ella fueran todos paralelos. El caso de uso común para esto es simular la luz del día; el sol está lo suficientemente lejos como para que su posición pueda considerarse infinita, y todos los rayos de luz que provienen de él son paralelos. Esta luz hace referencia al vector V de la imagen 15.

Dentro de la implementación del aplicativo se ha decidido poner este tipo de luz en la posición de la cámara y con su objetivo apuntando hacia el mismo objetivo de la cámara. De esta manera, siempre que se esté moviendo la cámara mediante los controles de uso por el *tour virtual*, también se añadirá este tipo de iluminación a lo que estemos visionando. A continuación veremos cómo cambia la escena gracias a este tipo de iluminación.



(a) Escena con varios objetos con luz AmbientLight.



(b) Escena con varios objetos con luz AmbientLight y DierctionalLight.

Figura 25: Comparativa de una escena con luz ambiental y con luz ambiental + luz direccional. Elaboración propia.

Se puede observar cómo ya vemos los colores especulares que antes no estaban (que también dependen del brillo del objeto, explicado en 9.2.2.5), junto con los colores de la componente difusa que simulan un sombreado en función de la posición de la luz y cómo toda la escena global gana en realismo.

9.2.5. Raycasting

Raycasting es un método de la librería *three js* que permite crear un vector a partir de un origen y un punto en la escena tridimensional y tiene la funcionalidad de detectar todos los objetos que el vector trazado intersecciona en toda la escena.

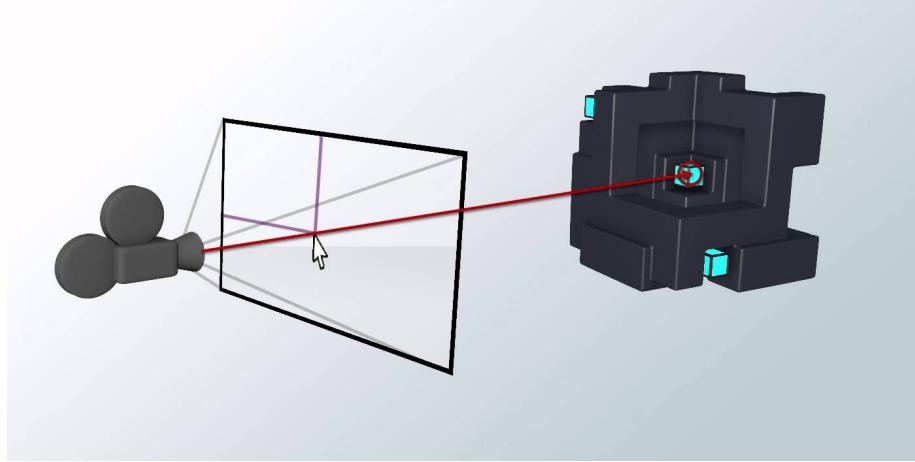


Figura 26: Representación de la interacción del *Raycasting* mediante la cámara, la posición del ratón en la pantalla y la escena 3D. Fuente: [37]

Three js permite crear el *raycaster* con origen la cámara del escenario y hacia el ratón, de la siguiente forma:

```
var raycaster = new THREE.Raycaster();
raycaster.setFromCamera(camera, mouse);
```

Y actualizando su posición haciendo lo siguiente dentro de la función de eventos ³ *mousemove*, la cual se ejecuta siempre que se haga cualquier movimiento con el ratón.

```
raycaster.setFromCamera( mouse, camera );
var intersects = raycaster.intersectObjects( scene.children );
```

³Una función de evento en *JavaScript* no es más que una función que se ejecuta siempre que sucede un evento particular, tanto por parte del usuario como por parte del navegador. En este proyecto, únicamente haremos uso de funciones de eventos que tienen por entrada el ratón

Así tenemos en la variable `intersects` un array con todos los objetos de la escena que son interseccionados por el trazado del rayo ordenado de forma ascendente (por ejemplo, si lo primero que se encuentra es una pared, ese objeto pared irá en la posición 0 del array, y si después de la pared existe una silla, irá en la posición 1 del array) y se mantendrá actualizado por la función de evento, es decir, se ejecutará siempre que el ratón se mueva.

Dado que hay varios elementos de los cuales no necesitamos información, a la hora de comprobar el primer objeto interseccionado, comprobaremos que es un objeto cuya información es relevante. Por ejemplo, si el primer objeto interseccionado por nuestro *raycaster* es un *hotspot*, incrementaremos el valor del índice del array con tal de encontrarnos con el siguiente objeto interseccionado por el *raycaster*.

En este proyecto, este método es de lo más importante a la hora de interactuar el usuario con el aplicativo. Principalmente, se ha implementado con dos finalidades: seleccionar objetos en la escena una vez establecidos y agregar los objetos a la posición deseada en la escena 3D.

9.2.5.1. Posicionar un objeto seleccionado

Como ya se ha explicado en el anterior punto 9.2.1 la información geométrica del objeto es cargada mediante el archivo OBJ y su material es cargado mediante su archivo MTL y sus opcionales archivos de imagen. Este proceso se hace en cuanto se hace click en una de las miniaturas de los modelos 3D disponibles.

También se puede seleccionar un objeto desde la escena, tal y como se explica en el primer punto del apartado 9.2.7.1.

Una vez tengamos el objeto seleccionado o cargado desde el catálogo, el objeto aparecerá en la escena centrado con su *Pivot point* en el centro del puntero del ratón, dado que se actualizará su posición respecto al origen, dentro de la función *mousemove* (es decir, cada vez que se mueva el ratón). Eso se ha mostrado en el apartado 9.2.3.2.

En función del tipo de objeto que estemos colocando y el tipo de objeto que esté interseccionando, aparecerá el objeto de una forma u otra. se ha decidido dejar el objeto seleccionado visible en el caso de que exista una coincidencia de tipos y no visible en cualquier otro caso, para que no se pueda ver el objeto en cualquier lugar de la escena.

Diremos que hay coincidencia de tipos cuando el *raycaster* intersecciona con un objeto cuyo tipo coincide con la superficie adecuada para el objeto seleccionado, es decir, para objetos de tipo *floor* un suelo, para objetos de tipo *wall* o *wallFloor* una pared y para objetos de tipo *item*, otro objeto.

A continuación se explican las consecuencias que tienen cada caso nombrado previamente:

- *Floor*. En este caso, únicamente se actualizará la posición del objeto a la del ratón. Dado que los objetos cargados siempre estarán de pie, es decir, sin ninguna inclinación, no es necesario rectificar el ángulo del modelo.
- *Wall*. Si el objeto es de tipo *wall*, hace falta rectificar la rotación del modelo en función de la rotación en Y que tenga la pared interseccionada. Para ello, se calcula la normal del punto de la pared interseccionada (línea verde en el ejemplo 27) y se genera un punto lo suficientemente lejano. Mediante la función `object.lookAt(puntoAuxiliar)`; el objeto mirará hacia esa dirección y por lo tanto, quedará acorde con la pared.

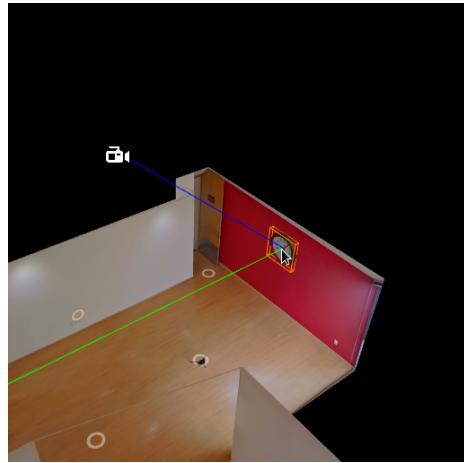


Figura 27: Ejemplo de posicionamiento con un objeto de tipo *wall*. Elaboración propia.

Pero hay ocasiones en las que la normal de la pared se interpreta de forma errónea, de forma que queda mirando hacia el lado contrario, como sucede en el ejemplo a continuación.

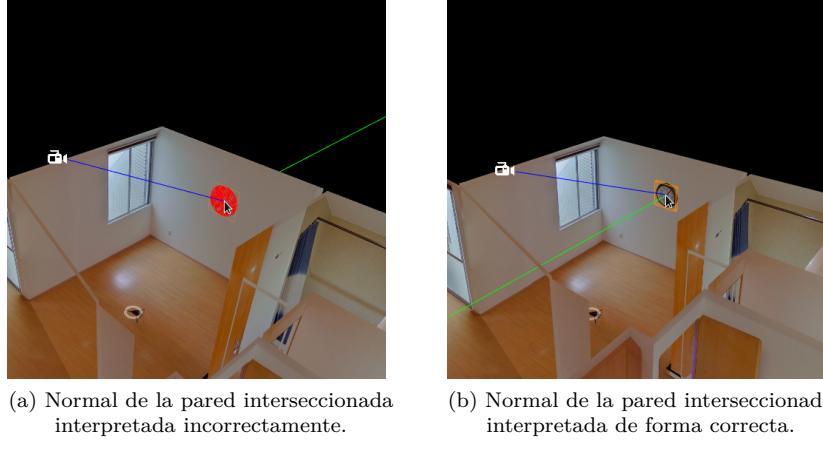


Figura 28: Representación de la solución al problema de normales invertidas en paredes. Elaboración propia.

Para solucionar este tipo de error, se calcula el vector que va desde la posición de la cámara hasta el punto de la pared intersecciónada (línea azul en el ejemplo 28) y el vector normal de la pared desde el punto (línea verde en el ejemplo 28). A continuación se hace el producto escalar de estos dos vectores.

El simple signo del producto escalar proporciona información sobre la relación geométrica de los dos vectores. Si el producto es positivo entonces el ángulo entre ambos vectores es menor de 90 grados y cada vector tiene un a componente en la misma dirección que el otro. Si el producto es negativo, implica que el ángulo entre vectores es mayor que 90 grados, y un vector tiene una componente en la dirección opuesta del otro.

En el caso de que el producto anterior fuese positivo (y, por lo tanto, no mirase hacia cámara), utilizaremos la normal negada, de esta forma la normal será interpretada de forma correcta.

Una vez tengamos este punto auxiliar, hacemos que el objeto rote hacia ese punto. En *three.js*, mediante el método `lookAt` conseguimos que el objeto rote hacia el punto auxiliar generado.

- *WallFloor*. Se sigue exactamente el mismo proceso que en los objetos de tipo *wall*, pero actualizando su posición, bajando en el eje *y* hasta que llegue a tocar el suelo. Por eso, objetos tipo bañeras o cocinas son colocadas en la pared pero están, a su vez, en el suelo.
- *Item*. Funciona igual que los objetos de tipo *floor*, pero en vez de interseccionar un suelo, intersecciona un objeto cualquiera. Si comprobamos que la normal del punto interseccionado en el objeto no es plana (es decir, no es un vector $(0,1,0)$), consideraremos una posición no válida para el ítem.

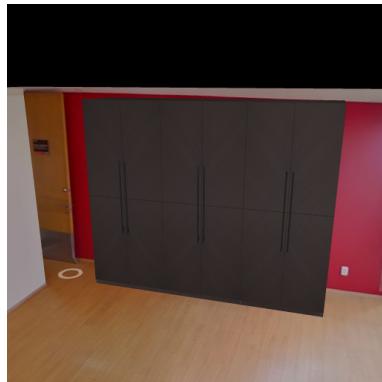
Una vez tengamos estas consideraciones en cuenta, aún dentro del método `mousemove`, debemos pasar por la función `checkValidPosition()`, que comprueba que el objeto, a parte de interseccionar mediante el *raycaster* en una posible posición valida, lo es. Este punto se explica en profundidad en la sección 9.2.6, y consta de diferentes puntos a analizar, como por ejemplo, la colisión entre otros objetos o los límites que tiene dentro de una propiedad.

9.2.5.2. Seleccionar un objeto de la escena

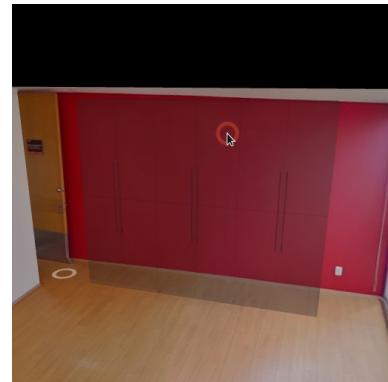
Gracias al método de *raycasting*, también podemos seleccionar objetos de la misma manera que lo hacemos en el anterior punto al interseccionar con una pared o suelo.

Esto sucede, en el caso de que no tengamos ningún objeto seleccionado, y el *raycaster* tenga en la primera posición del array (evitando algunos objetos) un modelo 3D, ya sea *floor*, *wall*, *wallFloor* o *item*.

