



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Facultat d'Informàtica de Barcelona



CREACIÓ D'UN VISUALIZADOR DE FOTOGRAFIES EN 3D

BERNAT BORRÀS CIVIL

Director/a: IMANOL MUÑOZ PANDIELLA (Departament de Ciències de la Computació)

Codirector/a: CARLOS ANDUJAR GRAN (Departament de Ciències de la Computació)

Titulació: Grau en Enginyeria Informàtica (Computació)

Memòria del treball de fi de grau

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

Agraïments

Vull destacar i agrair la importància de totes les persones que m'han donat el seu suport durant la realització d'aquest projecte.

En primer lloc, agrair tant al meu director, Imanol Muñoz, i al meu codirector Carlos Andujar per la dedicació, suport i consells que m'han donat setmana rere setmana per a tirar endavant el projecte.

En segon lloc, agrair en Gaetano Alfano per a fer les fotografies de l'església de Sant Quirze de Pedret, i a *EHEM Project* per a produir els models utilitzats.

En tercer lloc, agrair el centre de recerca *ViRVIG* per proveir tots els recursos necessaris, on s'inclouen els models i les fotografies utilitzades durant el desenvolupament.

A més a més, agrair els companys i amics del grau pels ànims i el suport mutu necessaris per a finalitzar el treball.

Per últim, agrair els meus familiars pel recolzament donat des del primer dia, i la motivació en aquells moments tan durs. Encara que no s'hagin implicat directament al treball, han sigut un pilar fonamental per dur-lo a terme.

Resum

Des de l’arribada de la càmera fotogràfica, els humans han vist la necessitat de trobar la manera de poder organitzar les fotografies. Avui en dia, la facilitat i senzillesa de prendre una fotografia fa que s’acumulin grans quantitats d’aquestes. Els visors de fotografies i les galeries més populars mostren una imatge rere l’altra, en una llista o quadrícula, dificultant l’organització i la cerca d’aquestes.

Un àmbit on tenen present aquest problema és la digitalització del patrimoni cultural. En aquesta àrea, historiadors i fotògrafs professionals prenen una gran quantitat de fotografies en llocs d’interès cultural i monuments al voltant del món. Aquesta àrea porta una dificultat afegida. Per tal de conservar els detalls del lloc d’interès, les fotografies preses tenen una gran qualitat, requerint un sistema eficient per a dispositius poc potents.

Aquest projecte té com a objectiu proposar una solució per a facilitar la tasca de la cerca i organització d’imatges de manera eficient d’un lloc d’interès, com és l’església de Sant Quirze de Pedret. S’estudiarà la relació que hi ha entre les imatges en dues dimensions, i l’espai 3D on es van capturar. També, s’estudiarà quina relació hi ha entre els espais de captació físics i els models en tres dimensions.

Amb aquests estudis fets, es pretén visualitzar aquestes imatges tenint en compte l’entorn 3D, conjuntament amb models 3D de l’església. La gran qualitat de les fotografies, fa que la fluïdesa de la solució sigui un factor important a tenir en compte, havent d’aplicar optimitzacions importants al producte desenvolupat.

Resumen

Desde la llegada de la cámara fotográfica, los humanos han visto la necesidad de encontrar la forma de poder organizar las fotografías. Hoy en día, la facilidad y sencillez de tomar una fotografía hace que se acumulen grandes cantidades de estas. Los visores de fotografías y las galerías más populares muestran una imagen tras otra, en una lista o cuadrícula, dificultando su organización y búsqueda.

Un ámbito en el que tienen presente este problema es la digitalización del patrimonio cultural. En esta área, historiadores y fotógrafos profesionales toman una gran cantidad de fotografías en lugares de interés cultural y monumentos en torno al mundo. Esta área trae una dificultad añadida. Con el fin de conservar los detalles del sitio de interés, las fotografías tomadas tienen una gran calidad, requiriendo de un sistema eficiente para dispositivos poco potentes.

Este proyecto tiene como objetivo proponer una solución para facilitar la labor de la búsqueda y organización de imágenes de manera eficiente de un lugar de interés, como es la iglesia de Sant Quirze de Pedret. Se estudiará la relación que existe entre las imágenes en dos dimensiones, y el espacio 3D donde se capturaron. También, se estudiará qué relación existe entre los espacios de captación físicos y los modelos en tres dimensiones.

Con estos estudios realizados, se pretende visualizar estas imágenes teniendo en cuenta el entorno 3D, conjuntamente con modelos 3D de la iglesia. La gran calidad de las fotografías hace que la fluidez de la solución sea un factor importante a tener en cuenta, debiendo aplicar optimizaciones importantes al producto desarrollado.

Abstract

Since the arrival of the photographic camera, humans have seen the need to find a way to organize photographs. Nowadays, the ease and simplicity of taking a photograph causes large quantities of them to accumulate. The most popular photo viewers and galleries display one image after another, in a list or grid, making it difficult to organize and searching them.

One area in which they are aware of this problem is the digitization of cultural heritage. In this area, historians and professional photographers take a large number of photographs at cultural sites and monuments around the world. This area brings added difficulty. In order to preserve the details of the site of interest, the photographs taken are of high quality, requiring an efficient system for less powerful devices.

This project aims to propose a solution to facilitate the work of searching and organizing images of a place of interest efficiently, such as the church of Sant Quirze de Pedret. I will study the relationship that exists between two-dimensional images and the 3D space where they were captured. Also, the relationship between physical collection spaces and three-dimensional model will be studied.

With these studies carried out, it is intended to visualize these images taking into account the 3D environment, together with 3D models of the church. The high quality of the photographs makes the fluidity of the solution an important factor to take into account, requiring important optimizations to be applied to the developed product.

Taula de Continguts

1	Introducció	16
1.1	Contextualització	16
1.2	Definició de conceptes	16
1.2.1	Galeria d'imatges	17
1.2.2	Visor d'imatges	17
1.2.3	Entorn 3D	17
1.3	Identificació del problema	17
1.4	Actors implicats	18
2	Justificació	19
2.1	Estudi d'alternatives	19
2.1.1	PhotoCloud	19
2.1.2	MeshLab	20
2.2	Selecció del programari	20
2.2.1	Gràfics 3D	20
2.2.2	Visor d'imatges	21
3	Abast	22
3.1	Objectius i subobjectius	22
3.2	Requeriments	22
3.2.1	Requeriments funcionals	22
3.2.2	Requeriments no funcionals	22
3.3	Obstacles	23
3.4	Riscos	23
4	Metodologia	24
4.1	Metodologia Àgil	24
4.2	Eines de seguiment	24
4.3	Mètode de validació	24
5	Planificació temporal	26
5.1	Recursos	26
5.1.1	Recursos humans	26
5.1.2	Recursos materials	26
5.2	Descripció de les tasques	27
5.2.1	Gestió del projecte [G]	27
5.2.2	Treball previ [P]	27
5.2.3	Desenvolupament entorn 3D [T]	28
5.2.4	Desenvolupament entorn 2D [O]	29
5.2.5	Elaboració de la memòria [M]	29
5.2.6	Defensa [D]	29
5.2.7	Reunions [R]	29
5.3	Representacions gràfiques de les tasques	30
5.4	Gestió del risc: Plans alternatius i obstacles	33
6	Gestió econòmica	34
6.1	Identificació i estimació de costos	34

6.1.1	Costos genèrics	34
6.1.2	Costos personals	36
6.1.3	Costos de contingència	38
6.1.4	Costos d'imprevistos	38
6.1.5	Cost total	38
6.2	Control de gestió	38
7	Sostenibilitat	40
7.1	Autoavaluació	40
7.2	Dimensió econòmica	40
7.3	Dimensió ambiental	41
7.4	Dimensió social	42
8	Disseny i implementació de l'entorn 3D	43
8.1	Càrrega del model 3D	43
8.2	Representació d'imatges	49
8.2.1	Càrrega d'imatges	53
8.2.2	Posicionament d'imatges	55
8.2.3	Rotacions d'imatges	57
8.2.4	Distribucions d'imatges en un model complet	58
8.3	Interacció amb imatges	61
8.3.1	Modificació de la mida d'una imatge	61
8.3.2	Modificació de la separació respecte a una imatge	62
8.3.3	Hover i selecció d'imatges	63
8.3.4	Mostrar filferros	67
8.4	Visualització d'imatges	72
8.4.1	Visualització d'una sola imatge	72
8.4.2	Visualització d'imatges seleccionades	72
8.4.3	Visualització d'imatges en un pla	75
8.4.4	Visualització d'imatges en mode esfèric	81
8.4.5	Visualització d'imatges en mode cilíndric	84
8.5	Mode Autor i Inspecció	89
8.5.1	Mode Autor	89
8.5.2	Mode Inspecció	91
8.5.3	Comutació entre modes	92
8.6	Interfície gràfica	93
8.6.1	Panell lateral	94
8.6.2	Barra d'informació	105
9	Disseny i implementació de l'entorn 2D	107
9.1	Càrrega d'imatges	107
9.1.1	Visualització d'una imatge	107
9.1.2	Visualització de múltiples imatges	109
9.2	Deep Zoom Images	111
9.3	Posicionament i mida d'imatges	114
9.3.1	Posicionament d'imatges	114
9.3.2	Mida d'imatges	115
9.3.3	Posicionament d'imatges segons target	119

9.3.4	Evitant l'encavalcament d'imatges	122
9.4	Navegació a ThreeJS	126
9.5	Interfície gràfica	130
10	Descripció dels resultats	133
10.1	Visualitzant una imatge	133
10.2	Visualitzant imatges seleccionades	134
10.3	Visualitzant imatges d'una esfera	135
10.4	Visualitzant imatges d'un cilindre	138
10.5	Visualitzant imatges d'un pla	140
10.6	Visualitzant imatges amb el mode Inspecció	143
11	Conclusions	148
11.1	Valoració personal	148
11.2	Reptes	148
11.3	Assoliment d'objectius	149
11.4	Treballs futurs	150

Llista de Figures

1	Captura de la galeria <i>Google Photos</i>	16
2	Imatges i un model 3D a <i>PhotoCloud</i>	19
3	Estàtua envoltada de càmeres a <i>Meshlab</i>	20
4	Graf de dependències de tasques	31
5	Diagrama de Gantt	32
6	Model de Sant Quirze de Pedret XII en format OBJ	44
7	Model de Sant Quirze de Pedret XIII en format OBJ	45
8	Model de Sant Quirze de Pedret XIII en format FBX	46
9	Model de Sant Quirze de Pedret XIII en format GLTF	47
10	Model de l'absidiola sud amb la rotació incorrecta	48
11	Model de l'absidiola sud amb la rotació correcta	48
12	Estructura del directori <i>Sant Quirze de Pedret by Zones</i>	51
13	Estructura del directori <i>MDCS - Central Apse</i>	51
14	Format del fitxer <i>Bundler</i>	52
15	Format d'una càmera del fitxer <i>Bundler</i>	52
16	Format d'un punt del fitxer <i>Bundler</i>	52
17	Valors del fitxer <i>MANC-AbsSud.out</i>	53
18	Representació d'una imatge a l'entorn 3D	54
19	Posició 3D d'una càmera	55
20	Imatges després d'aplicar la fórmula de la posició	56
21	Imatges posicionades correctament	56
22	Direcció d'una càmera	57
23	Imatges després d'aplicar la rotació	57
24	Imatges amb la rotació correcta	58
25	Model pedret_XII amb imatges	59
26	Selecció dels 8 punts a cada model	59
27	Model després d'aplicar-li la matriu de rotació obtinguda	61
28	Interfície gràfica d'interacció	61
29	Esquema vectors direcció i posició	63
30	Imatge ressaltada en fer <i>hover</i>	66
31	Grup d'imatges seleccionades	67
32	Conjunt d'imatges amb wireframe	69
33	Coordenades d'objecte d'una imatge	70
34	Coordenades esfèriques	74
35	Representació del pla a ThreeJS	76
36	Pla centrat i reajustat	77
37	Interfície gràfica amb opcions del pla	78
38	Projecció d'un punt al pla	79
39	Conversió de coordenades del pla a coordenades de món	80
40	Representació d'una esfera a ThreeJS	82
41	Interfície gràfica amb opcions de l'esfera	82
42	Paràmetres d'un cilindre	84
43	Representació d'un cilindre a ThreeJS	85
44	Interfície gràfica amb opcions del cilindre	86
45	Obtenció de coordenades cilíndriques	87

46	Inclusió de punts dins d'un cilindre	88
47	Escena en mode Autor	93
48	Escena en mode Inspecció	93
49	Interfície gràfica de l'entorn 3D	94
50	Panell de mode	94
51	Panell de Configuració d'imatges	96
52	Panell de figures	97
53	Panell d'imatges seleccionades	99
54	Panell de l'esfera	100
55	Panell del pla	102
56	Panell del cilindre	104
57	Visor 2D amb una imatge	108
58	Visor 2D amb diverses imatges	110
59	Mala conversió d'una imatge DZI	112
60	Imatge DZI girada	113
61	Imatge DZI ben formada	114
62	Imatges amb posicions assignades	115
63	Imatges amb la mateixa mida	116
64	Imatges amb la mida segons el zoom	117
65	Imatges amb detalls augmentades	118
66	Distribució d'imatges segons la posició real	119
67	Raig llançat des d'una imatge	120
68	Desplaçament d'una imatge segons la intersecció	122
69	Càcul de la intersecció	123
70	Distribució amb algunes oclusions	125
71	Distribució sense oclusions	126
72	Imatge seleccionada per a la navegació	127
73	Navegació amb rotació incorrecta	128
74	Navegació amb rotació correcta	129
75	Imatge seleccionada al mode Inspecció	129
76	Navegació a la part del model corresponent a la imatge	130
77	Interfície gràfica per defecte	130
78	Interfície gràfica final	132
79	Imatge seleccionada a l'entorn 3D	133
80	Imatge oberta al visor	134
81	Imatges seleccionades a l'entorn 3D	134
82	Botó <i>Open in 2d viewer</i>	135
83	Imatges seleccionades obertes al visor	135
84	Botó <i>Create Sphere</i>	136
85	Esfera creada a l'entorn 3D	136
86	Botó <i>Open in 2d viewer</i>	137
87	Imatges amb posició real	137
88	Imatges amb posició segons el target	138
89	Botó <i>Create Cylinder</i>	138
90	Cilindre creat a l'entorn 3D	139
91	Botó <i>Open in 2d viewer</i>	139
92	Imatges amb la mida segons el zoom	140

93	Imatges amb la mida augmentar detalls	140
94	Botó <i>Create Plane</i>	141
95	Pla creat a l'entorn 3D	141
96	Botó <i>Open in 2d viewer</i>	142
97	Imatges al visor	142
98	Imatge premuda per a navegar	143
99	Navegació a la imatge seleccionada	143
100	Botó <i>Create Sphere</i>	144
101	Esfera creada a l'entorn 3D	144
102	Botó <i>Save figure to Inspect mode</i>	145
103	Botó <i>Change to Inspect mode</i>	145
104	Esfera generada al mode Inspecció	146
105	Imatges al visor	146
106	Imatge premuda per a navegar	147
107	Navegació a la part del model corresponent	147

Llista de Taules

1	Resum de les tasques	30
2	Cost elèctric	34
3	Cost d'internet	34
4	Cost d'espai de treball	35
5	Cost material i de programari	36
6	Costos genèrics	36
7	Resum costos personals	37
8	Hores destinades per a cada rol	37
9	Costos totals per a cada rol	37
10	Costos d'imprevistos	38
11	Costos totals	38
12	Models proporcionats per <i>VirVIG</i>	43
13	Fitxers proporcionats per <i>VirVIG</i>	49

Llista de Fragments de codi

1	Carregant models OBJ i MTL	44
2	Carregant models FBX	45
3	Carregant models glTF	46
4	Llistat d'imatges sense convertir	49
5	Script per convertir rutes d'imatges	50
6	Llistat d'imatges convertides	50
7	Representació d'una imatge en ThreeJS	53
8	Reducció de mida d'imatges	55
9	Posicionament d'una imatge en ThreeJS	57
10	Càlcul i aplicació de la rotació d'una imatge en ThreeJS	58
11	Matrius de rotacions obtingudes amb el Meshlab	60
12	Càlcul i aplicació de la rotació d'una imatge en ThreeJS	60
13	Establir mida d'una imatge	62
14	Càlcul posició d'imatges segons la separació	63
15	Integració del ray casting	64
16	Implementació del hovering	65
17	Implementació de la selecció d'images	66
18	Implementació del hovering d'images	67
19	Representació en filferros de les imatges	68
20	Creació d'una imatge actualitzat	70
21	Creació de filferros actualitzat	71
22	Primera imatge del raycast	71
23	Objecte amb informació de la imatge	72
24	Assignació de la ruta d'imatge com a nom d'objecte	72
25	Assignació de valors en crear l'objecte imatge	73
26	Creació de l'objecte amb informació d'una imatge	73
27	Conversió de coordenades de món a coordenades esfèriques	74
28	Càlcul d'un pla a ThreeJS	75
29	Representació d'un prisma a l'entorn 3D	76
30	Càlcul d'alçada i amplada del pla	76
31	Càlcul del centre del pla	77
32	Modificar amplada del prisma	78
33	Conversió de coordenades de món a coordenades del pla	79
34	Conversió de coordenades del pla a coordenades de món	80
35	Booleà que ens indica si una imatge es troba dins del prisma	81
36	Creació de l'objecte amb informació d'una imatge	81
37	Obtenció del centre de l'esfera	81
38	Representació d'una esfera a l'entorn 3D	82
39	Modificar radi de l'esfera	83
40	Booleà que ens indica si una imatge es troba dins de l'esfera	83
41	Creació de l'objecte amb informació d'una imatge	83
42	Obtenció dels centres de les dues cares d'un cilindre	84
43	Representació d'un cilindre a l'entorn 3D	85
44	Modificar radi del cilindre	86
45	Conversió de coordenades de món a coordenades del cilindre	87
46	Booleà que ens indica si una imatge es troba dins del cilindre	88

47	Creació de l'objecte amb informació d'una imatge	89
48	Generar i desar una esfera	90
49	Generar i desar un cilindre	90
50	Generar i desar un pla	91
51	Obtenció de la primera figura incidida pel raig	92
52	Obtenció de la primera figura incidida pel raig	92
53	Establir visibilitat de les imatges	92
54	Establir visibilitat de les figures reduïdes generades	92
55	Creació del panell de mode	95
56	Creació del panell Configuració d'imatges	96
57	Creació del panell de figures	97
58	Creació del panell d'imatges seleccionades	99
59	Creació del panell de l'esfera	101
60	Creació del panell del pla	102
61	Creació del panell del pla	104
62	HTML per a la barra d'informació	105
63	CSS per a la barra d'informació	105
64	Actualitzar mida del visor i perspectiva de la càmera	106
65	Establir text a la barra d'informació	106
66	Fitxer HTML que defineix el visor 2D	107
67	Definició d'una imatge a OpenSeadragon	107
68	Enllaç per a visualitzar una imatge al visor	108
69	Enllaç codificat	108
70	Crear i obrir l'enllaç	108
71	Visualitzar una imatge especificada a l'enllaç	109
72	Construcció d'un enllaç amb múltiples imatges	109
73	Visualitzar imatges especificades a l'enllaç	109
74	Escriptura a localstorage	110
75	Visualitzar imatges de localStorage	111
76	Conversió d'imatges de JGP a DZI amb MagickSlicer	111
77	Visualitzar imatges en format DZI	112
78	Conversió d'imatges de JGP a DZI amb deepzoom.py	113
79	Assignació de posicions a OpenSeaDragon	114
80	Assignació de posicions i mida a OpenSeaDragon	116
81	Càcul d'alçada segons el factor de zoom	117
82	Càcul d'alçada segons el factor de zoom	117
83	Actualització de l'alçada d'imatges	118
84	Obtenció de la posició en què intersecciona el raig	120
85	Creació de l'objecte amb les dues posicions	121
86	Actualització de les posicions	121
87	Resolució d'encavalcaments	122
88	Càcul de la intersecció entre dues imatges	124
89	Actualització de les posicions	124
90	Inicialitzar càlculs d'encavalcament una vegada s'hagin carregat les imatges	124
91	Càcul de la posició de la cantonada esquerra superior	125
92	Obtenció i escriptura del nom de la imatge premuda	126

93	Establir el zoom en clicar a 1	127
94	Establir la posició i rotació de la imatge	127
95	Establir la posició i punt objectiu dels controls de la càmera	128
96	HTML de la interfície gràfica	130
97	Estil de la interfície gràfica	131
98	Establir la posició i punt objectiu dels controls	131
99	Envolcall dels dos elements en un de sol	132
100	Estil per a la interfície flotant	132

1 Introducció

Des de l'arribada de la càmera fotogràfica, els humans han vist la necessitat de trobar la manera de poder organitzar les fotografies. Abans de l'arribada de la informàtica, les fotografies revelades s'organitzaven en àlbums físics, arxivadors, carrets, entre altres.

Més endavant, durant l'època de la digitalització, les càmeres ja començaven a fer imatges digitals, és a dir, que en contres d'imprimir-les en un carret o en papers d'impressió, es podien visualitzar des d'una pantalla. Amb les càmeres digitals, van començar a aparèixer les primeres galeries d'imatges, tan incorporades a les mateixes càmeres, com als ordinadors.

Tant en l'època de les imatges impreses com en les digitals, les fotografies s'organitzaven d'una manera molt similar, mostrar un llistat de les imatges, una darrera l'altra amb un cert ordre. Fins i tot, en galeries actuals i populars com *Google Photos* [1], aquesta és la manera determinada de mostrar les fotografies, com a l'exemple que podeu veure a la Figura 1.

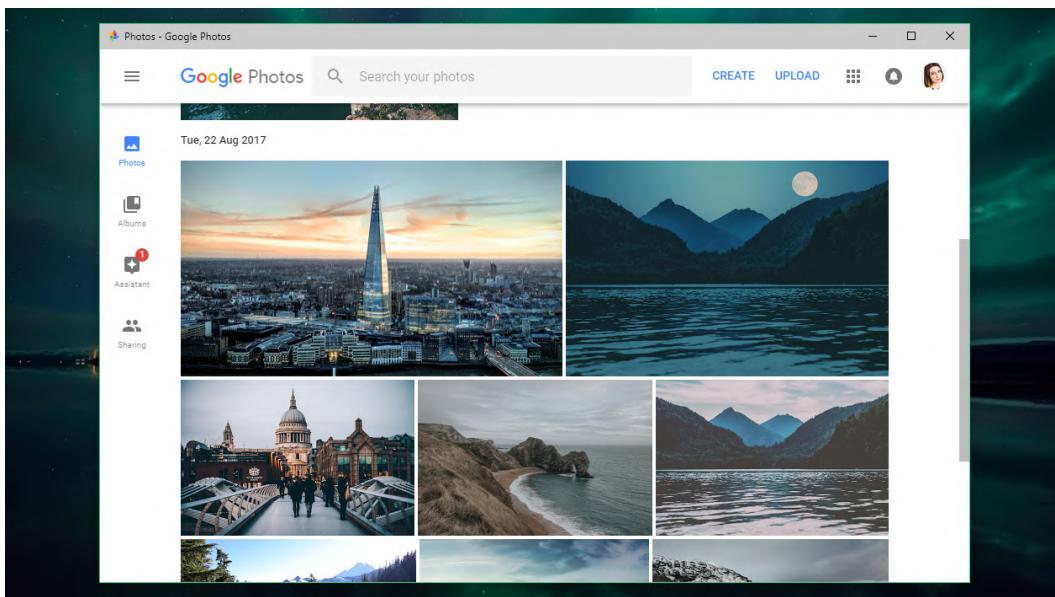


Figura 1: Captura de la galeria *Google Photos*
Font: techradar.com

1.1 Contextualització

Aquest Treball de Fi de Grau es cursa en Modalitat A, situat dins del marc de la Facultat d'Informàtica de Barcelona a la Universitat Politècnica de Catalunya. El projecte forma part de les línies de recerca del grup *ViRVIG* [2], que realitza activitats de recerca, educació i formació en visualització, modelatge geomètric i de volum, animació, renderització, realitat virtual i interacció avançada.

1.2 Definició de conceptes

Abans de definir el problema a resoldre és convenient que ens familiaritzem amb els conceptes de més rellevància.

1.2.1 Galeria d'imatges

Una galeria d'imatges és un programa informàtic que permet navegar, organitzar i gestionar imatges digitals. Habitualment aquestes galeries mostren les miniatures de les diverses imatges en format de quadrícula o de matriu, és a dir, organitzades en diferents files i columnes. En seleccionar les imatges, un explorador d'imatge permet obrir-les en un visor d'imatges.

1.2.2 Visor d'imatges

Un visor d'imatges és un programa informàtic que permet visualitzar imatges digitals. A més de mostrar les imatges, alguns visors d'imatges ofereixen funcions addicionals com ara: ampliar i reduir la imatge per a veure els detalls, retallar una part de la imatge, fer ajustaments bàsics com ajustar la brillantor i el contrast, visualitzar la imatge en pantalla completa, navegar a la imatge prèvia o següent, entre altres.

1.2.3 Entorn 3D

Un entorn 3D és un espai virtual que simula el món real en tres dimensions. A diferència dels entorns 2D, que només tenen alçada i amplada, els entorns 3D tenen alçada, amplada i profunditat. Això permet crear una experiència més immersiva i realista per a l'usuari.

Els entorns 3D s'utilitzen en una gran varietat d'aplicacions, com ara: els videojocs, la realitat virtual, la realitat augmentada, entra altres.

Principalment, aquests entorns es componen d'una càmera que defineix la perspectiva des de la qual es veu l'entorn, una geometria que defineix la forma dels objectes, uns materials que defineixen l'aspecte d'objectes, i una il·luminació que defineix quins efectes provoca als objectes la llum que els incideix.

1.3 Identificació del problema

Després de definir els principals conceptes, ja podem identificar el problema en qüestió.

Els exploradors d'arxius tradicionals permeten trobar fitxers segons el seu nom i la seva organització en directoris. Això és molt eficaç quan coneixes el nom de l'arxiu que busques, o coneixes el directori on se situa. Però què passa quan en un sol directori tens una gran quantitat de fitxers amb noms no representatius? Doncs que destines una gran quantitat de temps buscant entre els fitxers un a un.

El tema de la cerca d'imatges no és diferent d'aquest. Pots tenir en un sol directori centenars o milers de fotografies d'una certa temàtica, i per buscar la fotografia t'has de fixar en les miniatures que ofereix l'explorador d'imatges.

Aquest mateix problema és el que tenen els historiadors. Poden fer milers de fotografies a un monument d'interès per tal de no deixar escapar cap detall, i els resulta complicat trobar una imatge determinada.

El projecte pretén proposar una solució per a facilitar la cerca d'imatges d'un cert monument de la manera més ràpida i eficaç possible.

1.4 Actors implicats

En aquest projecte podem trobar tres grans grups d'actors implicats.

En primer lloc, trobem el personal de desenvolupament del projecte. Aquest grup és responsable de dur a terme la implementació del producte. Tot i ser un desenvolupament individual, l'equip està format per 3 integrants, l'estudiant, en aquest cas jo, i el director i codirector del projecte que s'asseguraran que el producte compleixi amb els objectius i fer un seguiment sobre aquest.

El projecte desenvolupat forma part del grup de recerca *ViRVIG* [2], que du a terme activitats de recerca, educació i formació en visualització, modelatge geomètric i de volum, animació, renderització, realitat virtual i interacció avançada. Aquest grup està interessat que el projecte es dugui a terme correctament.

Finalment, podem trobar els usuaris finals del producte, en el nostre cas són principalment els historiadors. Aquest grup és el que utilitzarà el programari.

2 Justificació

Una vegada feta la introducció i contextualització del treball, seguirem amb la justificació, on s'estudiaran les principals alternatives, i es raonarà la selecció del programari.

2.1 Estudi d'alternatives

Per a justificar la creació del nou visor de fotografies en 3D, avaluarem les principals alternatives existents. Principalment, podem trobar l'alternativa *PhotoCloud*, però també podem trobar *MeshLab*.

2.1.1 PhotoCloud

PhotoCloud [3] és un sistema client-servidor per a l'exploració interactiva de grans dimensions de dades que inclouen models 3D d'alta complexitat i milers de fotografies calibrades sobre les dades 3D.

Aquest programa permet trobar fàcilment les fotografies d'un monument segons la seva posició i orientació. Té moltes similituds amb el producte proposat en aquest treball. A continuació es visualitza com es relacionen les imatges amb el model 3D en aquest programa (vegeu Figura 2).

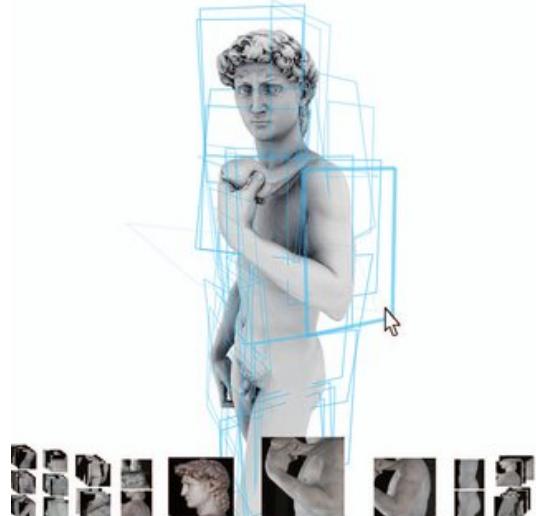


Figura 2: Imatges i un model 3D a *PhotoCloud*

Font: ISTI-CNR Visual Computing Lab. (2013). *PhotoCloud: interactive remote exploration of joint 2D and 3D datasets*.

Sent la principal alternativa, ens pot fer pensar si realment la nostra eina és necessària o no. Hem trobat dos grans desavantatges d'aquesta eina. El primer és que *PhotoCloud* és una eina enfocada a la fotogrametria, fent que incorpori eines que facilitin saber la posició des d'on s'ha fet la foto en contres de donar facilitats per trobar una certa imatge. El segon desavantatge és que l'eina no ofereix suport web, fent que no sigui tan accessible per als historiadors sense ordinadors potents.

Es descarta l'opció de modificar i adaptar *Photocloud* a les nostres necessitats, ja que al no tenir suport web, s'hauria de fer tota la implementació de nou amb un llenguatge enfocat al desenvolupament web.

2.1.2 MeshLab

Meshlab [4] és un sistema de codi obert per processar i editar malles triangulars 3D. Proporciona un conjunt d'eines per editar, netejar, inspeccionar, renderitzar, texturar i convertir malles. Ofereix funcions per processar dades en brut produïdes per eines/dispositius de digitalització 3D i per preparar models per a la impressió 3D.

Aquesta eina és capaç de mostrar els punts des d'on s'ha fet la foto, però hi ha una sèrie de desavantatges que ens fa decantar per a rebutjar aquesta opció. De fet, aquesta eina ens és molt útil per indicar des d'on s'ha fet la fotografia, fent-nos de suport en aquest projecte. A la Figura 3 es mostren les càmeres d'una estàtua, és a dir, els punts i direcció des d'on s'han pres les fotografies.

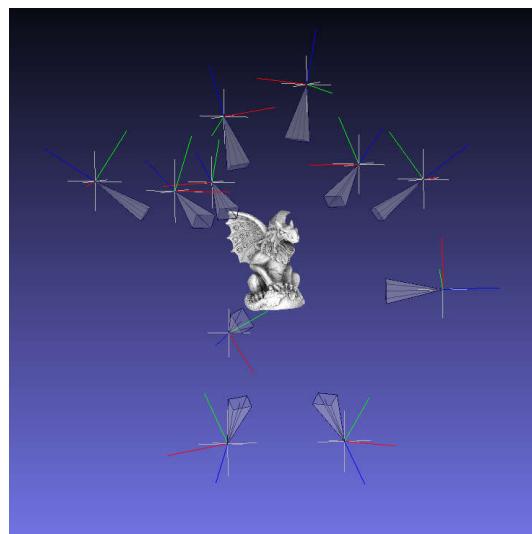


Figura 3: Estàtua envoltada de càmeres a *Meshlab*

Font: meshlab.net

Aquesta és una eina que no està enfocada pel nostre cas d'ús específic. Això fa que tingui una gran quantitat d'eines i opcions innecessàries pel nostre projecte. Tot i això, aquesta eina sí que té suport web, però la manca d'eines específiques per al nostre projecte, ens fa descartar l'ús d'aquesta alternativa.

També, es descarta l'opció de modificar i adaptar *MeshLab* a les nostres necessitats, ja que l'eina és molt complexa. Aquesta té tantes opcions i dependències, que ens resultaria molt difícil fer una adaptació correcta amb les funcionalitats desitjades.

2.2 Selecció del programari

Com s'explicarà més endavant, un dels objectius del projecte és que l'eina tingui suport web. Per aquest motiu, s'ha optat per fer el desenvolupament amb el llenguatge *JavaScript* juntament amb alguns frameworks que ofereix.

2.2.1 Gràfics 3D

Com s'explicarà més endavant, l'aplicació requerirà gràfics en 3D, per això s'han valorat les següents biblioteques i frameworks de *JavaScript* capaces de renderitzar aquesta classe de gràfics:

WebGL [5] és un API de *JavaScript* per mostrar gràfics interactius en 2D i 3D en qualsevol navegador. En treballar en baix nivell, dificulta el desenvolupament del projecte. Per aquest motiu es va decidir seleccionar un framework d'alt nivell que executés *WebGL*.

ThreeJS [6] és un framework d'alt nivell de *JavaScript* que s'utilitza per crear i mostrar gràfics animats 3D en un navegador web mitjançant *WebGL*.

Unity [7] és un motor 3D que permet exportar els projectes en múltiples plataformes, incloent en *WebGL*, ideal per al nostre treball.

Després de fer una anàlisi valorant *ThreeJS* i *Unity*, hem optat per seleccionar *ThreeJS*, ja que ens permetrà integrar més fàcilment integrar el framework per visualitzar les imatges en alta resolució, tal com veurem a continuació.

2.2.2 Visor d'imatges

Un altre dels objectius que definirem més endavant és poder visualitzar fotografies d'alta qualitat, podent interactuar amb aquestes. S'entén com a interactuar el fet de poder fer zoom, i moure's ns per la imatge. Per tal d'assolir aquest objectiu, s'han valorat les següents opcions:

ViewerJS [8] és un framework de *JavaScript* que permet visualitzar, fer zoom, girar i escalar imatges.

PhotoSwipe [9] és un framework de *Javascript* que permet integrar una galeria d'imatges. Aquest també permet fer les principals interaccions en una imatge.

OpenSeaDragon [10] és un visor web de codi obert per a imatges amb zoom d'alta resolució, implementat en *JavaScript* pur.

Després de fer una anàlisi valorant els diferents candidats, hem optat per seleccionar *OpenSeaDragon*, ja que està enfocat a les imatges d'alta resolució. Els altres candidats no estaven enfocats en l'alta resolució, fent que els descartéssim.

3 Abast

En aquesta secció definirem quin és l'abast del treball, llistant els diferents objectius i subobjectius, els requeriments, els obstacles i els riscos.

3.1 Objectius i subobjectius

En aquest projecte podem destacar dos grans objectius, amb els seus subobjectius corresponents, que veiem a continuació:

1. Estudiar la relació entre imatges 2D i models 3D.
 - (a) Estudiar la relació entre les imatges 2D amb l'espai 3D que es van capturar.
 - (b) Estudiar la relació entre els espais de captació físics i els models 3D.
2. Visualitzar les imatges 2D tenint en compte l'espai 3D.
 - (a) Visualitzar les imatges en un entorn 3D.
 - (b) Visualitzar les imatges amb els models 3D.

3.2 Requeriments

Després de descriure els objectius i els subobjectius del projecte, destacarem els diversos requeriments. Els podem dividir en dues categories, en requeriments funcionals i requeriments no funcionals.

3.2.1 Requeriments funcionals

Tot i que la majoria dels requeriments funcionals es trobin emmarcats dins dels objectius i subobjectius esmentats anteriorment, també podem trobar els següents, que van molt lligat entre ells:

Format d'imatges: És convenient que l'aplicació pugui llegir els principals formats d'imatges, com poden ser *JPG*, *PNG*, *GIF*, entre altres.

Format dels models 3D: De la mateixa manera, és oportú que l'aplicació pugui llegir els principals formats de models 3D, com poden ser *OBJ*, *FBX*, *GLTF*, entre altres.

3.2.2 Requeriments no funcionals

A més a més dels requeriments funcionals, també llistem quins són els requeriments no funcionals del projecte. Principalment en podem trobar dos:

Eficiència: La gran qualitat i quantitat de fotografies, juntament amb el model del monument en qüestió, fa que l'aplicació necessiti una gran quantitat de recursos per tal que s'executi amb fluïdesa. Usuaris amb màquines poc potents haurien de poder utilitzar l'aplicació. Per mesurar aquesta fluïdesa s'utilitzarà una extensió que mostra el nombre de *frames* per segon.

Usabilitat: La interacció amb un entorn 3D sol ser complex. Per això és important que l’aplicació sigui fàcil d’utilitzar-se, ja que els usuaris poden no tenir experiència prèvia en els moviments en entorns 3D. Per mesurar la usabilitat es farà enquestes sobre la facilitat d’ús, fixant-nos amb la ràtio de respostes satisfactòries sobre el total.

3.3 Obstacles

Aquest projecte, com tots els altres no està lliure d’obstacles. En aquest, podem trobar els següents:

El primer obstacle és la inexperiència dels frameworks de JavaScript utilitzats. Caldrà fer un aprenentatge de *ThreeJS* [6] i *OpenSeaDragon* [10] abans de començar amb el desenvolupament del projecte.

Un dels obstacles més importants és l’optimització. L’aplicació haurà de generar models 3D amb textures, i milers de fotografies d’alta qualitat. Això fa que en dispositius que no tenen una gran capacitat de computació, el temps de càrrega del programa sigui molt gran, i que l’aplicació no s’executi amb fluïdesa.

3.4 Riscos

Tot projecte, inclòs el nostre, no està lliure de riscos. En el nostre cas, podem trobar els següents:

Aquest projecte depèn de dades emmagatzemades al núvol, com el repositori de *GitHub*[11] que manté el codi, i *Overleaf* [12] on es desenvolupa la memòria del treball. Durant el desenvolupament podria que algun servidor d’aquests serveis no estigui disponible i no es pugui tenir accés immediat als recursos. Un dels pitjors casos seria la pèrdua permanent de les dades en algun servei. Per tal de reduir-lo, es mantindrà còpies de seguretat en el dispositiu local i en dispositius d’emmagatzematge extern.