En el caso de que se encuentre con uno de estos modelos, se le bajará la opacidad de 1.0 hasta 0.35 a todos los subobjetos del objeto interseccionado para dar al usuario un feedback de que ese objeto puede ser seleccionado mediante un click.



(a) Captura del objeto “Armario oscuro” sin el ratón encima.



(b) Captura del objeto “Armario oscuro” con el ratón encima.

Figura 29: Representación de la diferencia de opacidad al interseccionar un objeto. Elaboración propia.

Haciendo uso de otra función de evento llamada `mousedown`, la cual se ejecuta siempre que el ratón recibe un click en alguno de sus botones, le haremos saber que el objeto seleccionado debe ser el que se está interseccionando con el *raycaster* en ese momento.

Con un único click izquierdo del ratón, se selecciona el modelo al que estamos apuntando con el ratón y se reconoce porque aparecerá con menos opacidad. Es entonces, cuando veremos las opciones de objeto, las object options, y veremos el feedback que se recibe cuando un objeto es seleccionado, que es el contorno de su *bounding box*, explicado en el punto 9.2.6.1, junto con su control de rotación si lo necesita, como se ve en la siguiente imagen.

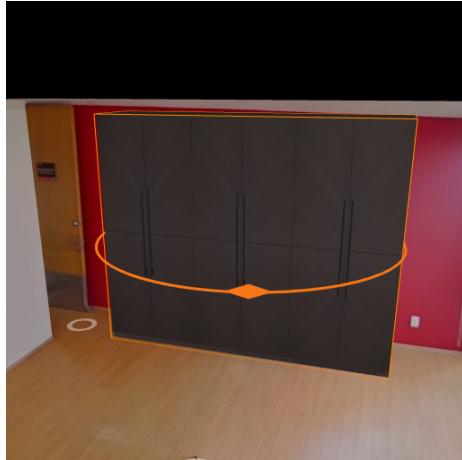


Figura 30: Captura del objeto "Armario oscuro" seleccionado. Elaboración propia.

Si en el momento de hacer click, el click se mantiene, se asumirá que el objeto es agarrado, y pasaremos a estar en en fase de posicionar un objeto en el entorno 3D, como se explica en 9.2.5.1.

9.2.6. Posición valida

Siempre que se esté ejecutando una de las tres transformaciones geométricas sobre un objeto (trasladar, rotar o escalar), se comprobará si el objeto permanece en una posición válida o inválida. Todo lo realizado en este apartado está relacionado con la tarea M3D.5 - Definir los límites en la propiedad.

Si el objeto se halla en una posición válida, se podrá quedar en esa posición, mientras que si no lo está, veremos un cambio en su apariencia, pues todos los subobjetos que forman el objeto tendrán un material rojo y con opacidad reducida al 0.35. Este es un recurso visual para que el usuario se de cuenta de que el objeto no está posicionado correctamente y se muestra de la siguiente manera:

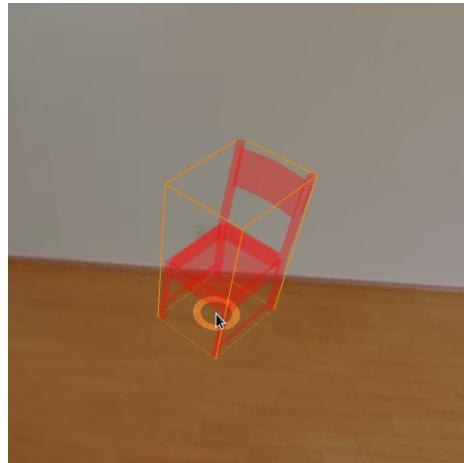


Figura 31: Captura del objeto “Silla normal” en una posición no válida. Elaboración propia.

La validez de la posición del objeto dependerá de si está bien colocado en la escena (en función del tipo de objeto del que hablamos) o si está interseccionando con cualquier objeto de la escena.

Para ello, necesitamos hablar de un concepto básico en el tema de colisiones en objetos tridimensionales: el *bounding box*.

9.2.6.1. *Bounding Box*

Un *bounding box* (en castellano, caja delimitadora) no es más que el paralelepípedo rectangular⁴ más pequeño que rodea completamente un objeto.

En *three js*, de manera natural, se puede generar el *bounding box* de un objeto llamando a la función *Box3*. Pero esta función devuelve únicamente el *Axis-Aligned Bounding Box* (AABB).

Lo particular de este tipo de *bounding box* es que las caras están alineadas con los ejes (*x*, *y*, *z*). Es decir, todas las caras del *boundning box* son paralelas a uno de los ejes principales. En la siguiente figura se puede observar el problema se supone hacer uso de este tipo de métodos.

⁴Un paralelepípedo es un poliedro de seis caras, en el que todas las caras son paralelogramos, paralelas e iguales dos a dos. Un paralelepípedo tiene 12 aristas, que son iguales y paralelas en grupos de cuatro, y 8 vértices. Si es rectangular, todos los ángulos entre aristas son de 90 grados.

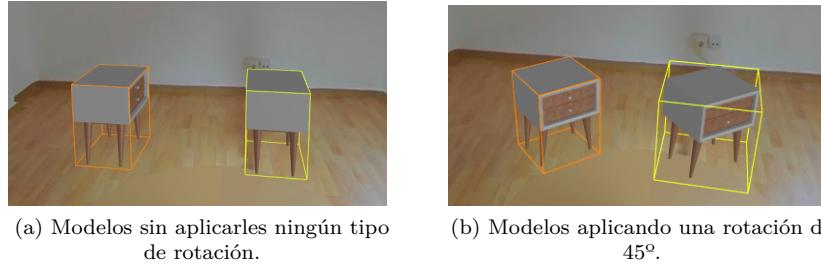


Figura 32: Dos modelos, uno con *bounding box* final (anaranjado) y otro con AABB por defecto (en amarillo). Elaboraci\xf3n propia.

Podemos ver que los *bounding box* inicialmente tienen el mismo valor y encajan perfectamente con nuestro modelo. Esto se debe a que todos los objetos tienen, por defecto, su construcci\xf3n alineada con los ejes de coordenadas. Pero el objeto con AABB, al ser modificada, su rotaci\xf3n nos da un resultado nada ajustado a la realidad, dado que sus ejes tienen la direcci\xf3n de los vectores b\xfasicos (*x*, *y*, *z*) independientemente de sus cambios en las transformaciones geom\xe9tricas.

Dado este problema, me vi en la necesidad de construir algo para que el *bounding box* sea lo m\xfas fiel a la realidad posible, dado que necesitaba sacar la informaci\xf3n del *bounding box* para calcular las colisiones y los l\xedmites en la propiedad de forma precisa y, evidentemente, con el AABB por defecto (color amarillo en la figura 32) no lograba obtener un resultado justo en cuanto el objeto rotaba.

Para ello, por cada objeto nuevo añadido a la escena, se guarda su *bounding box* inicial. Usando el m\xethodo de la librer\xfa *three.js BoxHelper*, se puede obtener la geometr\xfa del AABB de la siguiente forma:

```
var bboxAABB = new THREE.BoxHelper(object)
```

Y en la variable `bboxAABB` se guarda el AABB del objeto en ese momento. Dado que, como hemos visto en el ejemplo de la figura 32, el valor inicial del AABB coincide con lo deseado, tendr\xf3 unos valores inicialmente correctos. Una vez tengamos el AABB del objeto, se asigna al objeto en concreto, y en cada modifiaci\xf3n de la *transform matrix* (matriz de transformaci\xf3n en castellano), actualizamos la matriz con la del objeto.

Una *transform matrix* o matriz de transformaci\xf3n es una matriz 4x4 que guarda informaci\xf3n sobre las transformaciones geom\xe9tricas del objeto. Cada vez

que se escala, rota o se traslada el objeto, esta matriz es actualizada, aplicándole una matriz de escalado, de rotación o de traslación, respectivamente.

Es por eso que, internamente, en *three js*, se actualiza la matriz de transformación del objeto al realizar alguna de estas transformaciones geométricas. Por lo tanto, para actualizar el valor de la matriz de transformación del *bounding box* del objeto, solamente tenemos que aplicarle el valor de la matriz de transformación del objeto.

A nivel de implementación, quedaría de la siguiente forma:

```
function updateBBOX(object, bbox) {  
    object.updateMatrix()  
    bbox.matrix.copy(object.matrix)  
}
```

Para asegurarnos de que el objeto tenga la matriz de transformación actualizada, usamos `updateMatrix()` y después aplicamos la misma matriz del objeto a la matriz de transformación del *bounding box*. Con ello, conseguimos que la información de posicionamiento, tamaño y rotación sea la misma que tiene el objeto en todo momento.

El resultado final es una *bounding box* precisa como en el ejemplo con contorno naranja de la figura 32. Gracias a este elemento, podemos comprobar si un objeto está en una posición válida o no en función del tipo de objeto del que estemos hablando mediante la función `checkValidPosition()` que se ejecuta también dentro de la función de evento *mousemove*.

Esta función comprueba los límites de cada uno de los objetos en función de su tipo, como se explica a continuación.

9.2.6.2. Límites de objetos *floor*

Si el objeto seleccionado es de tipo *floor*, se utilizarán los cuatro vértices del *bounding box* con el valor en el eje *y* más bajo. De esta forma, tendremos los 4 vértices del *bounding box* del objeto que están en el suelo.

Una vez obtenidos estos 4 puntos, utilizaremos los puntos del suelo que se intersecciona mediante el *raycaster*. Este conjunto de puntos está ordenado según se forma el polígono de la geometría del suelo.

Una vez tengamos los 4 puntos del objeto y los puntos que forman parte de la geometría del suelo interseccionado (a los que, a partir de ahora, denominaremos

pointsObject y *pointsFloor*, respectivamente), se tiene que realizar un algoritmo que compruebe que todos los puntos están dentro del polígono.

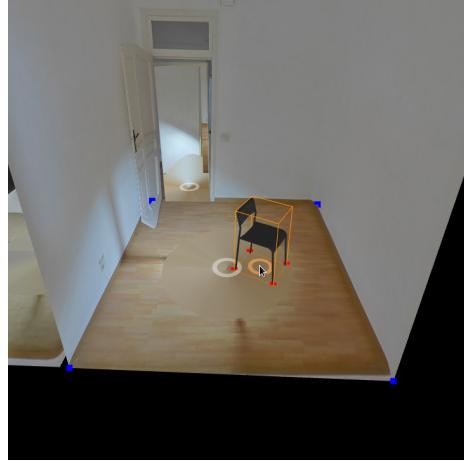


Figura 33: Representación de los *pointsObject* (en rojo) y los *pointsFloor* (en azul) en un objeto de tipo *floor*. Elaboración propia.

Para ello, se realizó un algoritmo que consta de los siguientes pasos:

- Por cada punto p_i de *pointsObject*:
 - Generar un punto *aux* con las mismas características que p_i pero con el valor de x *infinito*.
 - Crear el segmento que vaya desde p_i hasta *aux*.
 - Inicializar un contador de intersecciones = 0.
 - Por cada segmento $f_j f_{j+1}$ de *pointsFloor* (incluido el segmento que va desde el último punto hasta el primero):
 - Comprobar si el segmento p_i *aux* intersecciona con el segmento $f_j f_{j+1}$ mediante el uso de la función de determinante, la cual indica en qué lado de un segmento se encuentra un punto en el plano *xz*:

```
function determinant(p, q, r) {
    return (q.x - p.x) · (r.z - p.z) -
           (q.z - p.z) · (r.x - p.x);
}
```

Si devuelve un valor positivo, el punto *r* está a la derecha, si es negativo, a la izquierda, y si es 0, el punto está en la linea que forman *pq*. Se utilizan los valores de *x* y *z* porque el suelo está en el plano *xz*.

- A continuación se calculan las siguientes variables.

```
var det1 = determinant(f[j], f[j+1], p[i])
var det2 = determinant(f[j], f[j+1], aux)
var det3 = determinant(p[i], aux, f[j])
var det4 = determinant(p[i], aux, f[j+1])
```

Para cubrir todas las posibilidades de cruce entre los 4 *endpoints* de los segmentos.

- Si $\det1 * \det2$ es negativo y $\det3 * \det4$ es negativo, se afirma que un *endpoint* de un segmento está en un lado del otro segmento, y el otro *endpoint* en el lado opuesto, por lo tanto, hay una intersección respecto al plano *xz*. Entonces se incrementa el contador de intersecciones.
- Si en algún caso el número de intersecciones que hay en el contador es un número par, significa que o bien no se ha encontrado con ninguno de los segmentos del polígono *pointsFloor* o bien ha interseccionado con uno (entra en el polígono) y después con otro (sale del polígono) un número indeterminado de veces, por lo tanto, no está dentro del polígono y saldríamos de el algoritmo.
- En caso contrario, si ha interseccionado un número impar de veces, significa que el punto está dentro del polígono formado por *pointsFloor* y se seguiría comprobando que los demás puntos de *pointsObject* estén dentro.
- Si finaliza la comprobación con los 4 puntos de *pointsFloor*, afirmamos que el objeto está dentro del suelo.

Por lo tanto, si el algoritmo devuelve para todos los puntos de *pointsFloor* un número impar de intersecciones, diremos que el punto está dentro, en cualquier otro caso, está fuera. Si alguno de los cuatro vértices que forman *pointsObject* está fuera, se considerará una posición no válida para el objeto de tipo *floor*.

El coste del algoritmo es lineal en el número de vértices de *pointsFloor*, dado que recorremos 4 veces (una por cada elemento de *pointsObject*) los vértices que tiene *pointsFloor*.

Pero esto no es suficiente para afirmar que un objeto está dentro dentro o no de un suelo de un *tour virtual*, dado que tenemos que generar otro algoritmo que compruebe que ningún segmento de los *pointsObject* intersecciona con algún segmento que forma parte del polígono del suelo formado por *pointsFloor* en el plano *xz*, dado que podría darse el caso de que ningún vértice de *pointsObject* está fuera del polígono *pointsFloor* pero siga siendo una posición no válida, como ocurre en vértices cóncavos como en el ejemplo a continuación. 34.



(a) Objeto “Alfombra” con todos los *pointsObject* dentro del polígono que forma el suelo, pero claramente en una posición no válida.



(b) Objeto “Alfombra” con una comprobación correcta de su posición válida.

Figura 34: Representación de la comprobación que hace falta para que el algoritmo sea robusto en todos los posibles casos. Elaboración propia.

Por lo tanto, si con el algoritmo anterior llegamos a la conclusión de que los 4 *pointsObject* están dentro de *pointsFloor*, haremos uso de la misma función que detecta intersecciones (descrita en el anterior apartado) mediante el determinante por cada segmento de *pointsObject* y cada segmento de *pointsFloor*.

Para realizar este algoritmo, primero se ha necesitado ordenar en el sentido de agujas del reloj (*clockwise*) los puntos de *pointsObject* para obtener los segmentos que lo forman. Para ello, se ha gestionado mediante el algoritmo:

- Generar el punto *center* del cuadrilátero formado por los 4 puntos de *pointsObject* (simplemente $x = anchura/2$ y $z = profundidad/2$)
- Por cada punto p_i de *pointsObject*:
 - Generar el ángulo que forman p_i y *center* y asignárselo a p_i .

```
var angulo = Math.atan2(p[i].z - center.z, p[i].x - p[i].z)
```

- Ordenar los puntos respecto a su ángulo.

Una vez tengamos los puntos de *pointsObject* ordenados de esta manera, sabremos que los segmentos que lo forman serán $p_0\ p_1$, $p_1\ p_2$, $p_2\ p_3$ y $p_3\ p_0$

Entonces, podemos comenzar el algoritmo:

- Por cada segmento de *pointsObject* $p_i p_{i+1}$:
 - Por cada segmento de *pointsFloor* $f_j f_{j+1}$:
 - Comprobar si hay intersección mediante el uso del determinante (explicado previamente). Si existe una intersección con el segmento de *pointsObject* y el de *pointsFloor*, directamente salimos del algoritmo y decimos que la posición no es válida.
- Si acaba la ejecución del algoritmo por todos los puntos de *pointsObject*, entonces no hay intersecciones y la posición es válida.

El coste de este algoritmo también es lineal respecto al número de vértices del suelo, dado que el número de vértices de *pointsObject* siempre será una constante de 4 unidades.

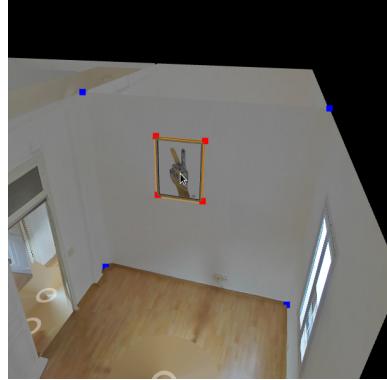
9.2.6.3. Límites de objetos *wall* y *wallFloor*

Para los objetos de tipo *wall* y de tipo *wallFloor* se sigue la misma estructura para comprobar si están en una posición válida o no, dado que se diferencian únicamente a la hora de posicionarse en una pared del tour.

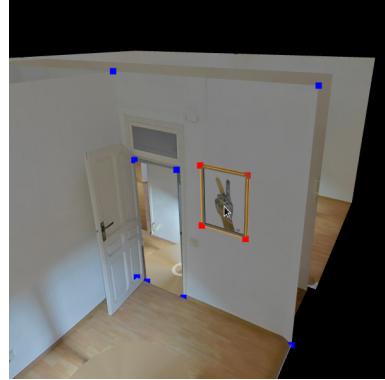
Hay que tener en cuenta que estos dos objetos van pegados a una pared y no a un suelo, como lo era en el caso de los objetos *floor*. También es importante añadir que el suelo siempre está en un plano paralelo al plano *xz*, y en este caso, no es suficiente comprobando si existen intersecciones en un único plano dado que las paredes son planos verticales que pueden estar en cualquier rotación en el eje *y*.

Para realizar el algoritmo, en este caso, se ha necesitado hacer algo parecido a lo que se hace con los objetos de tipo suelo, pero en lugar de usar el determinante entre puntos respecto al plano *xz*, se hace respecto *xy* y *zy*, de esta forma cubrimos todas las posibilidades.

Como en los objetos de tipo *floor*, primero debemos de identificar los 4 *pointsObjects* que se van a evaluar para saber si están o no dentro de la pared, en este caso. Para ello, utilizaremos los 4 vértices del *bounding box* que están en el mismo plano que la pared. Hay que tener en cuenta que las paredes pueden tener diversas formas y las puertas pueden delimitar su geometría poligonal, como se muestra en la segunda imagen de la siguiente figura:



(a) Objeto “Cuadro” en una pared.



(b) Objeto “Cuadro” en una pared con puerta.

Figura 35: Representación de los *pointsObject* (en rojo) y los *pointsWall* (en azul) en un objeto de tipo *wall*. Elaboración propia.

Para ello, simplemente calcularemos cuál es la cara del *bounding box* del objeto paralela a la pared (la frontal o posterior) cuya distancia es más pequeña a la pared. De esta forma se consigue obtener los 4 *pointsObject* acordes al *bounding box* que están pegadas a la pared.

También necesitaremos utilizar los *pointsWall*, que no son más que los vértices de la geometría de la pared interseccionado por el *raycaster*.

Una vez tengamos los 4 puntos del objeto y los puntos que forman parte de la geometría del suelo interseccionado, se tiene que realizar un algoritmo que compruebe que todos los *pointsObject* están dentro del polígono descrito por la pared. A partir de ahora, se referirán a estos puntos como *pointsObject* y *pointsWall*.

El algoritmo resultante, sigue los siguientes pasos:

- Por cada punto p_i de *pointsObject*:
 - Generar un punto *aux* que es perpendicular a la normal del punto de la pared interseccionado con *raycaster*. De esta manera, obtendremos un punto que está en el mismo plano que la pared. El punto se genera desde p_i y recorre una distancia suficientemente grande en dirección hacia la perpendicular a la normal.
 - De manera similar al algoritmo para objetos de tipo suelo, se traza un segmento desde p_i hacia *aux*.



Figura 36: Representación de la normal (línea azul) y segmento perpendicular a la normal (línea roja) que se ha utilizado como dirección para hacer el segmento desde p_i hacia aux . Elaboración propia.

- Inicializar contador de intersecciones = 0.
- Por cada segmento $w_j w_{j+1}$ de $pointsWall$ (incluido el segmento que va desde el último punto hasta el primer punto):
 - Comprobar si el segmento $p_i aux$ intersecciona con el segmento $w_j w_{j+1}$ mediante el uso de la modificación de la función que calcula el determinante, tendremos estas dos funciones:

```

function determinantXY(p, q, r) {
    return (q.x - p.x) * (r.y - p.y) -
           (q.y - p.y) * (r.x - p.x)
}

function determinantZY(p, q, r) {
    return (q.y - p.y) * (r.z - p.z) -
           (q.z - p.z) * (r.y - p.y)
}
  
```

Las cuales calculan si un punto está a la derecha o izquierda (dependiendo del signo del valor resultante) de un segmento pero en los planos xy y zy , respectivamente. Teniendo en cuenta ambos planos, será suficiente para concretar si una pared, en cualquier rotación sobre el eje y , contiene el punto p_i o no.

- Se calculan las siguientes variables:

```

var det1 = determinantXY(w[j], w[j+1], p[i])
var det2 = determinantXY(w[j], w[j+1], aux)
var det3 = determinantXY(p[i], aux, w[j])
  
```

```

var det4 = determinantXY(p[i], aux, w[j+1]})

var det5 = determinantZY(w[j], w[j+1], p[i]))
var det6 = determinantZY(w[j], w[j+1], aux))
var det7 = determinantZY(p[i], aux, w[j]))
var det8 = determinantZY(p[i], aux, w[j+1]))

```

```

var condition1 = det1*det2 < 0 && det3*det4 < 0;
var condition2 = det5*det6 < 0 && det7*det8 < 0;

```

Como en el algoritmo de los objetos de tipo floor, si `condition1` es cierta, hay intersección del segmento p_i `aux` con el segmento w_j w_{j+1} en el plano xy , y si `condition2` es cierta, la hay respecto el plano zy . Con que una de las dos condiciones se cumpla, es suficiente para incrementar el valor del contador de intersecciones

- Si el contador de intersecciones es par, saldremos del algoritmo alegando que el punto p_i está fuera del polígono que forma la geometría de `pointsWall`, en cualquier otro caso, seguiremos buscando el número de intersecciones que tiene el siguiente punto.
- Si hemos llegado hasta aquí, significa que no hemos salido del algoritmo por lo cual no hay un número impar de intersecciones en ninguno de los puntos de `pointsObject` y podemos afirmar que el objeto está dentro del polígono representado por la pared interseccionada.

Este algoritmo tiene coste lineal en función de los vértices de la pared, igual que el anterior algoritmo. También hay que tener en cuenta que no nos hace falta calcular si alguno de los segmentos de `pointsObject` intersecciona directamente con alguno de los segmentos de `pointsWall`, dado que, por definición de *bounding box*, las líneas que forman `pointsObject` serán paralelas dos a dos con el plano de la pared interseccionada, por lo tanto, es imposible que un segmento de `pointsObject` interseccione de forma no paralela con uno de `pointsWall` y no tiene sentido ejecutar este algoritmo.

Pero, dado que los objetos de tipo `wall` o `wallFloor` pueden llegar a ocupar mucho espacio, también hace falta que se calcule si el objeto cabe dentro del polígono que representa al suelo de la habitación en la que se encuentra la pared interseccionada. En la imagen siguiente se puede ver como tenemos un objeto de tipo `wallFloor` que debería dar posición no válida dado que no entra dentro de los límites del suelo, como se ve en la siguiente imagen.

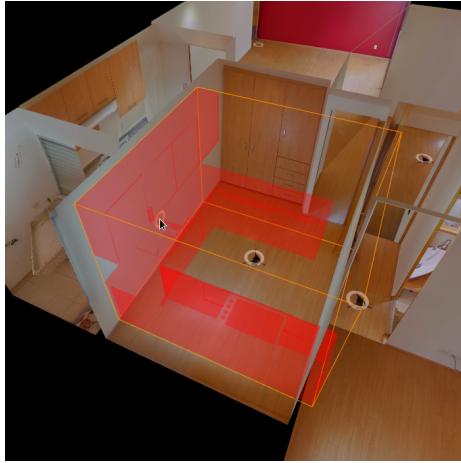


Figura 37: Ejemplo del objeto de tipo *wallFloor* “Cocina completa 1”, en el que se ve que la parte inferior del objeto no cabe en el suelo. Elaboración propia.

Es por esta razón que se utiliza el mismo algoritmo que se usa para los objetos de tipo suelo exactamente igual que en el punto 9.2.6.2, esta vez utilizando el suelo que forma parte de la habitación a la que pertenece la pared, cogeremos los vértices que forman su geometría y se usarán como *pointsFloor*. Y se utilizarán los *pointsObject* de la misma manera que lo hacemos con los objetos de tipo *floor*, los 4 con un valor en *y* más bajo.