Un altre risc a tenir en compte és l’endarreriment del desenvolupament a causa d’imprevistos produïts durant el desenvolupament. A conseqüència d’aquests fets, podríem no complir certs terminis marcats a la planificació. Fins i tot ens podríem veure forçats a descartar alguna funcionalitat o objectiu per tal de garantir l’entrega del treball abans de la data límit.

4 Metodologia

En aquesta secció es descriurà quina metodologia s'efectuarà per a aquest treball, i quin serà el mètode de seguiment.

4.1 Metodologia Àgil

Per a fer aquest projecte s'utilitzarà una metodologia àgil, concretament el mètode *SCRUM* [13]. La metodologia *SCRUM* és un marc de treball àgil per a la gestió de projectes, especialment útil per al desenvolupament de programari i productes complexos. El seu objectiu principal és facilitar la creació d'un producte final que satisfaci les necessitats dels clients de manera ràpida i flexible.

Aquesta metodologia s'organitza en *sprints*, cicles de treball curts que es divideixen en una sèrie de tasques. Normalment, aquestes tasques són la planificació, el disseny, el desenvolupament, la verificació, el control i la publicació.

En aquest projecte, cada *sprint* tindrà una duració aproximada d'una setmana. Cada *sprint* comença amb la reunió de seguiment, on es fa la planificació. Durant la setmana es fan les fases de disseny, implementació i testatge, i acaba amb la revisió a la reunió de seguiment. És a dir, cada reunió de seguiment es farà la revisió del *sprint* anterior, i la planificació del *sprint* posterior. Podeu veure amb més detall el propòsit de les reunions de seguiment a l'apartat 4.3.

S'ha seleccionat aquesta metodologia, ja que ofereix una major velocitat en el desenvolupament i una millor adaptació als canvis que es puguin produir.

4.2 Eines de seguiment

Per a fer la gestió i el seguiment del projecte s'utilitzaran una sèrie d'eines i aplicacions per facilitar-ho.

Per al sistema de control de versions s'utilitzarà l'eina *Git* [14] juntament amb un repositori de *GitHub* [11]. Aquestes eines facilitaran aplicar els canvis del projecte durant el desenvolupament. La branca principal (*main*) estarà protegida de manera que sigui necessari fer un *Pull Request* per a cada canvi a aquesta branca, i en cas que calgui desfer un canvi, es pot crear fàcilment un *Pull Request* desfent-lo.

Per tal de gestionar els objectius i establir tasques s'utilitzarà l'eina *Projects* [15] que ve integrat amb *GitHub*. Aquesta eina és un tauler que permet definir iteracions de treball amb diversos objectius, i s'integra amb les *issues* i els *Pull Requests* d'un repositori de *GitHub* determinat.

4.3 Mètode de validació

Una part de la validació del projecte s'efectuarà mitjançant una reunió setmanal amb el director i el codirector del treball. En aquestes sessions de seguiment el director i el codirector validen que projecte avanci correctament, complint els terminis establerts. En aquestes reunions també es discuteixen dubtes i problemes que puguin sorgir durant el desenvolupament, i possibles millors per tal d'incrementar la qualitat del producte.

D'altra banda, també s'estableix una comunicació directa mitjançant Discord, entre l'autor, el director i codirector per a discutir problemes i dubtes puntuals sense haver d'esperar a la pròxima reunió de seguiment.

5 Planificació temporal

Per tal de completar el projecte amb el termini establert i amb totes les funcionalitats s'ha fet una planificació temporal del projecte. La data d'inici d'aquest treball és el 30 de gener de 2024 amb la reunió inicial, i la de finalització és entre el 25 i el 28 de juny de 2024 segons el dia la defensa. En total la duració és de 147 a 150 dies, i considerant una dedicació diària de 3 hores i mitja, la dedicació total estaria entre les 514.5 hores (147×3.5) i les 525 hores (150×3.5).

A la facultat, la normativa sobre els Treballs de Fi de Grau [16] indica que el nombre de crèdits del treball són 18, i que la càrrega estimada és de 30 hores per cadascun d'aquests crèdits. Tenint en compte aquests valors, la dedicació total estimada és de 540 hores, fent que les hores estimades d'aquest projecte s'ajustin amb les estimes per la facultat.

5.1 Recursos

Com tots els projectes, necessitem una sèrie de recursos per tal de poder-lo dur a terme. Aquests els podem dividir en humans i materials. A continuació podeu veure els recursos vinculats al nostre projecte.

5.1.1 Recursos humans

En el cas dels recursos humans, podem trobar 4 rols implicats en aquest projecte: el *director de projectes*, el *programador*, l'*analista* i el *redactor*.

- El *director de projectes* [DP] és l'encarregat de fer la planificació del projecte i guiar les reunions amb l'equip.
- El *programador* [PR] s'encarrega de fer el desenvolupament de l'aplicació.
- L'*analista* o *tester* [AN] és l'encarregat de provar que l'aplicació funcioni correctament i trobi errors.
- El *redactor* [RE] és el que s'encarrega d'elaborar la memòria i de redactar els informes de la gestió del projecte.

5.1.2 Recursos materials

Els recursos materials i de programari necessaris per a l'elaboració del nostre projecte són els següents:

- Un *portàtil* [PO] per a cada integrant és necessari per dur a terme totes les tasques.
- L'*Overleaf* [OV] és un editor *LaTeX*, que s'utilitzarà per a l'elaboració dels informes i de la memòria.
- Els *frameworks de JavaScript* [JS] *ThreeJS* i *OpenSeaDragon* són necessaris per al desenvolupament de l'aplicació.
- El *paquet d'office* [OF] inclou programaris de full de càcul i de presentacions, necessaris per a la gestió del projecte, i per l'elaboració de la presentació

que dona suport a la defensa.

5.2 Descripció de les tasques

En aquest apartat es fa un desglossament de les tasques del projecte, juntament amb les seves descripcions, les precedències de cadascuna, i la seva duració estimada.

5.2.1 Gestió del projecte [G]

Abans de començar a desenvolupar el projecte és important fer estudis sobre alternatives existents, definir l'abast, planificar les tasques, entre altres. Aquesta tasca tindrà una duració aproximada de 90 hores, i es divideix en les 4 subtasques que podeu veure a continuació.

5.2.1.1 Context i Abast [G1]

La primera tasca a fer en un projecte d'aquestes característiques és estudiar alternatives existents, i marcar el seu abast. És necessari haver completat la lectura de treballs similars (tasca P3) abans de començar la tasca. Aquesta tasca també inclou la seva documentació, i per tant la seva duració aproximada és de 35 hores.

5.2.1.2 Planificació temporal [G2]

Una vegada s'han marcat els objectius del projecte, cal dividir aquest en tasques i subtasques, i fer una estimació de la duració de cadascuna d'aquestes. També cal definir les dependències de cada tasca, els recursos necessaris, entre altres. També, és necessari haver completat la lectura de treballs similars (tasca P3) abans de començar la tasca. De la mateixa manera que la subtasca anterior, també inclou la documentació de la tasca, i es calcula que tindrà una durada de 25 hores.

5.2.1.3 Pressupost i sostenibilitat [G3]

També, cal fer un pressupost del cost que té el projecte, i fer un informe sobre la sostenibilitat d'aquest. De la mateixa manera, és necessari haver completat la lectura de treballs similars (tasca P3) abans de començar la tasca. S'estima que la duració d'elaboració d'aquests dos informes sigui d'unes 25 hores.

5.2.1.4 Informe Gestió de Projecte [G4]

Per acabar la Gestió del projecte, cal recopilar els informes ja elaborats del context i abast, de la planificació, del pressupost i de la sostenibilitat. Caldrà fer algunes millors i canvis, fent que la duració pugui arribar a les 5 hores.

5.2.2 Treball previ [P]

Abans de començar amb el desenvolupament i l'elaboració de la memòria cal fer alguns preparatius addicionals, que arriben a les 85 hores. Les subtasques implicades es troben a continuació.

5.2.2.1 Aprenentatge de ThreeJS [P1]

Al no tenir cap coneixement previ del framework *ThreeJS*, caldrà fer un aprenentatge abans de començar el desenvolupament de l'entorn 3D. L'aprenentatge d'aquest framework tindrà una duració d'unes 30 hores. La tasca s'iniciarà una vegada que es planifiqui el seu inici (tasca G4).

5.2.2.2 Aprenentatge d'OpenSeaDragon [P2]

De la mateixa manera que el framework previ, tampoc tinc cap coneixement del framework *OpenSeaDragon* pel desenvolupament de l'entorn 2D. Caldrà fer un aprenentatge sobre aquest abans d'iniciar el desenvolupament. L'aprenentatge d'aquest framework tindrà una duració d'unes 15 hores, ja que els entorns 2D tenen menys complexitat que els 3D. De la mateixa manera que la subtasca anterior, s'iniciarà una vegada que es planifiqui el seu inici (tasca G4).

5.2.2.3 Lectures de Treballs de Fi de Grau previs [P3]

Abans de començar la gestió del projecte i l'elaboració de la memòria, es farà la lectura de diversos Treballs de Fi de Grau per fer-se una idea del procés d'un TFG. Es llegirà dos treballs de temàtiques relacionades amb aquest. Les lectures en profunditat i detingudament per a entendre'ls correctament poden arribar a les 5 hores a cadascun, destinant 10 hores en total. Aquesta tasca no té precedents.

5.2.2.4 Estudi de la relació entre imatges 2D i models 3D [P4]

Un dels objectius marcats era l'estudi de la relació entre les imatges 2D i els models 3D. S'estudiarà la relació entre les imatges 2D amb l'espai 3D que es van captar, i la relació entre els espais de captació físics i els models 3D. Aquest estudi pot ser complex, i pot arribar a les 45 hores de duració. De la mateixa manera, aquesta tasca no té precedents.

5.2.3 Desenvolupament entorn 3D [T]

Aquesta tasca correspon al desenvolupament de l'entorn 3D mitjançant el framework *ThreeJS*. Aquesta és la tasca de més pes de tot el projecte, arribant a les 115 hores. La podem dividir en 3 subtasques.

5.2.3.1 Disseny [T1]

Es determinarà totes les funcionalitats i aparences que ha de tenir l'entorn 3D, i quins components s'utilitzarà. El disseny per al desenvolupament de l'entorn 3D serà d'unes 10 hores. Cal haver completat l'aprenentatge de *ThreeJS* abans d'iniciar la tasca.

5.2.3.2 Implementació [T2]

Desenvolupament del codi encarregat de l'entorn mitjançant el framework *ThreeJS*. Aquesta tasca tindrà una duració llarga, d'unes 110 hores. El disseny (tasca T1) és necessari abans d'iniciar la tasca.

5.2.3.3 Proves [T3]

Durant el transcurs del desenvolupament de l'entorn 3D és important fer proves periòdicament, juntament amb una prova final en profunditat. Es calcula que aquesta subtasca tingui una duració d'unes 10 hores.

5.2.4 Desenvolupament entorn 2D [O]

Aquesta tasca correspon al desenvolupament de l'entorn 3D mitjançant el framework *OpenSeaDragon*. Aquesta és la tasca de més pes de tot el projecte, arribant a les 35 hores. La podem dividir en 3 subtasques.

5.2.4.1 Disseny [O1]

Es determinarà totes les funcionalitats i aparences que ha de tenir l'entorn 2D, i quins components s'utilitzarà. El disseny per al desenvolupament de l'entorn 2D serà d'unes 10 hores. Cal haver completat l'aprenentatge de *OpenSeaDragon* abans d'iniciar la tasca.

5.2.4.2 Implementació [O2]

Desenvolupament del codi encarregat de l'entorn mitjançant el framework *OpenSeaDragon*. Aquesta tasca tindrà una duració d'unes 20 hores. El disseny (tasca O1) és necessari abans d'iniciar la tasca.

5.2.4.3 Proves [O3]

Durant el transcurs del desenvolupament de l'entorn 2D és important fer proves periòdicament, juntament amb una prova final en profunditat. Es calcula que aquesta subtasca tingui una duració d'unes 10 hores.

5.2.5 Elaboració de la memòria [M]

Aquesta tasca inclou l'elaboració de la memòria del Treball de Fi de Grau, amb l'excepció de les seccions corresponents a la Gestió del Projecte, que ja inclouen aquesta part en les seves tasques. L'elaboració de la memòria és de 90 hores, tenint en compte que la llargada de la memòria sigui d'entre 60 i 100 pàgines. Aquesta elaboració serà present una vegada s'hagi completat la lectura de treballs amb temàtiques relacionades amb aquest (tasca P3).

5.2.6 Defensa [D]

Defensa del Treball de Fi de Grau. També inclou la preparació del material de la defensa del Treball, juntament amb la preparació de la defensa. La duració estimada per a la defensa i la seva preparació és de 40 hores. S'iniciarà una vegada s'hagi completat la memòria (tasca M).

5.2.7 Reunions [R]

Per tal de fer un seguiment del projecte, es farà una reunió setmanalment durant 19 setmanes, de duració aproximada d'una hora, amb el director i el codirector del treball. A més a més, hi haurà 3 reunions extraordinàries: la reunió inicial, la reunió

de fita intermèdia, i la reunió final. En total es calcula que el temps destinat a les reunions serà d'unes 25 hores.

5.3 Representacions gràfiques de les tasques

Per la fàcil comprensió de les tasques, s'ha elaborat una taula que les resumeix (Taula 1). Per facilitar la visualització de les precedències entre tasques, s'ha elaborat un diagrama de dependències (Figura 4). Finalment, s'ha elaborat un diagrama de Gantt (Figura 5) per mostrar la planificació temporal de les tasques i subtasques.

Id.	Tasca	Duració	Dependències	Recursos
G1	Context i Abast	35h	P3	DP, RE, PO, OV, OF
G2	Planificació temporal	25h	P3	DP, RE, PO, OV, OF
G3	Pressupost i sostenibilitat	25h	P3	DP, RE, PO, OV, OF
G4	Informe Gestió de Projecte	5h	G1, G2, G3	RE, PO, OV
G	Gestió del projecte	90h		
P1	Aprenentatge ThreeJS	30h	G4	PR, PO, JS
P2	Aprenetatge OpenSeaDragon	15h	G4	PR, PO, JS
P3	Lectures TFGs previs	10h	-	DP, RE, PO
P4	Estudi entre imatges i models	45h	-	DP, PR, RE, PO
P	Treball previ	100h		
T1	Disseny	10h	P1, P4	PR, PO, JS
T2	Implementació	110h	T1	PR, PO, JS
T3	Proves	10h	T1	PR, AN, PO, JS
T	Desenvolupament 3D	130h		
O1	Disseny	10h	P2, P4	PR, PO, JS
O2	Implementació	20h	O1	PR, PO, JS
O3	Proves	10h	O1	PR, AN, PO, JS
O	Desenvolupament 2D	40h		
M	Elaboració de la memòria	90h	P3	RE, PO, OV
D	Defensa	40h	M	DP, PO, OF
R	Reunions	25h	-	DP, PR, AN, RE, PO
	Total	515h		

Taula 1: Resum de les tasques

Font: Producció pròpia

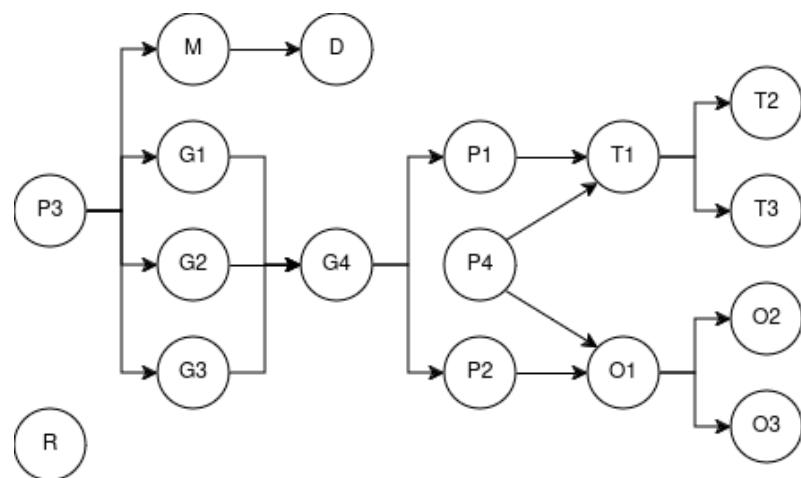


Figura 4: Graf de dependències de tasques

Font: Producció pròpria, fet amb *draw.io*

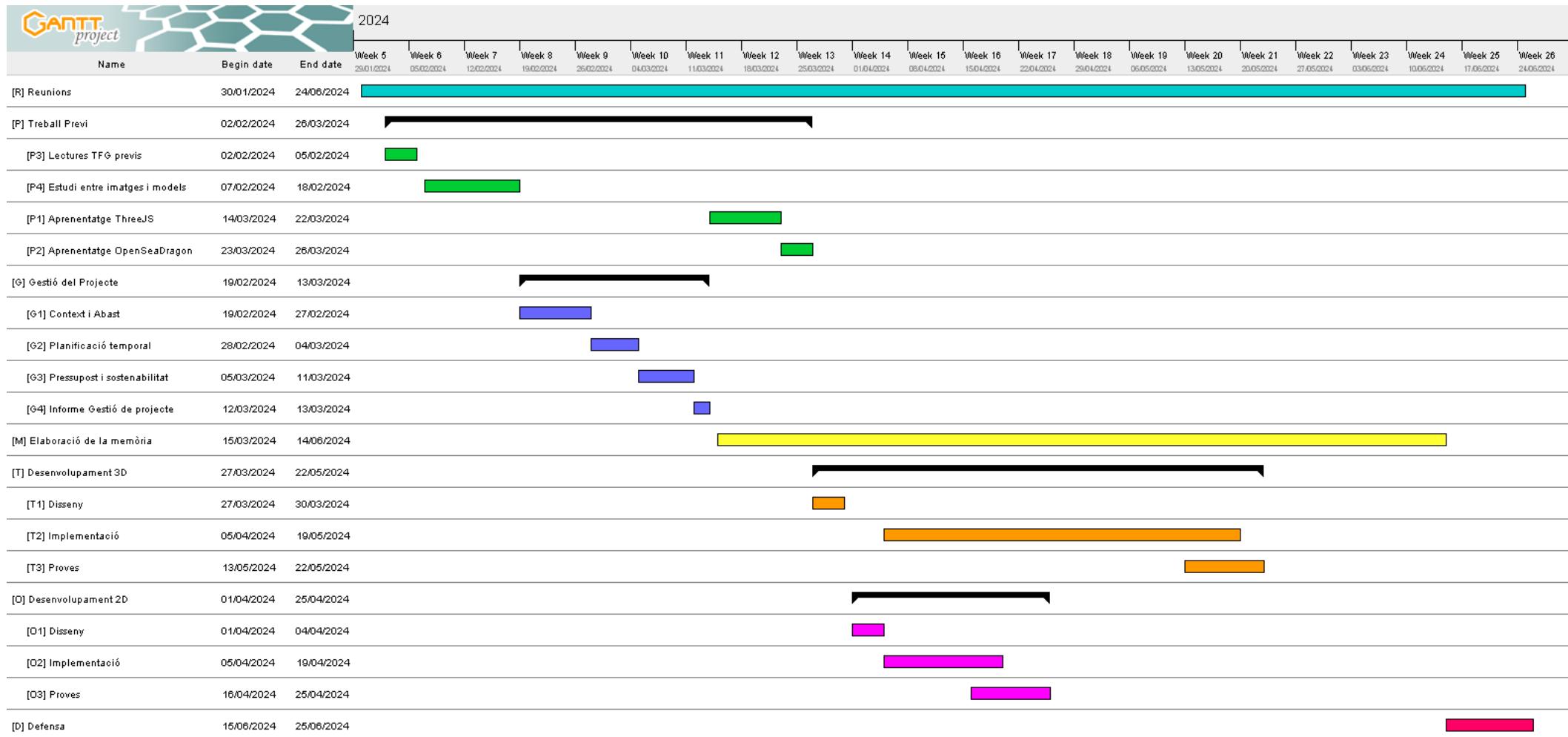


Figura 5: Diagrama de Gantt
Font: Producció pròpia, fet amb *GanttProject*

5.4 Gestió del risc: Plans alternatius i obstacles

Tal com s'ha explicat anteriorment, aquest projecte no està lliure de riscos. Es poden produir una sèrie d'imprevistos, fent que es modifiqui lleugerament parts del projecte. Per tant, a continuació es llisten una sèrie de possibles obstacles, juntament amb els plans alternatius per tal de reduir els efectes sobre el projecte.

- **Endarreriment desenvolupament entorn 3D:** La planificació temporal té en compte possibles imprevistos, incrementant l'estimació d'hores per tasca per tal de reduir els efectes sobre el projecte. En principi, aquestes hores addicionals planificades haurien de ser suficients per mitigar aquests problemes. En el cas que no fos així, ens veuríem vistos a incrementar la dedicació per al desenvolupament de l'entorn 3D. En el cas que siguem superats per la dificultat i anéssim endarrerits significativament, podríem prendre la decisió de sacrificar funcionalitats no crítiques com l'aparença de la interfície gràfica. La probabilitat que se succeeixi és d'un 35%.
- **Endarreriment desenvolupament entorn 2D:** Molt similar que a l'entorn 3D, es té en compte a la planificació temporal un increment de dedicació causat per imprevistos. En el cas que les hores no fossin suficients, ens veuríem vistos obligats a incrementar les hores dedicades per tal de no endarrerir el projecte. En aquest cas, no es preveu sacrificar funcionalitats. Aquest risc té la mateixa probabilitat de successió que l'anterior, aproximadament un 35%.
- **Problemes d'eficiència entorn 3D:** S'ha tingut en compte en la planificació temporal el temps necessari per poder optimitzar l'escena 3D, fent que corri de manera fluida. En el cas que no sigui així, ens podríem veure obligats a reduir la complexitat de l'entorn 3D i la qualitat les imatges. La probabilitat d'aquest risc és la més alta, arribant al 50%.
- **Problemes d'eficiència entorn 2D** De la mateixa manera que a l'entorn 3D, s'ha tingut en compte en la planificació temporal el temps necessari per poder optimitzar l'escena 2D. En aquest cas, hauríem de reduir mínimament la qualitat de les imatges. La complexitat, al ser significativament menor que l'entorn 3D, la probabilitat és més baixa, aproximadament un 15%.

6 Gestió econòmica

En aquesta secció es mostraran els costos que comporta l'execució del projecte, juntament amb una especificació del control d'aquests costos.

6.1 Identificació i estimació de costos

A continuació es mostren quins són els costos que comporta el projecte, dividits segons la tipologia d'aquest. Podem trobar els costos genèrics, costos de personal, costos de contingència i els costos d'imprevistos.

6.1.1 Costos genèrics

Els costos genèrics són aquells que no depenen de les tasques del projecte, però hi intervenen. En aquesta categoria podem trobar el cost elèctric, el cost d'internet, el cost d'espai de treball i el cost de material i programari.

6.1.1.1 Cost elèctric

Per calcular el consum energètic s'ha consultat el preu per Quilowatt hora de la tarifa contractada. En el nostre cas surt 0,123161€/kWh [17]. També cal considerar que l'ordinador utilitzat té un consum de 62W [18], i que s'utilitzarà durant les 515 hores planificades. A continuació es pot visualitzar les fórmules utilitzades i la taula 2 amb el cost elèctric.

$$\text{Consum} = \text{Potència} * \text{Hores} / 1000 = 62\text{W} * 515\text{h} / 1000 = 31,93\text{KWh}$$

$$\text{Cost_total} = \text{Consum} * \text{Cost_KWh} = 31,93\text{KWh} * 0,123161\text{€/kWh} = 3,93\text{€}$$

Concepte	Potència	Hores	Consum	Cost KWh	Cost total
Portàtil	62W	515h	31,93KWh	0,123161€/kWh	3,93€

Taula 2: Cost elèctric

Font: Producció pròpia

6.1.1.2 Cost d'internet

Per al càlcul del cost d'internet es té en compte una factura de 27€/mes [19]. Amb aquest valor calculem quin és el cost per hora de l'internet per tal de poder calcular el cost total. Les fórmules següents mostren el càlcul del cost, juntament amb una taula resum (taula 3).

$$\text{Hores_mensuals} = 24\text{h} * 30\text{dies} = 720\text{h/mes}$$

$$\text{Cost/h} = \text{Cost_mes} / \text{Hores_mensuals} = 27\text{€/mes} / 720\text{h/mes} = 0,0375\text{€/h}$$

$$\text{Cost_total} = \text{Cost/h} * \text{Hores} = 0,0375\text{€/h} * 515\text{h} = 19,32\text{€}$$

Concepte	Cost/mes	Cost/h	Hores	Cost total
Internet	27€/mes	0,0375€/h	515h	19,32€

Taula 3: Cost d'internet

Font: Producció pròpia

6.1.1.3 Cost d'espai de treball

En el cost d'espai de treball es té en compte una oficina d'una superfície d'onze metres quadrats localitzada al barri de Gràcia de Sabadell. El preu mitjà al metre quadrat en aquesta zona és de $2.104\text{€}/m^2$ [20]. Per a calcular el nostre cost, es considera un temps de vida útil de l'espai de treball de 60 anys. A les fórmules i a la taula 4 podeu veure el cost total de l'espai de treball.

$$\text{Cost_espai} = \text{Superficie} * \text{Preu/m}^2 = 11\text{m}^2 * 2.104\text{€}/\text{m}^2 = 23144\text{€}$$

$$\text{Mesos_amortització} = 12 \text{ mesos} * \text{anys_amortitzacio} = 12 \text{ mesos} * 60 \text{ anys} = 720 \text{ mesos}$$

$$\text{Cost/mes} = \text{Cost_espai}/\text{Mesos_amortització} = 23.144\text{€} / 720 \text{ mesos} = 32,14\text{€}/\text{mes}$$

$$\text{Cost_total} = \text{Mesos} * \text{Cost/mes} = 5 \text{ mesos} * 32,14\text{€}/\text{mes} = 160,70\text{€}$$

Concepte	Cost	Cost/mes	Mesos	Cost total
Espai de treball	23.144€	32,14€/mes	5 mesos	160,70€

Taula 4: Cost d'espai de treball
Font: Producció pròpria

6.1.1.4 Cost material i de programari

El primer cost material és l'ordinador portàtil utilitzat. Aquest és un *Medion Erazer P6705* [18] amb un cost de 749€. Es considera un temps de vida útil de 7 anys. Podeu trobar el càlcul a continuació:

$$\text{Mesos_amortització} = 12 \text{ mesos} * \text{anys_amortització} = 12 \text{ mesos} * 7 \text{ anys} = 84 \text{ mesos}$$

$$\text{Cost/mes} = \text{Cost}/\text{Mesos_amortització} = 749\text{€} / 84 \text{ mesos} = 8,92\text{€}/\text{mes}$$

$$\text{Cost_total} = \text{Mesos} * \text{Cost/mes} = 5 \text{ mesos} * 8,92\text{€}/\text{mes} = 44,60\text{€}$$

També, podem trobar l'escriptori utilitzat, que té un cost de 129€ [21]. Considerem un temps de vida de 12 anys. El càlcul és el següent:

$$\text{Mesos_amortització} = 12 \text{ mesos} * \text{anys_amortització} = 12 \text{ mesos} * 12 \text{ anys} = 144 \text{ mesos}$$

$$\text{Cost/mes} = \text{Cost} / \text{Mesos_amortització} = 129\text{€} / 144 \text{ mesos} = 0,90\text{€}/\text{mes}$$

$$\text{Cost_total} = \text{Mesos} * \text{Cost/mes} = 5 \text{ mesos} * 0,90\text{€}/\text{mes} = 4,50\text{€}$$

A més a més, la cadira utilitzada té un cost de 79€ [22] amb un temps de vida útil de 10 anys. A continuació es recull les fórmules per aquest càlcul:

$$\text{Mesos_amortització} = 12 \text{ mesos} * \text{anys_amortització} = 12 \text{ mesos} * 10 \text{ anys} = 120 \text{ mesos}$$

$$\text{Cost/mes} = \text{Cost} / \text{Mesos_amortització} = 79\text{€} / 120 \text{ mesos} = 0,66\text{€}/\text{mes}$$

$$\text{Cost_total} = \text{Mesos} * \text{Cost/mes} = 5 \text{ mesos} * 0,66\text{€}/\text{mes} = 3,30\text{€}$$

Finalment, tot el programari utilitzat és de codi obert, fent que el nostre cost sigui de 0€.

A continuació es mostra la taula 5, que recull aquests costos descrits.

Concepte	Cost	Cost/mes	Mesos	Cost total
Portàtil	749€	8,92€/mes	5 mesos	44,60€
Escriptori	129€	0,90€/mes	5 mesos	4,50€
Cadira	79€	0,66€/mes	5 mesos	3,30€
Programari	0€	0€/mes	5 mesos	0€
Total		10,48€/mes	5 mesos	52,40€

Taula 5: Cost material i de programari

Font: Producció pròpia

6.1.1.5 Costos genèrics totals

Podem recollir tots aquests costos específics a la taula 6, i calculem el total dels costos genèrics.

Concepte	Cost
Electricitat	3,39€
Internet	19,32€
Espai de treball	160,70€
Material i programari	52,40€
Total	235,81€

Taula 6: Costos genèrics

Font: Producció pròpia

6.1.2 Costos personals

Tal com s'ha descrit a la secció 5.1, hi ha 4 rols implicats: un programador, un cap de projecte, un analista, i un redactor. El sou mitjà anual brut per a cada rol és: 25.500€ [23] per al programador, 36.000€ [24] per al cap de projectes, 31.500€ [25] per l'analista, i 20.000€ [27] per al redactor.

S'ha de tenir en compte que el sou brut no correspon amb el cost real de cada-cun d'aquests. També cal afegir les contribucions addicionals que es fa a hisenda. Cal afegir les contingències comunes, les prestacions per desocupació, les contingències professionals, la formació i el Fons de Garanties Salarials (FOGASA). El valor es pot aproximar al 33,4% [28] addicional sobre el sou brut.

Per a calcular el cost per hora, considerem que a Espanya es treballen aproximadament 1800 hores anuals [29]. A continuació podem trobar les fórmules utilitzades:

$$\text{Contribucions} = 0.334 * \text{Salari_anual_brut}$$

$$\text{Cost_anual} = \text{Salari_anual_brut} + \text{Contribucions}$$

$$\text{Cost/h} = \text{Cost_anual} / 1800h$$

A la taula 7 es troben el cost per hora de cada rol.

Rol	Salari anual brut	Contribucions	Cost anual	Cost per hora
Programador	25.500€	8.517€	34.017€	18,90€/h
Cap de projectes	36.000€	12.024€	48.024€	26,68€/h
Analista	31.500€	10.521€	42.021€	23,35€/h
Redactor	20.000€	6.680€	26.680€	14,82€/h

Taula 7: Resum costos personals
Font: Producció pròpia

Una vegada calculats el cost per hora, es divideix cada tasca en el temps destinat a cada rol. Aquesta divisió es pot trobar en la taula 8.

Tasca	Programador	Cap de projectes	Analista	Redactor
G1	0h	20h	0h	15h
G2	0h	15h	0h	10h
G3	0h	15h	0h	10h
G4	0h	0h	0h	5h
P1	30h	0h	0h	0h
P2	15h	0h	0h	0h
P3	0h	5h	0h	5h
P4	15h	15h	0h	15h
T1	10h	0h	0h	0h
T2	110h	0h	0h	0h
T3	5h	0h	5h	0h
O1	10h	0h	0h	0h
O2	20h	0h	0h	0h
O3	5h	0h	5h	0h
M	0h	0h	0h	90h
D	0h	40h	0h	0h
R	0h	25h	0h	0h
Total	220h	135h	10h	150h

Taula 8: Hores destinades per a cada rol
Font: Producció pròpia

Finalment, podem calcular els costos personals totals, tal com podeu veure a la fórmula i taula 9.

$$\text{Cost_total} = \text{Hores} * \text{Cost/h}$$

Rol	Cost per hora	Hores	Cost total
Programador	18,90€/h	220h	4.158€
Cap de projectes	26,68€/h	135h	3.601,80€
Analista	23,35€/h	10h	233,50€
Redactor	14,82€/h	150h	2.223€
Total			10.216,30€

Taula 9: Costos totals per a cada rol
Font: Producció pròpia

6.1.3 Costos de contingència

Com tot projecte, cal tenir en compte que poden aparèixer complicacions i contratemps, fent que el cost final del projecte s'incrementi. En el nostre cas, en ser un projecte de desenvolupament de programari, escollirem un nivell de contingència del 20%. A continuació es troba el càlcul corresponent:

$$\text{Cost_contingencia} = 20\% * (\text{Costos_generics} + \text{Costos_personal}) = 0.2 * (235,81\text{€} + 10.216,30\text{€}) = 2.090,42\text{€}$$

6.1.4 Costos d'imprevistos

A més a més, cal tenir en compte possibles imprevistos que poden incrementar el cost total. Principalment, podem trobar 2 imprevistos: la malfunció de l'ordinador, i un increment del temps de desenvolupament, tal com s'ha definit anteriorment a la secció 3.4. Aquests costos es recullen a la *Taula 10*, mitjançant la fórmula següent:

$$\text{Cost.total} = \text{Cost} * \text{Probabilitat}$$

Concepte	Cost	Probabilitat	Cost total
Ordinador portàtil	800€	20%	160€
Temps desenvolupament (30h de programador)	567€	35%	198,45€
Total			358,45€

Taula 10: Costos d'imprevistos

Font: Producció pròpia

6.1.5 Cost total

A la taula 11 es recullen tots els costos implicats en el projecte, sumant el cost total de cada secció.

Concepte	Cost
Costos genèrics	235,81€
Costos personals	10.216,30€
Costos de contingència	2.090,42€
Costos d'imprevistos	358,45€
Total	12.900,98€

Taula 11: Costos totals

Font: Producció pròpia

6.2 Control de gestió

Un cop definits els costos del projecte, es descriuen quins són els mecanismes per dur a terme el seguiment d'aquests.

Per a fer aquest seguiment, s'apuntarà les hores destinades a cada tasca, juntament amb un control exhaustiu dels costos. A cada reunió es farà el seguiment de les hores destinades a cada tasca respecte a les hores estimades durant el *sprint* corresponent a aquella setmana. De la mateixa manera, també es farà el seguiment dels

costos en relació amb els estimats. Així, es podran detectar possibles desviaments, i en el cas que sigui necessari, decidir quines mesures cal aplicar.

En el cas que la desviació de tasques no sigui favorable, i que les hores reservades a imprevistos estiguin exhaurides, caldrà augmentar considerablement la dedicació al projecte per tal de posar-lo al dia. Tal com s'ha especificat a la secció 3.4, en el cas que no sigui factible l'augment de dedicació, s'haurà de sacrificar l'estètica de la interfície gràfica per tal de no comprometre cap funcionalitat.

En el cas favorable que la desviació de tasques sigui positiu, es tirarà endavant les tasques planificades, reservant-nos el temps guanyat per possibles contratemps i imprevistos de les tasques posteriors.

A més a més de la desviació de tasques, tenim la desviació del cost. Podem veure a la Taula 11, que el gran gruix del cost prové de costos personals, juntament amb els costos de contingència. Aquests costos venen directament relacionats amb les hores dedicades al projecte, fent que en el cas que s'hagin de dedicar més hores, el cost s'incrementarà.

Per tant, en el cas que la desviació de cost sigui desfavorable, destinarem el pressupost de contingència per cobrir-los. Si és el temps de desenvolupament el que ens fa augmentar aquest cost, es pot destinar el pressupost d'imprevistos reservats. En el cas que no siguin suficients, caldrà decidir si cal augmentar el pressupost total del projecte, o si en canvi, fer retallades és més factible.

En el cas que aquesta desviació ens sigui favorable, podríem destinar les quantitats excedents a augmentar el pressupost de costos de contingència per hipòtetics contratemps que es podrien causar més endavant.

Per tal de calcular la desviació del cost, la següent fórmula ens serà d'utilitat:

$$\text{Desviació_Cost} = (\text{cost_Estimat} - \text{cost_Real}) * \text{hores_Reals}$$

En el cas de la desviació de les tasques, s'utilitzarà la següent fórmula:

$$\text{Desviació_Tasques} = (\text{hores_Estimades} - \text{hores_Reals}) * \text{cost_Real}$$

7 Sostenibilitat

En aquest capítol es descriu l'anàlisi de sostenibilitat d'aquest projecte. Per primer lloc es farà una autoavaluació, i més endavant s'analitzen la dimensió econòmica, ambiental i social.

7.1 Autoavaluació

L'enquesta resposta m'ha fet adonar que la sostenibilitat és un àmbit que comprèn més aspectes dels que esperava. També, he vist que en general tinc menys coneixement sobre aquest tema, sobretot en l'àmbit social. Ja des de petit t'ensenyan que la sostenibilitat ambiental és un aspecte a tenir en compte, però l'enquesta remarcava que la sostenibilitat econòmica i social també són importants.

Podem dir que un dels punts forts que tinc en l'àmbit de la sostenibilitat és la distinció de les causes que perjudiquen la sostenibilitat ambiental, com poden ser les emissions de CO₂ causades per la generació d'electricitat. A més a més, considero que dono molta importància en l'àmbit de la sostenibilitat econòmica, però amb alguns problemes en fer els pressupostos.

En el cas dels punts febles, considero que moltes vegades deixo de banda l'aspecte de sostenibilitat social, és a dir, que no tinc en compte quins afectes pot comportar a la societat les accions que faig. També, en altres àmbits professionals no relacionats amb la informàtica tinc desconeixements de la sostenibilitat, sobretot en l'àmbit econòmic i social. Finalment, cal dir que els meus coneixements són limitats sobre les iniciatives per millorar els diversos àmbits de la sostenibilitat.

7.2 Dimensió econòmica

Costos

Els costos estimats que comporten l'execució d'aquest projecte han estat detallats anteriorment. Podem veure que el gruix més important correspon als sous de l'equip. Per tant, la manera de reduir els costos seria la reducció de la duració del projecte, és a dir reduint les funcionalitats del producte.