Teniendo la información de los valores de *pointsFloor* y *pointsObject* de esta manera, solo hace falta llamar a la misma función del algoritmo para los objetos de tipo suelo. Si nos retorna que está en una posición válida, el objeto será válido, en cualquier otro caso, se invalidará su posición.

9.2.6.4. Límites de objetos *item*

Los objetos de tipo *item* actúan igual que los de *floor* pero en lugar de yacer en un suelo, lo hacen encima de otro objeto. Es por esta razón por la que los *item* utilizan un algoritmo distinto, aprovechando que la superficie en la que se ponen tiene una geometría convexa.

Dado que la superficie en la que se pone un objeto *item* es la cara superior del *bounding box* de un objeto, tiene 4 vértices y es una geometría convexa, podemos utilizar el siguiente algoritmo:

Primero cogemos los puntos del objeto tipo *item* cuyos valores en *Y* sean mínimos, estos serán los 4 vértices del *bounding box* de la cara inferior, los cuales harán de *pointsObject*. También necesitaremos coger los 4 puntos con

valor y mas grande del objeto interseccionado (siempre y cuando la normal del punto en el objeto sea $(0,1,0)$). A estos puntos les llamamos *pointsFloor*.

- Por cada punto p_i de *pointsObject*:

- Cada dos vértices f_j y f_{j+1} consecutivos del polígono que forma *pointsFloor*:

- Comprobar en qué lado se encuentra el punto p_i respecto al segmento $f_j f_{j+1}$ mediante el uso del determinante para detectar la orientación.

```
function determinant(p, q, r) {
    return (q.x - p.x) · (r.z - p.z) -
           (q.z - p.z) · (r.x - p.x);
}
```

- Se calcula:

```
var det = determinant(f[j], f[j+1], p[i])
```

Si devuelve un valor positivo, el punto p_i está a la derecha del segmento $f_j f_{j+1}$, si es negativo, a la izquierda, y si es 0, el punto está en el propio segmento. Se utilizan los valores de x y z porque la superficie de la cara superior del *bounding box* del objeto está en el plano xz .

- Si el valor del signo cambia en una de las iteraciones, se asume que está fuera directamente y pasamos a finalizar el algoritmo alegando que hay al menos un vértice de *pointsObject* fuera de la geometría de *pointsFloor*.
- Si todos los lados coinciden (todos derecha o todos izquierda) se puede afirmar que el punto p_i está dentro del polígono que forma *pointsFloor*. Y si todos los puntos de *pointsObject* lo hacen, es que el objeto tiene todos los vértices dentro de la geometría que forma la cara superior del *bounding box* del objeto.

Inicialmente se consideró este algoritmo también para los anteriores puntos, pero rápidamente se desechó esa idea dado que en Floorfy existen inmuebles de varias características, entre ellas, viviendas con habitaciones de suelos y paredes con geometría no convexa, y este tipo de algoritmo no funciona correctamente con polígonos no convexos.

Este algoritmo, en este caso, tiene un coste constante dado que conocemos el número de *pointsObject* y *pointsFloor*, que es de 4 vértices cada uno y no depende de la entrada.

9.2.6.5. Colisión con otros objetos

Dentro de la función de evento *mousemove* también se tiene en cuenta si un objeto colisiona o no al tener un objeto navegando por la escena. De esta condición también depende si un objeto estará en una posición válida o inválida.

Para detectar la colisión entre objetos de la escena se utiliza una estructura donde se guarda la información de todos los objetos añadidos a la escena. Cada vez que un objeto se arrastra por la escena, ya sea nuevo o modificado, comprobaremos si tiene colisión con algún objeto de esta estructura que no sea él mismo.

Para ello, se hará uso de la función `checkCollision(vertices1, vertices2)`, donde `vertices1` son los vértices que forman la geometría *bounding box* del objeto que se está comprobando su posición válida y `vertices2` es la misma información pero de uno de los objetos guardados en la estructura. Esta comprobación se efectuará por cada uno de los objetos que ya están en la escena.

Inicialmente, para detectar la colisión entre objetos se utilizó una función simple y efectiva para cuando se utilizaba AABB (explicado en el punto 9.2.6.1). Esta función tenía la siguiente forma:

```
function checkCollision(vertices1, vertices2) {
    return (vertices1.minX <= vertices2 maxX && a.maxX >= vertices2.minX) &&
           (vertices1.minY <= vertices2 maxY && vertices1.maxY >= vertices2.minY) &&
           (vertices1.minZ <= vertices2.maxZ && vertices1.maxZ >= vertices2.minZ);
}
```

Dado que en AABB todos los puntos están alineados con los ejes (x , y , z), esta función funcionaba a la perfección para este tipo de casos. La función tiene como entrada en `vertices1`: los puntos máximos y mínimos del AABB de un objeto y en `vertices2` la misma información de otro objeto. Simplemente, la función comprueba que ningún punto esté solapado en la AABB del otro objeto.

Pero, como ya se ha explicado en el punto 9.2.6.1, este método de utilizar la *bounding box* que viene por defecto en *three js* no es para nada precisa y no es lo que nosotros buscamos en el proyecto. Entonces, se consideró un método para comprobar las colisiones para el nuevo tipo de *bounding box* más acorde y robusto que el anterior.

El algoritmo se basa en dos partes: la primera, se fundamenta en comprobar si alguno de los segmentos del *bounding box* del objeto A intersecciona con alguna de las caras del *bounding box* del objeto B (y viceversa), donde se aplica un algoritmo de detección de intersección entre segmento y plano.

A continuación se puede ver como en la colisión entre estos dos objetos, el objeto con *bounding Box* naranja tiene segmentos dentro interseccionando caras del objeto con *bounding box* amarilla y viceversa, por lo tanto, hay colisión.

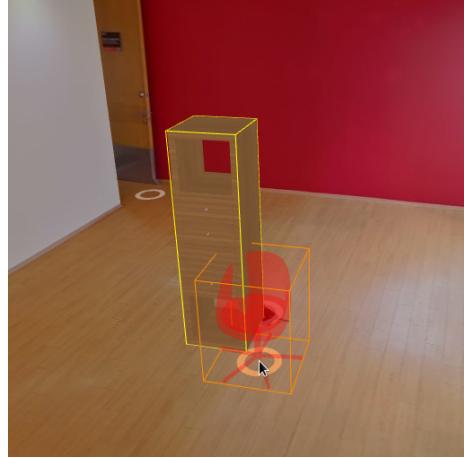


Figura 38: Representación de una colisión entre dos objetos. Elaboración propia.

La segunda parte es comprobar que no haya ningún vértice del *bounding box* del objeto A en el *bounding box* del objeto B.

Para la primera comprobación, este sería el pseudocódigo del algoritmo empleado:

- Por cada cara $cara_i$ del *bounding box* de A:
 - Por cada segmento $segmento_j$ del *bounding box* de B:
 - Se calcula la ecuación del plano de $cara_i$ y la ecuación paramétrica de la línea que forma el $segmento_j$.

Dado que estos dos elementos son infinitos, siempre existirá un punto de intersección entre la línea y el plano (siempre y cuando la línea no sea paralela al plano). A partir de esta información encontramos el punto de intersección entre el plano y la línea.

La forma paramétrica de la línea se encuentra mediante los valores de x , y , z de los dos puntos que forman el segmento y a continuación hay un ejemplo de su estructura.

$$\begin{cases} x = -t + 4 \\ y = 4t - 3 \\ z = 4t - 2 \end{cases}$$

Con 3 puntos es suficiente para encontrar la ecuación de un plano. Para ello, se han utilizado 3 puntos de $cara_i$, un ejemplo sería:

$$4x - 13y + 23z = 45$$

Para encontrar ese punto de intersección entre la línea y el plano, únicamente substituimos los valores de (x, y, z) de la forma paramétrica de la línea en los valores (x, y, z) de la ecuación del plano. En el ejemplo anterior, sería de esta forma:

$$4x - 13y + 23z = 45 \quad (1)$$

$$4 * (-t + 4) - 13 * (4t - 3) + 23 * (4t - 2) = 45 \quad (2)$$

Despejamos la variable t:

$$t = 1 \quad (3)$$

Entonces, para $t = 1$ el punto pertenece a la linea y al plano a la vez. Finalmente, substituimos el valor t en la forma paramétrica de la línea, por lo tanto:

$$\begin{cases} x = -1 + 4 \\ y = 4 - 3 \\ z = 4 - 2 \end{cases}$$

$$\begin{cases} x = 3 \\ y = 1 \\ z = 2 \end{cases}$$

El punto en el espacio 3D donde se produce la intersección del plano y la línea es el $(3, 1, 2)$. Ahora hay que comprobar que este punto pertenece al $segmento_j$ y a la $cara_i$, dado que el plano y la línea son componentes infinitos y es posible que el punto encontrado como punto de intersección no pertenezca a nuestro $segmento_j$ ni a nuestra $cara_i$.

Se ha decidido explicar la intersección entre plano y línea con un ejemplo para hacerlo más ilustrativo.

- Comprobamos que el punto de la intersección pertenece al $segmento_j$ haciendo uso del siguiente método:

```
function onSegment(p,q,r) {
    return q.x <= Math.max(p.x, r.x) && q.x >= Math.min(p.x, r.x) &&
           q.y <= Math.max(p.y, r.y) && q.y >= Math.min(p.y, r.y) &&
           q.z <= Math.max(p.z, r.z) && q.z >= Math.min(p.z, r.z);
}
```

Básicamente comprobamos que q está en el segmento pr mediante los valores de cada una de las variables x, y, z . Si el valor de una

de las coordenadas está entre el máximo valor de esa coordenada en q o r y el mínimo valor de esa coordenada en p o r , se afirma que el valor de esa coordenada del punto q pertenece al segmento.

Esta comprobación se hace por las 3 variables x , y , z . Si todas cumplen esta condición, está dentro del segmento, en cualquier otro caso, está fuera.

En el caso de que obtengamos que el punto no pertenece a $segmento_i$, directamente saldríamos de la función alegando que no hay intersección entre $segmento_j$ y $cara_i$. En caso contrario, tendremos que comprobar que el punto también pertenece a $cara_i$ con el siguiente método [38]:

- Dado un punto i que es el punto de intersección entre línea y plano calculado previamente, y los cuatro vértices de la geometría de $cara_i$: c_0, c_1, c_2, c_3 .

Calcularemos la suma de las áreas de los triángulos formados por: $\Delta i c_0 c_1, \Delta i c_1 c_2, \Delta i c_2 c_3, \Delta i c_3 c_0$.

Si la suma obtenida es igual al área que forma $cara_i$, el punto i está dentro de la superficie de $cara_i$ y por lo tanto existe un punto en el que el segmento $segmento_j$ y la superficie $cara_i$ interseccionan y finalizaría el algoritmo dado que ambas *bounding boxes* colisionan. En cualquier otro caso, no lo está y por lo tanto no existe intersección entre ambos elementos.

- En el caso de encontrar una sola intersección entre $cara_i$ y $segmento_j$, el algoritmo parará y devolverá que existe una colisión y por lo tanto, la posición del objeto seleccionado no es válida.
- Si el algoritmo se ejecuta hasta el final sin encontrar ninguna intersección, la posición es válida.

Con tal de que la comprobación de colisión entre objetos sea robusta, hace falta comprobar que no existe ningún vértice del *bounding box* del objeto A en el *bounding box* del objeto B, dado que podría suceder el extraño caso en el que la *bounding box* de un objeto está justo dentro de la *bounding box* de otro objeto y comprobando mediante el anterior algoritmo no obtendríamos colisión entre la *bounding box* del objeto A y la *bounding box* del objeto B cuando en realidad están colisionando.

Para comprobar que no existe ningún vértice del *bounding box* del objeto A dentro del *bounding box* del objeto B seguiremos el planteamiento del algoritmo

9.2.6.2, en el cual trazábamos una línea y contábamos el número de intersecciones entre los segmentos del polígono, pero esta vez, en tres dimensiones. Para ello, haremos:

- Por cada vértice v_i de *bounding box* de A:
 - Inicializar contador de intersecciones = 0.
 - Generamos el punto *aux*, que es igual que v_i pero con el valor en la coordenada x aumentado al máximo valor numérico para asegurarse de que está completamente fuera del *bounding box* de B.
 - A continuación se comprueba por cada $plano_j$ del *bounding box* de B:
 - Si el $plano_j$ intersersecciona con el segmento de v_i hasta *aux*, utilizando la misma función que se ha descrito en el algoritmo anterior 9.2.6.5 en los puntos 1, 2, 3, 4 y 5, pero en el caso de que encuentre una intersección incrementará en una unidad el valor del contador de intersecciones.
 - Finalmente, si el algoritmo encuentra en un vértice v_i un número impar de intersecciones, podemos decir que el punto perteneciente al *bounding box* de A está dentro del *bounding box* de B. Dado que únicamente intersecciona una vez, y asumiendo que el punto *aux* está completamente fuera, para pasar un número impar de veces tiene que haber estado dentro del otro *bounding box*.
- En caso de que el algoritmo finalice sin encontrar ningún un vértice v_i interseccionando un número impar de veces con todos las caras que representan el *bounding box* de B, afirmaremos que no hay colisión entre el objeto A y B.

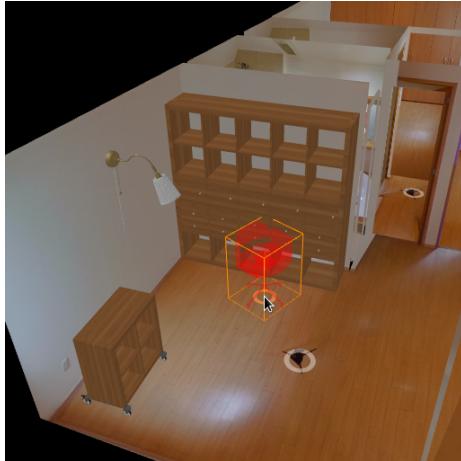


Figura 39: Ejemplo de un objeto colisionando en una escena del *tour virtual*. Elaboración propia.

Para concluir este apartado, hay que recordar que todo este proceso de comprobar si una posición es válida o no se hace siempre que se efectúa un nuevo cambio en las transformaciones geométricas de un objeto, es decir, al trasladar un objeto por el escenario (Ya sea cargando desde el catálogo o moviendo un objeto por toda la escena), o bien escalándolo (Ya sea mediante los controles de ratón explicados en el punto 9.2.7.1, o mediante el *slider* que modifica el tamaño del objeto seleccionado, en el punto 9.2.7.2) o rotándolo (mediante el click derecho cuando hay un objeto seleccionado 9.2.7.1, o mediante el círculo de rotación de los controles, explicado en 9.2.7.10).

En caso de dar una posición no válida, no se realiza el cambio y el objeto no puede estacionarse en esa posición. En el caso de que sea una posición válida, se efectúa el cambio y el objeto yace en esa posición con las transformaciones geométricas que se le han realizado.

9.2.7. Controles

Dado que el aplicativo es interactivo, se han tenido que diseñar e implementar una serie de controles para que el usuario pueda moldear el inmueble a su gusto de forma intuitiva. Un elemento importante en los controles son las transformaciones geométricas. Las transformaciones geométricas no son más que la o las operaciones geométricas que permiten crear una nueva figura a partir de una previamente dada.

A continuación se explican las transformaciones geométricas que se han im-

plementado [39], pese a que *three js* facilita mucho la implementación, se verá cómo es que *three js* lo recrea internamente. Esta parte del proyecto está relacionada con la tarea M3D.4 - Controles de transformaciones geométricas.

Partimos de la base de que *three js* utiliza matrices para codificar las transformaciones tridimensionales de un objeto (traslación, rotación y escalado). Toda instancia de Object3D tiene una matriz 4x4 (matriz de transformación) la cual guarda la información de posición, rotación y escalado [40].

Para efectuar un cambio en su transformación, se deberá multiplicar esa matriz de transformación por una de las matrices a continuación, en función de su cambio deseado, ya sea translación, rotación o escalado. Hay que recordar que todas las transformaciones se realizan respecto al punto de origen o *Pivot point*.

- Traslación. Esta transformación geométrica se basa en cambiar la posición en el espacio de un objeto. Internamente, se multiplica la matriz del objeto por la siguiente matriz:

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

Donde t_x , t_y y t_z son las coordenadas x , y , z de la nueva posición. Así cuando hagamos la operación `object.position.set(x, y, z)` se cambiará su posición a la indicada por x , y , z .

Esta transformación geométrica se realiza al colocar el objeto desde catálogo y al mover un objeto que ya está posicionado.

- Rotación. En este aplicativo, únicamente se podrá controlar la rotación en el eje y de los objetos de tipo *floor* e *item*, dado que los de tipo *wall* y *wallFloor* no tiene sentido que roten en ese eje porque su rotación se actualiza dependiendo de la pared y si se rotase en cualquier movimiento daría una posición no válida. Tampoco hay rotación en los ejes x y z dado que todos los objetos están enderezados hacia el eje y y no tendría sentido tumbarlos hacia ninguno de esos ejes.

Para efectuar una rotación en el eje y , internamente, *three js* multiplica la matriz de transformación del objeto por la siguiente:

$$R_{(\alpha)} = \begin{bmatrix} \cos(\alpha) & 0 & \sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Donde α es el ángulo en grados que se desea rotar en el eje y . Esta transformación geométrica se realiza mediante los controles de ratón explicados

en 9.2.7.1 y el control de rotación incorporado para cuando un objeto ya está en la escena, explicado en 9.2.7.10. La función que se utiliza en *three js* es `object.rotationY(α)`.

- Escalado. El escalado cambia el tamaño del objeto. En esta herramienta, se puede escalar un objeto uniformemente manteniendo la relación de aspecto, es decir, siempre que se escala, se escala con el mismo valor en x , y , z . Para cambiar la matriz de un objeto con tal de escalarlo, lo que hace *three js* internamente es multiplicar la matriz del objeto por la siguiente matriz:

$$S = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Donde S_x , S_y y S_z son los tres valores de escala de cada una de las coordenadas del objeto. En el caso de este aplicativo, $S_x = S_y = S_z$.

Esta modificación la podremos hacer mediante los controles básicos de ratón (con la rueda del ratón, profundamente explicado en el siguiente apartado) y mediante el *slider* implementado en el apartado de Object Options 9.2.7.2.

La función con la cuál se interactúa en el código es `object.scale.set(S_x , S_y , S_z)`.

A continuación se explican cada uno de los controles que hay disponibles al seleccionar un objeto de la escena.

9.2.7.1. Controles básicos mediante ratón

Estos controles están disponibles únicamente cuando agarramos un objeto mediante el click izquierdo o cuando se selecciona un objeto desde catálogo. Se dividen en 3.

- Traslación. Al mover el ratón se desplazará el objeto por la escena, en el caso de interseccionar correctamente con un objeto o suelo o pared correspondiente, se comprobará si está en una posición válida o no. Principalmente se multiplica la matriz del objeto seleccionado por la matriz de traslación explicada en el punto anterior con los valores de x , y , z iguales a la posición del ratón en el punto interseccionado mediante el *raycaster*. *Three js* permite actualizar la posición de un objeto mediante la operación: `object.position.set(x, y, z)`, pero internamente se hace mediante las matrices de transformaciones geométricas.

Mediante el doble click el objeto se coloca en la posición deseada. En el caso de ser una posición inválida, el objeto seguirá seleccionado en el ratón y no dejará de estarlo hasta que se haga doble click en una posición válida.

- Rotación. Cuando el objeto es seleccionado, podremos hacerle una rotación en 45° del eje *y* en el sentido de las agujas del reloj si hacemos click derecho. Eso es solo posible para objetos de tipo *floor* y tipo *item*.

En *three.js* se realiza ejecutando la siguiente línea de código:

```
object.rotationY(45)
```

Pero internamente se hace mediante las matrices de transformaciones geométricas.

- Escalado. El tamaño del objeto seleccionado se puede cambiar mediante la rueda del ratón. El movimiento es bastante intuitivo, por cada grado incrementado en la rueda, el tamaño del objeto sube un 5% en su tamaño original. Mientras sube o baja su tamaño, el valor en el *slider* se verá reflejado en función de la escala que tenga.

La función con la cual se interactúa en el código es:

```
object.scale.set(x, y, z)
```

Pero internamente se hace mediante las matrices de transformaciones geométricas. En este caso, como se mantiene la relación de aspecto, $x = y = z$, para evitar deformaciones.

Después de realizar una de estas transformaciones geométricas, se comprueba si el objeto sigue estando en una posición válida o no.

9.2.7.2. *Slider*

El slider es un elemento de HTML que sirve para escoger un valor entre un rango de valores. El que se utiliza en esta herramienta, se muestra en las Object Options al seleccionar un objeto. El valor que tiene por defecto está al 50% y cada vez que el usuario ajusta su valor (ya sea mediante *slider* o mediante el control básico de la rueda del ratón), el *slider* actualiza su valor.

Si hacemos click en un objeto con tamaño distinto al original, el *slider* se mostrará actualizado de todas maneras dado que coge el valor del tamaño del objeto seleccionado.

Como en el control de la rueda del ratón, el valor del tamaño del objeto seleccionado tiene un mínimo y un máximo, y cada valor que aumenta/decremente en el *slider* es un 5% más o menos que tiene el valor de la escala del objeto.

De la misma manera que con la rueda, se utiliza la función `object.scale.set(x, y, z)`, pero internamente se hace mediante las matrices de transformaciones geométricas.



Figura 40: Representación de la modificación del tamaño mediante el uso del *slider*. Elaboración propia.

9.2.7.3. Color

El color es otro elemento de HTML, mediante él se puede seleccionar un color de forma interactiva, que luego en código se refleja con su valor en hexadecimal. Este elemento aparecerá en las Object Options junto con las demás opciones una vez el objeto sea seleccionado.

En cuanto se hace click en un objeto, el color que aparece en este selector de color será igual al de los layers del objeto que seleccionamos previamente, explicado en el punto 9.2.2.2. Al cambiar este color, los layers del objeto, que no son más que una serie de subobjetos que forman el objeto seleccionado, cambiarán su color al color seleccionado.

El proceso, mediante código, es el siguiente:

```
var colorRGB = hexToRGB(colorpicker.value)
object.children[layer].material.color.r = normalizeRGB(colorRGB.r)
object.children[layer].material.color.g = normalizeRGB(colorRGB.g)
object.children[layer].material.color.b = normalizeRGB(colorRGB.b)
```

Se recoge el valor del color mediante `colorpicker.value`, que es donde se almacena su valor una vez haya cambiado por el usuario. Este valor se cambia de hexadecimal a RGB mediante la función `hexToRGB()`, dado que en *three js* se ajusta el color del material de un objeto con las componentes RGB. Una vez transformado ese valor a RGB, lo asignamos al color del material de cada subobjeto que forma parte de layers normalizando el valor de cada componente (ya que *three js* lo reconoce de esta manera).

Se realiza por cada uno de los layers de un objeto. La variable `layer` hace referencia al layer actual, que en realidad se recorre de forma iterativa por cada uno de los layers asignados.



(a) Objeto “Comoda” con su valor de color por defecto.



(b) Objeto “Comoda” con el color negro.



(c) Objeto “Comoda” con el color amarillo.

Figura 41: Objeto “Comoda” con diferentes colores. Elaboración propia.

También se puede cambiar el color una vez la textura ha sido cambiada,

como en el siguiente ejemplo:



(a) Objeto “Comoda” con el color por defecto y la textura “Madera”.



(b) Objeto “Comoda” con el color anaranjado y la textura “Madera”.

Figura 42: Objeto “Comoda” con diferente color y textura. Elaboración propia.

9.2.7.4. Textura

De forma similar al color, se cambia la textura de los layers que tiene el objeto asignado. Este componente está recreado en el catálogo igual que los modelos, en filas de 3 y con miniaturas para hacerte una idea de cómo se verán. Como las demás opciones, el cambio de textura solo será posible efectuarlo en el caso de que un objeto sea seleccionado. En este apartado se refleja el trabajo de la tarea MAT.3 - Texturas

Para texturizar un objeto, *three.js* aplica internamente técnicas de *UV mapping*. El *UV mapping* es el proceso de modelado 3D de proyectar una imagen 2D en la superficie de un modelo 3D para el mapeo de texturas. Las letras *u* y *v* denotan el eje de la textura 2D porque *x*, *y*, *z* ya se utilizan para denotar el eje del objeto 3D. Lo que se hace es relacionar una coordenada de la textura (*u*, *v*) con un vértice del objeto (*x*, *y*, *z*). Desde el archivo .OBJ, se pueden encontrar estas coordenadas (*u*, *v*) en las coordenadas de textura representadas como *vt*, explicado en 9.2.1.

Mediante las coordenadas de textura de los objetos y una imagen es suficiente para aplicar una textura a un subobjeto, pero existen algunos objetos que carecen de las componentes *vt*. Estos objetos, por defecto no tienen un mapeado asignado a ninguna de su superficie. Es por ello que se ha tenido que generar las coordenadas (*u*, *v*) de algunos de los objetos del catálogo.