Després de dur a terme el projecte, s'han revisat els costos que han comportat. Ens centrarem sobretot en el cost personal, ja que, tal com s'ha especificat, és el responsable del 80% del pressupost. En aquest cas, les hores estimades del cap de projectes, de l'analista i del redactor, s'han ajustat molt a les estimades, mantenint el seu cost. En canvi, s'han augmentat unes 25 hores en el rol de programador, és a dir, d'uns 475€, però, entren dins dels costos de contingència. També, cal especificar que ens hem pogut estalviar els costos d'imprevistos, i una bona part dels de contingència. En total, ens hem estalviat gairebé 2.000€ evitant els imprevistos i part dels costos de contingència, arribant a un cost lleugerament inferior als 11.000€.

Vida útil

En el cas de la viabilitat del projecte, cal especificar que considero que els costos que comporten no són prou elevats per cancel·lar-lo. Els beneficis d'eficiència que

el projecte aporta als historiadors justifica els costos d'aquest.

A llarg termini, no s'espera que el projecte tingui grans costos de manteniment. En el cas que s'opti per obrir l'accés al públic mitjançant un servidor, podria fer que els costos s'elevin, i que en algun moment no sigui econòmicament factible la seva utilització pública. Tot i això, aquests costos es dispararien si es volgués mantenir el programari. En el cas que volguéssim afegir millores al producte, caldria contractar personal que el dugui a terme. Aquest cost és molt difícil de quantificar, ja que depèn molt de les millores a fer.

Riscos

En el cas d'escenaris que poden posar en perill la viabilitat del projecte, només se me n'acudeix 1. L'obertura de l'accés al públic podria posar en perill la viabilitat en el cas que fos força popular. El cost del servidor es podria disparar, però és un escenari molt poc probable.

7.3 Dimensió ambiental

Costos

Aquest projecte no requereix cap dispositiu específic, només un ordinador portàtil comprat l'any 2018, pel qual s'amortitza amb una llarga vida útil. Tal com s'ha calculat anteriorment s'estima un consum elèctric d'uns $32KWh$. Tenint en compte les emissions de 259 grams de CO₂ per cada kWh de mitjana a Espanya el 2021 [30], s'emetran aproximadament 8,3 quilograms de CO₂ per a la creació d'aquest projecte.

En el tema de l'ús del paper, es destinarà solament per a la impressió de la memòria, ja que es farà ús de recursos digitals en tots els altres casos.

Finalment, també s'ha de considerar efectes ambientals indirectes, com seria el consum elèctric de servidors de GitHub o Overleaf. Aquest ens és difícil calcular aquests consums.

Després de realitzar el projecte, s'han comparat els costos amb els estimats. L'augment del 25 hores de desenvolupament fa que el consum d'electricitat pugi de $32KWh$ a $33.5Kwh$, és a dir un augment d'uns 400g de CO₂ emès.

Principalment, el cost ambiental del projecte es basa en l'electricitat utilitzada, que està directament relacionada amb el temps de desenvolupament. Per reduir-ho, l'única opció seria reduir les hores de desenvolupament, una mesura poc realista.

Vida útil

Durant la vida útil, els costos estimats que poden comportar és el manteniment d'un servidor que executi el producte. En el cas que s'obi l'accés al públic utilitzant un servidor, faria que el consum elèctric incrementés, emetent més CO₂ a l'atmosfera. Com bé s'ha dit anteriorment, si es decidís fer millores en el producte, les hores de desenvolupament s'incrementarien, augmentant el consum elèctric. Aquests dos costos afegits durant la vida útil són difícils de quantificar, ja que també dependrà

de les emissions de la producció d'electricitat produïda, que es pot reduir mitjançant fonts renovables d'energia.

La petjada ecològica probablement no es veurà afectada per l'impacte d'aquest projecte. El consum elèctric destinat per elaborar-lo ha afectat aquesta petjada, però no de manera significativa. A més a més, no hi ha cap ús del projecte que permeti reduir aquest impacte.

Riscos

En el cas d'escenaris que poden posar en perill la viabilitat del projecte, només es pot produir combinant 2 factors. El primer seria la popularitat del projecte després d'una obertura pública. Tant el servidor com els usuaris, destinarien recursos energètics a la seva execució, que es tradueixen amb més grams de CO₂ emesos. El segon factor seria l'increment considerable d'emissions per a produir l'electricitat, un fet que no dependria d'aquest projecte.

7.4 Dimensió social

Nivell individual

A nivell individual aquest projecte m'aporta coneixements tècnics sobre gràfics 3D, i més específicament sobre els frameworks de *JavaScript ThreeJS* [6] i *OpenSeaDragon* [10]. També m'aporta altres coneixements no tècnics com la planificació i gestió de projectes, i una millora de la comunicació oral i escrita.

Aquest projecte no ha implicat que cap de les persones implicades hagin fet cap reflexió personal, professional o ètic, amb relació a aquest.

Nivell col·lectiu

A nivell col·lectiu, aquest projecte beneficia als historiadors fent que la tasca de cerca d'imatges resulti més ràpida i senzilla. Cal indicar que aquest projecte no és 100% imprescindible, però molt útil. Finalment, cal indicar que el projecte no perjudica gairebé a ningú, amb l'excepció de l'alternativa de *PhotoCloud* [3]. Així i tot, *PhotoCloud* [3] és un projecte desenvolupat per un equip de recerca que no busca cap benefici, fent que no els perjudiqui.

El producte soluciona el problema que troben els historiadors a l'hora d'organitzar i buscar les fotografies.

Riscos

En aquests moments no es donen casos que pugui perjudicar cap usuari. Així i tot, un mal ús d'aquest programari pot comportar conflictes sobre drets d'imatges, si un usuari fotografia un lloc d'interès protegit sense els permisos adients. En aquest cas, el responsable seria l'usuari que ha infringit el conflicte, no del programari.

Finalment, en cap cas aquest projecte pot deixar persones en estat dèbil.

8 Disseny i implementació de l'entorn 3D

Aquest capítol recull la implementació i el disseny de l'entorn 3D. Tal com s'ha explicat a la secció 2.2, la implementació de l'entorn 3D es durà a terme utilitzant el framework de *JavaScript ThreeJS* [6]. Aquest té implementats mètodes que facilita la visualització de models 3D al navegador.

8.1 Càrrega del model 3D

La primera tasca amb relació a l'entorn 3D és la visualització de models. L'objectiu principal de la tasca és la visualització correcta d'un model de l'Església de Sant Quirze de Pedret. Aquest model ajudarà a entendre les posicions de cada fotografia que es visualitzarà.

Abans de començar, cal especificar que tots els models de l'Església de Sant Quirze de Pedret utilitzats en aquest treball, han sigut proporcionats pel grup de recerca *VirVIG* [2]. A continuació es llisten dels models proporcionats (vegeu Taula 12).

Model	Descripció
pedret_IX	Model de Sant Quirze de Pedret del segle IX en formats <i>FBX</i> , <i>OBJ</i> i <i>MTL</i>
pedret_X	Model de Sant Quirze de Pedret del segle X en formats <i>FBX</i> , <i>OBJ</i> i <i>MTL</i>
pedret_XI	Model de Sant Quirze de Pedret del segle XI en formats <i>FBX</i> , <i>OBJ</i> i <i>MTL</i>
pedret_XII	Model de Sant Quirze de Pedret del segle XII en formats <i>FBX</i> , <i>OBJ</i> i <i>MTL</i>
pedret_XIII	Model de Sant Quirze de Pedret del segle XIII en formats <i>FBX</i> , <i>OBJ</i> i <i>MTL</i>
MNAC-AbsNord-LowPoly	Model de l'absidiola nord de Sant Quirze de Pedret en formats <i>glTF</i>
MNAC-AbsSud-LowPoly	Model de l'absidiola sud de Sant Quirze de Pedret en formats <i>glTF</i>

Taula 12: Models proporcionats per *VirVIG*
Font: Producció pròpria

Tenim diverses opcions de format per tal de carregar un model a *ThreeJS* [6]. Principalment, s'han valorat 3 de les opcions més populars. A continuació s'explicarà quines són, i es justificarà l'elecció de tria.

Format OBJ + MTL

El format *Wavefront Objec* [31] o *OBJ* és un fitxer que conté informació sobre la geometria 3D de l'objecte. Aquest és un dels formats més antics i comuns que es troben en exportar un objecte de la majoria de programari de modelatge. Aquest format només defineix l'estruatura geomètrica, fent que el model carregat no contingui cap textura. Per això, és comú trobar models *OBJ* juntament amb fitxers de *Material Template Library* [31] o *MTL*. Aquest és un format de fitxer complementari de l'*OBJ* que defineix propietats de material dels objectes *OBJ*. El fitxer *MTL* no conté les textures, només indica com s'han de carregar a parir d'una imatge.

ThreeJS dona a disposició mètodes per carregar els dos formats de fitxers, utilitzant *OBJLoader* [32] i *MTLLoader* [33]. Aquests mètodes ens permeten carregar els models de forma senzilla. El Fragment 1 conté el codi utilitzat per carregar els models.

```

1 const mtlLoader = new MTLLoader()
2 mtlLoader.load("models/pedret/pedret_XII.mtl", function(
3     materials) {
4         materials.preload();
5         var objLoader = new OBJLoader();
6         // Apliquem les textures
7         objLoader.setMaterials(materials);
8         objLoader.load("models/pedret/pedret_XII.obj", function(
9             object) {
10                 // Afegim el model llegit a l'escena
11                 scene.add( object );
12             });
13 });

```

Fragment 1: Carregant models OBJ i MTL

El resultat d'aquest model és correcte, tal com veiem a la Figura 6.



Figura 6: Model de Sant Quirze de Pedret XII en format OBJ
Font: Producció pròpia

Cal especificar que tots els models van ser carregats correctament, amb l'accepció de Sant Quirze de Pedret XIII, que no es carregava completament (vegeu Figura 7).

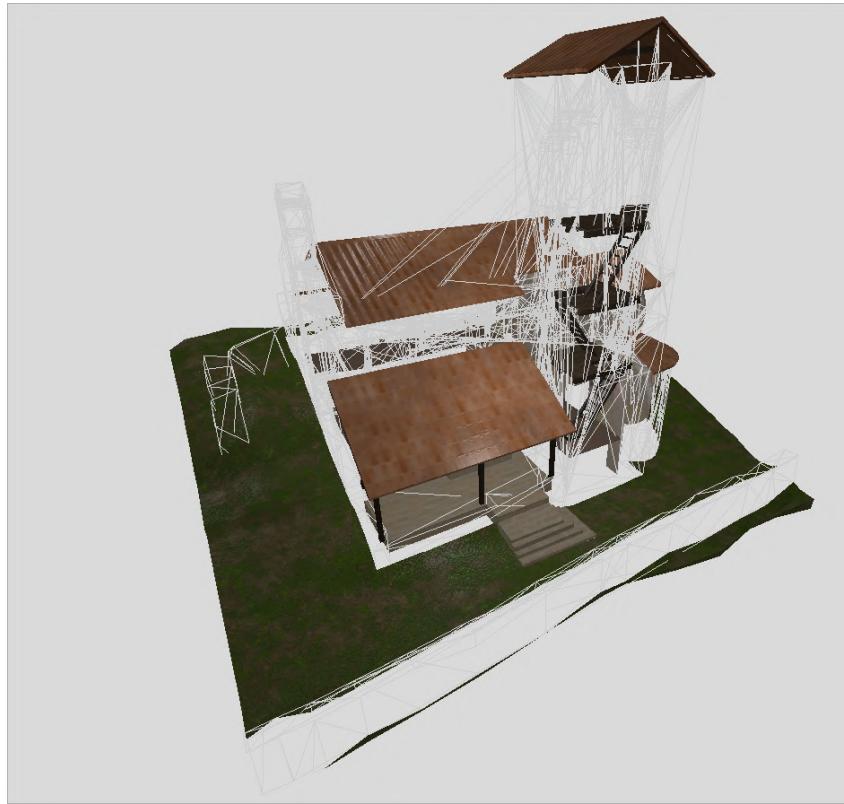


Figura 7: Model de Sant Quirze de Pedret XIII en format OBJ
Font: Producció pròpia

Després d'estar investigant, es va concloure que el problema provenia del model. En el cas que s'hagués triat aquest format per a visualitzar els models, hauríem de tornar a exportar el model.

Aquest format ha sigut un possible candidat, ja que s'ha pogut visualitzar els models correctament (amb l'accepció del segle XIII), però és un format antic. Finalment, vam descartar el format a favor d'un més modern.

Format FBX

El format *Filmbox* [34] o *FBX* és un format desenvolupat per *Kaydara* i actualment propietat d'*Autodesk* que defineix la geometria 3D de l'objecte juntament amb la descripció de les textures. De la mateixa manera que el format *MTL*, el fitxer no conté la textura, defineix com s'ha de crear a parir d'una imatge.

ThreeJS dona a disposició mètodes per carregar aquest format, utilitzant *FBXLoader* [35]. Aquests mètodes ens permeten carregar els models de forma senzilla. El codi utilitzat per carregar els models el trobareu al Fragment 2

```

1 const fbxLoader = new FBXLoader();
2 fbxLoader.load("models/pedret/pedret_XIII.fbx", (object) => {
3     // Reduim la mida
4     object.scale.set(0.01, 0.01, 0.01);
5     // Afegim el model llegit a l'escena
6     scene.add(object);
7 });

```

Fragment 2: Carregant models FBX

El resultat d'aplicar aquest fragment (vegeu Figura 8) no és l'esperat.

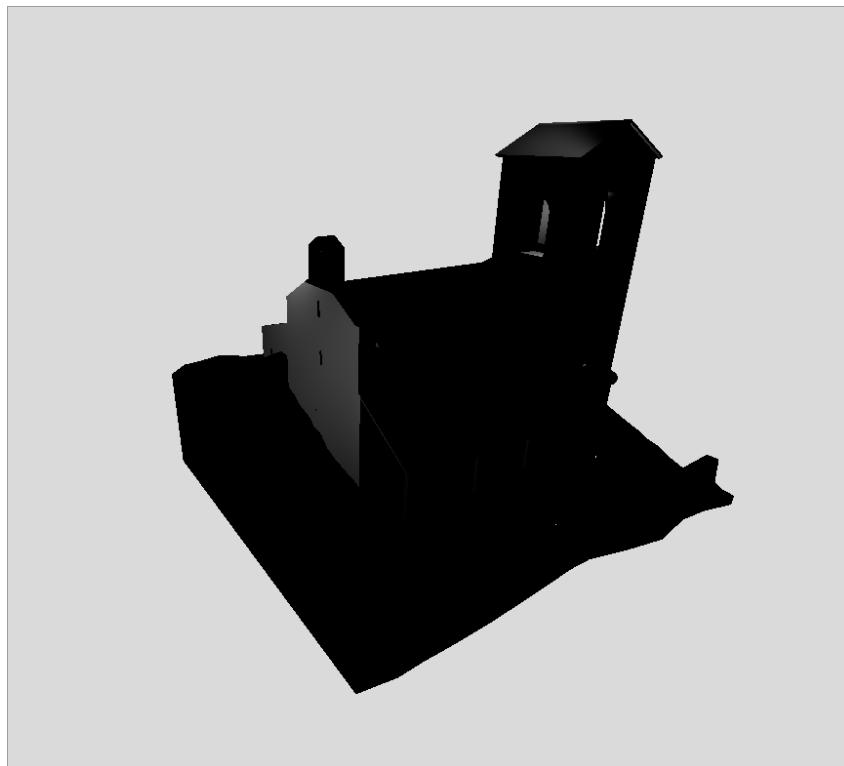


Figura 8: Model de Sant Quirze de Pedret XIII en format FBX
Font: Producció pròpria

Els models estaven exportats correctament, ja que Blender [36] els obria amb la textura. Reexportant-los, seguien carregant-se sense textura. Per no perdre tant temps, vam optar per provar el format que ThreeJS recomana [37], el *glTF*.

Format *glTF*

El format *Graphics Library Transmission Format* o *glTF* és un format de fitxer estàndard per a escenes 3D. Aquests fitxers contenen la geometria del model 3D, l'aparença, les animacions, entre altres. A diferència dels formats anteriors, aquest incorpora la textura dins del fitxer, fent que la mida d'aquest augmenti considerablement. Aquest format pot ser utilitzat utilitzant un fitxer *JSON* (extensió *gltf*) o un fitxer binari (extensió *glb*).

En aquest cas, *VirVIG* [2] no té els models exportats en aquest format. Per tal d'obtenir-los, vaig convertir-los utilitzant Blender [36], de *FBX* a *GLB*.

ThreeJS dona a disposició mètodes per carregar aquest format, utilitzant *GLTFLoader* [38]. Aquests mètodes ens permeten carregar els models de forma senzilla. El Fragment 3 conté el codi utilitzat per carregar els models.

```
1 const gltfLoader = new GLTFLoader();
2 gltfLoader.load("models/pedret/pedret_XIII.glb", (object) => {
3     // Afegim l'escena llegida
4     scene.add(object.scene);
5 });


```

Fragment 3: Carregant models *glTF*

El resultat de la càrrega de models en format *FBX* no va ser correcte. Primer, els models tenien una mida molt gran, però es va corregir fàcilment reduint la mida en el mateix codi. Però, com podeu veure a la Figura 9, les textures del model no es carregaven.



Figura 9: Model de Sant Quirze de Pedret XIII en format GLTF
Font: Producció pròpria

Finalment, es va optar per carregar els models amb aquest format, ja que els podia obrir tots correctament, inclosos el del segle XIII.

Absidioles

A part de mostrar els models sencers de l'església, es va creure convenient poder carregar models de les dues absidioles. En decidir que *glTF* seria el format a seguir en aquest treball, *VirVIG* [2] va proporcionar els models de les absidioles directament amb aquest format. En carregar les absidioles, vam localitzar 3 problemes.

Tal com podem veure a la Figura 10, el primer problema és una rotació incorrecta. Aquest problema s'ha pogut solucionar aplicant a l'objecte una rotació de $-\pi/2$ radians en l'eix X, equivalent a girar 90° en direcció a les agulles del rellotge en l'eix X.



Figura 10: Model de l'absidiola sud amb la rotació incorrecta
Font: Producció pròpia

Després d'aplicar la rotació corresponent, obtenim el següent model (vegeu Figura 11), on podem visualitzar els altres 2 problemes.



Figura 11: Model de l'absidiola sud amb la rotació correcta
Font: Producció pròpia

Un dels problemes és que podem visualitzar l'interior de l'absidiola, és a dir, que no

es mostren les parets exteriors. L'altre problema és que les textures no es generen correctament, tenen un ordre de prioritats incorrecte. Aquest problema es pot solucionar canviant aquest ordre utilitzant un programari com *Blender* [36]. No vaig solucionar aquests problemes, ja que aquest model només s'utilitza per fer proves.

8.2 Representació d'imatges

La següent tasca important és la representació de les imatges a l'entorn 3D, ubicades a la posició corresponent i amb una orientació adequada. Per fer-ho, *VirVIG* [2] ha proporcionat els fitxers que veiem a la taula 13.

Fitxer	Descripció
MNAC-AbsisSud.lst	Llistat d'imatges de l'absidiola sud
MANC-AbsSud.out	Fitxer Bundler de l'absidiola sud
MNAC-AbsisNord.lst	Llistat d'imatges de l'absidiola nord
MANC-AbsNord.out	Fitxer Bundler de l'absidiola nord
Sant Quirze de Pedret by Zones/*	Directori d'imatges separat per parts de l'església
MNAC/North/*	Directori d'imatges de l'absidiola nord
MNAC/South/*	Directori d'imatges de l'absidiola sud
Pedret-Ext/2018/*	Directori d'imatges exteriors 2018
Pedret-Ext/2019/*	Directori d'imatges exteriors 2019
Pedret-Int/2018/*	Directori d'imatges interiors 2018
Pedret-Int/2019/*	Directori d'imatges interiors 2019

Taula 13: Fitxers proporcionats per *VirVIG*

Font: Producció pròpia

A continuació es descriuen els fitxers proporcionats, començant pel Llistat d'imatges.

Llistat d'imatges

Tal com indica aquest nom, el fitxer conté el llistat d'imatges a representar. Aquest fitxer conté el *path* o ruta de cada imatge a visualitzar. Cada línia correspon a una imatge. Al Fragment 4 es mostra un retall de les primeres línies del fitxer *MNAC-AbsisSud.lst*.

- ¹ D:\Photogrammetries\AbsisSudMNAC\Imatges\ga210623_s-quinze-de-pedret_0011.jpg
- ² D:\Photogrammetries\AbsisSudMNAC\Imatges\ga210623_s-quinze-de-pedret_0012.jpg
- ³ D:\Photogrammetries\AbsisSudMNAC\Imatges\ga210623_s-quinze-de-pedret_0013.jpg

Fragment 4: Llistat d'imatges sense convertir

Aquest fitxer va ser generat automàticament utilitzant el programari *Meshlab* [4], fent que el *path* de cada imatge correspongui al *path* absolut de les imatges que tenia l'usuari. Aquest *path* no correspon amb l'específic en la màquina on es desenvolupa aquest treball. A més a més, destacar que la màquina utilitzada és Linux, fent incompatible les rutes d'arxius de Windows. A més a més, cada fitxer .lst tenia

rutes de diferents ordinadors. Per aquests motius, calia convertir el fitxer, fent que les rutes corresponguin amb les de la màquina utilitzada. Per fer-ho, vaig fer un petit script en *Python* que podeu veure al Fragment 5.

```

1 file_path = input("File path of picture list: ") # Exemple: ./MNAC-AbsisSud.lst
2 destination_path = input("File of destination: ") # Exemple: ./MNAC-AbsisSud-converted.lst
3 prefix = input("File of destination: ") # Exemple: Sant Quirze de Pedret by Zones/MNAC - South Apse/
4 f = open(file_path, "r")
5 f2 = open(destination_path, "w")
6 content = f.read()
7
8 lines = content.split('\n')
9 for line in lines:
10     # Dividim la línia amb els delimitadors '\'
11     words = line.split('\\')
12     # Escrivim la nova ruta concatenant el prefix amb el nom d'imatge
13     f2.write(prefix + words[len(words) - 1] + '\n')
```

Fragment 5: Script per convertir rutes d'imatges

Bàsicament, aquest script llegeix cada línia del document, extreu el nom de la imatge,afegeix el prefix de la ruta on tens les imatges, i l'escriu en un nou fitxer. Cal destacar, que vaig creure oportú no imprimir el *path* complet, només la ruta relativa des del directori arrel de les imatges. Per exemple, després d'executar el script sobre el retall anterior del fitxer *MNAC-AbsisSud.lst*, el resultat seria el que veiem al Fragment 6.

```

1 Sant Quirze de Pedret by Zones/MNAC - South Apse/ga210623_s-quinze-de-pedret_0011.jpg
2 Sant Quirze de Pedret by Zones/MNAC - South Apse/ga210623_s-quinze-de-pedret_0012.jpg
3 Sant Quirze de Pedret by Zones/MNAC - South Apse/ga210623_s-quinze-de-pedret_0013.jpg
```

Fragment 6: Llistat d'imatges convertides

Directori d'imatges

En iniciar el projecte, *VirVIG* [2] va proporcionar el directori d'imatges *Sant Quirze de Pedret by Zones*, que té l'estrucció que veiem a la Figura 12.

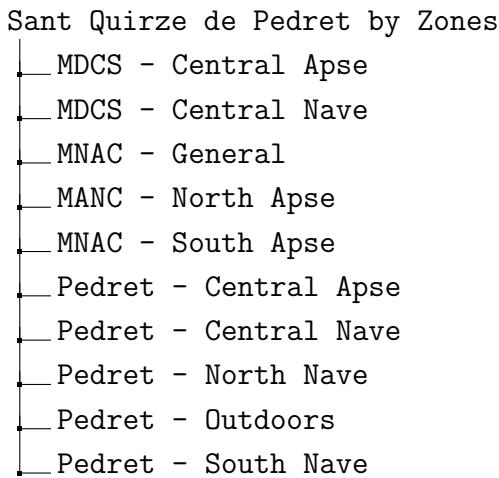


Figura 12: Estructura del directori *Sant Quirze de Pedret by Zones*

Font: Producció pròpia

Cadascun d'aquests directoris tenia la mateixa estructura. Per exemple, *MDCS - Central Apse* conté els directoris que mostra la Figura 13.

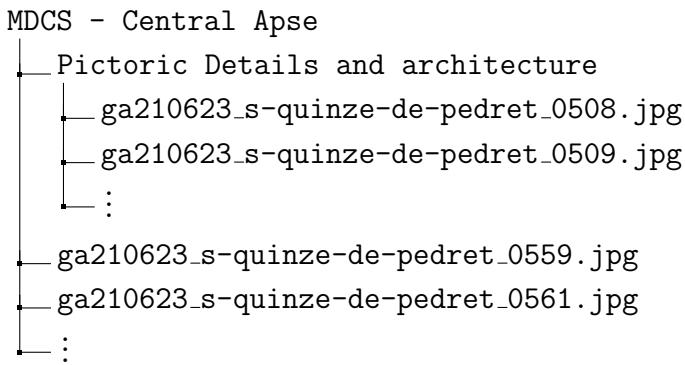


Figura 13: Estructura del directori *MDCS - Central Apse*

Font: Producció pròpia

Per simplificar l'estructura vaig decidir moure totes les fotografies dins de *Pictoric Details and architecture* al directori pare corresponent. Aquest pas simplifica la feina en l'obtenció del fitxer convertit del *Llistat d'imatges*.

Durant el desenvolupament em vaig adonar que els fitxers que contenen els llistats d'imatges apareixien noms imatges que no existien. Llavors, *VirVIG* [2] va proporcionar els altres directoris d'imatges. Simplement, vaig moure les imatges dels directoris *MNAC/North* dins de *MANC - North Apse*, *MNAC/South* dins de *MANC - South Apse*, *Pedret-Ext/2018* i *Pedret-Ext/2019* dins de *Pedret - Outdoors*. Els directoris *Pedret-Int/2018* i *Pedret-Int/2019* tenien les imatges barrejades, fent que les hagués de repartir entre els directoris corresponents de *Sant Quirze de Pedret by Zones*.

Bundler

El format de fitxers *Bundler* [39] és un sistema que representa les càmeres de les quals s'han pres les fotografies, i en reconstrueix una escena. *Bundler* pren un

conjunt d'imatges, i les seves característiques com a entrada i produeix una reconstrucció en 3D de la càmera i la geometria de l'escena com a sortida.

El fitxer té el format especificat a la Figura 14.

```

1   # Bundle file v0.3
2   <num_cameras> <num_points>    [two integers]
3   <camera1>
4   <camera2>
5   ...
6   <cameraN>
7   <point1>
8   <point2>
9   ...
10  <pointM>
11

```

Figura 14: Format del fitxer *Bundler*
Font: cs.cornell.edu

Cada entrada de <cameraI> conté informació sobre la càmera. En el nostre cas, aquest fitxer va ser generat utilitzant *Meshlab* [4] juntament amb el fitxer que llista les imatges. Això implica que el nostre fitxer *Bundler* tindrà tantes càmeres com imatges contingudes al fitxer que les llista. A més a més, les càmeres estan ordenades de manera que la primera correspongui amb la primera imatge de la llista, la segona càmera amb la segona imatge, així successivament. El format de cada càmera és el que trobem a la Figura 15.

```

1   <f> <k1> <k2>    [the focal length, followed by two radial
2   distortion coeffs]
3   <R>                  [a 3x3 matrix representing the camera
4   rotation]
5   <t>                  [a 3-vector describing the camera
6   translation]
7

```

Figura 15: Format d'una càmera del fitxer *Bundler*
Font: cs.cornell.edu

I per a cada <pointI>, el fitxer conté els paràmetres mostrats a la Figura 16.

```

1   <position>      [a 3-vector describing the 3D position of
2   the point]
3   <color>          [a 3-vector describing the RGB color of the
4   point]
5   <view list>     [a list of views the point is visible in]
6

```

Figura 16: Format d'un punt del fitxer *Bundler*
Font: cs.cornell.edu

A efectes pràctics, en el treball no s'utilitzaran els punts definits. A la Figura 17 hi ha un retall del fitxer *MANC-AbsSud.out* amb els diferents valors identificats. No apareix la part on es defineixen els punts, ja que no és necessari en aquest treball.

```

num_cameras=445      num_points=2313583
# Bundle file v0.3   f=8698.5918    k1=0    k2=0
445 2313583
camera1
8698.591776963744 0 0
-0.9966587719530035 -6.61828737764827e-002 -4.786564015886159e-002
5.290771279593129e-002 -0.9695825819011787 0.2389778039913898
-6.222592880652494e-002 0.2356468631077657 0.9698445698624265
0.1411940028941951 -1.797409698363241 0.9393406260790326
t=[0.1412, -1.7974, 0.9394]
camera2
9089.648403047435 0 0
-0.8181419604236451 -0.1877535235821872 -0.5435000892149217
-0.1068689841396278 -0.8790743610182853 0.4645506302108063
-0.564998011344393 0.4381516657309047 0.6991425927477992
-0.1597502813036902 -1.432818162971776 1.695899528931121
9082.359599706964 0 0
-0.876206956175688 -0.1701867826545129 -0.4508856051805618
-7.247357900067383e-002 -0.8784040481781987 0.472391689692977
camera3

```

Figura 17: Valors del fitxer *MANC-AbsSud.out*
Font: Producció pròpria

8.2.1 Càrrega d'imatges

Per poder representar totes i cadascuna de les imatges especificades, es llegeix el fitxer .lst que llista totes les imatges a visualitzar. Això es pot aconseguir fàcilment utilitzant el mètode *FileLoader* [40] que ens proporciona *ThreeJS*. Aquest ens retorna una cadena de caràcters amb el contingut d'un document. Després, podem fraccionar aquesta cadena usant el delimitador de salt de línia '\n' per separar els *paths* de les imatges.

Per representar una imatge a l'entorn 3D, ens ajudarem d'una primitiva que ens proporciona el framework. La geometria *PlaneGeometry* [41], tal com ens indica el seu nom, correspon a un pla.

Per tal d'obtenir una textura que aplicarem al pla, utilitzarem el mètode *TextureLoader* [42]. Aquest, en passar la ruta d'una imatge, ens retornarà la textura en qüestió. Si en aquests moments assignem aquesta textura com a material del pla, no s'aplicarà correctament, la textura quedarà deformada. Això és degut al fet que la primitiva del pla és un quadrat 1x1, i les imatges són rectangulars. Per solucionar-ho, podem definir la mida del pla a partir de les dimensions de la imatge que ens retorna la textura. En aquest cas, imatges amb una qualitat major, en tenir més píxels, apareixerien amb una mida major que la resta. Per tal que les imatges tinguin la mateixa mida independent de la seva qualitat, calculem la ràtio d'alçada respecte amplada. Amb aquesta ràtio podem assignar la mida de la imatge de manera que totes les imatges verticals tinguin una alçada d'una unitat, i totes les imatges horitzontals tinguin aquesta unitat a l'amplada. A continuació trobem un Fragment de codi (Fragment 7) on ens mostra com es crea el pla i s'aplica la textura en aquest.

```

1 const image_loader = new THREE.TextureLoader();
2 const texture = image_loader.load(image_path, function () {
3     texture.colorSpace = THREE.SRGBColorSpace;
4     // Calculem la ratio alcada respecte amplada
5     const heightToWidthRelation =
6         texture.image.height / texture.image.width;
7     let width = 1;
8     let height = 1;

```

```

9         // Escalem la imatge correctament depenent si es
10        vertical o horitzontal
11        if (heightToWidthRelation < 1)
12            height = heightToWidthRelation;
13        else width = 1 / heightToWidthRelation;
14        // Creem la geometria amb les mides corresponents
15        const geometry = new THREE.PlaneGeometry(width, height)
16        ;
17        // Creem el material
18        const material = new THREE.MeshBasicMaterial({
19            map: image_texture,
20        });
21        // Creem l'objecte
22        const object = new THREE.Mesh(geometry, material);
23        // Afegim l'objecte a l'escena
24        scene.add(object);
25    );

```

Fragment 7: Representació d'una imatge en ThreeJS

En el Fragment 7, en el cas que *image_path* equivalgui a *Sant Quirze de Pedret by Zones/MNAC - South Apse/ga210623_s-quinze-de-pedret_0011.jpg*, obtindrem el resultat mostrat a la Figura 18.

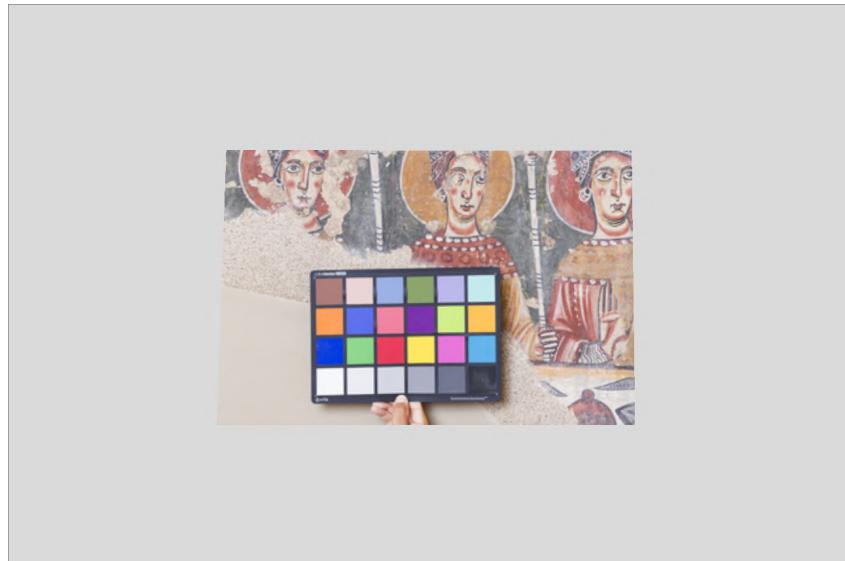


Figura 18: Representació d'una imatge a l'entorn 3D
Font: Producció pròpia

Per mostrar la resta d'imatges, només ens cal iterar el Fragment 7 amb cadascuna de les imatges del llistat.

Ara bé, si féssim aquest procés, tindríem un problema. Aquestes imatges tenen una resolució molt alta, moltes són de 6240x4160, gairebé 8K. Amb el mètode especificat, amb la majoria d'ordinadors, ens quedaríem sense memòria RAM, i el navegador no seria capaç de suportar-ho, i es bloquejaria. En el cas que la màquina tingüés prou memòria, el més probable és que el visor aniria tan lent que no seria usable.

Vam considerar que a l'entorn 3D, com que l'objectiu no era visualitzar els detalls de les imatges (això se n'encarrega el visor 2D), seria reduir la mida de les imatges.

Per fer-ho, ens vam ajudar del programari *ImageMagick* [43]. Vaig crear un script que reduíss les imatges fins al 10% de la mida original. Un valor que feia visualitzar les imatges amb prou qualitat, i que *ThreeJS* era capaç de suportar en la majoria de màquines. El petit codi utilitzat és el que trobem al Fragment 8.

```

1 #!/bin/bash
2 for f in `find . -name "*.jpg"`
3 do
4     convert $f -resize 10% $f
5 done
6
7 for f in `find . -name "*.JPG"`
8 do
9     convert $f -resize 10% $f
10 done

```

Fragment 8: Reducció de mida d’imatges

8.2.2 Posicionament d’imatges

Una vegada hem aconseguit mostrar les imatges, cal col·locar-les a la posició adequada. Per a obtenir les posicions corresponents a cada imatge, ens ajudarem del fitxer *Bundler*. Tal com hem descrit anteriorment, per a cada imatge o càmera obtenim un vector de translació. Utilitzar aquest vector com a posició no seria correcte. Per a obtenir la posició correcta, cal fer una multiplicació amb la matriu de rotació, tal com descriu la fórmula de la Figura 19.

```

1     3D position of a camera is
2         -R' * t
3

```

Figura 19: Posició 3D d’una càmera

Font: cs.cornell.edu

Tal com s’ha especificat anteriorment, cada imatge li correspon a un vector translació t i una matriu de rotació R . Aquests valors s’han llegit del fitxer *Bundler* d’una manera similar al llistat d’imatges. El llistat i el fitxer *Bundler*, en estar ordenats de la mateixa manera, ens ha facilitat molt l’assignació de cada vector i matriu amb la seva imatge corresponent.

Però, aplicant la fórmula de la Figura 19, podem veure que les posicions obtingudes no són les correctes (vegeu Figura 20).



Figura 20: Imatges després d'aplicar la fórmula de la posició
Font: Producció pròpia

Podem observar que les imatges estan girades 180° respecte a l'eix X, que és equivalent a establir la posició negant la component Y i Z. Després d'aplicar aquesta rotació, observem que les imatges estan ben posicionades (vegeu Figura 21).