Las texturas son imágenes cuadradas que se aplican encima de una superficie

de un modelo 3D y se repite por todo el modelo. Están pensadas para que cuando acabe una textura en uno de los lados, el lado opuesto la continúe sin dar una sensación extraña. Al hacer lo siguiente, cambiará la textura de aquellos subobjetos del objeto seleccionado que formen parte de los layers del objeto:

```
var loader = new THREE.TextureLoader();
var tex = loader.load(url);
tex.wrapS = THREE.RepeatWrapping;
tex.wrapT = THREE.RepeatWrapping;
tex.repeat.set(2, 2);
for (var i = 0; i < layers.length; i++) {
    if (id == "none") object.children[layers[i]].material.map = null;
    object.children[layers[i]].material.map = tex;
}
```

En *three.js* se utiliza el método `TextureLoader` para cargar la imagen que hace referencia a la textura, `url` sería la dirección donde se encuentra dicha imagen. Al hacer el `load`, tendremos la información de esa imagen de textura en la variable `tex`. Utilizamos `RepeatWrapping` para *S* y *T* para que la textura se repita en vertical y en horizontal, respectivamente. De esa forma, la superficie del subobjeto será recubierta completamente por esa textura. Por último, a la textura `tex` se le hace un `repeat` con los valores 2 y 2. Esto se hace para que el tamaño de la textura encaje en esas unidades de las coordenadas *UV* del modelo. Se ha ido variando este parámetro hasta conseguir algo más realista en el tamaño de la textura.

Finalmente, en el bucle que recorre el array `layers`, se le asigna esta textura a los layers del objeto. Si la id de la textura es “none”, es decir, la textura representada con una línea diagonal roja, el objeto pierde esa textura y dará una sensación de tener un material totalmente plano simulando al plástico.



(a) Objeto “Cocina completa 2” con su valor de textura por defecto.

(b) Objeto “Cocina completa 2” con el valor de textura “Madera 1”.



(c) Objeto “Cocina completa 2” con el valor de textura “Madera 2”.

Figura 43: Objeto “Cocina completa 2” diferentes texturas. Elaboración propia.

9.2.7.5. Reset

El botón reset aparece en las Object Options y sirve para que un objeto cuya textura y/o color ha sido modificado, pueda volver a sus valores por defecto. A continuación se muestra un ejemplo de su uso.



(a) Objeto “Mesa” con la textura “Marmol” y color blanquecino.

(b) Objeto “Mesa” después del reset, con su valor por defecto sin textura y con color oscuro.

Figura 44: Ejemplo del uso del botón de reset en un objeto. Elaboración propia.

Esto es posible dado que al cargar un objeto se guardan los materiales de los layers, dado que son los únicos que se pueden modificar, y cuando el botón es usado, se cambia el material de todos los layers involucrados al material que traían por defecto.

9.2.7.6. Tour Controls

Cuando un objeto es seleccionado, se pierden los controles de visualización del *tour virtual*. Es por ello que se decidió poner este *switch* para cambiar entre los controles básicos de modificación del objeto seleccionado (transformaciones geométricas) y los del tour. Las principales diferencias son las siguientes:

- Rueda del ratón. Si las transformaciones geométricas están activadas, mediante la rueda del ratón se puede modificar el tamaño del objeto seleccionado. En el caso de que la rueda suba, el objeto incrementa en tamaño, y si va hacia abajo, decrementa. En el caso de que los controles del entorno estén activados, si la rueda sube, incrementará el zoom de la vista del *tour virtual* y si baja, decrementará.
- Click derecho. Cuando el objeto está seleccionado, al hacer click derecho del ratón, el objeto gira 45°. En cambio, si se hace click derecho desde los controles de visualización, se modificará la posición de la cámara al trasladar el click a otra posición.

- Click izquierdo. Con los controles de transformaciones geométricas, no ocurre nada. Pero usando los controles de visualización del entorno, podremos navegar entre los hotspots del habitáculo.
- Doble click. Con el objeto seleccionado, al hacer doble click el objeto se posicionará en la escena. En cambio, con los controles de entorno no ocurre nada.

9.2.7.7. Clonar

Cuando un objeto es seleccionado, si se hace click en el botón de clonar, aparecerá un nuevo objeto como si lo hubiésemos sacado desde catálogo en las mismas condiciones que el objeto seleccionado, es decir, con su tamaño, rotación (si la tiene) y el color y textura en los layers.

Esta pequeña funcionalidad es útil si estamos retocando un objeto por mucho tiempo y queremos tener una copia exactamente igual de él. Como en la imagen a continuación.



Figura 45: Ejemplo del uso del botón clone con varios taburetes con el mismo color y textura. Elaboración propia

En código, simplemente cargamos el mismo objeto seleccionado desde catálogo y le cambiamos la rotación, escala, y se le aplica el mismo material que tiene en el objeto seleccionado.

9.2.7.8. Lock in place

Cuando esta opción está activada en un objeto seleccionado, cuando se haga click para mover ese objeto, la posición no cambiará de lugar. Simplemente ayuda a evitar errores al trazar un click sobre ese objeto sin querer cuando estamos navegando por el *tour virtual*.

En código, simplemente se activa un booleano asignado al objeto que no permite mover el objeto cuando su valor sea cierto.

9.2.7.9. Eliminar

Se elimina el objeto seleccionado al apretar el botón. Como esta funcionalidad es destructiva, se ha decidido añadir una alerta de dialogo con una confirmación con tal de evitar que un usuario elimine un objeto sin querer. Si el usuario acepta esta confirmación, el objeto se eliminará para siempre de la escena, en el caso contrario, el objeto permanecerá en la escena.

En código se realiza de la siguiente manera:

```
scene.remove(object)
```

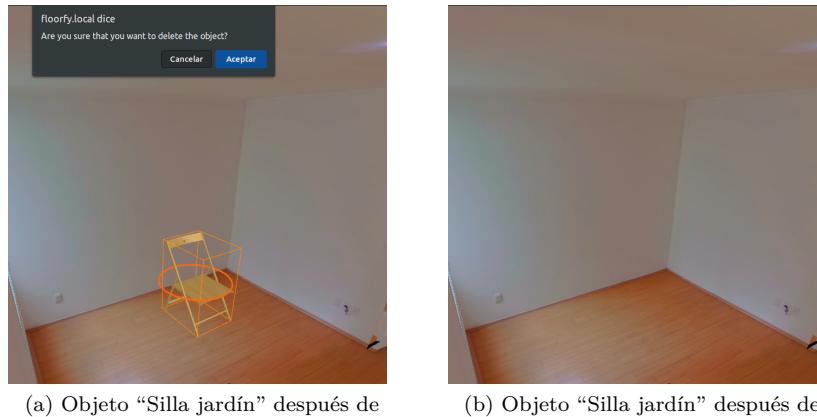


Figura 46: Ejemplo del uso del control de rotación con un objeto. Elaboración propia.

9.2.7.10. Rotación

Esta es el único control que aparece en la escena y no en la parte de Object Options. Aparece también cuando un objeto es seleccionado, a la vez que las Object Options.

Se trata de un círculo completo que está al rededor del objeto a media altura, tiene un rombo para facilitar su uso y es de color anaranjado como el *branding* de la empresa. Está hecho mediante el método de rotación de los **TransformControls** que incluye *three js*. Por defecto, tiene tres círculos con los colores correspondientes en cada eje (x , y , z), pero se ha modificado debidamente para que encaje con el aplicativo. Dado que no es necesario el eje en x y en z , se han eliminado y el círculo ahora es completo y de color naranja.

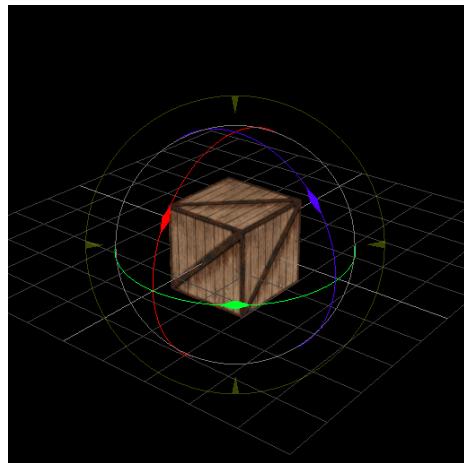


Figura 47: Ejemplo oficial de *three js* del TransformControl de rotación por defecto. Fuente: [14].

Lo que se hace internamente es realizar el producto de matrices con la matriz de rotación de las transformaciones geométricas, usando un ángulo que dependerá de la distancia recorrida del ratón en cuanto se hace click en el círculo.

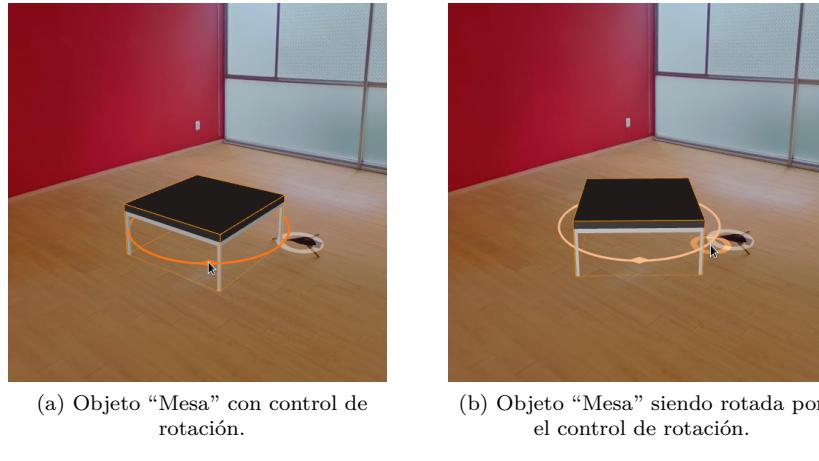


Figura 48: Ejemplo del uso del control de rotación con un objeto. Elaboración propia.

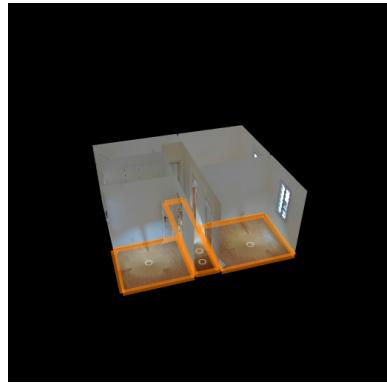
9.3. Paredes y suelos

A parte de la funcionalidad de amueblar inmuebles en el entorno 3D que ofrece Floorfy, en este proyecto también se ha añadido la posibilidad de cambiar de textura y de color las paredes y suelos que forman parte de un inmueble.

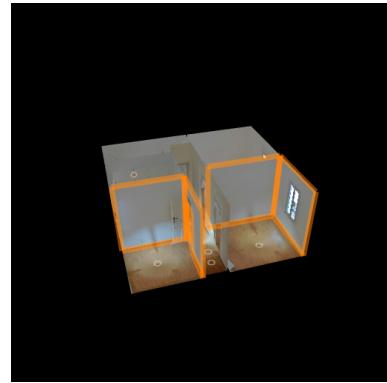
Se puede entrar en este modo si se hace click en el icono de la brocha en la parte superior del catálogo. Una vez dentro, podremos seleccionar cualquier pared o suelo haciéndole click derecho encima de uno de ellos y mas adelante, poder seleccionar la textura que queremos aplicar y el color que deseamos.

9.3.1. Selección de paredes y suelos

El primer paso es seleccionar las paredes o suelos. Exactamente de la misma forma en la que se utilizaba el *raycaster* en el punto 9.2.5, esta vez, al no tener ningún objeto seleccionado y al estar dentro del modo de pintar paredes y suelos, al hacer click derecho en una de las paredes o suelos (las paredes y suelos no son más que objetos 3D en forma de plano), el contorno de la geometría poligonal del objeto interseccionado se tornará de color anaranjado con una leve opacidad y se abrirá el menú con las texturas de pared o suelo, dependiendo del objeto seleccionado. De esta forma, se le comunica al usuario de que esta pared o suelo ha sido seleccionada.



(a) Selección de varios suelos de un *tour virtual*.



(b) Selección de varias paredes de un *tour virtual*.

Figura 49: Ejemplo del uso del selector de suelos o paredes. Elaboración propia.

Se puede observar en la primera imagen, en la cual seleccionamos suelos, que la línea anaranjada que forma el contorno del suelo de en medio se sobreponen por encima de la escena. Esto está hecho adrede, dado que se pretende dar la información de que un suelo o pared está seleccionado, y de esta manera siempre se sabrá esa información independientemente de las posibles occlusiones que puedan haber con la escena.

Esto es posible asignándole al material que forman las líneas anaranjadas el valor `false` a la variable `depthTest`, de la siguiente forma:

```
line.material.depthTest = false;
```

`depthTest` hace que se active el test de profundidad a la hora de renderizar un material, al ponerlo en falso, no se evalúa y se pinta sobre la escena.

Si el objeto interseccionado es una pared, podremos seguir seleccionando otras paredes, pero si primero seleccionamos paredes y luego un suelo, las paredes dejarán de estar seleccionadas y solo se mostrará el suelo seleccionado. Esta funcionalidad también se aplica a la inversa, seleccionando suelos en lugar de paredes. Se ha decidido hacer de esta manera dado que no tiene sentido poner una textura de tipo suelo en la pared y viceversa, de esta manera se seleccionan únicamente paredes o únicamente suelos.

Para lograrlo, se ha utilizado una estructura array donde se guardan los objetos seleccionados, y si el nuevo objeto seleccionado es de un tipo diferente a los que ya hay añadidos, se vacía por completo y se añade de nuevo el último objeto seleccionado. A demás, siempre que se añade un nuevo suelo/pared al array, se dibuja su contorno en la escena.

Si hacemos click derecho en una pared o suelo seleccionado, este se deseleccionará, dándole a entender al usuario que está deselegionado porque ya no se dibuja su contorno en la escena. Simplemente, eliminamos ese objeto del array de paredes seleccionadas.

9.3.2. Paint Options

Al igual que con los objetos y modelos 3D, también tenemos un apartado de opciones que es utilizable en cuanto más de una pared o suelo es seleccionada. A continuación se explican las diversas funcionalidades que se han implementado en esta parte del aplicativo.

9.3.2.1. Textura de pared y suelo

De la misma forma que en los objetos, tenemos texturas a añadir. El proceso es el mismo que se ha explicado en el punto 9.2.7.4 pero con alguna pequeña variación. En el caso de los objetos de suelo y pared, no tenemos coordenadas u , v en la geometría dado que las imágenes aplicadas por defecto de las habitaciones tienen un material de tipo `ShaderMaterial`⁵ y carecen de coordenadas u , v , por lo tanto se han tenido que recrear.

El nuevo material a aplicar en la pared/suelo es un `MeshLambertMaterial`. Este material es igual que Phong pero sin la componente especular, únicamente con la difusa, como se puede ver en la imagen 14. Se ha decidido añadir este tipo de material para evitar especularidad (por mínima que sea) en una pared o suelo.

También hay que fijarse que una pared es visible por sólo un lado. Esto estaba preestablecido en el código para que los usuarios del *tour virtual* pudiesen observar con claridad el inmueble por dentro, de lo contrario, solo se verían paredes exteriores en el caso de navegar por la cámara en tercera persona.

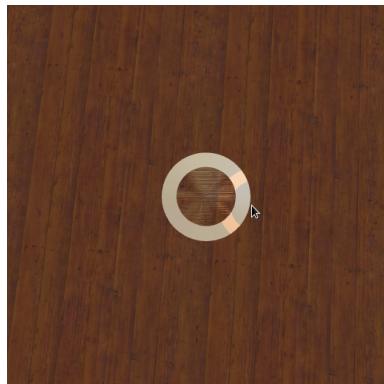
Al cambiar este material, se ha tenido que preservar esta condición. Mediante `three js`, el material tiene un parámetro denominado como `material.side`, que hace referencia al lado por el cual el material va a ser renderizado. Sus posibles valores son de frente, por atrás o ambas. (`FrontSide`, `BackSide`, `DoubleSide`, respectivamente). Dado que esta información también la guarda el material anterior, lo que se hace es darle el valor del `side` del material anterior.

⁵Un `ShaderMaterial` es un material renderizado con *shaders* personalizados. Un *shader* es un pequeño programa escrito en GLSL que se ejecuta en la GPU. La razón por la cual se utilizan `ShadersMaterials` para aplicar esa imagen en las paredes y suelos del *tour virtual* es porque se ha necesitado implementar un efecto no incluido con ninguno de los materiales incorporados por defecto.

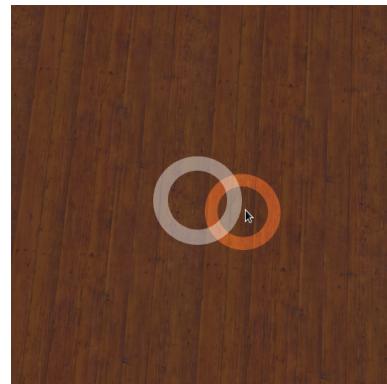
```
newMaterial.side = oldMaterial.side;
```

Al realizar este cambio, el material de los *hotspots* se interponían con los nuevos materiales de los suelos añadidos a la escena, dado que los *hotspots* son objetos con material opaco. Para solucionarlo se ha hecho uso del **renderOrder** de cada objeto. El **renderOrder** es un valor que permite que se anule el orden de representación predeterminado de los objetos del gráfico de escena, aunque los objetos opacos y transparentes permanecen ordenados de forma independiente, es por ello que los *hotspots* no aparecían correctamente. La clasificación es de menor a mayor **renderOrder**.

Es por ello que se ha aumentado el valor de **renderOrder** de los *hotspots*, y por lo tanto, aparecen por encima del suelo utilizando esta solución.



(a) *Hotspot* con el suelo texturizado con la textura “Parquet”.



(b) *Hotspot* con el suelo texturizado con la textura “Parquet” y solucionado con la modificación en **renderOrder**.

Figura 50: Solución a los hotspots con **renderOrder**. Elaboración propia.

También teníamos el mismo problema con el indicador anaranjado que sigue el cursor en la escena del *tour virtual*, utilizando el mismo método, también se ha solventado el problema.

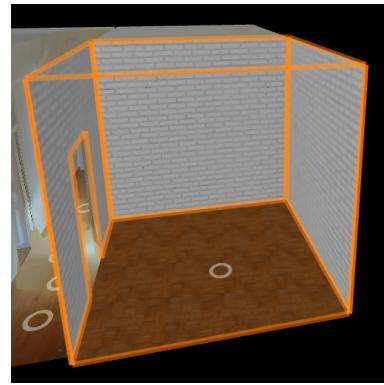
En este caso, se han necesitado dos catálogos de texturas, uno para las paredes y otro para los suelos. No tenía mucho sentido que una textura de suelo como parquet o baldosas estuviesen en la pared y viceversa, de esta forma, cuando se seleccionen suelos se abrirá el menú con las texturas de suelos y cuando se estén seleccionando paredes, se abrirá el de paredes.

A demás, se ha tenido que añadir un valor de repetición a cada textura, porque a veces una textura no era del todo coherente con su tamaño. El valor se ha ido ajustando hasta encontrar un resultado más apropiado.

En cuanto se hace click en una de las texturas disponibles, los materiales de las paredes/suelos que estén seleccionadas en ese momento, se cambiarán a la textura aplicada.



(a) Habitación sin texturizar.



(b) Habitación texturizada.

Figura 51: Comparación de una habitación del un *tour virtual* sin texturizar y con las paredes y suelos texturizados mediante el aplicativo. Elaboración propia.

9.3.2.2. Color

Aplicamos color a las paredes cuyo material haya sido modificado previamente. Es decir, las paredes seleccionadas que siguen con el ShaderMaterial que representaba al inmueble original no pueden ser modificadas en color.

En el caso de que haya paredes/suelos seleccionadas con el material cambiado, se podrá modificar el color. Este proceso se hace de la misma manera que en los modelos 3D, pero en vez de hacer uso de layers, solo hay que pintar un único objeto. Queda explicado en este punto: 9.2.7.3.

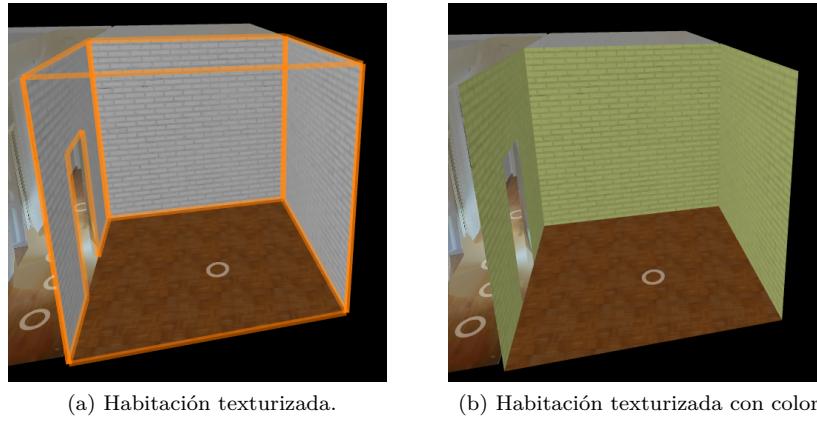


Figura 52: Comparación de una habitación del un *tour virtual* texturizada y texturizada con color. Elaboración propia.

9.3.2.3. Unselect

Deseleccionamos todas las paredes/suelos que estuviesen seleccionados. Esto internamente lo que hace es vaciar el array con todas las paredes/suelos seleccionados y quitar de la escena las líneas que daban a entender que esas paredes/suelos están seleccionadas.

9.3.2.4. Reset

Si en las paredes/suelos que hay seleccionadas existe alguna con una textura aplicada, el aplicativo lanzará una ventana emergente para confirmar que queremos volver a su valor predeterminado. Esto se hace dado que puede ser una opción destructiva porque eliminaremos la textura y color aplicado.

En el caso de que el usuario confirme, las paredes/suelos cuyo material ha sido modificado volverá al de antes. Esto se hace en código guardando los ShaderMaterials que hemos ido substituyendo por nuevos materiales.

10. Conclusiones finales

En el siguiente apartado se exponen las futuras modificaciones del proyecto, las conclusiones finales sobre el desarrollo del proyecto y se finaliza con una

explicación acerca de cada una de las competencias técnicas sobre este proyecto de fin de grado.

10.1. Futuro del proyecto

Una de las cosas que me gustaría mejorar personalmente es el tema de la iluminación y sombras. Una de las cosas que haré al seguir el proyecto en Floorfy es añadir puntos de luz a objetos como lámparas o ventanas para que el entorno tenga un estilo más realista. A demás de esto, se pueden añadir sombras que reflejen estas nuevas luces añadidas a la escena. Esto dará un todo más realista a lo que hay actualmente y también el usuario podría configurar estas luces en términos de color, dirección etcétera.

Desde Floorfy, lo que se quiere hacer es contactar con empresas que comercializan con muebles como IKEA para que en el *tour virtual* se puedan escoger objetos que existen en la realidad, y mediante esta herramienta, poder comprar el modelo 3D directamente. Esto se debería de hacer conjuntamente con IKEA y es una idea de lo que puede acabar siendo este proyecto.

Dado que los muebles de IKEA tienen un tamaño específico y fijo, la opción de cambiar el tamaño dejaría de estar disponible y se trataría de hacer que el mueble tuviese escala 1:1 respecto la realidad dentro del *tour virtual*. De esta manera, el modelo sería fiel a la realidad y el usuario se podría hacer una idea de cómo quedaría su inmueble con una determinada decoración real basada en modelos reales.

Por último, una de las cosas que en Floorfy tenemos en mente es poder amueblar un inmueble de origen de forma automática mediante las herramientas diseñadas en este proyecto (límites en una habitación, colisión de objetos, etcétera). Este proceso se haría a través de algoritmos de *Machine Learning* y el uso de *transformers*. De momento, se ha valorado esta opción y se tratará de implementar en un futuro próximo.

10.2. Conclusiones

Para empezar, considero que este proyecto de fin de carrera me ha ayudado a entrar y comprender el entorno laboral desde dentro de una empresa, dado que lo he desarrollado a través de la compañía Floorfy, a demás de tener la oportunidad de poder utilizar su entorno para desarrollar mi proyecto, me ayudaron en encontrar un proyecto acorde a lo que yo deseaba y en lo que yo quería aprender y formarme.

Principalmente he expandido mis conocimientos sobre los gráficos por computadora y sus aplicaciones en la industria.

tador que traía desde la universidad, dado que he tenido que trabajar directamente con ello.

Para empezar, la creación del catálogo me ha hecho conocer en más profundidad las herramientas de HTML y CSS, para así dar con un diseño más intuitivo para el usuario, y que se pueda manejar con facilidad por el aplicativo. Toda esta parte de *frontend* me ha hecho adquirir nuevos conocimientos para el lenguaje entre una herramienta y el usuario que la va a utilizar, pese a ser consciente de que los diseños que he desarrollado no son del todo profesionales y se debería modificar cuando el aplicativo salga al mercado.