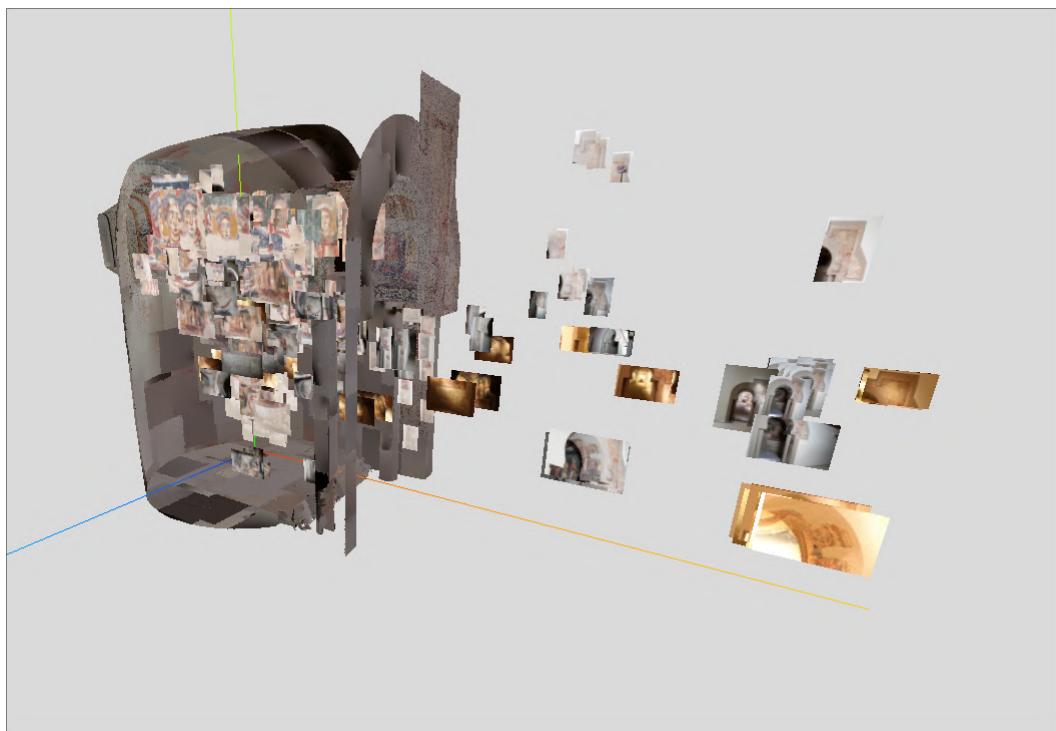


Figura 21: Imatges posicionades correctament
Font: Producció pròpia

Cal mencionar, que per fer algunes de les operacions matemàtiques del projecte, s'ha utilitzat la llibreria de JavaScript *MathJS* [44]. A continuació podeu consultar el Fragment 9, que s'encarrega de posicionar una imatge.

```

1 // pos = -R' * t
2 const position = math.multiply(math.unaryMinus(math.transpose(R
    )), t);
3 // object correspon a la imatge afegida a l'escena
4 object.position.set(pos.get([0]), -pos.get([1]), -pos.get([2]))
    ;

```

Fragment 9: Posicionament d'una imatge en ThreeJS

8.2.3 Rotacions d'imatges

Una vegada hem posicionat correctament les imatges, cal orientar-les correctament. Per fer-ho, ens ajudarem d'una altra fórmula que ens dona el manual de *Bundler* [39], sobre com calcular el view vector o vector direcció (vegeu Figura 22).

```

1 the camera viewing direction is:
2     R' * [0 0 -1]' (i.e., the third row of -R or third
column of -R')
3

```

Figura 22: Direcció d'una càmera
Font: cs.cornell.edu

Després d'obtenir la direcció de la càmera, hauríem de calcular la rotació en cada eix, però *ThreeJS* ens facilita el càlcul amb un mètode anomenat *lookAt* [45], que aplica la rotació d'un objecte a partir d'un vector direcció. De la mateixa manera que en el posicionament d'imatges, cal aplicar una rotació de 180° respecte a l'eix X. Aquesta, l'aplicarem negant les components Y i Z del vector direcció.

El resultat següent no és el correcte, tal com podem veure a la Figura 23.

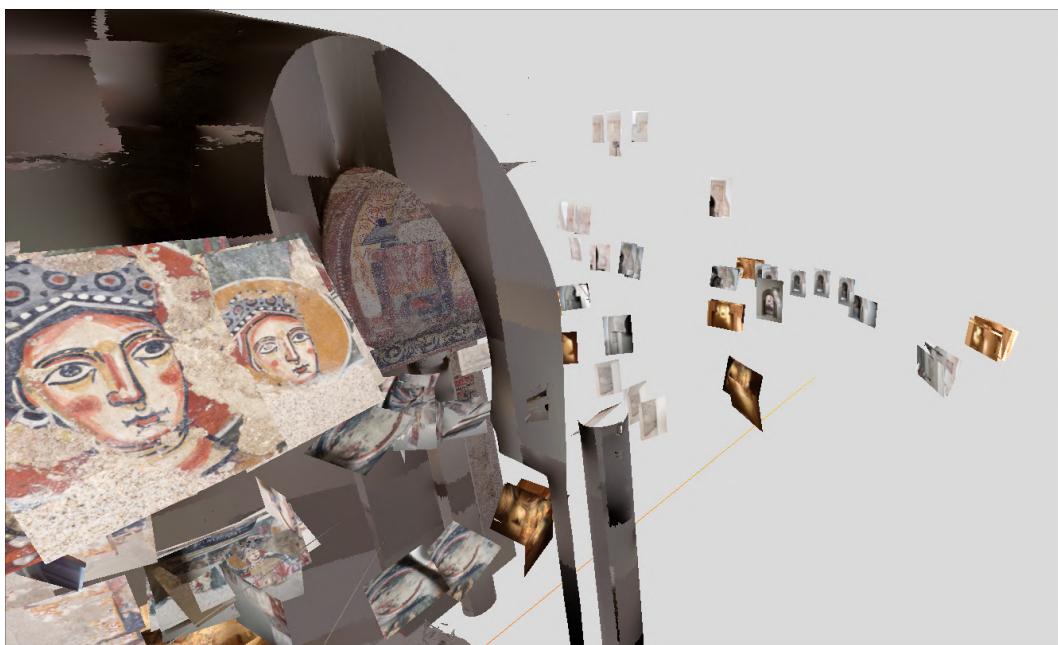


Figura 23: Imatges després d'aplicar la rotació
Font: Producció pròpia

Podem observar que la rotació en l'eix Y està desfasat 180°, és a dir que el pla pinta la cara que no és correcta. Per exemple, les imatges que es troben fora de

l'absidiola, haurien d'enfocar l'absidiola, cosa que no ho fan. Per solucionar-ho, en comptes de sumar 180° en l'eix Y a la rotació obtinguda per *lookAt* [45], he aplicat la rotació directament a la primitiva del pla. Així, una vegada es crea l'objecte del pla, ja vindrà inicialitzat amb la rotació correcta. A la Figura 24 veiem el resultat d'aplicar la rotació de 180° a l'eix Y de la primitiva del pla.

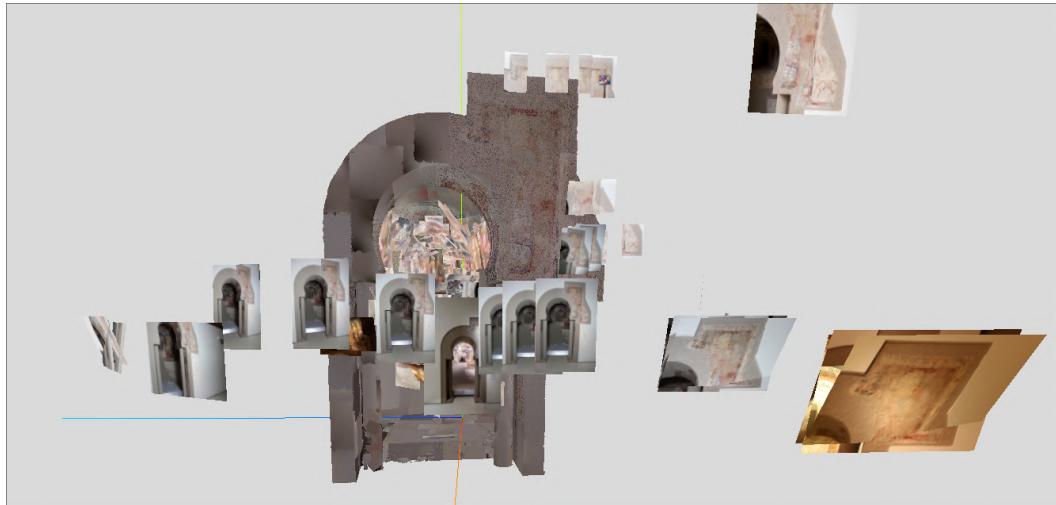


Figura 24: Imatges amb la rotació correcta
Font: Producció pròpia

Podem veure que ara, les imatges de l'entrada de l'absidiola ja apunten a l'absidiola. A continuació podeu veure el Fragment 10, que s'encarrega de calcular i aplicar la rotació correcta en una imatge.

```

1 // Abans de crear l'objecte, cal aplicar la rotacio a la
   primitiva del pla
2 geometry.rotateY(Math.PI)
3 const object = new THREE.Mesh(geometry, material);
4
5 // dir = R' * [0,0,1]
6 const dir = math.multiply(math.transpose(R), math.transpose(
    math.matrix([0, 0, -1])));
7 // object correspon a la imatge afegida a l'escena
8 object.position.lookAt(dir.get([0]), -dir.get([1]), -dir.get
    ([2]));

```

Fragment 10: Càlcul i aplicació de la rotació d'una imatge en ThreeJS

8.2.4 Distribucions d'imatges en un model complet

Per al posicionament i rotacions d'imatges s'ha utilitzat el model de l'absidiola, ja que va ser aquest el que va ser utilitzat per generar el fitxer *Bundler*. Els altres models tenen una escala, rotació i desplaçament diferent. L'objectiu és posicionar les imatges en un model complet de *Sant Quirze de Pedret*. Per exemple, si volem visualitzar el model *pedret XII*, el resultat és el que trobem a la Figura 25.



Figura 25: Model pedret XII amb imatges
Font: Producció pròpia

Per solucionar-ho, haurem d'obtenir una matriu de rotació, i aplicar-li al model. Utilitzarem el programa *MeshLab* [4], que ens permet trobar una matriu de rotació entre dos models.

Primer de tot, obrirem els dos models en qüestió al *MeshLab* [4], i seleccionarem l'opció *Align*. Allà, s'obriran els models un al costat de l'altre, i haurem de seleccionar almenys 4 punts en el primer model, i seleccionar els seus corresponents al segon model, seguint el mateix ordre. És a dir, el primer punt del model 1 ha de correspondre amb el primer del model 2, el segon amb el segon, etc. A la Figura 26 es mostra els 8 punts seleccionats en els dos models.

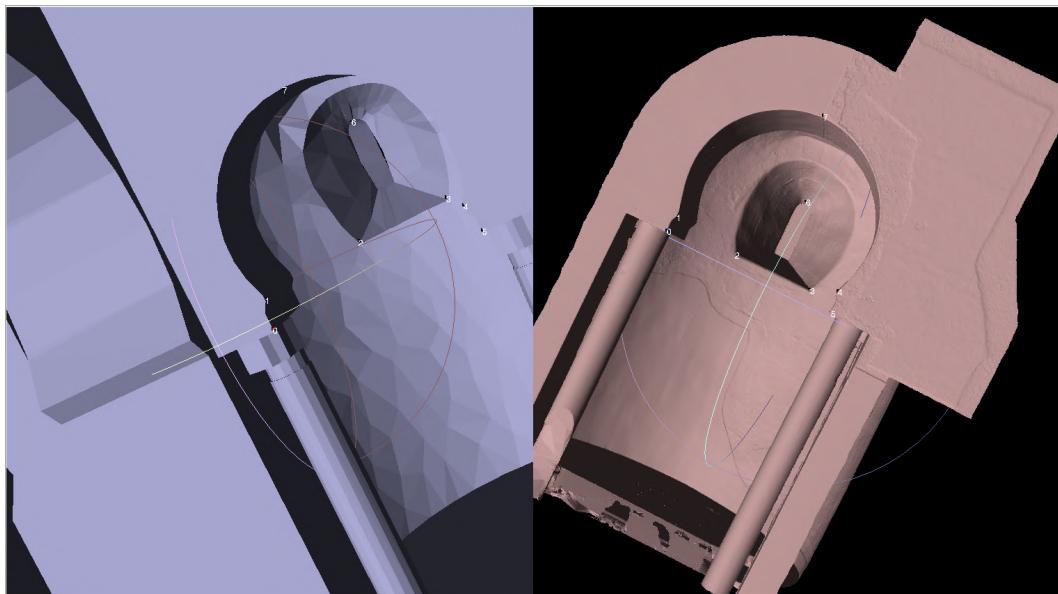


Figura 26: Selecció dels 8 punts a cada model, pedret XII a l'esquerra, absidiola sud a la dreta
Font: Producció pròpia

Finalment, premem el botó *Process*, i desem l'arxiu. L'arxiu desat conté la matriu de rotació de cadascun dels models. Nosaltres agafarem la del model *pedret XII*. El

fitxer obtingut és el que trobem al Fragment 11.

```
1 2
2 models/pedret/pedret_XII.glb
3 #
4 -0.996301 0.069092 -0.051103 6.341516
5 -0.052068 -0.012240 0.998569 13.402682
6 0.068368 0.997535 0.015792 1.105989
7 0.000000 0.000000 0.000000 1.000000
8 models/pedret10/MNAC-AbsSud-LowPoly.glb
9 #
10 0.997940 0.014458 0.062500 -0.078124
11 -0.013743 0.999835 -0.011851 0.035270
12 -0.062662 0.010968 0.997975 0.012606
13 0.000000 0.000000 0.000000 1.000000
14 0
```

Fragment 11: Matrius de rotacions obtingudes amb el Meshlab

Una vegada hem obtingut la matriu de rotació per a sincronitzar l'escala, posició, i rotació, ens caldrà aplicar-li al model. Per aplicar-la, podríem seguir els passos de posicionament i rotació d'imatges, però aquesta vegada utilitzarem mètodes ja implementats per *ThreeJS* anomenats *setFromMatrixPosition* [46], que ens retorna la posició d'una matriu de rotació, *setFromMatrixScale* [47], que ens retorna l'escala, i *setFromRotationMatrix* [48], que ens retorna la rotació. Com que el model de l'absidiola estava girat 90°, també li aplicarem aquesta rotació. Com que el model estarà desplaçat, i la rotació la voldrem aplicar des de l'origen de coordenades, l'inclourem dins d'un objecte que estigui centrat abans d'aplicar-li la rotació. El Fragment 12 és el responsable de carregar un model amb l'aplicació de la matriu de rotació.

```
1 gltfLoader.load("models/pedret/pedret_XII_text4K.gltf", (object) => {
2   // Matriu de rotació obtinguda
3   const matrix = new THREE.Matrix4().set(
4     -0.996301, 0.069092, -0.051103, 6.341516,
5     -0.052068, -0.012240, 0.998569, 13.402682,
6     0.068368, 0.997535, 0.015792, 1.105989,
7     0.0, 0.0, 0.0, 1.0);
8   // Obtenim i apliquem la posició, rotació i escalat
9   const pos = new THREE.Vector3().setFromMatrixPosition(
10    matrix);
11   const scale = new THREE.Vector3().setFromMatrixScale(matrix);
12   const rotation = new THREE.Quaternion().
13     setFromRotationMatrix(matrix);
14   object.scene.position.copy(pos);
15   object.scene.scale.copy(scale);
16   object.scene.quaternion.copy(rotation);
17   // Afegim el model a un objecte centrat, i el girem 90
18   // graus
19   const wrapper = new THREE.Object3D();
20   wrapper.name = "model";
21   wrapper.add(object.scene);
```

```

19     wrapper.rotateX(-Math.PI / 2);
20     scene.add(wrapper);
21 });

```

Fragment 12: Càlcul i aplicació de la rotació d'una imatge en ThreeJS

El resultat obtingut és el que trobem a la Figura 27.



Figura 27: Model després d'aplicar-li la matriu de rotació obtinguda
Font: Producció pròpia

8.3 Interacció amb imatges

Després de representar les imatges correctament a l'escena 3D, s'ha afegit una sèrie d'interaccions possibles. Aquestes són la modificació de la seva mida, la separació respecte a la posició de la càmera, i la selecció d'imatges. Algunes d'aquestes interaccions requereixen una interfície gràfica. Per fer-ho, s'ha optat per utilitzar la biblioteca de *JavaScript lil-gui* [49], que permet crear-les de forma senzilla. A la Figura 28 es pot observar la part de la interfície gràfica que s'encarrega de les interaccions.

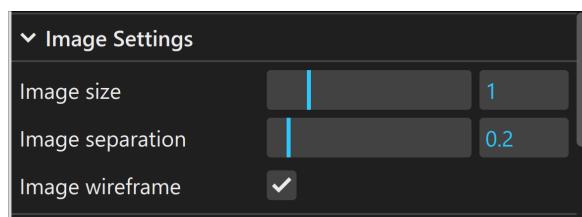


Figura 28: Interfície gràfica d'interacció
Font: Producció pròpia

A continuació es descriuen cadascuna d'aquestes interaccions.

8.3.1 Modificació de la mida d'una imatge

Tal com es pot observar a la Figura 28, per modificar la mida de les imatges mostres a l'entorn, s'ha afegit un *slider* que permet establir la mida desitjada. Cada vegada que el valor del *slider* canvia, es crida una funció amb el valor com a paràmetre per modificar la mida de les imatges.

La lògica per a modificar la mida de les imatges és molt senzilla, només cal modificar els components X i Y de la mida de la imatge. En primer lloc, es restableix la mida original de la imatge, i després s'aplica la nova mida de la imatge. A continuació podem veure el Fragment 13, que conté el codi simplificat que s'encarrega d'establir aquesta mida.

```
1 // Variable que conte la mida actual
2 var currentSize;
3 function setSize(size) {
4     // Restablim a la mida original
5     object.scale.set(1/currentSize, 1/currentSize, 1);
6     // Establim la nova mida
7     object.scale.set(size, size, 1);
8     currentSize = size;
9 }
```

Fragment 13: Establir mida d'una imatge

Així i tot, en uns inicis no va ser tan senzill. En aquells moments aplicava la rotació de la imatge a la primitiva, no en l'àmbit d'objecte. Això feia que l'escalat no quedés uniforme, ja que la imatge no coincidia amb el pla XY de l'objecte. Per aconseguir l'escalat correcte aplicava la matriu de rotació inversa, escalava segons l'eix X i Y, i tornava a aplicar la rotació. Una vegada vaig trobar que aplicar la rotació en l'àmbit d'objecte facilitaria les coses, vaig poder simplificar l'escalat, juntament amb la selecció de separació respecte a la imatge.

8.3.2 Modificació de la separació respecte a una imatge

La separació de la imatge respecte a la posició de la càmera és una distància que es troba entre aquestes dues. La posició de la càmera la definim com a la posició del dispositiu en el moment de prendre la imatge. Amb un *slider* a la interfície gràfica podem modificar aquesta separació.

Per a calcular la nova posició de la imatge segons la separació establerta utilitzarem el vector direcció de la imatge. El càlcul d'aquest vector equival a la transposada de la matriu rotació amb el vector [0,0,-1]', tal com s'ha explicat al capítol 8.2.3. Aquest vector l'escalarem segons el valor de la separació, i finalment el sumarem amb el vector posició d'imatge (vegeu Figura 29).

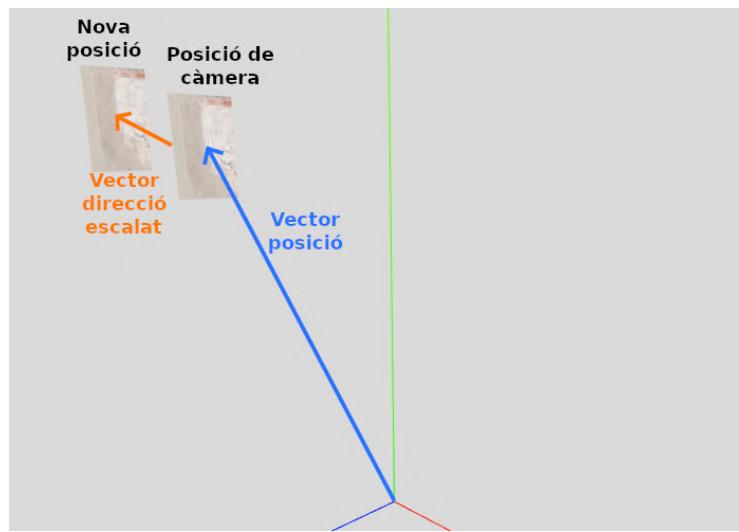


Figura 29: Esquema vectors direcció i posició
Font: Producció pròpria

El Fragment 14 s'encarrega de calcular el nou posicionament d'una imatge.

```

1 // Variable que conte la mida actual
2 var currentOffset;
3 // Matriu rotacio de la imatge
4 var R;
5 // Posicio de camera de la imatge
6 var pos;
7 function setOffset(offset) {
8     // Calcul del vector direccio, dir = R' * [0,0,1]
9     let dir = math.multiply(math.transpose(R),
10         math.transpose(math.matrix([0, 0, -1])));
11     // Escalat vector direccio
12     let view_vector = math.multiply(math.number(offset), dir);
13     // Suma vector posicio amb vector direccio
14     let new_pos = math.add(math.matrix(pos), view_vector);
15     // Establim nova posicio
16     object.position.set(new_pos.get([0]), -new_pos.get([1]),
17         -new_pos.get([2]));
18     currentOffset = offset;
19 }
```

Fragment 14: Càlcul posició d'imatges segons la separació

Cal recordar que en utilitzar la matriu de rotació R obtinguda del fitxer *Bundler*, hem utilitzat la posició de càmera calculada en comptes de la posició de l'objecte. Altrament, la posició final no seria correcta. Finalment, la posició obtinguda s'ha de girar 180° respecte a l'eix X.

Finalment, cal destacar que aquest mètode no és el definitiu, ja que més endavant es van trobar dificultats en implementar la representació en filferro (vegeu capítol 8.3.4).

8.3.3 Hover i selecció d'imatges

La següent interacció afegida és el *hover* i la selecció d'imatges. Aquestes dues van molt relacionades, ja que comparteixen part del mecanisme que permet identificar

la imatge que se selecciona o la qual es fa el *hover*. Les dues funcionalitats utilitzen el *ray casting* per a identificar les imatges.

Raycasting

El *ray casting* [50] és una tècnica utilitzada en la il·luminació d'escenes per ordinador. Els rajos de llum són traçats des del punt focal d'una càmera a través de cada píxel del sensor de la càmera per determinar què és visible al llarg del raig a l'escena 3D. Podem modificar aquest l'ús d'aquest raig de manera que ens indiqui quin objecte de l'escena estem seleccionant.

En el nostre cas, llançarem un raig des de la càmera fins a la punta del ratolí, i obtindrem el primer objecte que incideixi amb el raig. Per fer-ho, ens ajudarem amb *Raycaster* [51], una classe que ens proporciona *ThreeJS* amb la implementació del *ray casting* feta.

Una vegada ja integrat el *ray casting*, vaig implementar les funcionalitats de *howering* i selecció, i em vaig adonar que el raig traspassava la interfície gràfica, i a vegades seleccionava imatges en interaccionar amb botons i sliders de la interfície. Després de fer una sèrie de proves, vaig veure que el *tagName* [52] de l'aplicació de *ThreeJS* i del panell de la interfície gràfica eren diferents. Puc obtenir l'element que el ratolí es troba a sobre, i comprovar si el *tagName* de l'element equivalia a *CANVAS* (*ThreeJS*) o *DIV*, *BUTTON*, *CHECKBOX*, *SLIDER*, etc. (interfície gràfica). En el cas que el *tagName* no equivalguia a *CANVAS*, puc desactivar el *ray cast*.

A continuació podeu observar el Fragment 15, que mostra l'obtenció d'un objecte mitjançant *ray casting*.

```
1 event.preventDefault();
2 // Prevenir interaccions amb imatges darrere la interficie
   grafica
3 if (event.target.tagName != "CANVAS") return;
4 mouse.x = (event.clientX / window.innerWidth) * 2 - 1;
5 mouse.y = -(event.clientY / window.innerHeight) * 2 + 1;
6 raycaster.setFromCamera(mouse, camera);
7 var intersects = raycaster.intersectObject(scene, true);
8 if (intersects.length == 0) return;
9 // Obtenim primer objecte del raycast
10 var object = intersects[0];
11
12 // Logica de les accions amb l'objecte, veure seccions
   posteriors
13
14 render();
```

Fragment 15: Integració del ray casting

Hover

El *hover* o *mouseover* [53] el definim com a una acció activada quan el ratolí passa per damunt d'un objecte determinat. En aquest treball, ressaltarem una imatge quan el ratolí es mogui per damunt d'aquesta. La funcionalitat és purament visual, i ens ajuda a distingir quina és la imatge que el ratolí està seleccionant.

Una vegada tenim el *ray cast* implementat, afegir la funcionalitat de *hovering* és molt senzilla. Només cal que cada vegada que el ratolí es mogui, llancem un raig i comprovem si l'objecte incidit és una imatge o no. En el cas que en el *frame* anterior cap imatge estigui en mode *hover* l'objecte imatge, aquesta passarà a *hover*. Altrament, si l'objecte en mode *hover* en el *frame* anterior és diferent de l'objecte incidit, es desactivarà el *hover* de l'objecte anterior, i s'habilitarà en la nova imatge. Finalment, si el raig no ha incidit a cap objecte o a un objecte que no és imatge, es desactivarà el *hover* en el cas que hi hagi una imatge que el tingui habilitat. A continuació veiem el Fragment 16, que correspon al *hovering* de les imatges.

```

1 // Cridem la funció cada vegada que es mogui el ratoli
2 window.addEventListener("pointermove", onHover);
3
4 // Ressaltem la imatge amb un nou color
5 function hoverIn(object) {
6     object.material.color.setHex(HOVER_COLOR);
7     hover = object;
8 }
9
10 // Restablism el color de la imatge
11 function hoverOut() {
12     hover.material.color.setHex(NEUTRAL_COLOR);
13     hover = undefined;
14 }
15
16 function onHover() {
17     // Definició del raycast vist a la secció anterior
18
19     var intersects = raycaster.intersectObject(scene, true);
20     if (intersects.length > 0) {
21         var object = intersects[0];
22         // Si l'objecte es una imatge
23         if (object != null && object.name.startsWith("Sant
Quirze de Pedret by Zones")) {
24             // Nou hover
25             if (hover === undefined) hoverIn(object);
26             // Substituir hover
27             else if (hover.name !== object.name) {
28                 hoverOut();
29                 hoverIn(object);
30             }
31         }
32         // Hovering d'un objecte no-imatge, restablism l'actual
33         else if (hover !== undefined) hoverOut();
34     }
35     // No hi ha hovering de cap objecte, restablism l'actual
36     else if (hover === undefined) hoverOut();
37 }
```

Fragment 16: Implementació del hovering

El resultat d'aplicar el Fragment 16, és el que veiem a la Figura 30.

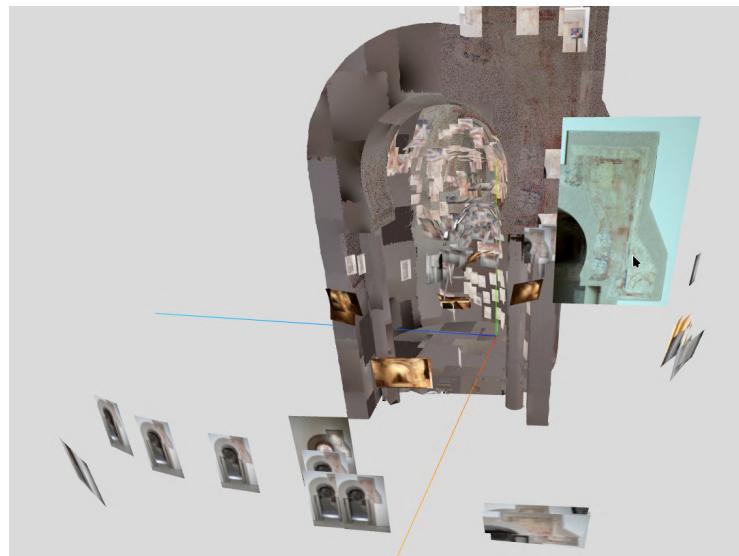


Figura 30: Imatge ressaltada en fer *hover*
Font: Producció pròpia

Selecció d'imatges

Entenem com a selecció d'imatges el fet d'escol·lir una o diverses imatges per tal de visualitzar-les en un visor 2D. Més endavant s'explicarà el procés d'obrir aquestes imatges amb el visor fet amb *OpenSeaDragon*, però en aquest capítol ens centrarem en la fase de selecció.

De la mateixa manera que el *hover*, la selecció d'imatges també utilitza el *ray cast*. De fet, la implementació de la selecció comparteix moltes similituds amb el *hover*. En aquest cas, el raig es llançarà cada vegada que el ratolí detecti un clic. En el cas que s'hagi premut el botó esquerre del ratolí, la imatge s'obrirà automàticament al visor 2D. Altrament, la imatge quedarà ressaltada amb un color magenta, indicant que queda seleccionada per tal de poder obrir un grup d'imatges amb el visor 2D en un futur. A més a més, en el cas que seleccionis una imatge amb el botó dret per segona vegada, la imatge quedarà desseleccionada, i es restablirà el seu color. El Fragment 17 s'encarrega d'implementar aquesta funcionalitat.

```

1 // Cridem la funció quan es deixi anar qualsevol boto i el
   ratoli estigui quiet
2 window.addEventListener("mouseup", function () {
3     if (dragging === false) onClick();
4 });
5
6 function onClick() {
7     // Definició del raycast vist a la secció anterior
8
9 var intersects = raycaster.intersectObject(scene, true);
10 if (intersects.length == 0) return;
11 var object = intersects[0];
12 if (object == null || !object.name.startsWith("Sant Quirze
de Pedret by Zones")) return;
13 // Obrir imatge si s'ha pres el boto esquerra
14 if (event.button == 0) openImage(object);
15 // Desselecció del imatge
16 else if (imagesSelected.has(object)) {

```

```

17         imagesSelected.delete(object);
18         object.material.color.setHex(HOVER_COLOR);
19     }
20     // Seleccionem imatge
21     else {
22         imagesSelected.add(object);
23         object.material.color.setHex(SELECTION_COLOR);
24     }
25 }
```

Fragment 17: Implementació de la selecció d'imatges

El resultat d'aplicar el Fragment 17, és el que es mostra a la Figura 31.



Figura 31: Grup d'imatges seleccionades

Font: Producció pròpia

Finalment, s'ha de resoldre un conflicte que es produeix quan es fa *hover* d'una imatge seleccionada. En el meu cas, he decidit que les imatges no es marquin com a *hover* quan la imatge ja està seleccionada. Simplement, he afegit una condició que no es marqui ni es desmarqui del *hover* quan aquesta està seleccionada. Les funcions *hoverIn* i *hoverOut* queden implementades al Fragment 18.

```

1 function hoverIn(object) {
2     if (!imagesSelected.has(object)) object.material.color =
3         setHex(HOVER_COLOR);
4     hover = object;
5 }
6 function hoverOut() {
7     if (!imagesSelected.has(hover)) hover.material.color.setHex
8         (NEUTRAL_COLOR);
9     hover = undefined;
9 }
```

Fragment 18: Implementació del hovering d'imatges

8.3.4 Mostrar filferros

Per tal de visualitzar la separació d'imatges, s'ha optat per dibuixar una sèrie de línies que ajuden a veure la posició de la càmera per a cada imatge. Aquestes

línies o filferros es poden habilitar i deshabilitar mitjançant un *checkbox* present a la interfície gràfica, tal com podeu veure a la *Figura 28*.

Per a dibuixar aquestes línies es va seguir una guia sobre les *línies a ThreeJS* [54]. Aquesta, utilitza *BufferGeometry* [55] per a la primitiva, i *LineBasicMaterial* [56] per al material. El mètode per dibuixar-les, tracta d'especificar els dos vèrtexs del segment al *BufferGeometry*, i *ThreeJS* s'encarregarà de dibuixar una línia entre aquests dos.

Per dibuixar la representació en filferros, necessitem la posició de la càmera de la imatge, i les 4 cantonades o vèrtex de la imatge. La posició de la càmera correspon a la llegida anteriorment. En canvi, els 4 vèrtexs s'han d'obtenir a partir de la geometria de l'objecte. En el cas d'objectes a partir d'un *PlaneGeometry* [41], els 4 vèrtexs corresponen als 4 primers vèrtexs especificats al *BufferGeometry* [55] de l'objecte. Aquests vèrtexs estan en coordenades d'objecte, i per tal de passar-les a coordenades de món apliquem una matriu de transformació que ens proporciona l'objecte. A més a més de dibuixar les línies de la posició de la càmera a cada-cun dels vèrtexs, també es dibuixa el contorn de la imatge. Això ens és útil per representar les imatges quan es visualitzen d'esquena. A continuació es mostra el Fragment de codi simplificat per tal de dibuixar la representació en filferro (vegeu Fragment 19).

```

1 // Obtenim BufferGeometry dels vertexs de la imatge
2 const vertexPositions = object.getAttribute("position");
3 var wireFrameObject = new THREE.Object3D();
4 const material = new THREE.LineBasicMaterial({color: 0x0000ff})
5 ;
6 // Lílies camera-vertexs
7 for (let k = 0; k < 4; k++) {
8     // Obtenim el vertex del BufferGeometry
9     let vertex = new THREE.Vector3().fromBufferAttribute(
10        vertexPositions, k);
11     // Conversio a coordenades de mon
12     vertex.applyMatrix4( object.matrixWorld );
13     // Dibuixem la línia vertex-camera
14     const points = [vertice, pos];
15     const geometry = new THREE.BufferGeometry().setFromPoints(
16        points);
17     const line = new THREE.Line(geometry, material);
18     wireFrameObject.add(line);
19 }
20 //Contorn de la imatge
21 for (let k = 0; k < 4; k++) {
22     let vertex1 = new THREE.Vector3().fromBufferAttribute(
23        vertexPositions, k);
24     // Conversio a coordenades de mon
25     vertex1.applyMatrix4( object.matrixWorld );
26     // Els vertexs del BufferGeometry segueixen l'ordre 0,1,3,2
27     let a = 1;
28     if (k == 1) a = 3;
29     else if (k == 2) a = 0;
30     else if (k == 3) a = 2;
31     const vertex = new THREE.Vector3().fromBufferAttribute(
32        vertexPositions, a);
33 }
```

```

28     // Conversio a coordenades de món
29     vertex2.applyMatrix4( object.matrixWorld );
30     const points = [vertex1, vertex2];
31     // Dibuixem la línia vertex-vertex
32     const geometry = new THREE.BufferGeometry().setFromPoints(
33         points);
34     const line = new THREE.Line(geometry, material);
35     wireFrameObject.add(line);
36 }
36 scene.add(wireFrameObject);

```

Fragment 19: Representació en filferros de les imatges

El resultat de dibuixar la representació en filferro a un nombre reduït d'imatges el trobem a la Figura 32.

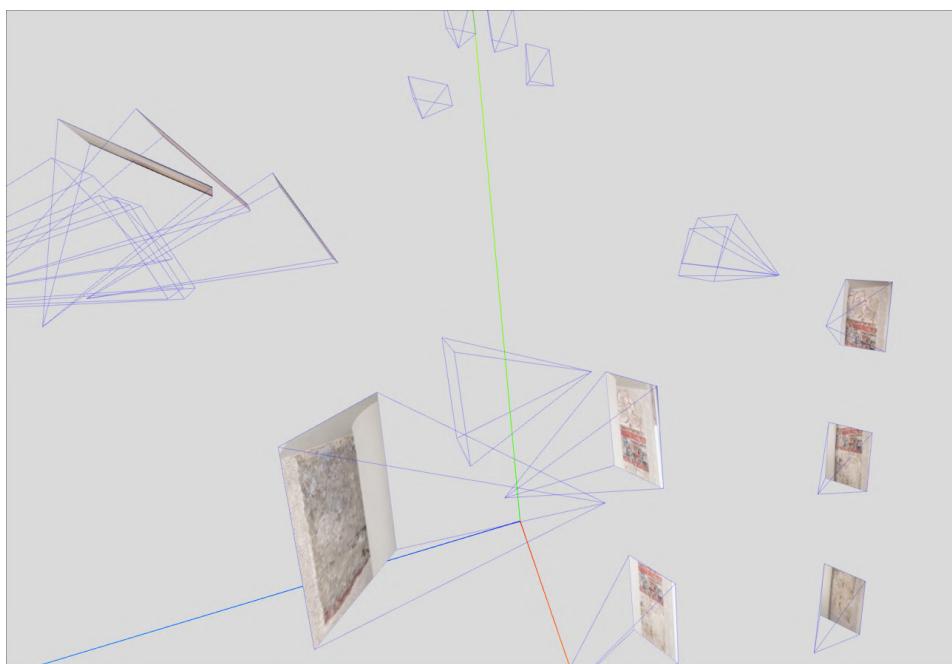


Figura 32: Conjunt d'imatges amb wireframe
Font: Producció pròpria

Després de dibuixar en filferros, calia actualitzar-los cada vegada que les imatges canvien de mida o de separació. Per fer-ho vaig decidir esborrar i tornar a dibuixar els filferros cada vegada que canviaven. El problema era que en moure el *slider*, *ThreeJS* calculava tots els filferros de nou, i alentia significativament l'execució. Per solucionar-ho i millorar l'eficiència vaig optar per fer la representació en filferros utilitzant un altre mètode.

La idea és que la modificació de l'escalat i de la separació ja actualitzés automàticament el filferro en comptes d'esborrar-lo i tornant-lo a dibuixar. Per fer-ho, vaig decidir incloure les línies dins l'objecte. Així, en modificar l'escalat de l'objecte, ja les escalaria correctament. I així va ser, en escalar l'objecte, les línies s'escalaven correctament. Cal comentar, que en afegir les línies a l'objecte, calia calcular-les en coordenades d'objecte. Per fer-ho, es va agafar el (0,0,-separació) com a posició de càmera, i els vèrtexs ja no calia convertir-los en coordenades de món. Però, en modificar la separació utilitzant la interfície gràfica, els filferros es desplaçaven juntament amb la imatge, és a dir, que la posició de càmera de la imatge es desplaçava.

Per poder solucionar aquest problema, es va decidir modificar la implementació del càlcul de la separació. La idea és escalar l'eix Z de l'objecte per calcular automàticament la seva posició, ja que l'objecte de la imatge ja està orientat respecte al vector direcció. Per fer-ho, la primitiva del pla estarà separada 1 unitat en l'eix Z, i s'escalarà l'eix Z segons el valor de la separació. Per visualitzar-ho millor, podeu observar la Figura 33.