Principalmente se ha trabajado con modelos 3D y texturas, en ese aspecto también he enriquecido mis conocimientos, dado que he aprendido cuáles son las componentes que tiene un fichero OBJ y un MTL, y cómo se pueden ver en función de los atributos que se les da. También he aprendido a cómo afectan sus modificaciones en brillo, escala, *Pivot point*, etcétera.

También he podido tocar pequeñas cosas de *Blender*, en este software, he aprendido un poco sobre la modificación de modelos 3D y he logrado hacer cambios a algunos modelos con tal de ver un resultado deseable.

Sobretodo he aprendido a utilizar la librería *three js*, la cual es fundamental en este proyecto, dado que es el intermediario entre el programador (en este caso yo mismo) y la API de WebGL. Evidentemente no me considero un experto de su uso, pero puedo afirmar que he aprendido bastantes métodos que implican su uso que antes desconocía y me gustaría seguir aprendiendo de ello.

He aprendido cosas tales como la función del *Raycast* para interseccionar objetos mediante el uso del ratón, los componentes de los modelos 3D (textura, color, cómo se relacionan las características de un .OBJ y .MTL con un Object3D en *three js*), las transformaciones geométricas que se le pueden aplicar mediante métodos de la librería a la hora de hacer los controles, la función que tiene la luz, etcétera.

También se han tenido que estudiar la forma en la que la luz infiere en los materiales y la importancia que tiene que un material tenga unas propiedades para que la luz que se le añade tenga una repercusión visual, como la reflexión de Phong, que ya la había estudiado en la carrera. También he añadido conocimientos sobre la técnica de *UV mapping* que *three js* aplica al aplicar una textura a un modelo 3D de forma interna y como se relaciona con los parámetros de vértices de textura que tienen un objeto en su fichero .OBJ.

También se han estudiado las distintas formas de aplicar algoritmos para el método de colisión, y comprobar los límites que tiene un objeto dentro de una habitación. Gracias a ello, he tenido que aprender a diferenciar si un algoritmo es acorde en función de solventar lo que se desea y eficiencia en tiempo.

Sobretodo en este caso, en el que los *bounding boxes* son generados de forma manual dado que la librería no permite crear *bounding boxes* de forma fiel a la realidad conforme las rotaciones que sufre el objeto. Considero que la fase del proyecto que más horas he dedicado ha sido a la hora de realizar los algoritmos para encajar un modelo dentro de una propiedad, siguiendo los 4 tipos de modelos que existen (*floor*, *wall*, *wallFloor* e *item*).

Primeramente, para sacar la información de los vértices del *bounding box* y luego, por generar el algoritmo de la mejor forma posible, dado que en el *tour virtual* ya existe una información acerca de las paredes y suelos que hay en el entorno, he tenido que adaptarme a lo establecido y saber substrair la información que ya existe en el entorno 3D para dar con unos algoritmos funcionales.

Finalmente, considero que todo el proyecto de fin de carrera ha sido realizado de forma satisfactoria, se han podido cumplir todas las tareas que se propusieron inicialmente, aunque se ha tenido que alargar en la parte de búsqueda de recursos.

También considero que las funcionalidades realizadas son acordes a un uso profesional dado que funcionan correctamente para los *tours virtuales* que dispone Floorfy y tiene potencial suficiente para que la empresa lo utilice y lo ponga en práctica en su mercado, puliendo algunas cosas de diseño, como el catálogo o la interacción entre el usuario y el aplicativo.

A continuación se muestran algunas de las cosas que se pueden hacer con el aplicativo:



(a) Cocina realizada con el aplicativo.



(b) Salón realizado con el aplicativo.



(c) Habitación realizada con el aplicativo.

Figura 53: Habitaciones modificadas tanto con modelos 3D como con texturas.
Elaboración propia.

En términos generales, actualmente me gustaría centrar mi carrera profesional en la programación 3D o formar parte de algo parecido, y este proyecto, junto con la colaboración y la oportunidad que me brinda la empresa Floorfy, me ha ayudado a tenerlo aún más claro.

10.3. Competencias técnicas

En este apartado, se presentan y justifican las distintas competencias técnicas que se seleccionaron al inicio del proyecto y se han aplicado durante el desarrollo

del mismo.

10.3.1. CCO1.1

Evaluar la complejidad computacional de un problema, conocer estrategias algorítmicas que puedan conducir a su resolución, y recomendar, desarrollar e implementar la que garantice el mejor rendimiento de acuerdo con los requisitos establecidos. [Bastante].

Dado que durante este proyecto se ha necesitado en varias ocasiones hacer uso de diferentes algoritmos, como por ejemplo, a la hora de saber cuándo un objeto intersecciona con otro o cuándo un objeto está fuera de los límites de una propiedad, se ha tenido que identificar un problema a resolver, saber cuál es el método mas eficiente e implementarlo.

10.3.2. CCO2.3

Desarrollar y evaluar sistemas interactivos y de presentación de información compleja, y su aplicación a la resolución de problemas de diseño de interacción persona computadora. [Bastante].

El aplicativo desarrollado durante este proyecto está pensado para ser accesible de cara al público, por lo tanto, se ha requerido de cierta capacidad para que la interacción entre el usuario y la herramienta sea lo más amena e intuitiva posible. Desde el catálogo, contando con sus secciones, menús y menús de opciones hasta en las modificaciones del objeto cuando está seleccionado.

10.3.3. CCO2.6

Diseñar e implementar aplicaciones gráficas, de realidad virtual, de realidad aumentada y videojuegos. [En profundidad].

Se considera la competencia técnica que ha sido mas aplicada a lo largo de este proyecto, ya que todas las partes del proyecto se ven involucradas de alguna manera en los gráficos por computador. Se puede afirmar que ha sido un proyecto de gráficos porque está claramente relacionado con objetos 3D, texturas, iluminación, métodos de intersección y colisión entre modelos, etcétera. El propósito principal de la nueva aplicación ha sido desarrollar, diseñar e implementar esta forma de aplicar a los tours virtuales un amueblado virtual y la posibilidad de cambiar texturas y colores mediante herramientas de gráficos por computador.

10.3.4. CCO3.1

Implementar código crítico siguiendo criterios de tiempo de ejecución, eficiencia y seguridad. [Bastante]

Como en todo proyecto que tiene expectativa de llegar a ser mostrado de cara al público, la eficiencia y los tiempos de ejecución son importantes. En el caso de este proyecto, se ha tratado de implementar siguiendo los criterios de eficiencia e intentando que los tiempos de ejecución que implican cargar un nuevo modelo 3D dentro del *tour virtual* sea de lo más razonable posible.

Referencias

- [1] Floorfy S.L. *Tours virtuales para inmobiliarias - Floorfy*. URL: <https://floorfy.com/es/>.
- [2] Mercedes Calma. *¿Qué es un modelo 3D? Historia, función y más.* URL: https://tecnoinformatic.com/c-avances-tecnologicos/que-es-un-modelo-3d/#Que_es_un_modelo_3D.
- [3] *Blender 3D: Noob to Pro/Materials and Textures.* URL: https://en.wikibooks.org/wiki/Blender_3D:_Noob_to_Pro/Materials_and_Textures#:~:text=%20Additional%20Resources-,Material%20versus%20Texture,world%20have%20completely%20uniform%20surface.
- [4] davidkent. *davidkent portfolio.* URL: <https://davidkent.cgsociety.org/>.
- [5] Unity Technologies. *Unity.* URL: <https://unity.com/es>.
- [6] Blender Fundation. *blender.org - Home of the Blender project - Free and Open 3D Creation Software.* URL: <https://www.blender.org/>.
- [7] Dan Smigrod. *3D Models: Matterport Versus Floored — We Get Around.* URL: <https://www.we-get-around.com/wegetaround-atlanta-our-blog/2015/3/1/3d-models-matterport-photography-compared-versus-floored>.
- [8] *floored.* URL: <https://www.cbre.es/es-es/floored>.
- [9] *Shapespark.* URL: <https://www.shapespark.com/>.
- [10] *Roomle.* URL: <https://www.roomle.com/en>.
- [11] *Roomy.* URL: <https://rooomy.com/>.
- [12] *Realisti.co — Virtual Tours, Made Easy.* URL: <https://realisti.co/es/>.
- [13] *Fotogrametría: Qué es, ventajas y metodología.* URL: <https://www.globalmediterranea.es/fotogrametria-que-es/>.
- [14] *Three.js - JavaScript 3D library.* URL: <https://threejs.org/>.
- [15] *Getting Started - WebGL Public Wiki.* URL: https://www.khronos.org/webgl/wiki/Getting_Started.
- [16] *Jira — Software de seguimiento de proyectos e incidencias.* URL: <https://www.atlassian.com/es/software/jira>.
- [17] *Bitbucket — The Git solution for professional teams.* URL: <https://www.atlassian.com/es/software/bitbucket>.
- [18] BarD s.r.o. *GanttProject: free project management tool for Windows, macOS and Linux.* URL: <https://www.ganttpoint.biz/>.
- [19] *Log in to Overleaf - Overleaf, Editor de LaTeX online.* URL: <https://es.overleaf.com/project>.

- [20] Sergio Mora. *Herramientas para la investigación: 4 ¿Qué es LaTeX?* URL: <http://desarrolloweb.dlsi.ua.es/cursos/2015/herramientas-investigacion/que-es-latex>.
- [21] *PhpStorm: el IDE rápido e inteligente para programación en PHP* de JetBrains. URL: <https://www.jetbrains.com/es-es/phpstorm/>.
- [22] Glassdoor. *Glassdoor: Búsqueda de empleo en Glassdoor.* URL: <https://www.glassdoor.es/index.html>.
- [23] neuvoo. *Convertir euros por hora en Salario anual.* URL: <https://neuvoo.es/convert/?&from=hour&to=year>.
- [24] Suzanne Wales. *Inside Barcelona's 11 best coworking spaces.* URL: <https://thespaces.com/best-barcelona-coworking-spaces/>.
- [25] Polantis. *INTEGRATE REAL-WORLD PRODUCTS BY LEADING MANUFACTURERS INTO YOUR ARCHITECTURAL DESIGN.* URL: <https://www.polantis.com/>.
- [26] Food Drink. *3D Models for Professionals :: TurboSquid.* URL: <https://www.turbosquid.com/>.
- [27] Free3D. *3D Models for Free - Free3D.com.* URL: <https://free3d.com/es/modelos-3d/obj>.
- [28] CGTrader. *Buy Professional 3D Models — CGTrader.* URL: <https://www.cgtrader.com/es/gratis-3d-modelos>.
- [29] Textures.com. *Textures for 3D, graphic design and Photoshop!* URL: <https://www.textures.com/library>.
- [30] 3dviewer.net. *Online 3D Viewer.* URL: <https://3dviewer.net/>.
- [31] Paul Bourke. *Object Files (.obj).* URL: <http://paulbourke.net/dataformats/obj/>.
- [32] People.math.sc.edu. *MTL Files - Material Definitions for OBJ Files.* URL: <https://people.math.sc.edu/Burkardt/data/mtl/mtl.html>.
- [33] Scratchapixel. *The Phong Model, Introduction to the Concepts of Shader, Reflection Models and BRDF.* URL: <https://www.scratchapixel.com/lessons/3d-basic-rendering/phong-shader-BRDF>.
- [34] en.wikipedia.org. *Phong reflection model - Wikipedia.* URL: https://en.wikipedia.org/wiki/Phong_reflection_model.
- [35] Spare Time Coding Adventures. *Example 01.04: Phong reflection model – specular lighting.* URL: <https://daviddegeyter.wordpress.com/2015/10/18/example-01-04-phong-reflection-model-specular-lighting/>.
- [36] Docs.blender.org. *Introducción — Blender Manual.* URL: https://docs.blender.org/manual/es/latest/getting_started/about/introduction.html.

- [37] How cursor. *How To Get 3D Position Of The Mouse Cursor - Godot Engine*. URL: <https://godotengine.org/qa/25922/how-to-get-3d-position-of-the-mouse-cursor>.
- [38] lab bhattacharjee. *Mathematics Stack Exchange - How to check if a point is inside a rectangle?* URL: <https://math.stackexchange.com/questions/190111/how-to-check-if-a-point-is-inside-a-rectangle>.
- [39] 3D Transformation - Tutorialspoint. *Tutorialspoint.com*. URL: https://www.tutorialspoint.com/computer_graphics/3d_transformation.htm.
- [40] Threejs.org. *three.js docs - Matrix transformations*. URL: <https://threejs.org/docs/#manual/en/introduction/Matrix-transformations>.