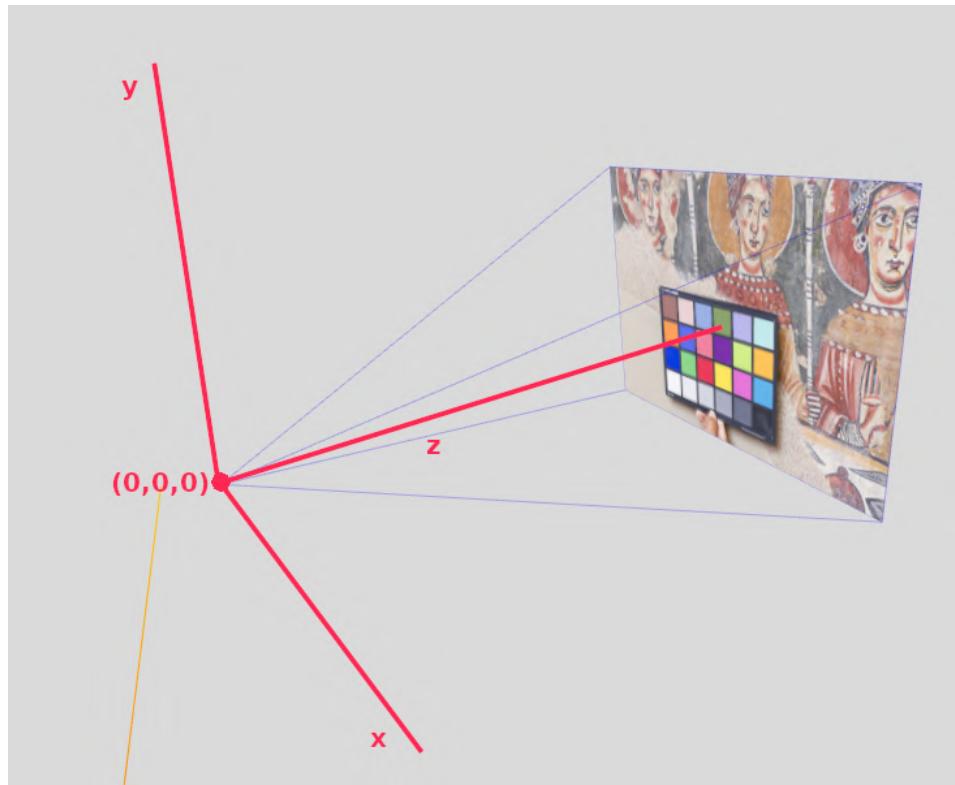


Figura 33: Coordenades d'objecte d'una imatge
Font: Producció pròpia

D'aquesta manera, quan no es vulgui tenir separació, l'objecte tindrà una mida de 0 unitats a l'eix Z, fent que la posició de la imatge coincideixi amb la posició de la càmera. Altrament, la separació coincidirà amb l'escalat de l'eix Z.

Amb aquesta modificació de la implementació de la separació d'imatges, els filferros s'escalen correctament, i en modificar la separació el punt on incideixen les línies ja no es desplaça, ja que està a l'origen de coordenades de l'objecte.

El Fragment 20 conté el codi definitiu simplificat de la creació de l'objecte imatge amb els canvis rellevants amb comentaris.

```

1 const geometry = new THREE.PlaneGeometry(texture.image.width/
    SCALE,
    texture.image.height/SCALE).rotateY(Math.PI);
2 // Traslladem la primitiva en l'eix Z
3 geometry.translate(0, 0, 1);
4 const material = new THREE.MeshBasicMaterial({map: texture,});
5 const object = new THREE.Mesh(geometry, material);
6 object.lookAt(x, -y, -z);
7 object.position.set(pos.get([0]), -pos.get([1]), -pos.get([2]))
8 ;
9 // Escalem l'eix Z per a obtenir la separació adequada

```

```
10 image_plane.scale.set(imageSize, imageSize, imageOffset);
```

Fragment 20: Creació d'una imatge actualitzat

En el cas dels filferros, els canvis es poden veure al Fragment 21.

```
1 for (let k = 0; k < 4; k++) {
2     let vertex = new THREE.Vector3().fromBufferAttribute(
3         vertexPositions, k);
4     // Esborrem la conversio a coordenades de mon
5     //vertex.applyMatrix4( object.matrixWorld );
6     // Substituim la posicio de camera per l'origen de l'
7     //objecte
8     const points = [vertice, new THREE.Vector3(0, 0, 0)];
9     const geometry = new THREE.BufferGeometry().setFromPoints(
10        points);
11    const line = new THREE.Line(geometry, material);
12    wireFrameObject.add(line);
13 }
14 for (let k = 0; k < 4; k++) {
15     let vertex1 = new THREE.Vector3().fromBufferAttribute(
16         vertexPositions, k);
17     // Esborrem la conversio a coordenades de mon
18     //vertex1.applyMatrix4( object.matrixWorld );
19     let a = 1;
20     if (k == 1) a = 3;
21     else if (k == 2) a = 0;
22     else if (k == 3) a = 2;
23     const vertex = new THREE.Vector3().fromBufferAttribute(
24         vertexPositions, a);
25     // Esborrem la conversio a coordenades de mon
26     //vertex2.applyMatrix4( object.matrixWorld );
27     const points = [vertex1, vertex2];
28     const geometry = new THREE.BufferGeometry().setFromPoints(
29        points);
30     const line = new THREE.Line(geometry, material);
31     wireFrameObject.add(line);
32 }
33 // Afegim el filferro dins de l'objecte de la imatge en comptes
34 // de l'escena
35 object.add(wireFrameObject);
```

Fragment 21: Creació de filferros actualitzat

Amb aquests canvis, els filferros ja es coordinaven amb l'escalat i la separació de les imatges, però havia sorgit un altre problema. Ara el *hover* i la selecció d'imatges no funcionava correctament. En la majoria dels casos, el *raycast* incidia en el filferro abans de la imatge. Per solucionar-ho, podem identificar quin objecte és el que ha incidit *raycast*. El *raycast* de *ThreeJS* ens retorna un *array* amb els objectes que ha incidit ordenats per proximitat. Llavors, només ens cal iterar els objectes retornats fins a trobar la primera imatge. Recordem que els objectes imatge tenen un nom assignat que comença per "Sant Quirze de Pedret by Zones". El següent codi ens retorna el primer objecte imatge del llistat (vegeu Fragment 22).

```
1 function firstImage(objects) {
```

```

2     for (let i = 0; i < objects.length; i++) {
3         if (objects[i].object.name.startsWith("Sant Quirze de
4             Pedret by Zones"))
5             return objects[i].object;
6     }
7 }
```

Fragment 22: Primera imatge del raycast

8.4 Visualització d’imatges

Aquest capítol descriu el procés d’obtenció de les dades necessàries per a poder representar les imatges en un visor en 2D. Es generarà un objecte per a cada imatge que contindrà informació com la ruta on podem trobar-la, la seva posició, la mida que té, etc. Concretament, l’objecte creat té els següents camps:

```

1 - name (cadena de caracters amb el nom o ruta de la imatge)
2 - x (nombre decimal que indica coordenada X de la posicio)
3 - y (nombre decimal que indica coordenada Y de la posicio)
4 - isLandscape (boolea que indica si la imatge es vertical o
    horitzontal)
5 - heightToWidthRatio (nombre decimal que conte la ratio alcada
    respecte amplada)
6 - zoom (nombre decimal que indica el zoom)
```

Fragment 23: Objecte amb informació de la imatge

8.4.1 Visualització d’una sola imatge

La visualització d’una sola imatge es dona quan l’usuari prem el botó esquerre del ratolí sobre una imatge. Tal com s’ha explicat anteriorment a la secció 8.3.3, el *raycast* ens permet identificar la imatge premuda. En aquest cas, ens és indiferent la posició i la mida, ja que aquests no afecten la visualització en dues dimensions. Tenint només una sola imatge, el visor 2D s’encarregarà d’escalar-la de manera que ocipi l’espai del visor. Per tant, en la visualització d’una sola imatge només ens cal obtenir el nom o ruta de la imatge premuda.

Durant la creació de l’objecte de la imatge, podem assignar la ruta de la imatge com a nom d’objecte (vegeu Fragment 24).

```
1 const image_path = object.name
```

Fragment 24: Assignació de la ruta d’imatge com a nom d’objecte

Com es pot veure a la secció 8.3.3, el *raycast* ens retorna l’objecte que incideix amb el raig. Ja que anteriorment havíem desat la ruta com a nom d’objecte, només ens cal llegir la propietat *name* per a obtenir-la.

8.4.2 Visualització d’imatges seleccionades

La visualització d’imatges seleccionades es dona quan l’usuari selecciona diverses imatges seguint el procediment especificat a la secció 8.3.3. En aquest cas ens in-

teressa obtenir la posició i mida de cada imatge per tal de distribuir-les correctament al visor 2D. A banda del nom, posició i mida, ens cal obtenir el zoom.

El zoom [57] és un factor que relaciona la distància focal amb la distància focal mínima. És a dir, en el cas que la distància focal correspongui amb la mínima, el factor de zoom serà 1, i en el cas que la distància focal sigui el doble que la mínima, el factor serà 2. La fórmula per a obtenir-lo és la següent:

$$zoom = \frac{distancia_focal}{distancia_focal_minima}$$

En el nostre cas, podem obtenir la distància focal de la imatge utilitzant el fitxer *Bundler*, tal com s'explica a la secció 8.2. Sabem que la distància focal mínima és de 25mm, obtingut fent el mínim de totes les distàncies focals obtingudes del fitxer, i comprovat amb les metadades de les imatges. Els valors *isLandscape* i *heightToWidthRatio* ja han estat calculats anteriorment.

Durant la creació de l'objecte de la imatge, podem assignar aquests valors a l'objecte, fent ús de la propietat *userData*, que ens permet desar informació. El Fragment 25 conté el codi on es fa l'assignació de valors en crear l'objecte.

```

1 const zoom = focalLength / 25;
2 object.name = image_path
3 object.userData = {
4     zoom,
5     isLandscape: isLandscape,
6     heightToWidthRatio: heightToWidthRelation,
7 };

```

Fragment 25: Assignació de valors en crear l'objecte imatge

L'objecte generat que conté les dades que utilitzarà el visor 2D té la construcció que es pot consultar al Fragment 26.

```

1 const C = camera.position;
2 const pos = get2DCoords(C, object.position);
3 const info = {
4     name: object.name,
5     x: pos.x,
6     y: pos.y,
7     isLandscape: object.userData.isLandscape,
8     heightToWidthRatio: object.userData.heightToWidthRatio,
9     zoom: object.userData.zoom,
10};

```

Fragment 26: Creació de l'objecte amb informació d'una imatge

En el Fragment de codi anterior podem observar que utilitzem la funció *get2DCoords*, que fa la conversió de coordenades de món a coordenades en 2 dimensions. A continuació es descriu el procés d'obtenció d'aquestes coordenades.

8.4.2.1 Càcul de les coordenades en 2 dimensions

El visor en 2 dimensions requereix dues coordenades per a representar la posició de la imatge, però nosaltres tenim la posició de la imatge en coordenades de mòn. Hem de trobar una manera de fer aquesta conversió fent que les posicions en el visor 2D s'acostin a les que ens podem trobar en el mòn 3D. Una manera de fer-ho és utilitzant una projecció esfèrica.

La *projecció esfèrica* [58] és la projecció de la superfície d'una esfera sobre un pla. Una manera d'obtenir-la és calcular les components (θ, ϕ) de les coordenades esfèriques dels punts que conformen la superfície de l'esfera. Els càlculs per convertir les coordenades de mòn (x, y, z) a coordenades esfèriques (r, θ, ϕ) es poden consultar a la Figura 34.

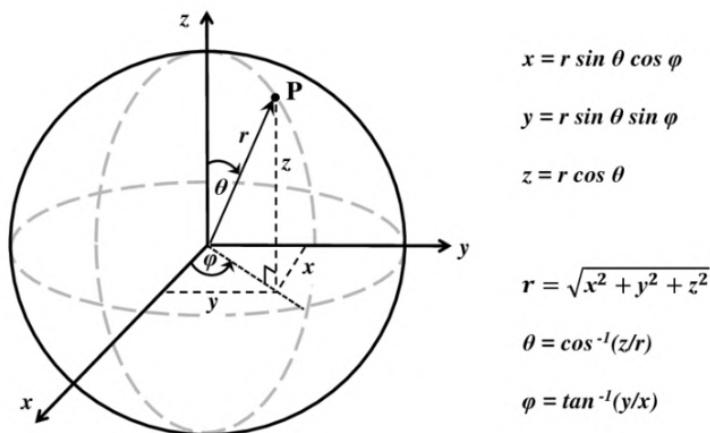


Figura 34: Coordenades esfèriques
Font: Gary Ward, Austin Peay State University

En el nostre cas hem agafat la posició de la càmera de l'escena com a centre de l'esfera. Per tal que totes les posicions de les imatges formin part de la superfície de l'esfera, normalitzarem els vectors del centre a les posicions, fent que tinguin una longitud d'una unitat. És a dir, projectarem les posicions de les imatges sobre la superfície d'una esfera de radi 1 centrada a la posició de la càmera.

A més a més, cal tenir en compte que el sistema de coordenades mencionat a la imatge no correspon al sistema de *ThreeJS*. En el visor de *ThreeJS* és la coordenada Y la que indica l'alçada, i és la Z la que ens indica la profunditat. Per això, en el Fragment 27, les components del vector estan ordenades de forma diferent.

```

1 function get2DCoords(C, P) {
2     // Vector centre a posicio d'imatge de longitud 1
3     const V = new THREE.Vector3().subVectors(P, C).normalize();
4     const phi = math.acos(V.y);
5     const theta = math.atan2(V.x, V.z);
6     return { x: theta, y: phi };
7 }
```

Fragment 27: Conversió de coordenades de mòn a coordenades esfèriques

8.4.3 Visualització d'imatges en un pla

La visualització d'imatges en un pla es dona quan l'usuari crea un pla a l'entorn 3D, i desitja visualitzar diverses imatges plasmades al pla creat. En aquesta secció es descriu el procediment de la representació d'un pla a l'entorn 3D, el càlcul de quines imatges s'inclouen, i finalment la conversió de coordenades de món a coordenades d'un pla.

8.4.3.1 Representant la figura

Un *pla* [59] és un espai de dues dimensions que s'estén a l'infinit. A l'espai 3D, podem calcular l'equació del pla $Ax + By + Cz + D = 0$ utilitzant 3 punts de l'espai sempre que no siguin co-linears, és a dir que no hi hagi cap recta que passi per aquests 3 punts. Per a obtenir l'equació mitjançant 3 punts $p_1(x_1, y_1, z_1)$, $p_2(x_2, y_2, z_2)$ i $p_3(x_3, y_3, z_3)$, podem utilitzar les fórmules següents [60]:

$$\begin{aligned} P &= p_2 - p_1 = (x_2, y_2, z_2) - (x_1, y_1, z_1) \\ Q &= p_3 - p_1 = (x_3, y_3, z_3) - (x_1, y_1, z_1) \\ N &= P \times Q \\ A, B, C &= N_1, N_2, N_3 \\ D &= -(A * x_1 + B * y_1 + C * z_1) \end{aligned}$$

És a dir, donats 3 punts $p1$, $p2$ i $p3$, calculem dos vectors del pla. Fent el producte vectorial d'aquests obtenim el vector normal, que les seves components corresponen a les components A, B i C del pla. Per trobar la component D només ens cal substituir un dels punts a l'equació del pla.

Aplicat al nostre projecte, l'usuari pot seleccionar 3 imatges tal com s'ha descrit a la secció 8.3.3. Podem accedir a la posició de les 3 imatges seleccionades. *ThreeJS* posa a disposició la classe *Plane* [61], que conté el mètode *setFromCoplanarPoints* [62], que et retorna el pla obtingut a partir de 3 punts. D'aquesta manera no ens cal utilitzar les fórmules descrites anteriorment. El codi per calcular el pla d'una certa imatge el trobem al Fragment 28.

```
1 const A = selectedImages[0].position;
2 const B = selectedImages[1].position;
3 const C = selectedImages[2].position;
4 const plane = new THREE.Plane().setFromCoplanarPoints(A, B, C);
```

Fragment 28: Càlcul d'un pla a ThreeJS

El següent objectiu és representar aquesta figura a l'entorn 3D. Es va veure convenient afegir un paràmetre que definís la distància en la qual s'incluïa una imatge o no al visor 2D, decidint que la millor manera de representar el pla seria mitjançant un prisma.

Per generar un prisma utilitzarem la primitiva *BoxGeometry* [63] que ens proporciona *ThreeJS*. La posició d'aquest prisma correspondrà a qualsevol de les posicions

dels 3 punts. Per a orientar el prisma, utilitzarem el mètode *lookAt* [45] amb el punt focal. Aquest punt s'obté sumant el vector normal al punt on situem el pla. El codi simplificat per a representar el prisma a l'entorn 3D és el que trobem al Fragment 29:

```

1 const A = selectedImages[0].position;
2 var focalPoint = new THREE.Vector3().addVectors(A,
    abstractPlane.normal);
3 const boxGeometry = new THREE.BoxGeometry(1, 1, 1, 10, 10, 10);
4 const box = new THREE.Mesh(boxGeometry, boxMaterial);
5 scene.add(box);
6 box.lookAt(focalPoint);
7 box.position.set(A.x, A.y, A.z);

```

Fragment 29: Representació d'un prisma a l'entorn 3D

A continuació es pot observar el resultat de representar un prisma que representa un pla a *ThreeJS* (vegeu Figura 35).

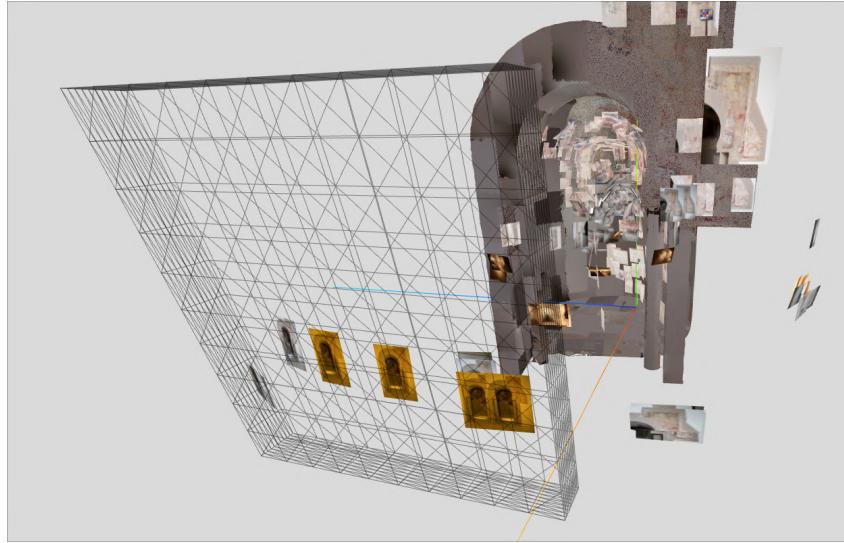


Figura 35: Representació del pla a ThreeJS
Producció pròpria

El prisma representat, en no tenir una extensió infinita, vaig decidir que en generar el prisma, que quedés centrat entre les 3 imatges seleccionades, i amb la mida justa per tal que englobés les 3 imatges. Per aconseguir-ho cal obtenir les coordenades del pla dels 3 punts. La conversió de coordenades de món a coordenades 2D mitjançant la projecció d'un pla es descriurà a la secció 8.4.3.2.

Amb aquestes coordenades, podem obtenir l'alçada del pla fent la diferència del màxim i el mínim de la component Y dels 3 punts. De la mateixa manera podem obtenir l'amplada del pla amb les components X. Finalment podem assignar la mida del prisma amb aquests valors. El codi corresponent el trobem al Fragment 30.

```

1 const A2d = worldCoordsToPlaneCoords(A);
2 const B2d = worldCoordsToPlaneCoords(B);
3 const C2d = worldCoordsToPlaneCoords(C);
4 const amplada =
5     max(A2d.x, B2d.x, C2d.x) - min(A2d.x, B2d.x, C2d.x);

```

```

6 const alcada =
7     max(A2d.y, B2d.y, C2d.y) - min(A2d.y, B2d.y, C2d.y);
8 // Assignem la mida del prisma
9 object.scale.set(amplada, alcada, distància);

```

Fragment 30: Càcul d'alçada i amplada del pla

Per a calcular el centre del pla, podem fer la mitjana aritmètica amb la màxima i mínima component X dels 3 punts, i amb la màxima i mínima component Y. Una vegada obtingut aquest punt, cal convertir aquest en coordenades de món. Aquesta conversió es descriurà a la secció 8.4.3.3. Aquesta posició serà la nova posició del pla. El codi simplificat corresponent és el que trobem al Fragment 31.

```

1 const centerX =
2     (max(A2d.x, B2d.x, C2d.x) + min(A2d.x, B2d.x, C2d.x)) / 2;
3 const centerY =
4     (max(A2d.y, B2d.y, C2d.y) + min(A2d.y, B2d.y, C2d.y)) / 2;
5 const center3d =
6     planeToWorldCoords(new THREE.Vector2(centerX, centerY));
7 // Assignem la nova posició del pla
8 object.position.set(center3d.x, center3d.y, center3d.z);

```

Fragment 31: Càcul del centre del pla

Aquests canvis ens dona una representació del pla centrat entre les 3 imatges, i amb la mida justa, tal com veiem a la Figura 36.



Figura 36: Pla centrat i reajustat
Producció pròpria

A més a més, vam decidir que l'usuari tingués l'opció de definir la mida del prisma, és a dir, que pogués seleccionar l'amplada i l'alçada, a banda de la profunditat o distància respecte al pla. Això es du a terme amb uns *sliders* incorporats a la interfície gràfica (vegeu Figura 37).

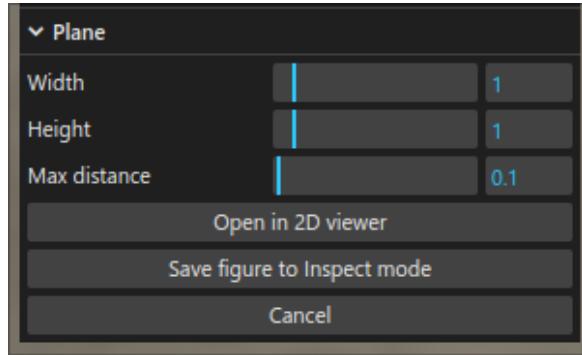


Figura 37: Interfície gràfica amb opcions del pla
Font: Producció pròpia

Cada vegada que algun d'aquests *sliders* canvia de valor, es crida una funció que modifica les dimensions del prisma. Simplement, escala l'invers dels valors actuals, i l'escala amb els nous valors. En el cas de l'amplada, la funció és la que podem trobar al Fragment 32.

```

1 function changePlaneWidth(nova_amplada) {
2     boxObject.scale.set(1/amplada, 1/alcada, 1/distancia);
3     boxObject.scale.set(a, alcada, distancia);
4     amplada = nova_amplada;
5 }
```

Fragment 32: Modificar amplada del prisma

8.4.3.2 Càcul de les coordenades en 2 dimensions

El visor en 2 dimensions i el càlcul de l'alçada, amplada i centre del pla requereix dues coordenades per a representar la posició de la imatge, però nosaltres tenim la posició de la imatge en coordenades de món. En aquest cas, utilitzarem la projecció en un pla.

La *projecció en un pla* o *projecció planar* [64] és la projecció d'un punt de l'espai sobre un pla. Els punts projectats passen de tenir 3 coordenades, a només dues. Hi ha diversos tipus de projeccions en un pla. Nosaltres utilitzem la *projecció ortogràfica* [65], la projecció més habitual. Els càlculs per obtenir les coordenades en un pla a partir de coordenades de món són els següents (sempre que T no coincideixi amb N):

Donats un punt P , el vector normal del pla N , i l'origen de coordenades del pla C ,

$$V = P - C$$

$$T = (0, 1, 0)$$

$$B = N \times T$$

$$x = -V \cdot T$$

$$y = C \cdot B$$

La Figura 38 ens ajuda a entendre les fórmules anteriors.

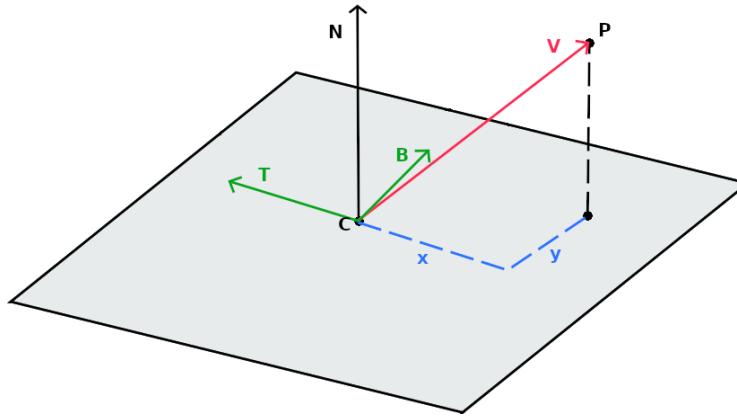


Figura 38: Projecció d'un punt al pla
Font: Producció pròpria

La funció encarregada de fer la conversió de coordenades és la que es troba del Fragment 33.

```

1 const N = new THREE.Vector3().copy(plane.normal).normalize();
2 const T = new THREE.Vector3().copy(N).cross(new THREE.Vector3
    (0, 1, 0)).normalize();
3 const B = new THREE.Vector3().copy(N).cross(T).normalize();
4
5 function worldCoordsToPlaneCoords(P) {
6     // C correspon a un punt del pla en coordenades de món
7     const V = new THREE.Vector3().subVectors(P, C);
8     const tv = new THREE.Vector3().copy(V).dot(T);
9     const bv = new THREE.Vector3().copy(V).dot(B);
10    return { x: -tv, y: bv };
11 }
```

Fragment 33: Conversió de coordenades de món a coordenades del pla

8.4.3.3 Conversió de coordenades de pla a coordenades de món

Com s'ha especificat anteriorment, necessitem una funció que faci la conversió de coordenades del pla a coordenades de món. A continuació es mostren les fórmules utilitzades per a aquesta conversió.

Donats un punt del pla Q, l'origen de coordenades C del pla, i els vectors T i B,

$$I = Q.x * T$$

$$J = Q.y * B$$

$$P = A + I + J$$

La Figura 39 ens ajuda a entendre aquests càlculs:

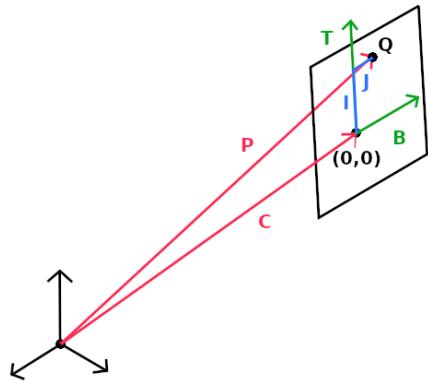


Figura 39: Conversió de coordenades del pla a coordenades de món
Font: Producció pròpria

Al Fragment 34 trobem la implementació de la funció que ens fa la conversió d'un punt en coordenades de pla a coordenades de món.

```

1 function planeCoordsToWorldCoords(Q) {
2     const I = new THREE.Vector3().copy(T).multiplyScalar(Q.x);
3     const J = new THREE.Vector3().copy(B).multiplyScalar(Q.y);
4     const P = new THREE.Vector3().copy(C).add(I).add(J);
5     return P;
6 }
```

Fragment 34: Conversió de coordenades del pla a coordenades de món

8.4.3.4 Càcul de les imatges incloses

Una vegada podem obtenir les coordenades del pla de cada imatge, ens cal fer un filtratge de quines imatges volem visualitzar en el visor 2D. Les imatges que inclourem són aquelles que la seva posició es troba dins del prisma generat.

La primera condició que la imatge ha de complir per ser inclosa, és que la mínima distància entre el punt i el pla ha de ser inferior al paràmetre que defineix la profunditat del pla. Per fer-ho, ens cal fer una projecció del punt al pla, en coordenades 3D. Per sort, *ThreeJS* ens proporciona el mètode *projectPoint* [66], que ens permet projectar un punt al pla obtenint les coordenades 3D de la projecció. Calculant la distància que hi ha entre el punt i la seva projecció al pla, podem comparar si és menor o major a la distància establerta per l'usuari. Si la distància és major, es rebutja.

La segona condició és que el punt es trobi dins de l'alçada i amplada definida per l'usuari. Per fer-ho, convertirem les coordenades del punt a coordenades 2D del pla. Sabent que l'origen del pla és el punt (0,0), podem comprovar si entra dins del pla utilitzant l'amplada i l'alçada d'aquest. En el cas que la component X de la coordenada del punt al pla sigui menor a la meitat de l'amplada, aquesta component entrerà dins. El mateix podem fer amb la component Y de la projecció i l'alçada del pla. Estem dividint entre 2 l'alçada i l'amplada, ja que el pla està centrat al (0,0). El Fragment 35 calcula un booleà que ens indica si la posició d'una imatge es troba dins del prisma.

```

1 const P = object.position;
2 let Q = new THREE.Vector3();
3 plane.projectPoint(P, Q);
4 const P2d = worldCoordsToPlaneCoords(P);
5 const isInside =
6     P.distanceTo(Q) < planeDistance
7     && abs(P2d.x) < planeWidth / 2
8     && abs(P2d.y) < planeHeight / 2

```

Fragment 35: Booleà que ens indica si una imatge es troba dins del prisma

Finalment, l'objecte generat que conté les dades que utilitzarà el visor 2D té la construcció mostrada a la Figura 36.

```

1 const P = object.position;
2 const P2d = get2DCoords(P);
3 let info = {};
4
5 if (isInside) {
6     info = {
7         name: object.name,
8         x: P2d.x,
9         y: P2d.y,
10        isLandscape: object.userData.isLandscape,
11        heightToWidthRatio: object.userData.heightToWidthRatio,
12        zoom: object.userData.zoom,
13    };
14 }

```

Fragment 36: Creació de l'objecte amb informació d'una imatge

8.4.4 Visualització d'imatges en mode esfèric

La visualització d'imatges en mode esfèric es dona quan un usuari crea una esfera a l'entorn 3D, i desitja visualitzar diverses imatges plasmades al pla que forma la superfície d'una esfera. En aquesta secció es descriu el procediment de la representació d'una esfera a l'entorn 3D, el càlcul de quines imatges s'inclouen, i finalment la conversió de coordenades de món a coordenades esfèriques.

8.4.4.1 Representant la figura

Una *esfera* [67] és un espai on tots els punts es troben a la mateixa distància respecte a un punt en concret. Aquest punt l'anomenem centre de l'esfera, i la distància és el radi. Per tant, podem definir l'esfera amb un punt P , i un radi r .

Aplicat al nostre projecte, l'usuari pot seleccionar una imatge tal com s'ha descrit a la secció 8.3.3. Podem accedir a la posició de la imatge seleccionada de la manera mostrada a la Figura 37.

```

1 const centre = selectedImages[0].position;

```

Fragment 37: Obtenció del centre de l'esfera

El següent objectiu és representar aquesta figura a l'entorn 3D. La posició de l'esfera correspon a la posició de la imatge seleccionada. En el nostre cas, l'esfera té

per defecte el radi de mitja unitat. Per representar l'esfera a l'entorn 3D, podem utilitzar la primitiva *sphereGeometry* [68] que ens proporciona *ThreeJS*. El Fragment 38 s'encarrega de la representació d'una esfera a l'entorn 3D.

```

1 const centre = selectedImages[0].position;
2 // Esfera de radi 1
3 const geometry = new THREE.SphereGeometry(1);
4 const sphereObject = new THREE.Mesh(geometry, material);
5 scene.add(sphereObject);
6 sphereObject.scale.set(0.5, 0.5, 0.5);
7 sphereObject.position.set(centre.x, centre.y, centre.z);

```

Fragment 38: Representació d'una esfera a l'entorn 3D

A la Figura 40 es pot observar el resultat de representar una esfera a *ThreeJS*.



Figura 40: Representació d'una esfera a ThreeJS
Producció pròpia

A més a més, vam decidir que l'usuari tingués l'opció de definir la mida de l'esfera, és a dir el radi. Això es du a terme amb un *slider* incorporats a la interfície gràfica (vegeu Figura 41).

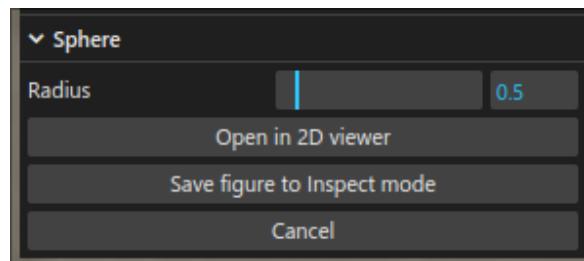


Figura 41: Interfície gràfica amb opcions de l'esfera
Font: Producció pròpria

Cada vegada que aquest *slider* canvia de valor, es crida una funció que modifica el radi de l'esfera. Simplement, escala l'esfera amb l'invers del radi actual, i l'escala amb el nou valor. La funció que modifica el radi de l'esfera és la que trobem al Fragment 39.

```

1 function applySphericalRadius(nou_radi) {
2     sphereObject.scale.set(1 / radi, 1 / radi, 1 / radi);
3     sphereObject.scale.set(nou_radi, nou_radi, nou_radi);
4     radi = nou_radi;
5 }
```

Fragment 39: Modificar radi de l'esfera

8.4.4.2 Càcul de les coordenades en 2 dimensions

El visor en 2 dimensions requereix dues coordenades per a representar la posició de la imatge, però nosaltres tenim la posició de la imatge en coordenades de món. En aquest cas, utilitzarem la *projecció esfèrica* [58], que projecta la superfície d'una esfera sobre un pla. Aquesta projecció ha estat explicada anteriorment a la secció 8.4.2.1.

Per a la conversió de coordenades s'utilitza el codi especificat a 8.4.2, però amb una petita diferència. En aquest cas, en contres d'establir la posició de la càmera com a centre de l'esfera, utilitzarem la posició de la imatge seleccionada com a centre.

8.4.4.3 Càcul de les imatges incloses

Una vegada podem obtenir les coordenades esfèriques de cada imatge, ens cal fer un filtratge de quines imatges volem visualitzar en el visor 2D. Les imatges que inclourem són aquelles que la seva posició es troba dins de l'esfera generada.

La condició que la imatge ha de complir per ser inclosa, és que la mínima distància entre el punt i el centre de l'esfera ha de ser inferior al radi establert. Si la distància és major al radi, la imatge es rebutja. El Fragment 40 calcula un booleà que ens indica si la posició d'una imatge es troba dins de l'esfera.

```

1 const P = object.position;
2 const isInside = centre.distanceTo(P) < radi
```

Fragment 40: Booleà que ens indica si una imatge es troba dins de l'esfera

Finalment, l'objecte generat que conté les dades que utilitzarà el visor 2D té la construcció que mostra el Fragment 41.

```

1 const P = object.position;
2 let info = {};
3
4 if (isInside) {
5     const P2d = get2DCoords(P);
6     info = {
7         name: object.name,
8         x: P2d.x,
9         y: P2d.y,
10        isLandscape: object.userData.isLandscape,
11        heightToWidthRatio: object.userData.heightToWidthRatio,
12        zoom: object.userData.zoom,
13    };
14}
```

Fragment 41: Creació de l'objecte amb informació d'una imatge

8.4.5 Visualització d'imatges en mode cilíndric

La visualització d'imatges en un cilindre es dona quan l'usuari crea un cilindre a l'entorn 3D, i desitja visualitzar diverses imatges plasmades a la superfície lateral del cilindre. En aquesta secció es descriu el procediment de la representació d'un pla a l'entorn 3D, el càlcul de quines imatges s'inclouen, i finalment la conversió de coordenades de móv a coordenades cilíndriques.

8.4.5.1 Representant la figura

Un *cilindre* [69] és un espai de tres dimensions format per un rectangle revolucionat al voltant d'un dels seus costats. Hi ha diverses maneres de representar un cilindre, però nosaltres el definim amb dos punts i un radi. Aquests 2 punts corresponen al centre de les dues bases. L'alçada d'aquest correspon a la distància entre els 2 punts, i el radi queda definit per l'usuari.

Aplicat al nostre projecte, l'usuari pot seleccionar 2 imatges tal com s'ha descrit a la secció 8.3.3. Podem accedir a la posició de les 2 imatges seleccionades. Podem accedir a la posició de les dues imatges seleccionades de la manera mostrada al Fragment 42.

```
1 const A = selectedImages[0].position;  
2 const B = selectedImages[1].position;
```

Fragment 42: Obtenció dels centres de les dues cares d'un cilindre

El següent objectiu és representar aquesta figura a l'entorn 3D. La posició del cilindre correspondrà al punt central C del segment format entre els dos punts A i B seleccionats. L'alçada h correspon a la distància entre els dos punts, i el radi r agafa un valor predeterminat de 0,5 unitats. Finalment, el vector direcció D del cilindre correspon al vector entre el punt central C , i un dels punts A o B . La Figura 42 mostra un esquema del cilindre amb aquests paràmetres.

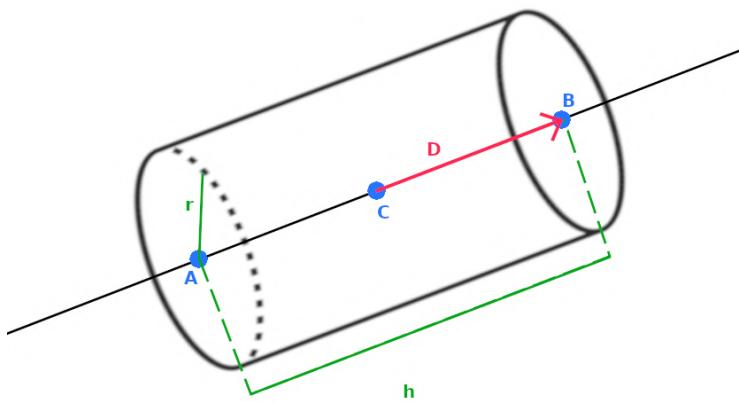


Figura 42: Paràmetres d'un cilindre
Font: Producció pròpia

Per generar un cilindre utilitzarem la primitiva *CylinderGeometry* [70] que ens proporciona *ThreeJS*. La posició d'aquest prisma correspondrà al centre del cilindre

C. Per a orientar-lo, utilitzarem el mètode *lookAt* [45] amb el vector direcció D. També, escalarem el cilindre utilitzant la distància entre els punts A i B i com a alçada, i el radi r com a amplada i profunditat de la geometria. Cal rotar la primitiva del cilindre 90° respecte a l'eix X per inicialitzar-lo correctament. El codi simplificat per a representar el cilindre a l'entorn 3D és el que trobem al Fragment 43.

```

1 const A = selectedImages[0].position;
2 const B = selectedImages[1].position;
3 const C = new THREE.Vector3((A.x + B.x)/2, (A.y + B.y)/2, (A.z
+ B.z)/2);
4 const D = new THREE.Vector3().subVectors(P2, P1);
5 const h = V.length();
6 // Cal rotar el cilindre 90 graus per inicialitzar-lo
// horitzontalment
7 const geometry = new THREE.CylinderGeometry(1, 1, 1, 10, 1,
true).rotateX(Math.PI / 2);
8 let cylinderObject = new THREE.Mesh(geometry, material);
9 // Al haver rotat el cilindre 90 graus, la profunditat de la
figura marca l'alcada
10 cylinderObject.scale.set(0.5, 0.5, h);
11 cylinderObject.lookAt(D.normalize());
12 cylinderObject.position.set(C.x, C.y, C.z);

```

Fragment 43: Representació d'un cilindre a l'entorn 3D

A continuació es pot observar el resultat de representar un cilindre a *ThreeJS* (vegeu Figura 43).

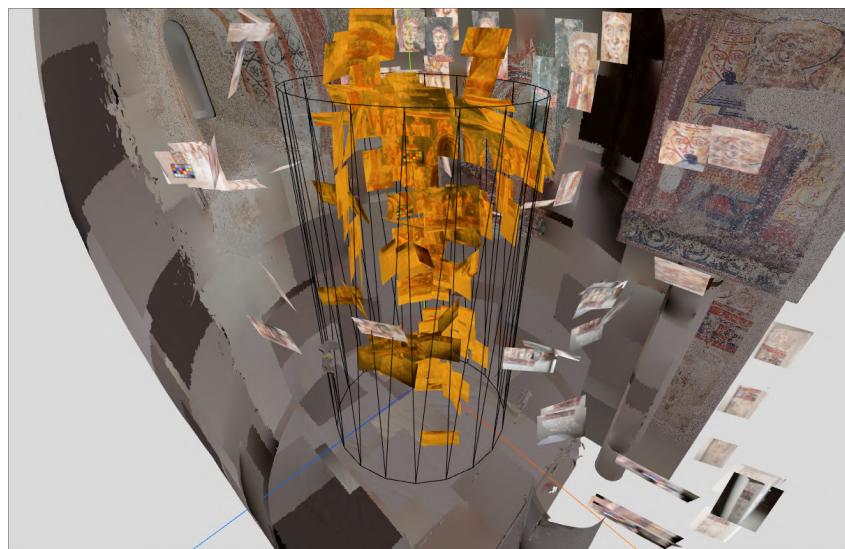


Figura 43: Representació d'un cilindre a ThreeJS
Producció pròpia

A més a més, vam decidir que l'usuari tingués l'opció de definir la mida del cilindre, és a dir, que pogués seleccionar el radi i l'alçada. Això es du a terme amb uns *sliders* incorporats a la interfície gràfica (vegeu Figura 44):

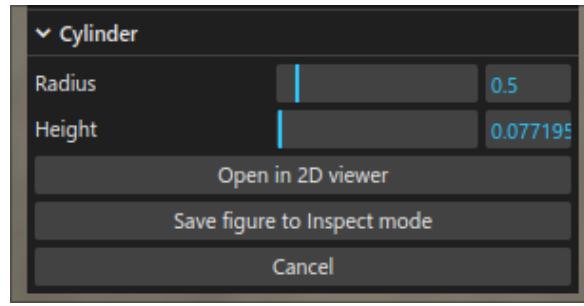


Figura 44: Interfície gràfica amb opcions del cilindre
Font: Producció pròpria

Cada vegada que algun d'aquests *sliders* canvia de valor, es crida una funció que modifica les dimensions del cilindre. Simplement, escala l'invers dels valors actuals, i l'escala amb els nous valors. En el cas del radi, la funció és la que podem trobar al Fragment 44.

```

1 function changeRadius(nou_radi) {
2     cylinderObject.scale.set(1 / radi, 1 / radi, 1 / height);
3     cylinderObject.scale.set(nou_radi, nou_radi, height);
4     radi = nou_radi;
5 }
```

Fragment 44: Modificar radi del cilindre

8.4.5.2 Càcul de les coordenades en 2 dimensions

El visor en 2 dimensions requereix dues coordenades per a representar la posició de la imatge, però nosaltres tenim la posició de la imatge en coordenades de món. En aquest cas, utilitzarem la projecció en el pla format per la cara lateral del cilindre. Projectarem el punt en qüestió sobre la superfície de la cara lateral, per tal d'obtenir les coordenades en el *sistema de coordenades cilíndriques* [71]. Els càlculs per obtenir les coordenades de món (x, y, z) a coordenades cilíndriques (ϕ, k) són els següents:

Primer de tot necessitarem el punt P projectat sobre la recta. Donats el punt P , i els punts centrals A, B de les dues cares,

$$\begin{aligned} AP &= P - A \\ AB &= B - A \\ P' &= A + \frac{AP \cdot AB}{AB \cdot AB} * AB \end{aligned}$$

Donats un punt P , el punt projectat P' , el punt central A de la cara d'un cilindre, i un vector O perpendicular a la recta,

$$\begin{aligned} k &= |A - P'| \\ V &= P' - P \\ \phi &= \text{acos} \frac{O \cdot V}{|O| * |V|} \end{aligned}$$

La Figura 45 ens ajuda a entendre les fórmules anteriors.

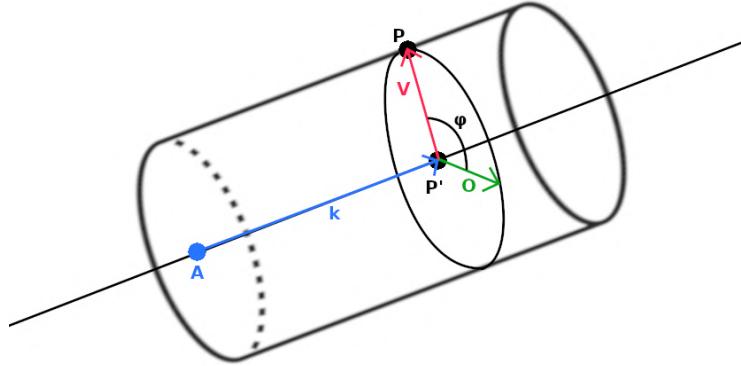


Figura 45: Obtenció de coordenades cilíndriques
Font: Producció pròpria

Per a la implementació de la funció encarregada per a la conversió de coordenades, utilitzaré un mètode ja implementat anomenat *closestPointToPoint* [72], que retorna el punt projectat en un segment. A més a més, podem calcular l'angle entre O i V utilitzant el mètode *angleTo* [73]. El codi simplificat de la funció és el que trobem al Fragment 45.

```

1 function get2dCoords(P) {
2     // Obtencio d'un vector O qualsevol, perpendicular al
3     // segment
4     const segment = new THREE.Line3(A, B);
5     const origin = new THREE.Vector3(0, 0, 0);
6     var origin_projected = new THREE.Vector3();
7     segment.closestPointToPoint(origin, false, origin_projected);
8
9     // P'
10    const P2 = new THREE.Vector3();
11    segment.closestPointToPoint(P, true, P2);
12
13    const k = A.distanceTo(P2);
14    const V = new THREE.Vector3().subVectors(P, P2).normalize();
15    const phi = O.angleTo(V);
16    return {k: k, phi: phi};
17 }
```

Fragment 45: Conversió de coordenades de món a coordenades del cilindre

8.4.5.3 Càcul de les imatges incloses

Una vegada podem obtenir les coordenades del pla de cada imatge, ens cal fer un filtratge de quines imatges volem visualitzar en el visor 2D. Les imatges que inclourem són aquelles que la seva posició es troba dins del cilindre generat.

La primera condició que la imatge ha de complir per ser inclosa, és que la distància

mínima de la posició de la imatge amb la recta infinita sigui menor al radi. Si aquesta distància és major, la imatge es rebutja.

La segona condició és que el punt es trobi entre els dos plans que formen les cares del cilindre. Per estalviar-nos la feina de calcular els dos plans, utilitzarem un truc utilitzant distàncies entre un punt i un segment. Calcularem la distància entre el punt i la recta infinita del cilindre. A més a més, calcularem la distància entre el punt i el segment entre les dues cares d'aquest. En el cas que aquestes dues distàncies siguin equivalents, significarà que el vector de menys distància a la recta infinita és el mateix que el vector del punt cap al segment. En aquest cas, la imatge s'inclourà. Altrament, significarà que el vector del punt a la recta infinita és més curt, indicant-nos que aquest queda fora del cilindre. La Figura 46 ens ajuda a visualitzar aquesta condició.

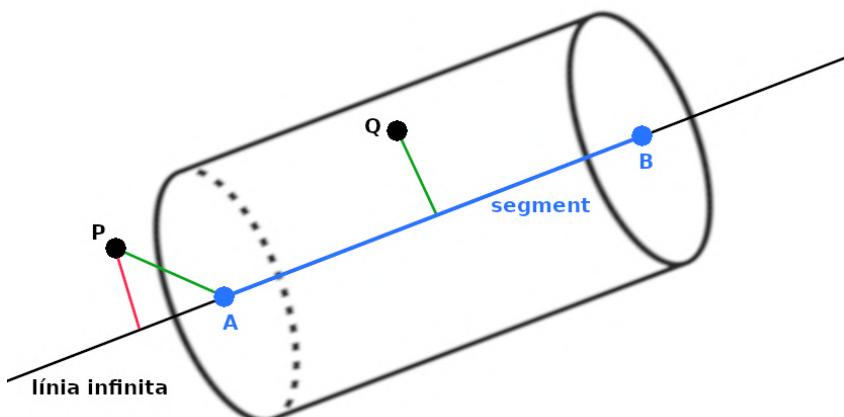


Figura 46: Inclusió de punts dins d'un cilindre
Producció pròpia

La Figura 46 ens ajuda a visualitzar que en el cas que un punt quedi fora de les dues cares del cilindre, la distància entre aquest i el segment sempre serà major que la distància mínima entre el punt i la recta. En el cas que el punt estigui entre les dues cares, la distància del punt amb la recta i el segment sempre serà la mateixa, ja que el segment està sobreposat a la recta.

Per a calcular-ho amb *ThreeJS*, crearem un segment i una recta. Com que *ThreeJS* no disposa d'una classe recta infinita, construirem un vector llarg. Utilitzarem el mètode *closestPointToPoint* [72] per tal d'obtenir el punt més pròxim del segment donat un punt qualsevol. Fent la distància d'aquests dos punts, obtindrem la distància mínima d'un punt respecte a un segment. El Fragment 46 calcula un booleà que ens indica si la posició d'una imatge es troba dins del prisma. Recordem que C és el centre del cilindre, i D el vector CB .

```

1 const P = object.position;
2 const segment = new THREE.Line3(A, B);
3 // ThreeJS no disposa de rectes infinites, farem un segment
  llarg

```

```

4 const vector_inifinit = new THREE.Vector3(D.x, D.y, D.z).
    multiplyScalar(1000);
5 const extremRecta1 = new THREE.Vector3(C.x, C.y, C.z).add(
    vector_inifinit);
6 const extremRecta2 = new THREE.Vector3(C.x, C.y, C.z).sub(
    vector_inifinit);
7 const recta = new THREE.Line3(extremRecta1, extremRecta2);
8
9 const puntSegment = new THREE.Vector3();
10 segment.closestPointToPoint(P, true, puntSegment);
11 const distanciaSegment = puntSegment.distanceTo(P);
12
13 const puntRecta = new THREE.Vector3();
14 recta.closestPointToPoint(P, true, puntRecta);
15 const distanciaRecta = puntRecta.distanceTo(P);
16
17 const inside = distanciaRecta < radi && distanciaRecta ==
    distanciaSegment

```

Fragment 46: Booleà que ens indica si una imatge es troba dins del cilindre

Finalment, l'objecte generat que conté les dades que utilitzarà el visor 2D té la construcció mostrada al Fragment 47.

```

1 const P = object.position;
2 let info = {};
3
4 if (isInside) {
5     const P2d = get2DCoords(P);
6     info = {
7         name: object.name,
8         x: P2d.k,
9         y: P2d.phi,
10        isLandscape: object.userData.isLandscape,
11        heightToWidthRatio: object.userData.heightToWidthRatio,
12        zoom: object.userData.zoom,
13    };
14 }

```

Fragment 47: Creació de l'objecte amb informació d'una imatge

8.5 Mode Autor i Inspecció

Els modes *Autor* i *Inspecció* pretenen fer una distinció de dos casos d'ús diferents. El mode Autor està dirigit a usuaris més avançats, donant la possibilitat de generar unes figures predefinides, que contenen imatges d'una part de l'església. En canvi, el mode Inspecció està dirigit a usuaris que volen visualitzar un conjunt d'imatges predefinides per un usuari amb el mode Autor.

8.5.1 Mode Autor

El mode Autor està dirigit a usuaris avançats que volen tenir un control total sobre la selecció d'imatges a visualitzar. Aquest mode conté totes les funcionalitats descrites fins ara. El mode mostra totes les imatges disponibles distribuïdes per l'escena, i incorporen totes les interaccions descrites al capítol 8.3. També té la possibilitat de

visualitzar les imatges de totes les maneres descrites a 8.4. A més a més d'aquestes funcionalitats, disposarà l'opció de crear una figura per al mode Inspecció.

Per a implementar-ho, afegirem un botó que permeti desar la figura, una vegada s'ha creat, i s'ha definit la mida desitjada. Una vegada es premi el botó, es generarà una figura de mida reduïda amb tots els paràmetres necessaris. A més a més, desarem paràmetres que defineixen la figura creada, però no representats per la figura reduïda. És a dir, es desarà la mida real de la figura encara que no sigui utilitzada directament. Aquests paràmetres ens seran d'utilitat a l'hora de mostrar les imatges que engloba la figura. Per generar les figures, podeu consultar els capítols 8.4.3.1, 8.4.4.1 i 8.4.5.1. Els fragments 48, 49 i 50 contenen el codi per a generar i desar les figures.

```

1 // Vector que conte totes les esferes desades
2 var spheres = [];
3 // C: centre de l'esfera, r: radi
4 function saveSphere(C, r) {
5     // Generem l'esfera reduïda
6     const geometry = new THREE.SphereGeometry(0.05, 10, 10);
7     const material = new THREE.MeshBasicMaterial({
8         color: NORMAL_COLOR,
9     });
10    const sphereObject = new THREE.Mesh(geometry, material);
11    // Establim la mida i posicio
12    sphereObject.position.set(C.x, C.y, C.z);
13    sphereObject.userData.radius = r;
14    sphereObject.name = "Sphere" + spheres.length;
15    // L'afegim a l'escena de forma oculta, i la desem al
16    // vector
17    sphereObject.visible = false;
18    scene.add(sphereObject);
19    spheres.push(sphereObject);
}

```

Fragment 48: Generar i desar una esfera

```

1 // Vector que conte tots els cilindres desats
2 var cylinders = [];
3 // C: centre del cilindre, r: radi, h: alcada, V: vector eix
4 function saveCylinder(C, r, h, V) {
5     // Generem el cilindre reduït
6     const geometry = new THREE.CylinderGeometry(0.05, 0.05,
7         0.1, 10).rotateX(Math.PI / 2);
7     const material = new THREE.MeshBasicMaterial({
8         color: NORMAL_COLOR,
9     });
10    const cylinderObject = new THREE.Mesh(geometry, material);
11    // Establim la mida, posicio i rotacio
12    cylinderObject.lookAt(V.normalize());
13    cylinderObject.position.set(C.x, C.y, C.z);
14    cylinderObject.userData.radius = r;
15    cylinderObject.userData.height = h;
16    cylinderObject.userData.vector = V;
17    cylinderObject.name = "Cylinder" + cylinders.length;
18    // L'afegim a l'escena de forma oculta, i el desem al
19    // vector

```

```

19     cylinderObject.visible = false;
20     scene.add(cylinderObject);
21     cylinders.push(cylinderObject);
22 }
```

Fragment 49: Generar i desar un cilindre

```

1 // Vector que conte totes les esferes desades
2 var planes = [];
3 // C: centre del pla, plane: pla abstracte, h: alcada, w:
4 // amplada, d: profunditat, t: vector t, b: vector b
5 function savePlane(C, plane, h, w, d, t, b) {
6     var coplanarPoint = plane.coplanarPoint(new THREE.Vector3()
7         .copy(C));
8     var focalPoint = new THREE.Vector3().addVectors(
9         coplanarPoint, plane.normal);
10    // Generem el pla reduit
11    const boxGeometry = new THREE.BoxGeometry(0.1, 0.1, 0.02,
12        10, 10, 10);
13    const boxMaterial = new THREE.MeshBasicMaterial({
14        color: NORMAL_COLOR,
15    });
16    const box = new THREE.Mesh(boxGeometry, boxMaterial);
17    // Establim la mida, posicio i rotacio, i desem altres
18    // parametres
19    box.lookAt(focalPoint);
20    box.position.set(C.x, C.y, C.z);
21    box.userData.abstractPlane = plane;
22    box.userData.height = h;
23    box.userData.width = w;
24    box.userData.distance = d;
25    box.userData.t = t;
26    box.userData.b = b;
27    box.name = "Plane" + planes.length;
28    // L'afegim a l'escena de forma oculta, i el desem al
29    // vector
30    box.visible = false;
31    scene.add(box);
32    planes.push(box);
33 }
```

Fragment 50: Generar i desar un pla

8.5.2 Mode Inspecció

El mode Inspecció està dirigit a usuaris que simplement volen visualitzar un conjunt d'imatges predefinit. Aquest mode conté la funcionalitat de seleccionar una figura de l'escena, i visualitzar les imatges al visor.

Per implementar la primera funcionalitat, haurem d'afegir les interaccions de selecció i *hover* a les figures generades. Aquests tipus d'interaccions són descrites amb més detalls al capítol 8.3.3. La principal modificació a fer, és canviar la funció que et retorna la primera imatge del *raycast*, per una funció que et torni la primera figura incidida pel raig. En contres de comprovar que el nom de l'objecte comenci per *Sant Quirze de Pedret by Zones*, comprovarem que comencin per *Sphere*, *Cylinder* o *Plane*. El nom dels objectes s'han afegit durant la generació de les figures,

tal com podem comprovar als tres fragments de codi anteriors. La funció que ens retorna la primera figura és la que es troba al Fragment 51.

```

1 function firstFigure(objects) {
2     for (let i = 0; i < objects.length; i++) {
3         const o = objects[i];
4         if (o.object.name.startsWith("Sphere") || o.object.name
5             .startsWith("Cylinder") || o.object.name.startsWith("Plane"))
6             return o.object;
7     }
8     return null;
9 }
```

Fragment 51: Obtenció de la primera figura incidida pel raig

El mode d'inspecció, al ser més senzill, ens permet eliminar el botó per obrir les imatges al visor. Simplement, les podrem obrir en fer clic sobre les figures. Una vegada obtenim la figura, només cal extreure els paràmetres d'aquestes, i cridar la funció que fa visualitzar les imatges al visor (vegeu Fragment 52). Aquestes funcions han estat descrites prèviament als capítols 8.4.3, 8.4.4 i 8.4.5.

```

1 // Object correspon a la figura incidida pel raig
2 if (object.name.startsWith("Sphere")) openSphericalImages(
3     object.position, object.userData.radius);
4 else if (object.name.startsWith("Cylinder"))
5     openCylindricalImages(object.userData.vector, object.
6     position, object.userData.radius, object.userData.height);
7 else if (object.name.startsWith("Plane")) openPlaneImages(
8     object.userData.abstractPlane, object.userData.height,
9     object.userData.width, object.userData.distance, object.
10    position, object.userData.t, object.userData.b);
```

Fragment 52: Obtenció de la primera figura incidida pel raig

8.5.3 Commutació entre modes

Per a canviar de mode, s'ha afegit un botó de commutació. En canviar de mode Autor a mode Inspecció, l'entorn amaga totes les imatges, elimina les figures creades, mostra les figures reduïdes desades, i estableix la interfície de mode Inspecció.

En canviar de mode Inspecció a mode Autor, és al revés, l'entorn amaga les figures generades, mostra les imatges i estableix la interfície de mode Autor.

Per mostrar i amagar les imatges i figures, simplement es fa una iteració que estableix la visibilitat d'aquestes (vegeu Fragments 53 i 54).

```

1 function setImageVisibility(show) {
2     images.forEach((i) => (i.visible = show));
3 }
```

Fragment 53: Establir visibilitat de les imatges

```

1 function setFiguresVisibility(show) {
2     if (spheres.length > 0) spheres.forEach((s) => (s.visible =
3         show));
```

```

3     if (cylinders.length > 0) cylinders.forEach((c) => (c.
4         visible = show));
5     if (planes.length > 0) planes.forEach((p) => (p.visible =
show));
5 }

```

Fragment 54: Establir visibilitat de les figures reduïdes generades

A continuació es mostren dues imatges de com és l'escena en mode Autor i mode Inspecció. A l'exemple mostrat, es crea una esfera en mode Autor que és desada (vegeu Figura 47). En el mode Inspecció, es mostra l'esfera generada corresponent (vegeu Figura 48).

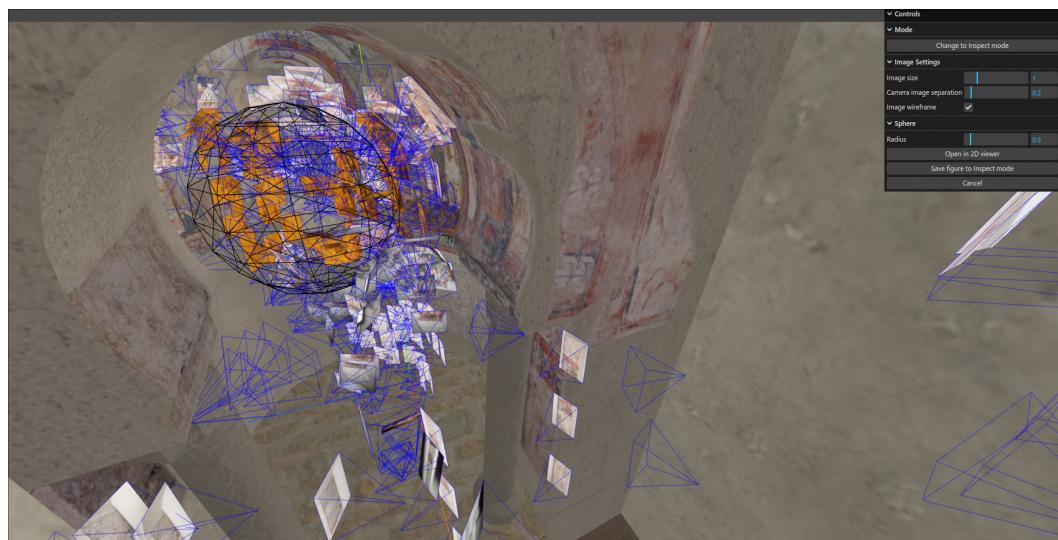


Figura 47: Escena en mode Autor
Producció pròpia

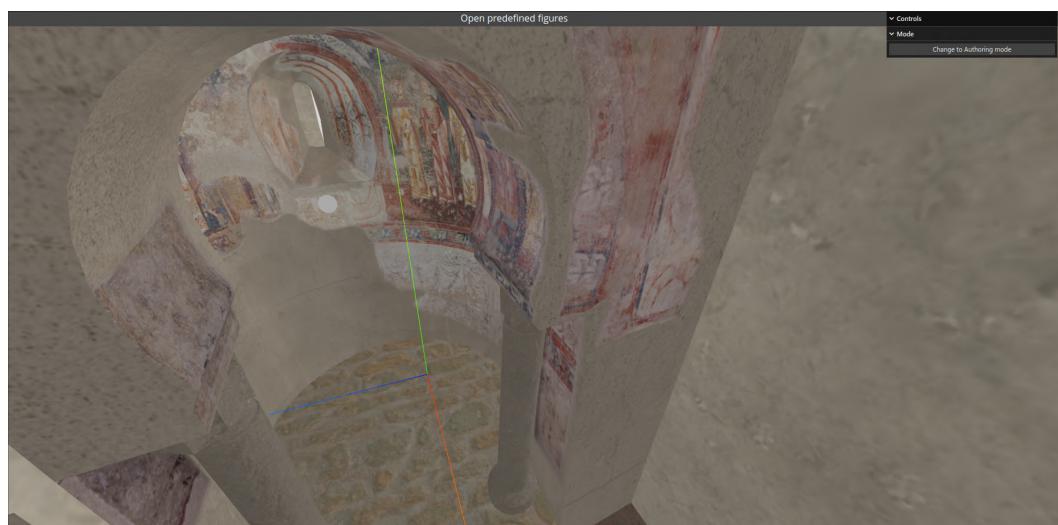


Figura 48: Escena en mode Inspecció
Producció pròpia

8.6 Interfície gràfica

Prèviament, en seccions com la 8.3 o la 8.4 s'ha esmentat l'ús d'una interfície gràfica per tal de dur a terme certes accions. En aquest capítol s'aprofundirà sobre

aquest panell utilitzat en aquestes seccions. A més a més, es descriurà l'ús i la implementació d'una barra d'informació. A la Figura 49 podeu visualitzar la interfície de forma general.



Figura 49: Interfície gràfica de l'entorn 3D
Producció pròpria

8.6.1 Panell lateral

L'element principal de la interfície gràfica és un panell que conté elements que permeten la interacció amb la plataforma. Aquests elements van des dels botons, als *sliders*, fins a *checkboxes*.

La implementació de la interfície s'ha dut a terme mitjançant la biblioteca de *JavaScript lil-gui* [49]. Aquesta ens proporciona un panell on li pots afegir els elements interactius de forma senzilla.

A continuació es descriuen cadascuna de les parts del panell.

Panell de mode

El *Panell de mode* conté l'opció de canviar del mode Autor al mode Inspecció, i al revés. Aquest panell sempre està habilitat, i per defecte mostra el botó *Change to Inspect mode*. A la Figura 50 es mostra una captura d'aquest panell, i una descripció de cadascun dels elements que el formen.

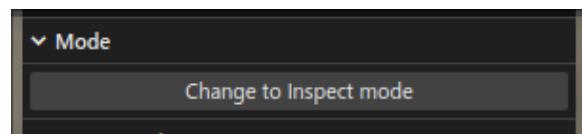


Figura 50: Panell de mode
Font: Producció pròpria

- **Change to Inspection mode.** Botó que canvia el mode Autor a Inspecció. Aquesta funció està definida al capítol 8.5.3. A més a més, oculta els panells del mode Autor, que inclou el *Panell d'imatges seleccionades*, *Panell Configuració d'imatges* i el *Panell de figures*. Finalment, substitueix aquest botó pel *Change to Authoring mode*.

- **Change to Authoring mode.** Botó que canvia el mode Inspecció a Autor. Aquesta funció està definida a la secció 8.5.3. A més a més, mostra els panells del mode Autor, que inclou el *Panell d'imatges seleccionades*, *Panell Configuració d'imatges* i el *Panell de figures*. Finalment, substitueix aquest botó pel *Change to Inspect mode*. Per defecte, aquest botó no es mostra.

El Fragment 55 mostra la configuració del panell.

```

1 const panel = new GUI({ width: 290 });
2
3 // Afegim el mode al panell principal
4 const folder0 = panel.addFolder("Mode");
5
6 let settings0 = {
7   "Change to Inspect mode": function () {
8     // Substitueix el boto
9     showController("Mode", "Change to Authoring mode");
10    hideController("Mode", "Change to Inspect mode");
11    // Oculta els panells de mode Autor
12    hideFolder("Individual Selection");
13    hideFolder("Image Settings");
14    hideFolder("Sphere");
15    hideFolder("Plane");
16    hideFolder("Cylinder");
17    // Establir mode Inspeccio
18    setAuthoringMode(false);
19  },
20  "Change to Authoring mode": function () {
21    // Substitueix el boto
22    hideController("Mode", "Change to Authoring mode");
23    showController("Mode", "Change to Inspect mode");
24    // Mostra els panells de mode Autor
25    showFolder("Image Settings");
26    showFolder("Sphere");
27    showFolder("Plane");
28    showFolder("Cylinder");
29    // Establir mode Autor
30    setAuthoringMode(true);
31  },
32};
33
34 // Afegim cada element a la seccio i la funcio a cridar
35 folder0.add(settings0, "Change to Inspect mode");
36 folder0.add(settings0, "Change to Authoring mode");
37 // Ocultem el boto
38 hideController("Mode", "Change to Authoring mode");

```

Fragment 55: Creació del panell de mode

Panell Configuració d'imatges

El *Panell Configuració d'imatges* conté les funcionalitats de modificació de mida d'una imatge, modificació de separació d'una imatge, i mostrar o amagar els fil-ferros, descrits al capítol 8.3. Aquest panell sempre està habilitat i visible. A la Figura 51 es mostra una captura d'aquest panell, i una descripció de cadascun dels

elements que el formen.

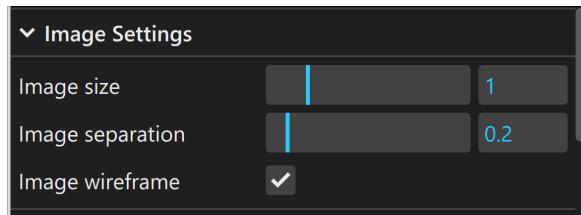


Figura 51: Panell de Configuració d’imatges
Font: Producció pròpria

- **Image size.** *Slider* que comprèn els valors d’entre 0 i 5, que pren un valor per defecte d’una unitat. Cada vegada que es modifica aquest valor, es crida a la funció *setSize* amb el nou valor com a paràmetre. Aquesta funció modifica la mida de les imatges, i s’ha descrit a la secció 8.3.1.
- **Image separation.** *Slider* que comprèn els valors d’entre 0 i 2, que pren un valor per defecte de 0.2 unitats. Cada vegada que es modifica aquest valor, es crida a la funció *setOffset* amb el nou valor com a paràmetre. Aquesta funció modifica la separació d’una imatge respecte a la seva posició, i s’ha descrit a la secció 8.3.2.
- **Image wireframe.** *Checkbox* que per defecte està habilitat. Cada vegada que es modifica aquest valor, es crida a la funció *setOffset* amb el nou valor com a paràmetre. Aquesta funció mostra o amaga els filferros de les imatges, i s’ha descrit a la secció 8.3.4.

El Fragment 56 mostra la configuració del panell.

```

1 const panel = new GUI({ width: 290 });
2
3 // Afegim la seccio al panell principal
4 const folder1 = panel.addFolder("Image Settings");
5
6 // Establim els valors predeterminats
7 let settings1 = {
8   "Image size": 1.0,
9   "Image separation": 0.2,
10  "Image wireframe": true,
11 };
12
13 // Afegim cada element a la seccio, definint els valors maxims
14 // i minims, i la funcio a cridar
14 folder1.add(settings1, "Image size", 0.0, 5.0, 0.01).onChange(
15   setSize);
15 folder1.add(settings1, "Image separation", 0.01, 2.0, 0.01).onChange(
16   setOffset);
16 folder1.add(settings1, "Image wireframe").onChange(setWireframe);

```

Fragment 56: Creació del panell Configuració d’imatges

Panell de figures

El *Panell de figures* conté els botons que permeten afegir les figures d'esfera, pla i cilindre descrits al capítol 8.4. Aquest panell està habilitat sempre que no hi hagi cap imatge seleccionada, ni cap figura a l'entorn. A la Figura 52 es mostra una captura d'aquest panell, i una descripció de cadascun dels elements que el formen.

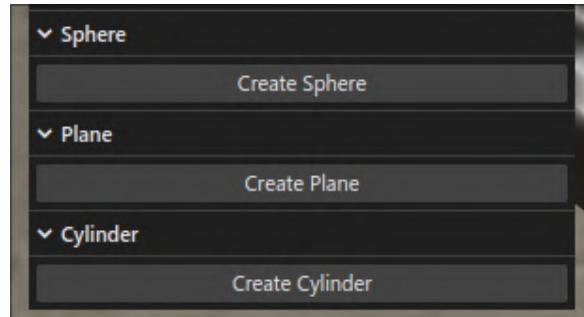


Figura 52: Panell de figures
Font: Producció pròpria

- **Create Sphere.** Botó que habilita la creació d'una esfera. Aquest indica a la lògica de l'aplicació, que en seleccionar una imatge, que creï l'esfera, tal com s'ha explicat a la secció 8.4.4. En seleccionar el botó, oculta aquest panell, i mostra un botó de cancel·lar. En prémer aquest últim, desactivarà l'ordre de creació d'una esfera, i tornarà a mostrar el panell.
- **Create Plane.** Botó que habilita la creació d'un pla. Aquest indica a la lògica de l'aplicació, que en seleccionar 3 imatges, que creï el pla, tal com s'ha explicat al capítol 8.4.3. En seleccionar el botó, oculta aquest panell, i mostra un botó de cancel·lar. En prémer aquest últim, desactivarà l'ordre de creació d'un pla, i tornarà a mostrar el panell.
- **Create Cylinder.** Botó que habilita la creació d'un cilindre. Aquest indica a la lògica de l'aplicació, que en seleccionar dues imatges, que creï el cilindre, tal com s'ha explicat a la secció 8.4.5. En seleccionar el botó, oculta aquest panell, i mostra un botó de cancel·lar. En prémer aquest últim, desactivarà l'ordre de creació d'un cilindre, i tornarà a mostrar el panell.

El Fragment 57 mostra la configuració del panell.

```
1 const panel = new GUI({ width: 290 });
2
3 // Cada boto es troba a la seva seccio
4 const folder3 = panel.addFolder("Sphere");
5 const folder4 = panel.addFolder("Plane");
6 const folder5 = panel.addFolder("Cylinder");
7
8 let settings3 = {
9     "Create Sphere": function () {
10         // Mostrar boto Cancel i ocultar panell figures
11         showController("Sphere", "Cancel");
12         hideController("Sphere", "Create Sphere");
13         hideFolder("Plane");
14         hideFolder("Cylinder");
```

```

15      // Indicar a la logica la creacio d'una esfera en
16      seleccionar una imatge
17      setSelectionMode("sphere");
18  },
19  Cancel: function () {
20      // Esborrar esfera en cas que estigui creada
21      cancelSphere();
22      // Ocultar boto Cancel i mostrar panell figures
23      hideController("Sphere", "Cancel");
24      showController("Sphere", "Create Sphere");
25      showFolder("Plane");
26      showFolder("Cylinder");
27      // Desactivar la creacio d'una esfera en seleccionar
28      una imatge
29      setSelectionMode("multi");
30  },
31 let settings4 = {
32  "Create Plane": function () {
33      // Mostrar boto Cancel i ocultar panell figures
34      showController("Plane", "Cancel");
35      hideController("Plane", "Create Plane");
36      hideFolder("Sphere");
37      hideFolder("Cylinder");
38      // Indicar a la logica la creacio d'un pla en
39      seleccionar 3 imatges
40      setSelectionMode("plane");
41  },
42  Cancel: function () {
43      // Esborrar pla en cas que estigui creat
44      cancelPlane();
45      // Ocultar boto Cancel i mostrar panell figures
46      hideController("Plane", "Cancel");
47      showController("Plane", "Create Plane");
48      showFolder("Sphere");
49      showFolder("Cylinder");
50      hideFolder("Individual Selection");
51      // Desactivar la creacio d'un pla en seleccionar 3
52      imatges
53      setSelectionMode("multi");
54  },
55 let settings5 = {
56  "Create Cylinder": function () {
57      // Mostrar boto Cancel i ocultar panell figures
58      showController("Cylinder", "Cancel");
59      hideController("Cylinder", "Create Cylinder");
60      hideFolder("Individual Selection");
61      hideFolder("Sphere");
62      hideFolder("Plane");
63      // Indicar a la logica la creacio d'un cilindre en
64      seleccionar 2 imatges
65      setSelectionMode("cylinder");
66  },
67  Cancel: function () {

```

```

67     // Esborrar cilindre en cas que estigui creat
68     cancelCylinder();
69     // Ocultar boto Cancel i mostrar panell figures
70     hideController("Cylinder", "Cancel");
71     showController("Cylinder", "Create Cylinder");
72     showFolder("Sphere");
73     showFolder("Plane");
74     hideFolder("Individual Selection");
75     // Desactivar la creacio d'un cilindre en seleccionar 2
76     // imatges
77     setSelectionMode("multi");
78 },
79
80 // Afegim cada element a la seva seccio corresponent
81 folder3.add(settings3, "Create Sphere");
82 folder3.add(settings3, "Cancel");
83 folder4.add(settings4, "Create Plane");
84 folder4.add(settings4, "Cancel");
85 folder5.add(settings5, "Create Cylinder");
86 folder5.add(settings5, "Cancel");

```

Fragment 57: Creació del panell de figures

Panell d'imatges seleccionades

El *Panell d'imatges seleccionades* conté l'opció de visualitzar imatges seleccionades al visor 2D, tal com s'ha descrit al capítol 8.4.2. Aquest panell s'habilita sempre que no hi hagi cap ordre de creació d'una figura, i que s'hagi seleccionat almenys una imatge. A la Figura 53 es mostra una captura d'aquest panell, i una descripció de cadascun dels elements que el formen.

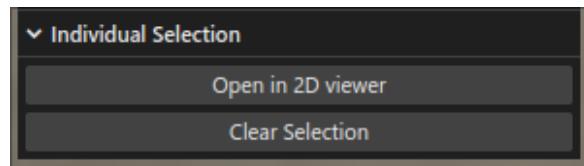


Figura 53: Panell d'imatges seleccionades
Font: Producció pròpria

- **Open in 2D viewer.** Botó que mostra les imatges seleccionades al visor 2D. La funció que fa visualitzar les imatges al visor 2D s'ha descrit a la secció 8.4.2. A més a més, en ser pres, s'esborra la selecció feta, i es mostra el panell de figures.
- **Clear Selection.** Botó que esborra la selecció d'imatges feta. Oculta aquest panell, i mostra el panell de figures.

El Fragment 58 mostra la configuració del panell.

```

1 const panel = new GUI({ width: 290 });
2
3 // Afegim la seccio al panell principal
4 const folder2 = panel.addFolder("Individual Selection");
5

```

```

6 let settings2 = {
7     "Clear Selection": function () {
8         // Deselecció les imatges
9         clearSelection();
10        // Desabilitem aquest panell, i habilitem el panell
11        figures
12            hideFolder("Individual Selection");
13            showFolder("Sphere");
14            showFolder("Plane");
15            showFolder("Cylinder");
16        },
17        "Open in 2D viewer": function () {
18            // Visualitzem les imatges seleccionades al visor 2D, i
19            // les deseleccionem
20            openImagesToOpenSeaDragon();
21            // Desabilitem aquest panell, i habilitem el panell
22            figures
23                hideFolder("Individual Selection");
24                showFolder("Sphere");
25                showFolder("Plane");
26                showFolder("Cylinder");
27        },
28    };
29
30 // Afegim cada element a la secció, definint els valors maxims
31 // i minims, i la funció a cridar
32 folder2.add(settings2, "Open in 2D viewer");
33 folder2.add(settings2, "Clear Selection");

```

Fragment 58: Creació del panell d'imatges seleccionades

Panell de l'esfera

El *Panell de l'esfera* conté les funcionalitats de modificació de la mida de la figura i l'opció de visualitzar les imatges al visor 2D, descrits a la secció 8.4.4. Aquest panell s'habilita després de seleccionar el botó *Create Sphere* i de seleccionar la imatge que conforma el seu centre. A la Figura 54 es mostra una captura d'aquest panell, i una descripció de cadascun dels elements que el formen.

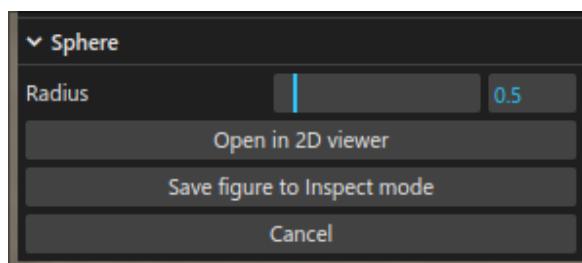


Figura 54: Panell de l'esfera

Font: Producció pròpia

- **Radius.** *Slider* que comprèn els valors d'entre 0 i 5, que pren un valor per defecte de 0,5 unitats. Cada vegada que es modifica aquest valor, es crida una funció amb el nou valor com a paràmetre. Aquesta funció modifica el radi de l'esfera, tal com s'ha descrit al capítol 8.4.4.

- **Open in 2D viewer.** Botó que mostra les imatges incloses a l'esfera al visor 2D. La funció que fa visualitzar les imatges al visor 2D s'ha descrit a la secció 8.4.4.
- **Save figure to Inspect mode.** Botó que genera i desa una figura pel mode Inspecció. Crida la funció descrita a la secció 8.5.1.
- **Clear Selection.** Botó que esborra l'esfera, també, oculta aquest panell, i mostra el panell de figures.

El Fragment 59 mostra la configuració del panell.

```

1 const panel = new GUI({ width: 290 });
2
3 // Afegim la seccio al panell principal
4 const folder3 = panel.addFolder("Sphere");
5
6 // Establim els valors predeterminats
7 let settings3 = {
8     "Open in 2D viewer": function () {
9         // Obrim les imatges al visor 2D
10        openSphericalImages();
11    },
12    // Definim el valor predeterminat del radi
13    Radius: 0.5,
14    "Save figure to Inspect mode": function () {
15        saveSphereToInspectMode();
16    },
17    Cancel: function () {
18        // Esborrem l'esfera creada
19        cancelSphere();
20        // Ocultem aquest panell, i mostrem el de figures
21        hideController("Sphere", "Cancel");
22        showController("Sphere", "Create Sphere");
23        hideController("Sphere", "Radius");
24        hideController("Sphere", "Open in 2D viewer");
25        hideController("Sphere", "Save figure to Inspect mode")
26        ;
27        showFolder("Plane");
28        showFolder("Cylinder");
29    },
30
31 // Afegim cada element a la seccio, definint els valors maxims
32 // i minims, i la funcio a cridar
33 folder3.add(settings3, "Radius", 0.0, 5.0, 0.01).onChange(
34     applySphericalRadius);
35 folder3.add(settings3, "Open in 2D viewer");
36 folder3.add(settings3, "Save figure to Inspect mode");
37 folder3.add(settings3, "Cancel");

```

Fragment 59: Creació del panell de l'esfera

Panell del pla

El *Panell del pla* conté les funcionalitats de modificació de la mida de la figura i l'opció de visualitzar les imatges al visor 2D, descrits al capítol 8.4.3. Aquest panell s'habilita després de seleccionar el botó *Create Plane* i de seleccionar les 3 imatges que el conformen. A la Figura 55 es mostra una captura d'aquest panell, i una descripció de cadascun dels elements que el formen.

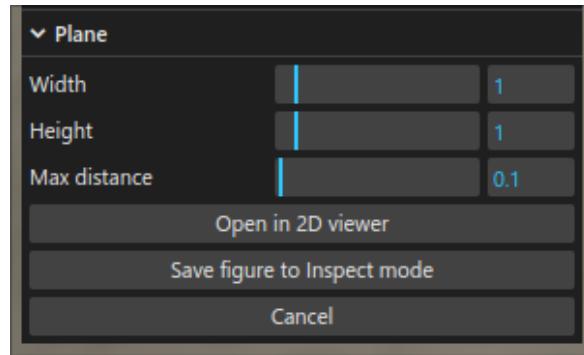


Figura 55: Panell del pla

Font: Producció pròpria

- **Width.** *Slider* que comprèn els valors d'entre 0 i 10, que pren un valor per defecte d'una unitat. Cada vegada que es modifica aquest valor, es crida una funció amb el nou valor com a paràmetre. Aquesta funció modifica l'amplada del pla, tal com s'ha descrit a la secció 8.4.3.
- **Height.** *Slider* que comprèn els valors d'entre 0 i 10, que pren un valor per defecte d'una unitat. Cada vegada que es modifica aquest valor, es crida una funció amb el nou valor com a paràmetre. Aquesta funció modifica l'alçada del pla, tal com s'ha descrit al capítol 8.4.3.
- **Max distance.** *Slider* que comprèn els valors d'entre 0 i 5, que pren un valor per defecte de 0,1 unitats. Cada vegada que es modifica aquest valor, es crida una funció amb el nou valor com a paràmetre. Aquesta funció modifica la profunditat del pla, tal com s'ha descrit a la secció 8.4.3.
- **Open in 2D viewer.** Botó que mostra les imatges incloses al pla al visor 2D. La funció que fa visualitzar les imatges al visor 2D s'ha descrit a la secció 8.4.3.
- **Save figure to Inspect mode.** Botó que genera i desa una figura pel mode Inspecció. Crida la funció descrita al capítol 8.5.1.
- **Clear Selection.** Botó que esborra el pla, també, oculta aquest panell, i mostra el panell de figures.

El Fragment 60 mostra la configuració del panell.

```
1 const panel = new GUI({ width: 290 });
2
3 // Afegim la seccio al panell principal
4 const folder4 = panel.addFolder("Plane");
5
6 // Establim els valors predeterminats
```

```

7 let settings3 = {
8     "Open in 2D viewer": function () {
9         // Obrim les imatges al visor 2D
10        openPlane();
11    },
12    // Definim els valor predeterminats de la mida del pla
13    Width: 1,
14    Height: 1,
15    "Max distance": 0.1,
16    "Save figure to Inspect mode": function () {
17        saveSphereToInspectMode();
18    },
19    Cancel: function () {
20        // Esborrem el pla creat
21        cancelPlane();
22        // Ocultem aquest panell, i mostrem el de figures
23        hideController("Plane", "Width");
24        hideController("Plane", "Height");
25        hideController("Plane", "Max distance");
26        hideController("Plane", "Open in 2D viewer");
27        hideController("Plane", "Save figure to Inspect mode");
28        hideController("Plane", "Cancel");
29        showController("Plane", "Create Plane");
30        showFolder("Sphere");
31        showFolder("Cylinder");
32    },
33 };
34
35 // Afegim cada element a la secció, definint els valors maxims
36 // i minims, i la funció a cridar
36 folder4.add(settings4, "Width", 0.0, 10.0, 0.01).onChange(
37     changePlaneWidth);
37 folder4.add(settings4, "Height", 0.0, 10.0, 0.01).onChange(
38     changePlaneHeight);
38 folder4.add(settings4, "Max distance", 0.0, 5.0, 0.01).onChange(
39     changePlaneDistance);
39 folder4.add(settings4, "Open in 2D viewer");
40 folder4.add(settings4, "Save figure to Inspect mode");
41 folder4.add(settings4, "Cancel");

```

Fragment 60: Creació del panell del pla

Panell del cilindre

El *Panell del cilindre* conté les funcionalitats de modificació de la mida de la figura i l'opció de visualitzar les imatges al visor 2D, descrits a la secció 8.4.5. Aquest panell s'habilita després de seleccionar el botó *Create Cylinder* i de seleccionar les 2 imatges que el conformen. A la Figura 56 es mostra una captura d'aquest panell, i una descripció de cadascun dels elements que el formen.

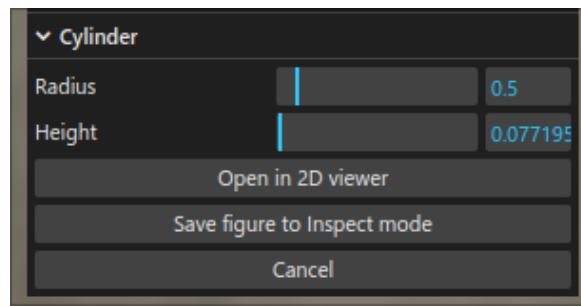


Figura 56: Panell del cilindre
Font: Producció pròpria

- **Radius.** *Slider* que comprèn els valors d'entre 0 i 5, que pren un valor per defecte de 0,5 unitats. Cada vegada que es modifica aquest valor, es crida una funció amb el nou valor com a paràmetre. Aquesta funció modifica el radi del cilindre, tal com s'ha descrit al capítol 8.4.5.
- **Height.** *Slider* que comprèn els valors d'entre 0 i 5, que pren un valor per defecte d'una unitat. Cada vegada que es modifica aquest valor, es crida una funció amb el nou valor com a paràmetre. Aquesta funció modifica l'alçada del cilindre, tal com s'ha descrit a la secció 8.4.5.
- **Open in 2D viewer.** Botó que mostra les imatges incloses al pla al visor 2D. La funció que fa visualitzar les imatges al visor 2D s'ha descrit a la secció 8.4.5.
- **Save figure to Inspect mode.** Botó que genera i desa una figura pel mode Inspecció. Crida la funció descrita a la secció 8.5.1.
- **Clear Selection.** Botó que esborra el cilindre, també, oculta aquest panell, i mostra el panell de figures.

El Fragment 61 mostra la configuració del panell.

```

1 const panel = new GUI({ width: 290 });
2
3 // Afegim la seccio al panell principal
4 const folder5 = panel.addFolder("Cylinder");
5
6 // Establim els valors predeterminats
7 let settings3 = {
8     "Open in 2D viewer": function () {
9         // Obrim les imatges al visor 2D
10        openCylindricalImages();
11    },
12    // Definim els valor predeterminats de la mida del
13    cilindre
14    Radius: 0.5,
15    Height: 1.0,
16    "Save figure to Inspect mode": function () {
17        saveSphereToInspectMode();
18    },
19    Cancel: function () {
20        // Esborrem el pla creat
21        cancelCylinder();
22    }
23}
```

```

21     // Ocultem aquest panell, i mostrem el de figures
22     hideController("Cylinder", "Radius");
23     hideController("Cylinder", "Height");
24     hideController("Cylinder", "Open in 2D viewer");
25     hideController("Cylinder", "Save figure to Inspect mode
26     ");
27     hideController("Cylinder", "Cancel");
28     showController("Cylinder", "Create Cylinder");
29     showFolder("Sphere");
30     showFolder("Plane");
31 },
32
33 // Afegim cada element a la seccio, definint els valors maxims
34 i minims, i la funcio a cridar
35 folder5.add(settings5, "Radius", 0.0, 5.0, 0.01).onChange(
36     applyCylindricalRadius);
37 folder5.add(settings5, "Height", 0.0, 5.0, 0.01).onChange(
38     applyCylindricalHeight);
39 folder5.add(settings5, "Open in 2D viewer");
40 folder5.add(settings5, "Save figure to Inspect mode");
41 folder5.add(settings5, "Cancel");

```

Fragment 61: Creació del panell del pla

8.6.2 Barra d'informació

La *Barra d'informació* és la part de la interfície gràfica que té com a objectiu ajudar a l'usuari final amb les opcions disponibles. Aquest tracta d'una barra que mostra per pantalla un text. Algun cas d'ús és mostrar el nombre d'imatges a seleccionar per a la creació d'una certa figura.

Aquesta barra s'ha creat utilitzant *HTML* per a la seva estructura i *CSS* per a l'estil (vegeu Fragments 62 i 63).

```

1 <div id="info">
2     Open an image with left click, select images with right
3     click or create a figure
4 </div>

```

Fragment 62: HTML per a la barra d'informació

```

1 #info {
2     text-align: center;
3     height: 25px;
4     background-color: rgb(70, 70, 70);
5     color: aliceblue;
6     font-family: 'Open Sans', sans-serif;
7 }

```

Fragment 63: CSS per a la barra d'informació

La implementació de la barra d'informació d'aquesta manera té un problema. Per defecte, el visor de l'entorn 3D estableix la seva mida amb la de la finestra, és a dir que ocupa tota la finestra. En afegir un element addicional, el visor mantindrà la seva mida, fent que tot el contingut de la finestra sigui més alt que la finestra. Això

fa que aparegui una barra de desplaçament que empitjora la usabilitat de l'aplicació. Per arreglar-ho, s'ha d'escalar el visor 3D de manera que sigui 25 pixels més baix, el nombre de pixels verticals que ocupa la barra d'informació. A part, s'ha de canviar la perspectiva de la càmera amb la nova mida del visor, i indicar que cada vegada que es redimensioni la finestra, que redueixi aquests 25 pixels. El Fragment 64 mostra el procediment.

```
1 // Actualitzar la mida durant la inicialitzacio
2 camera.aspect = window.innerWidth / (window.innerHeight - 25);
3 renderer.setSize(window.innerWidth, window.innerHeight - 25);
4
5 // Actualitzar la mida en fer resize de la finestra
6 function onWindowResize() {
7     camera.aspect = window.innerWidth / (window.innerHeight -
25);
8     camera.updateProjectionMatrix();
9     renderer.setSize(window.innerWidth, window.innerHeight -
25);
10    render();
11 }
```

Fragment 64: Actualitzar mida del visor i perspectiva de la càmera

Per a modificar el text que mostra la barra, només cal accedir a l'element, i canviar el text (vegeu Fragment 65).

```
1 const infoElement = document.getElementById("info");
2 infoElement.innerText = message;
```

Fragment 65: Establir text a la barra d'informació

9 Disseny i implementació de l'entorn 2D

Aquest capítol recull la implementació i el disseny de l'entorn 2D. Tal com s'ha explicat a la secció 2.2.2, la implementació de l'entorn 2D es durà a terme utilitzant el framework de *JavaScript OpenSeaDragon* [10]. Aquest té implementats mètodes que facilita la visualització d'imatges d'alta resolució.

9.1 Càrrega d'imatges

A continuació es descriurà el procés de visualització de les imatges. Començarem amb la càrrega d'una sola imatge, i més endavant, amb múltiples imatges.

9.1.1 Visualització d'una imatge

Per començar, crearem un nou fitxer d'*HTML*, i referenciarem el framework d'*OpenSeaDragon* [10]. Una vegada creat, li indicarem que mostri una imatge qualsevol al visor 2D, tal com indica la documentació [74]. El Fragment 66 importa el visor, defineix la seva mida, i referencia un fitxer de *JavaScript* que defineix el comportament d'aquest.

```
1 <div id="openseadragon1" style="width: 100vw"></div>
2 <script src="openseadragon/openseadragon.min.js"></script>
3 <script src="openseadragon.js"></script>
```

Fragment 66: Fitxer HTML que defineix el visor 2D

El fitxer *openseadragon.js* conté la ruta de la imatge a visualitzar(vegeu Fragment 67).

```
1 var viewer = OpenSeadragon({
2   id: "openseadragon1",
3   prefixUrl: "openseadragon/images/",
4   tileSources: [
5     {
6       type: "image",
7       url: "images/Sant Quirze de Pedret by Zones/MNAC - South Apse/ga210623_s-quinze-de-pedret_0078.jpg",
8     },
9   ],
10});
```

Fragment 67: Definició d'una imatge a OpenSeadragon

Com veureu a la Figura 57, el resultat d'aquest pas és un visor simple que mostra una interfície amb els botons bàsics de navegació, i inclou les accions habituals com moure's per la imatge i fer zoom mitjançant el ratolí.

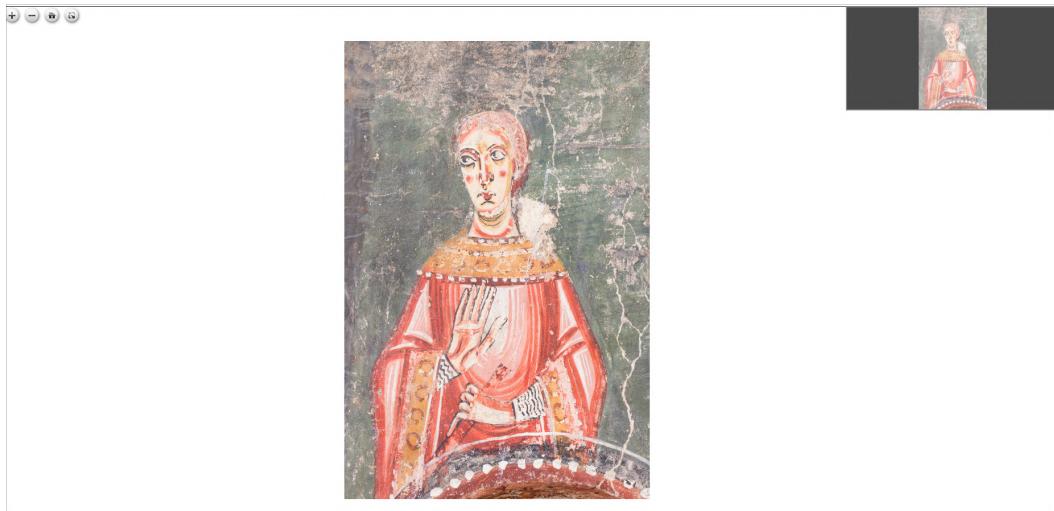


Figura 57: Visor 2D amb una imatge

Font: Producció pròpia

La següent tasca és iniciar el visor des de l'entorn 3D amb la imatge desitjada. Tal com s'ha descrit a la secció *Selecció d'imatges* de la 8.4.1, l'objectiu és obrir al visor una imatge utilitzant el nom de l'objecte com a ruta d'imatge.

El primer mètode que se'm va ocórrer va ser que l'entorn 3D obrís una nova pestanya indicant la ruta del fitxer *HTML*. Per indicar el nom de la imatge a visualitzar vaig optar per indicar-la com a paràmetre d'URL. Per exemple, per visualitzar la imatge anterior faríem ús de l'enllaç del Fragment 68.

```
1 http://localhost:5173/openseadragon.html?image=Sant Quirze de
  Pedret by Zones/MNAC - South Apse/ga210623_s-quinze-de-
  pedret_0078.jpg
```

Fragment 68: Enllaç per a visualitzar una imatge al visor

Realment l'enllaç anterior no seria vàlid, ja que conté caràcters invàlids com l'espai. L'espai té la codificació `%20`. L'enllaç vàlid, sense caràcters, una vegada convertit seria el que es mostra a la Figura 69.

```
1 http://localhost:5173/openseadragon.html?mode=single&image=Sant
  %20Quirze%20de%20Pedret%20by%20Zones%2FMNAC%20-%20South%20
  Apse%2Fga210623_s-quinze-de-pedret_0078.jpg
```

Fragment 69: Enllaç codificat

El Fragment 70 conté el codi per crear i obrir l'enllaç a partir d'un nom d'imatge obtingut a l'entorn 3D.

```
1 const url = "openseadragon.html?mode=single&image=" +
  encodeURIComponent(object.name);
2 window.open(url, "_blank"); // Obrim l'url a una nova pestanya
```

Fragment 70: Crear i obrir l'enllaç

L'últim pas a fer és indicar al visor que la imatge a visualitzar és l'especificada als paràmetres de l'enllaç. Obtenir un paràmetre d'URL des de *JavaScript* és molt

senzill en utilitzar *URLSearchParams*. El Fragment 71 conté el codi per visualitzar una imatge indicada a l'enllaç.

```
1 const urlParams = new URLSearchParams(window.location.search);
2 const image_name = urlParams.get('image');
3
4 var viewer = OpenSeadragon({
5   id: "openseadragon1",
6   prefixUrl: "openseadragon/images/",
7   tileSources: [
8     {
9       type: "image",
10      url: "images/" + image_name,
11    },
12  ],
13});
```

Fragment 71: Visualitzar una imatge especificada a l'enllaç

9.1.2 Visualització de múltiples imatges

Per tal de visualitzar més imatges al nostre visor podem seguir fent ús dels paràmetres de l'adreça web. En aquest cas, ens caldria codificar el nom d'imatges com a llista o conjunt d'imatges. Se'm va acudir codificar un objecte *JSON* com a cadena de caràcters, i establir-ho com a paràmetre d'URL. D'aquesta manera podia enviar un nombre indefinit d'imatges al visor mitjançant l'enllaç. *JavaScript* ja proporciona mètodes per crear un objecte *JSON* i codificar-lo a l'enllaç com a paràmetre (vegeu Fragment 72).

```
1 const url = 'openseadragon.html?images=' + encodeURIComponent(
2   JSON.stringify(Array.from(imagesSelected)));
3 window.open(url, '_blank')
```

Fragment 72: Construcció d'un enllaç amb múltiples imatges

Cal indicar que la variable *imagesSelected* és un conjunt de noms d'imatges obtinuts de les diferents seccions del capítol 8.4.

Per tal d'afegir les imatges al visor, crearem un objecte *sources*, que contindrà el llistat de rutes d'imatges. Aquest objecte el llegirà el visor per tal de poder mostrar-les (vegeu Fragment 73).

```
1 const urlParams = new URLSearchParams(window.location.search);
2 const images = urlParams.get('images');
3
4 var sources = [];
5 images.forEach((i) => {
6   sources.push({
7     type: "image",
8     url: "images/" + image_name,
9   });
10 });
11
12 var viewer = OpenSeadragon({
13   id: "openseadragon1",
```

```

14     prefixUrl: "openseadragon/images/",
15     tileSources: sources
16 });

```

Fragment 73: Visualitzar imatges especificades a l'enllaç

El resultat que es mostra a la Figura 58, les imatges estan sobreposades, ja que encara no se li han assignat una posició.

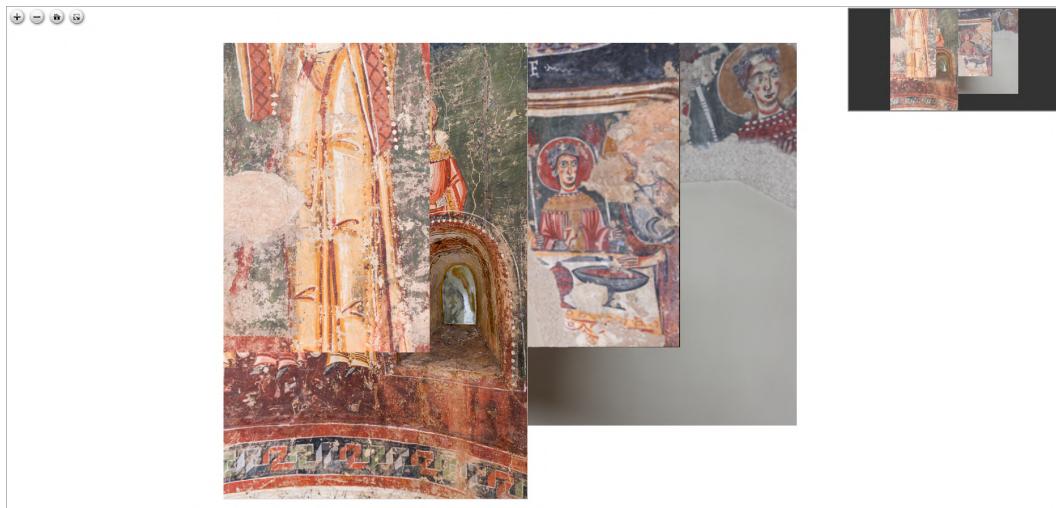


Figura 58: Visor 2D amb diverses imatges
Font: Producció pròpia

Aquest mètode té un problema, i és greu. Els navegadors suporten un nombre determinat de caràcters que ha de tenir l'adreça. Cada navegador suporta un nombre diferent de caràcters màxims, però en el cas de Microsoft Edge, el número màxim és de 2083 caràcters [75]. És fàcil veure, que en voler visualitzar un cert nombre d'imatges, la longitud del nostre enllaç superarà els 2083 caràcters. Per tal que el nostre visor sigui compatible amb tots els navegadors, haurem d'utilitzar un altre mètode per a enviar els noms d'imatges al visor.

El mètode triat per enviar els noms de les imatges al visor és utilitzant *LocalStorage*. *LocalStorage* [76] és un mecanisme de la interfície *window* que permet emmagatzemar dades de la sessió. Aquest mecanisme permet a pàgines del mateix domini a llegir i escriure informació en format clau-valor. El mecanisme és molt similar a *SessionStorage* [77], però aquest últim neteja l'emmagatzematge una vegada la sessió finalitza. Podem fer ús del *LocalStorage* [76], ja que l'entorn 3D i el visor d'imatges comparteixen el mateix domini.

L'entorn 3D escriurà el fitxer *JSON* al *LocalStorage* amb la clau *images*. Tot seguit, obrirà l'enllaç especificat anteriorment però sense paràmetres. El Fragment 74 conté aquesta implementació descrita.

```

1 let jsonContent = JSON.stringify(createJSON(imagesSelected));
2 localStorage.setItem("images", jsonContent); // Afegim l'
    objecte JSON al localStorage
3 const url = "openseadragon.html";
4 window.open(url, "_blank");

```

Fragment 74: Escriptura a localstorage

Per tal d'obtenir el conjunt de noms d'imatge al visor, llegim l'element *images* de *localStorage* (vegeu Fragment 75).

```

1 const retrievedObject = localStorage.getItem("images");
2 const images = JSON.parse(retrievedObject);
3
4 var sources = [];
5 images.forEach((i) => {
6     sources.push({
7         type: "image",
8         url: "images/" + image_name,
9     });
10 });
11
12 var viewer = OpenSeadragon({
13     id: "openseadragon1",
14     prefixUrl: "openseadragon/images/",
15     tileSources: sources
16 });

```

Fragment 75: Visualitzar imatges de localStorage

Amb aquesta implementació ja podem obrir més imatges al visualitzador, però, si intentem visualitzar un nombre més alt d'imatges, el navegador quedarà bloquejat. El visor estarà intentant mostrar un cert nombre d'imatges d'alta resolució (4160x6240 píxels), i no pot suportar la càrrega que comporta. Per solucionar-ho, farem ús de les *Deep Zoom Images*.

9.2 Deep Zoom Images

Les imatges *Deep Zoom* [78] és una tecnologia creada per *Microsoft* que permet visualitzar i transmetre imatges de manera eficient. Aquesta tracta de dividir la imatge en diverses cel·les o *tiles* amb diferents nivells de detall [79], i només carregar les necessàries. Per exemple, quan visualitzem la imatge des de lluny, només es carregarà les textures amb el nivell de detall més baix, i a mesura que ens acostem, el visor només carregarà les cel·les que entrin dins de la càmera, i cada vegada amb més detall.

Per tal de poder fer la conversió d'imatges *JPG* a *DZI* vaig optar per revisar el llistat d'eines recomanades [80] per *OpenSeaDragon*, i em vaig decantar per *MagickSlicer* [81], ja que només necessitava executar un script en *Bash*. *MagickSlicer* és de codi obert, i el podeu consultar al seu repositori [81]. Aquesta eina pren com a paràmetre la imatge a convertir. Per a automatitzar-ho, vaig fer un petit script similar a *Fragment 8: Reducció de mida d'imatges*, per tal que llegís totes les imatges d'un directori, i les convertís en format *DZI*. Aquest petit script que crida al convertidor d'imatges (vegeu Fragment 76).

```

1 #!/bin/bash
2 for f in `find . -name "*.jpg"`
3 do
4     ./magick-slicer.sh $f
5 done

```

```

6
7 for f in `find . -name "*.JPG"`
8 do
9     ./magick-slicer.sh $f
10 done

```

Fragment 76: Conversió d’imatges de JGP a DZI amb MagickSlicer

Finalment, cal indicar a *OpenSeaDragon* que utilitzés la imatge *DZI* en comptes de la *JPG*. Per fer-ho, cal aplicar 2 canvis al nostre codi. El primer és canviar l’extensió de la imatge a llegir, que es pot fer de forma senzilla amb operacions de cadenes de caràcters. La segona és indicar-li que el visor llegeix no és una imatge, és un fitxer *DZI*. Això s’indica substituint els paràmetres *type* i *url* per *tileSource*, on indicarem la ruta del fitxer *DZI* (vegeu Fragment 77).

```

1 const retrievedObject = localStorage.getItem("images");
2 const images = JSON.parse(retrievedObject);
3
4 var sources = [];
5 images.forEach((i) => {
6     var imageName = image.name.substr(0, image.name.lastIndexOf(
7         ".") + ".dzi");
8     sources.push({
9         titleSource: "images/" + imageName
10    });
11 });
12 var viewer = OpenSeadragon({
13     id: "openseadragon1",
14     prefixUrl: "openseadragon/images/",
15     tileSources: sources
16 });

```

Fragment 77: Visualitzar imatges en format DZI

A l’hora d’obrir aquestes imatges *DZI*, la majoria es visualitzaven correctament, però en algun cas, les cel·les estaven mal formades, i la imatge no es veia bé. A la Figura 59 podeu veure com es visualitzava una d’aquestes imatges mal formades.

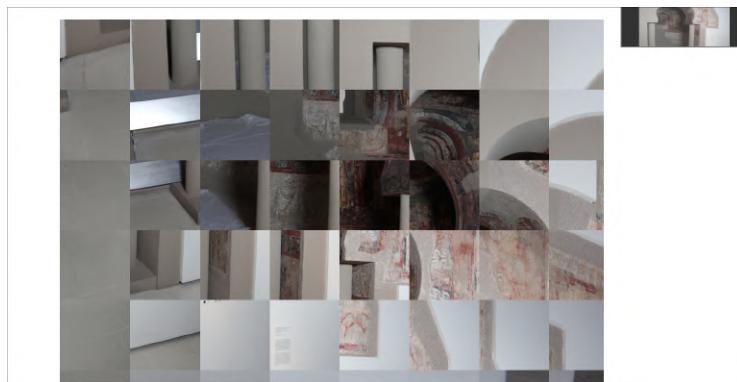


Figura 59: Mala conversió d’una imatge DZI
Font: Producció pròpia

La primera idea que vaig tenir per solucionar-ho, és revisar el llistat d’eines recomanades per [80] per *OpenSeaDragon*, i seleccionar-ne una altra. L’eina escollida

va ser *deepzoom.py* [82], un mòdul de *Python* amb la mateixa funcionalitat que *MagickSlicer* [81]. Vaig fer un script (vegeu Fragment 78) en *Python* que llegís totes les imatges del directori on es troba el fitxer, i les convertís a *DZI* utilitzant els paràmetres de l'exemple.

```

1 import os
2 import deepzoom
3
4 # Definim els paràmetres
5 creator = deepzoom.ImageCreator(
6     tile_size=128,
7     tile_overlap=2,
8     tile_format="jpg",
9     image_quality=1,
10    resize_filter="bicubic",
11)
12 # Obtenim tots els fitxers del directori
13 all_files = os.listdir()
14 for image_name in all_files:
15     # Comprovem que el fitxer sigui una imatge jpg
16     extension = os.path.splitext(image_name)[1]
17     if extension != 'jpg' and extension != 'JPG':
18         continue
19     # Obtenim el fitxer de sortida canviant l'extensió
20     output = os.path.splitext(image_name)[0] + '.dzi'
21     # Generem el DZI
22     creator.create(image_name, output)
```

Fragment 78: Conversió d'imatges de *JGP* a *DZI* amb *deepzoom.py*

Després d'aplicar la conversió a totes les imatges, les imatges que ja es veien bé abans segueixen visualitzant-se de la mateixa manera. Però, les imatges que abans estaven mal formades, ara es veien girades 90 graus, tal com podeu veure a la Figura 60.



Figura 60: Imatge *DZI* girada
Font: Producció pròpia

Després d'estar investigant possibles causes, vam trobar que a les imatges *JPG* contenen unes metadades *EXIF* [83] on es defineix la rotació de la imatge. Les imatges que es visualitzen correctament tenien una rotació definida a les metadades

de 0 graus. D'altra banda, les altres imatges tenien una rotació de 90 graus, que els conversors de *DZI* no llegien. Aquests conversors agafaven directament la imatge sense llegir les metadades, fent que no apliquessin aquesta rotació. En ser poques les imatges que tenien aquesta rotació, vaig optar per girar-les adequadament abans de fer la conversió. El resultat després d'aplicar aquesta rotació és la que trobem a la Figura 61.



Figura 61: Imatge DZI ben formada
Font: Producció pròpria

9.3 Posicionament i mida d'imatges

En aquest capítol es descriu el disseny i implementació de l'assignació de les posicions de les imatges. A més a més, la secció tracta el càlcul de les mides d'aquestes.

9.3.1 Posicionament d'imatges

La següent tasca a fer és el posicionament de les imatges al visor. Les posicions en dues coordenades ja han sigut calculades al capítol 8.4. Aquest, les col·locava a l'objecte que s'escrivia al *localStorage*, com a un paràmetre més. De la mateixa forma que obtenim el nom de les imatges, obtindrem les seves posicions accedint a les components *x* i *y*. Tal com descriu la documentació [84], podem especificar les components de les posicions de cada *source* assignant un valor al paràmetre *x* i *y*. Utilitzant el Fragment 79 obtenim la distribució de les imatges al visor.

```

1 const retrievedObject = localStorage.getItem("images");
2 const images = JSON.parse(retrievedObject);
3
4 var sources = [];
5 images.forEach((i) => {
6     var imageName = image.name.substr(0, image.name.lastIndexOf
7         (".")) + ".dzi";
8     sources.push({
9         titleSource: "images/" + imageName
10        x: image.x,
11        y: image.y,
12    });
13 });
14 var viewer = OpenSeadragon({

```

```

15     id: "openseadragon1",
16     prefixUrl: "openseadragon/images/",
17     tileSources: sources
18 });

```

Fragment 79: Assignació de posicions a OpenSeaDragon

La Figura 62 mostra el resultat de distribuir diverses imatges aplicant el Fragment 79.

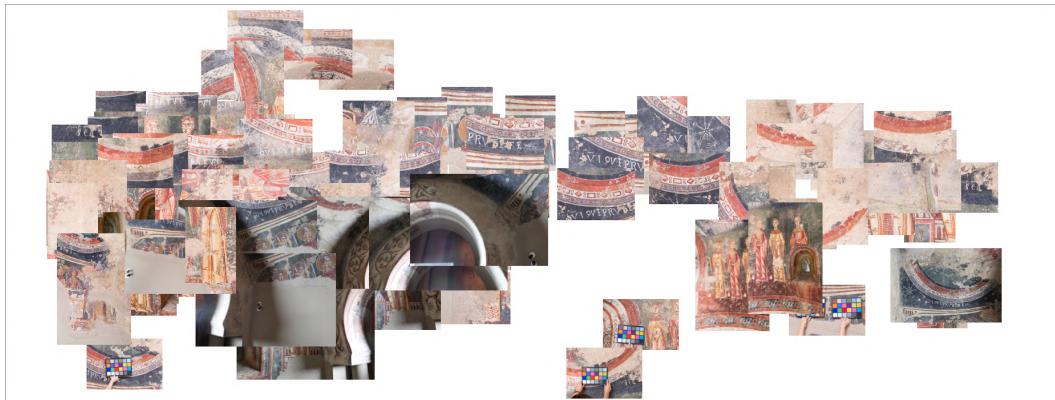


Figura 62: Imatges amb posicions assignades
Font: Producció pròpia

9.3.2 Mida d'imatges

Sense especificar la mida de les imatges, el visor les escala automàticament segons el nombre de píxels de cadascuna. És a dir, imatges de més resolució es visualitzaran amb una mida major. L'objectiu és que la mida de les imatges només depengui del factor de zoom de cadascuna. Per fer-ho, primer hem d'escalar les imatges de forma que totes tinguin la mateixa mida. Tal com s'exposa a la documentació d'*OpenSeaDragon* [84], podem definir l'amplada i alçada de cada imatge mitjançant els paràmetres *height* i *width*. En el cas que només assignem una de les components, les imatges s'escalarà d'acord amb la relació entre l'amplada i l'alçada.

Una de les maneres seria assignar a una de les components una mida d'una unitat. Això funcionaria en el cas que totes les imatges tinguessin la mateixa orientació. En tenir imatges verticals i horizontals, podem assignar aquesta unitat a la component més gran. En el cas que la imatge sigui vertical, aquesta tindrà una alçada d'una unitat, i en el cas que sigui horitzontal, aquesta tindrà una amplada d'una unitat. Per facilitar la implementació de característiques que es veurà més endavant com *Evitant l'encaixallament d'imatges* de la secció 9.3.4, és convenient assignar a totes les imatges només l'alçada o només l'amplada. D'aquesta manera, assignaré l'alçada d'una unitat a les imatges verticals, i una alçada per determinar a les imatges horitzontals. Aquesta alçada a determinar ha d'escalar la imatge de forma que la seva amplada prengui un valor d'una unitat. Aquest factor ja va ser calculat anteriorment a l'entorn 3D, i només cal obtenir-lo del *localStorage*, ja que s'escriu durant l'enviament de dades. A més a més, l'objecte que s'escriu conté el paràmetre *isLandscape*, que ens indica si la imatge és horitzontal. El Fragment 80 correspon a l'assignació de la mida juntament amb la de les posicions.

```

1 const retrievedObject = localStorage.getItem("images");
2 const images = JSON.parse(retrievedObject);
3
4 var sources = [];
5 images.forEach((i) => {
6     var imageName = image.name.substr(0, image.name.lastIndexOf
7     (".")) + ".dzi";
8     sources.push({
9         titleSource: "images/" + imageName
10        x: image.x,
11        y: image.y,
12        height: getHeight(image),
13    });
14 });
15 var viewer = OpenSeadragon({
16     id: "openseadragon1",
17     prefixUrl: "openseadragon/images/",
18     tileSources: sources
19 });
20
21 function getHeight(a) {
22     if (a.isLandscape) return a.heightToWidthRatio;
23     return 1;
24 };

```

Fragment 80: Assignació de posicions i mida a OpenSeaDragon

La Figura 63 conté les imatges escalades utilitzant el codi anterior, totes amb la mateixa mida.



Figura 63: Imatges amb la mateixa mida
Font: Producció pròpria

Una vegada tenim totes les imatges amb la mateixa escala, les podem escalar individualment segons el seu zoom calculat al capítol 8.4. Podem escalar les imatges de dues formes diferents. El primer és mostrar les imatges de manera que les imatges amb detalls siguin petites, i les que englobin l'escena siguin més grans. És a dir, que un mateix element tindrà aproximadament la mateixa mida independentment si s'ha fet amb molt zoom o amb molt poc. El segon mètode és invertir aquest escalat. Aquest cas ens permetrà veure les imatges amb més zoom (les imatges que contenen els detalls) de forma més gran.

9.3.2.1 Escalat segons el zoom

L'escalat segons el zoom té com a objectiu mostrar les imatges de manera realista. Un cert element de l'escena hauria de tenir sempre aproximadament la mateixa mida. Això s'aconsegueix fent que a més zoom tingui la imatge, més petita es veurà. Les imatges que mostren els detalls tindran una mida inferior que les que mostren l'escena en global. Per calcular la mida de les imatges dividirem la mida calculada anteriorment entre el factor de zoom. Modificarem la funció *getHeight* que hem implementat anteriorment aplicant aquests canvis (vegeu Fragment 81).

```
1 function getHeight(a) {  
2     if (a.isLandscape) return a.heightToWidthRatio / a.zoom;  
3     return 1 / a.zoom;  
4 };
```

Fragment 81: Càcul d'alçada segons el factor de zoom

La Figura 64 conté el resultat obtingut.

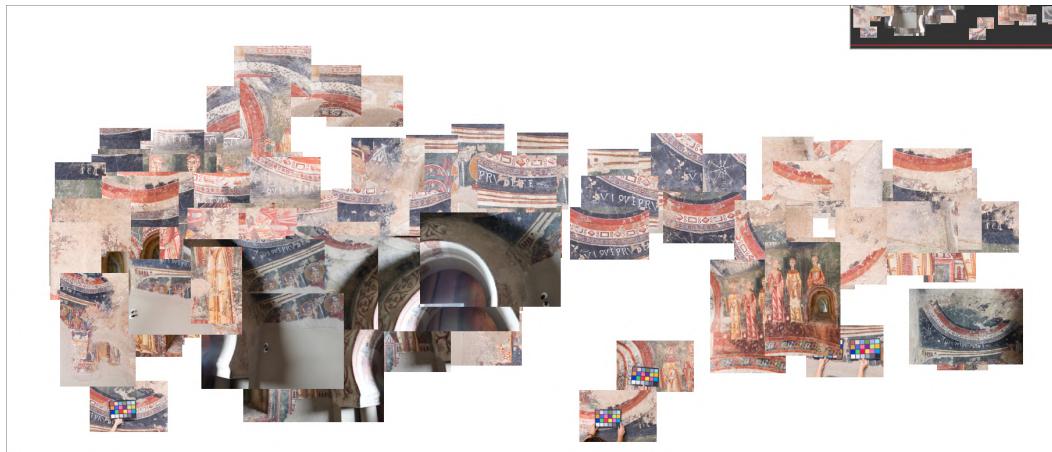


Figura 64: Imatges amb la mida segons el zoom

Font: Producció pròpria

9.3.2.2 Augmentar detalls

El segons mètode d'escalat té com a objectiu mostrar les imatges de manera que les que contenen els detalls es mostrin amb una mida major. Aquest mètode pot ser interessant en el cas que un historiador estigui interessat en visualitzar-los. Un cert element de l'escena hauria de tenir sempre aproximadament la mateixa mida. Això s'aconsegueix fent que a més zoom tingui la imatge, més gran es veurà. Les imatges que mostren els detalls tindran una mida superior que les que mostren l'escena en global. Per calcular la mida de les imatges multiplicarem la mida unitària calculada anteriorment amb el factor de zoom. Modificarem la funció *getHeight* que hem implementat anteriorment aplicant aquests canvis (vegeu Fragment 82).

```
1 function getHeight(a) {  
2     if (a.isLandscape) return a.heightToWidthRatio * a.zoom;  
3     return a.zoom;  
4 };
```

Fragment 82: Càcul d'alçada segons el factor de zoom

La Figura 65 conté el resultat obtingut.

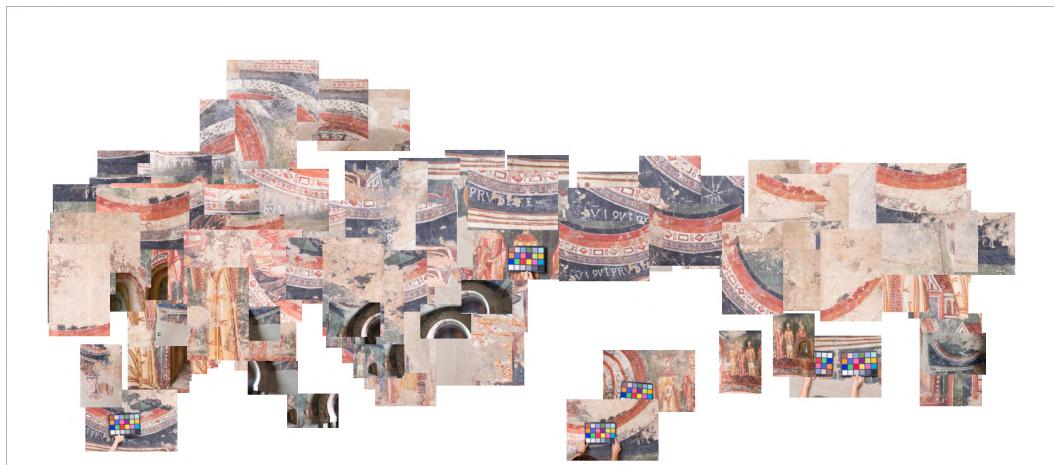


Figura 65: Imatges amb detalls augmentades
Font: Producció pròpia

9.3.2.3 Botó de commutació

Per tal de poder canviar la mida de les imatges en temps real, s'ha afegit un botó de commutació. Aquest botó es descriu amb més detall al capítol 9.5, i en aquesta secció es descriu l'actualització de la mida de les imatges.

En prémer el botó, es crida una funció que té com a objectiu actualitzar aquestes mides. Bàsicament, només cal iterar totes les imatges del visor, i assignar-li la nova mida. Podem obtenir les imatges del visor utilitzant el mètode *getItemAt* [85] de *viewer.world* [86]. A més a més, podem modificar l'alçada mitjançant *setHeight* [87]. Per a obtenir aquesta alçada, només en cal fusionar les dues funcions *getHeight* descrites anteriorment, i seleccionar el càlcul d'alçada corresponent. La implementació de la funció cridada pel botó, i la nova funció *getHeight* és la que trobem al Fragment 83.

```
1 var regularZoom = true;
2
3 function invertZoom() {
4     regularZoom = !regularZoom;
5     for (let i = 0; i < parsedImages.length; i++) {
6         const a = parsedImages[i];
7         var item = viewer.world.getItemAt(i); // Obtenim la
8             imatge
9             item.setHeight(getHeight(a)); // Apliquem la nova mida
10    }
11
12 // Reimplementació de getHeight
13 function getHeight(a) {
14     if (regularZoom) {
15         if (a.isLandscape) return a.heightToWidthRatio / a.zoom
16         ;
17         return 1 / a.zoom;
18     }
19     if (a.isLandscape) return a.heightToWidthRatio * a.zoom;
20     return a.zoom;
```

Fragment 83: Actualització de l'alçada d'imatges

9.3.3 Posicionament d'imatges segons target

En aquests moments, les posicions de les imatges del visor són obtingudes a partir de la posició de la imatge en l'escena. Aquesta manera d'obtenir les posicions és totalment vàlida, però, en certs casos ens pot ser limitant. Ens pot interessar posicionar les imatges segons la posició de l'element que mostren. Un dels casos on la posició des d'on la imatge s'ha pres no correspon amb l'element que enfoquen, és quan el fotògraf pren una fotografia del sostre. Aquesta estarà feta des de sota, fent que en el visor la mostri més avall. A la Figura 66 es mostra una imatge on les fotografies que enfoquen al sostre estan posicionades per sota d'altres imatges.



Figura 66: Distribució d'imatges segons la posició real

Font: Producció pròpia

Hem de trobar una manera per tal que aquestes imatges siguin col·locades a sobre de la resta. Una de les maneres és calcular la posició de l'element que enfoquen és utilitzant el *ray casting* [50] des de la fotografia. Aquesta tècnica ja s'ha explicat anteriorment al capítol 8.3.3. Per a cada imatge del model, es llençarà un raig des de la seva posició, amb la direcció on la imatge enfoca. A la Figura 67 es mostra en vermell el raig llançat des d'una imatge fins a incidir amb el model.

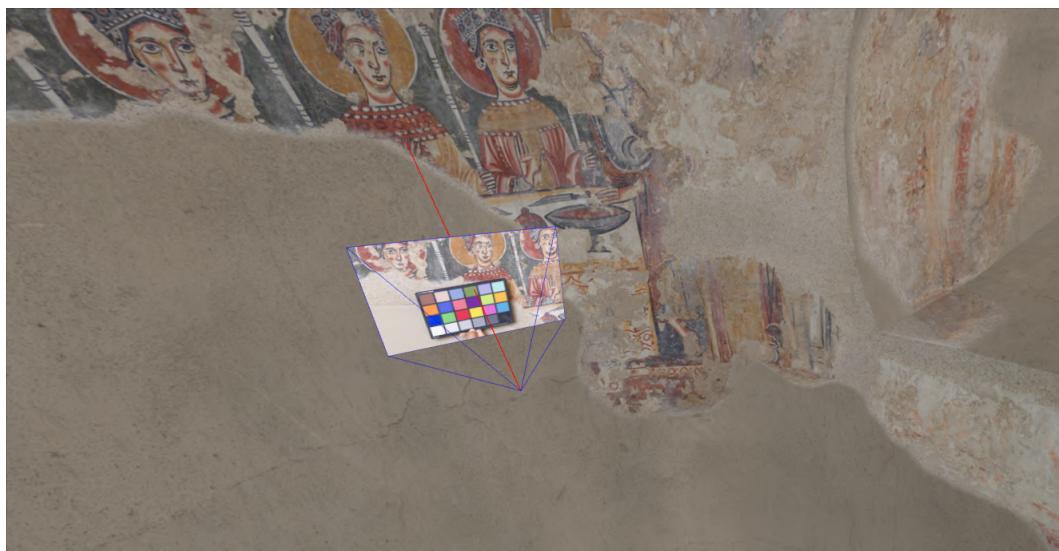


Figura 67: Raig llançat des d'una imatge
Font: Producció pròpia

Una vegada llancem el raig des de la posició de la imatge amb la seva direcció, el raig ens retorna els punts en què ha incidit. La primera intersecció del raig mai és el model, gairebé sempre és la imatge que correspon al raig. A més a més, en alguns casos incideix amb altres imatges abans de col·lidir amb el model. Per això, iterarem els punts amb què ha incidit, i comprovarem el nom de l'objecte amb què ha incidit. En el cas que el nom correspongui amb el model, agafarem aquell punt com a posició, altrament, comprovarem el punt següent (vegeu Fragment 84).

```

1 images.forEach((i) => {
2     // Obtenim posicio i direccio de la imatge
3     let position = i.position;
4     let direction = i.userData.direction;
5     const intersection = getIntersectionPosition(position,
6         direction);
7     // Desem el punt d'interseccio dins de l'objecte imatge
8     i.userData.intersection = new THREE.Vector3().copy(
9         intersection);
10    });
11
12    function getIntersectionPosition(scene, position, direction) {
13        // Llancem el raig
14        raycaster.set(position, direction);
15        var intersections = raycaster.intersectObject(scene, true);
16        if (intersections.length == 0) return null;
17        // Iterem tots els punts en que intersecciona
18        for (let i = 0; i < intersections.length; i++) {
19            // Retornem el primer punt del model
20            if (intersections[i].object.name == "model") return
21            intersections[i].point;
22        }
23        return null;
24    }

```

Fragment 84: Obtenció de la posició en què intersecciona el raig

Aquests càlculs es computen una sola vegada durant la càrrega d'imatges. Una vegada es volen obrir una sèrie d'imatges al visor en 2 dimensions, es calculen les posicions en 2 dimensions amb els 2 casos, amb la posició de la imatge, i la posició d'intersecció amb el model. Aquests dos valors s'escriuen a l'objecte que serà eniat a *OpenSeaDragon*. El Fragment 85 mostra el codi equivalent a cada projecció mencionada al capítol 8.4.

```

1 const P = object.position;
2 // Posicio interseccio
3 const P2 = object.userData.intersection;
4 let info = {};
5
6 if (isInside) {
7     const P2d = get2DCoords(P);
8     const P_inter = get2DCoords(P2)
9     info = {
10         name: object.name,
11         x: P2d.x,
12         y: P2d.y,
13         // Posicio interseccio
14         x_inter: P_inter.x,
15         y_inter: P_inter.y,
16         isLandscape: object.userData.isLandscape,
17         heightToWidthRatio: object.userData.heightToWidthRatio,
18         zoom: object.userData.zoom,
19     };
20 }

```

Fragment 85: Creació de l'objecte amb les dues posicions

Una vegada a *OpenSeaDragon*, només ens cal seleccionar quina de les dues posicions volem habilitar. Afegirem un botó que ens permeti seleccionar quin mode volem visualitzar. Aquest botó és molt similar al botó de commutació del capítol 9.3.2.2. Les imatges es distribuiran segons la posició original, i una vegada es premi el botó de commutació, les imatges prendran la posició d'intersecció amb el model. Accedirem a les imatges del visor de la mateixa manera que el botó de la secció 9.3.2.2, però modificant la posició en contres de la mida. Utilitzarem el mètode *setPosition* [88] per a assignar la nova posició (vegeu Fragment 86).

```

1 var realPosition = true;
2
3 function togglePosition() {
4     realPosition = !realPosition;
5     for (let i = 0; i < parsedImages.length; i++) {
6         const a = parsedImages[i];
7         var item = viewer.world.getItemAt(i); // Obtenim la
8             imatge
9         const pos = getImagePosition(a);
10        item.setPosition(new OpenSeadragon.Point(pos.x, pos.y))
11    ;
12    }
13 // Retorna la posicio segons el mode habilitat

```

```

14 function getImagePosition(a) {
15     if (realPosition) return { x: a.x, y: a.y };
16     return { x: a.x_inter, y: a.y_inter };
17 }

```

Fragment 86: Actualització de les posicions

9.3.4 Evitant l'encavalcament d'imatges

És fàcil veure en figures anteriors, que es produeixen encavalcaments entre les imatges del visor. Per poder mostrar totes les imatges senceses sense que s'ocultin parts, s'ha dissenyat un algorisme que distribueixi les imatges de manera que no es produixin oclusions, respectant la posició que els pertoca en la mesura més gran possible.

L'algorisme dissenyat iterarà per tots els parells d'interseccions. Si no existeix cap intersecció entre les dues imatges seleccionades, continuarem amb el següent parell. D'altra banda, si existeix una intersecció, es desplaçarà una de les imatges tantes unitats com equivalgui la intersecció. Si la distància d'encavalcament horitzontal és menor a la vertical, desplaçarem la imatge *a* aquesta distància horitzontal, altrament la desplaçarem verticalment amb el valor de la distància d'encavalcament vertical. La Figura 68 representa la resolució de l'encavalcament entre dues imatges.

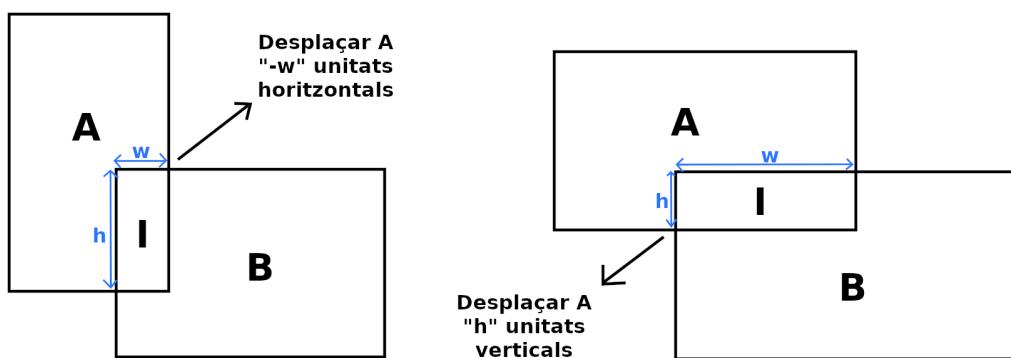


Figura 68: Desplaçament d'una imatge segons la intersecció

Font: Producció pròpria

Si s'han resolt interseccions durant la iteració de parells d'imatges, s'iniciarà una altra iteració amb tots els parells. S'anirà iterant fins que no hagi trobat cap encavalcament entre tot parell d'imatges diferents. A continuació trobem el Fragment 87, que resol tots els encavalcaments.

```

1 var overlapping;
2
3 function distribute(images) {
4     overlapping = true;
5     for (let i = 0; overlapping; i++) {
6         // Restablim indicador d'interseccions
7         overlapping = false;
8         // Iterem tots els parells
9         for (let j = 0; j < images.length; j++) align(images[i],
10             images, j);

```

```

10      }
11  }
12
13 function align(a, images, i) {
14   images.forEach((b) => {
15     // Si la imatge es la mateixa a comprovar, seguim
16     if (a.name === b.name) return;
17     var intersection = getIntersection(a, b);
18     // Si la intersecció es buida, seguim
19     if (!intersection) return;
20     // Indiquem que hem trobat una intersecció
21     overlapping = true;
22     var diff = getDistance(a, b);
23     var output = { x: 0, y: 0 };
24     // Resolem oclusió. La funció "sign" ens retorna 1 si
25     // el valor es positiu, -1 si es negatiu
26     if (intersection.width < intersection.height) output.x
27     += sign(diff.x) * intersection.width;
28     else output.y += sign(diff.y) * intersection.height;
29     // Despalcем la imatge
30     moveImage(a, output, i);
31   });
32 }

```

Fragment 87: Resolució d'encavalcaments

A continuació, calculem el polígon intersecció entre dues imatges. En el cas que no hi hagi cap intersecció retornarem nul. Per a obtenir la mida horitzontal del polígon, restarem el costat dret mínim amb el costat esquerre màxim. Per a l'alçada, restarem el costat superior mínim amb el costat inferior màxim. La Figura 69 ens ajuda a entendre el càlcul.

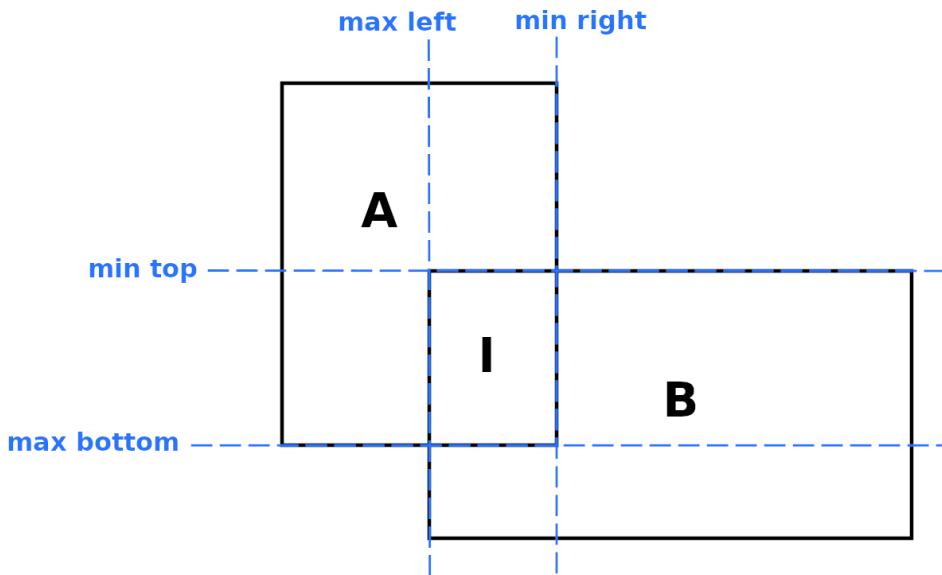


Figura 69: Càlcul de la intersecció
Font: Producció pròpia

La implementació de la funció `getIntersection` mencionada al Fragment 87, és al

Fragment 88.

```
1 function getIntersection(a, b) {
2     var left = max(getLeft(a), getLeft(b));
3     var top = min(getTop(a), getTop(b));
4     var right = min(getRight(a), getRight(b));
5     var bottom = max(getBottom(a), getBottom(b));
6
7     // Si hi ha interseccio, retorna la mida
8     if (bottom < top && right > left) {
9         return {
10            height: top - bottom,
11            width: right - left,
12        };
13    }
14    return null;
15 }
```

Fragment 88: Càcul de la intersecció entre dues imatges

Finalment, només ens queda actualitzar les posicions de les imatges. Es modifiquen d'una manera molt similar al posicionament descrit al capítol 9.3.3. Calcularem la nova posició amb la distància vertical i horitzontal calculades anteriorment. Accedirem a la imatge del visor utilitzant el mètode *getItemAt* [85] de *viewer.world* [86], i l'assignarem utilitzant el mètode *setPosition* [88]. El Fragment 89 conté la implementació del mètode *moveImage* mencionat al Fragment 87.

```
1 function moveImage(a, output, i) {
2     // Calcul de la nova posicio
3     a.x += output.x;
4     a.y += output.y;
5     var item = viewer.world.getItemAt(i);
6     // Assignacio de la posicio
7     item.setPosition(new OpenSeadragon.Point(a.x, a.y));
8 }
```

Fragment 89: Actualització de les posicions

Si executem el visor veurem que ens saltarà un error avisant-nos que no s'ha trobat l'element de la posició *i*. Això es deu al fet que estem calculant les noves posicions abans que les imatges siguin carregades. Per tant, haurem d'indicar al visor que no comenci els càlculs fins que totes les imatges s'hagin carregat. *OpenSeaDragon* llança un *Event* [89] *open* [90] una vegada s'han carregat totes les imatges. Li indicarem que comenci aquests càlculs quan rebem l'*Event* [89] *open* [90] (vegeu Fragment 90).

```
1 viewer.addHandler("open", function () {
2     distribute(parsedImages);
3 });
```

Fragment 90: Inicialitzar càlculs d'encavalcament una vegada s'hagin carregat les imatges

Cal mencionar que cal fer la crida a *distribute* cada vegada que s'activen els botons de commutació modificadors de posicions i de mida.

Però, el resultat obtingut no és l'esperat. Les imatges no estan separades correctament, i encara hi ha algunes oclusions, tal com podeu veure a la Figura 70.

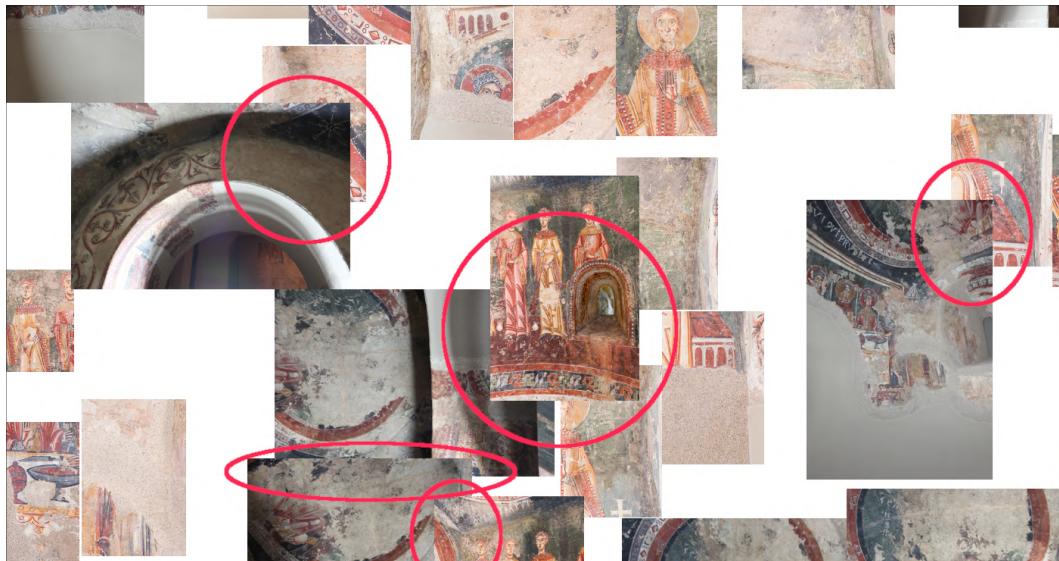


Figura 70: Distribució amb algunes oclusions

Font: Producció pròpia

L'anàlisi del codi m'indicava que les posicions de les imatges al visor no corresponien amb les que havia establert. Després de diverses investigacions, vaig trobar que l'origen de coordenades de les imatges a *OpenSeaDragon* no era el centre de les imatges, és la cantonada de dalt a l'esquerra [91]. Per solucionar-ho, només calia modificar la funció *getImagePosition* per tal que et retornés la posició a la cantonada esquerra superior. Aquesta s'aconsegueix restant la meitat de l'amplada, i la meitat de l'alçada (vegeu Fragment 91).

```

1 function getImagePosition(a) {
2     if (realPosition) return { x: a.x - getWidth(a)/2, y: a.y -
3         getHeight(a)/2};
4     return { x: a.x_inter - getWidth(a)/2, y: a.y_inter -
5         getHeight(a)/2};
6 }
7 );

```

Fragment 91: Càcul de la posició de la cantonada esquerra superior

Una vegada aplicats el canvi, el resultat és l'esperat. Com es mostra a la Figura 71, ja no hi ha oclusions entre les imatges.

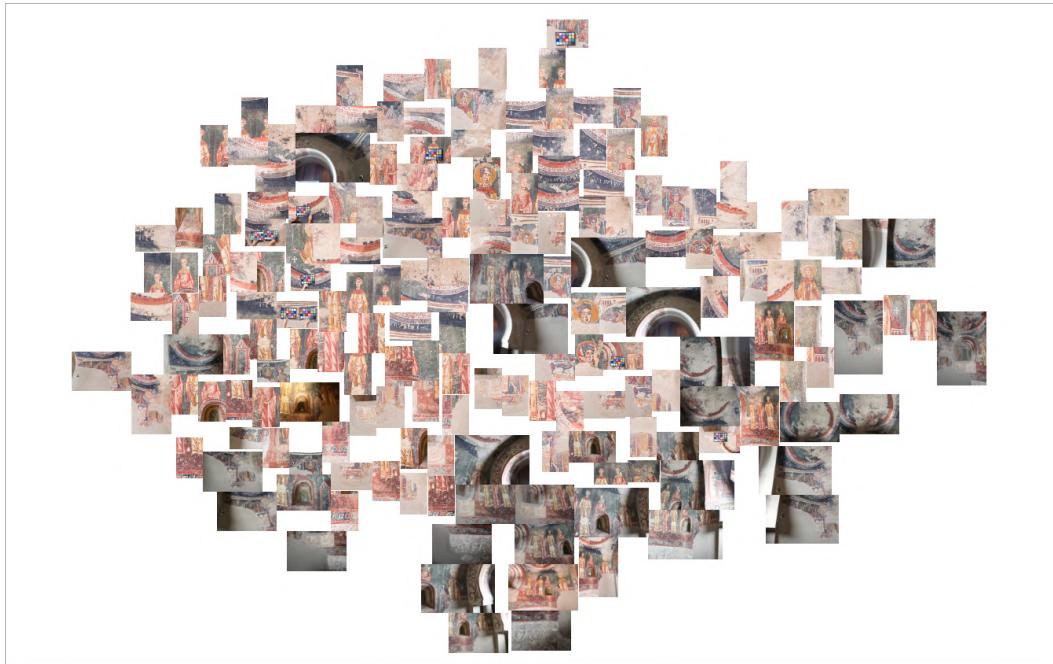


Figura 71: Distribució sense oclusions
Font: Producció pròpia

9.4 Navegació a ThreeJS

La navegació a *ThreeJS* té l'objectiu de fer una integració més forta entre l'entorn 3D i el visor 2D. Ara, hi ha la possibilitat de donar instruccions de l'entorn 3D al visor 2D, indicant quines imatges visualitzar, i com mostrar-les. En la direcció inversa, del visor 2D al 3D no hi ha cap instrucció.

Aquesta navegació tracta de situar la càmera de l'entorn 3D a la fotografia que li indiquis al visor 2D. L'usuari podrà premer qualsevol imatge del visor, i automàticament, la càmera de l'entorn 3D es posicionarà davant de la imatge en qüestió.

El primer pas és obtenir el nom de la imatge què s'ha premut a sobre. Per fer-ho, *OpenSeaDragon* ens proporciona l'*Event* [89] *canvas-click* [92], que es crida en fer clic sobre el visor. Aquest, té com a argument el píxel de la finestra que s'ha pres. Mitjançant el mètode *pointFromPixel* podem obtenir les coordenades del visor on s'ha pres el ratolí. Finalment, només ens cal iterar totes les imatges, i n'hi ha alguna on el punt entri dins dels 4 costats del polígon. En el cas que entri dins d'una imatge, escriurem el seu nom al *localStorage* [76], ja mencionat al capítol 9.1.2. El Fragment 92 és l'encarregat de trobar el nom de la imatge que s'ha premut.

```

1 viewer.addHandler("canvas-click", function (event) {
2     // Si s'ha mantingut el ratoli, retornem
3     if (!event.quick) return;
4     // Obtenim coordenades del visor
5     var viewportPoint = viewer.viewport.pointFromPixel(event.
position);
6     for (let i = 0; i < viewer.world.getItemCount(); i++) {
7         let a = viewer.world.getItemAt(i);
8         // Si la posicio esta fora de la imatge, seguiex
9         let bounds = a.getBounds();
10        if (

```

```

11         viewportPoint.x < bounds.x ||
12         viewportPoint.x > bounds.x + bounds.width ||
13         viewportPoint.y < bounds.y ||
14         viewportPoint.y > bounds.y + bounds.height
15     ) continue;
16     // Escrivim el nom de la imatge a localStorage, i
17     // retornem
18     localStorage.setItem("navigate", parsedImages[i].name);
19     return;
20 }
20 );

```

Fragment 92: Obtenció i escriptura del nom de la imatge premuda

Cal mencionar, que per defecte, el visor aplica un zoom cada vegada que prems el botó. Per desactivar-ho, he establert el valor de zoom en clicar a 1 (vegeu Fragment 93).

```
1 viewer.zoomPerClick = 1;
```

Fragment 93: Establir el zoom en clicar a 1

L'entorn 3D haurà de llegir el nom de la imatge des del *localStorage* [76], i aplicar la posició i rotació corresponent per tal que mostri la imatge de l'entorn. Cada vegada que s'escriu un valor a *localStorage*, es llança *storage event* [94]. Cada vegada que *ThreeJS* el rebi, llegirà el nom de la imatge del *localStorage*, i aplicarà a la càmera la posició i rotació que té aplicada la imatge (vegeu Fragment 94).

```

1 window.addEventListener("storage", (event) => {
2     let images = getAllImages();
3     // Llegim el valor de localStorage
4     const image_name = localStorage.getItem("navigate");
5     images.forEach((i) => {
6         if (i.name != image_name) return;
7         // Establim la posicio i rotacio de la imatge a la
7         // camera
8         camera.position.copy(i.position);
9         camera.rotation.copy(i.rotation);
10    });
11 });

```

Fragment 94: Establir la posició i rotació de la imatge

El resultat obtingut no és l'esperat. En prémer la imatge de la Figura 72,



Figura 72: Imatge seleccionada per a la navegació
Font: Producció pròpria

Podem veure a la Figura 73, que la posició és la correcta, però la rotació no.



Figura 73: Navegació amb rotació incorrecta
Font: Producció pròpria

La càmera, tot i indicar-li una rotació, es queda mirant al punt $(0,0,0)$ de l'escena. Després d'estar investigant, vaig trobar que els controls de la càmera mitjançant el ratolí sobreescrivien la rotació, per tal de mirar sempre mirant al punt $(0,0,0)$. Per tant, calia indicar que el punt objectiu dels controls de la càmera fos la posició de la imatge. El punt no pot ser el mateix en què es localitza la càmera, fent que hagués de calcular un desplaçament en la direcció on se situa la imatge. Finalment, el codi complet per a llegir el nom de la imatge de *localStorage* i l'establiment de la posició i rotació a la càmera, és el que trobem al Fragment 95.

```

1 window.addEventListener("storage", (event) => {
2     let images = getAllImages();
3     // Llegim el valor de localStorage
4     const image_name = localStorage.getItem("navigate");
5     images.forEach((i) => {
6         if (i.name != image_name) return;
7         // Establim la posicio i rotacio de la imatge a la
8         // camera
9         camera.position.copy(i.position);
10        // Calcul punt objectiu dels constos
11        let target = new THREE.Vector3().copy(i.position).add(i
12        .userData.direction);
13        // Establir punt objectiu
14        controls.target.copy(target);
15    });
16 });

```

Fragment 95: Establir la posició i punt objectiu dels controls de la càmera

Amb aquest canvi, el resultat és l'esperat (vegeu Figura 74).



Figura 74: Navegació amb rotació correcta
Font: Producció pròpia

Aquesta navegació també és possible en el mode Inspecció. Simplement, en seleccionar una imatge del visor, situarà la càmera enfocant la part del model que mostra la imatge. El funcionament és exactament el mateix. L'únic canvi és que les imatges de l'entorn 3D estan ocultes. La càmera segueix agafant la posició i orientació de la imatge seleccionada. Per exemple, en seleccionar la imatge del visor (vegeu Figura 75),



Figura 75: Imatge seleccionada al mode Inspecció
Font: Producció pròpia

La càmera navega a la posició de la imatge, i l'orienta. En contres de mostrar la imatge de l'entorn, la càmera mostra la part del model que representa la imatge (vegeu Figura 76).

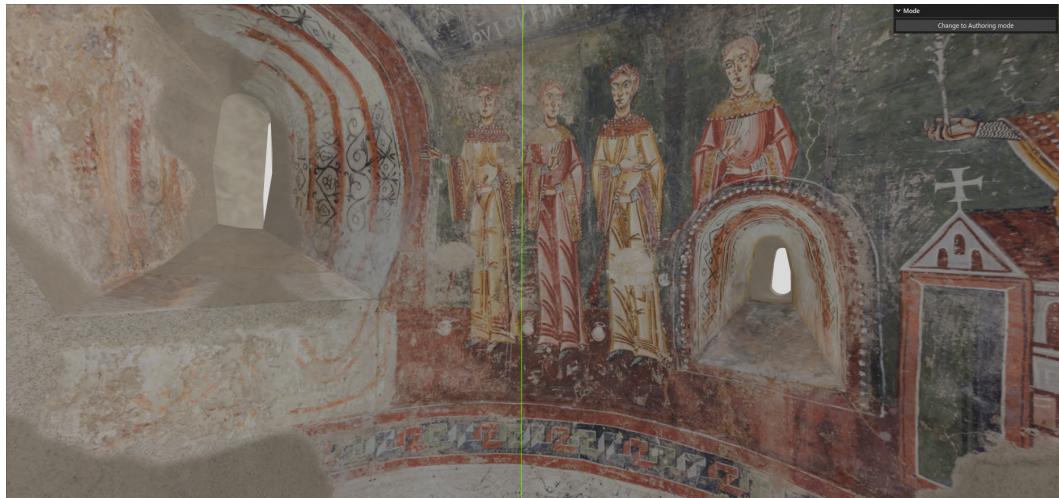


Figura 76: Navegació a la part del model corresponent a la imatge
Font: Producció pròpia

9.5 Interfície gràfica

OpenSeaDragon proporciona una interfície gràfica amb els botons de zoom, de pantalla completa, etc. Es veu una mica antiquat, però el motiu de substituir-lo no és el de l'estètica. Tal com s'ha descrit a les seccions 9.3.2.2 i 9.3.3, s'han afegit dos botons que commuten estats del visor. Una nova interfície gràfica integraria aquests botons amb els ja existents al visor. A la Figura 77 es mostra la interfície per defecte del visor.



Figura 77: Interfície gràfica per defecte
Font: Producció pròpia

La interfície està construïda com a un element *HTML* que incorpora els botons per defecte juntament amb els botons de commutació de les seccions 9.3.2.2 i 9.3.3. Aquests dos botons realment en són 4, i es mostren i ocullen en fer la commutació, ja que tenen noms i icones diferents. Els botons de commutació criden a les funcions ja especificades a les seves seccions corresponents, mentre els altres els aplicarem la funcionalitat dels botons originals, que s'explicarà més endavant. Les icones que componen cada botó de la interfície han sigut extretes de *FontAwesome* [95]. El Fragment 96 correspon a l'*HTML* de la interfície gràfica.

```

1 <div id="ui">
2     <!-- Botons predeterminats -->
3     <a id="home"><i class="fa-solid fa-camera"></i> Center
4         camera</a>
5     <a id="zoom-in"><i class="fas fa-magnifying-glass-plus"></i>
6         > Zoom In</a>
7     <a id="zoom-out"><i class="fas fa-magnifying-glass-minus"></i>
8         > Zoom Out</a>
9     <a id="full-page"><i class="fa-solid fa-expand"></i> Full
10        Screen</a>

```

```

7      <!-- Boto Increase/Decrease details -->
8      <a id="invert-zoom" href="javascript:invertZoom()" title=
9      Increase details"><i class="fa-solid fa-up-right-and-down-
10     left-from-center"></i> Increase Deatils</a>
11      <a id="regular-zoom" style="display: none" href="javascript:
12      :invertZoom()" title="Reduce details" ><i class="fa-solid
13     fa-down-left-and-up-right-to-center"></i> Reduce Details</a>
14
15      <!-- Boto Set Real/Intersection position -->
16      <a id="real-position" style="display: none" href="
17      javascript:togglePosition()" title="Set real position"><i
18      class="fa-solid fa-location-dot"></i> Set real position</a>
19      <a id="intersection-position" href="javascript:
20      togglePosition()" title="Set intersection position"><i
21      class="fa-solid fa-location-dot"></i> Set intersection
22      position</a>
23  </div>

```

Fragment 96: HTML de la interfície gràfica

Al Fragment 97 trobem l'estil aplicat.

```

1 a {
2     color: 000000;
3     text-decoration: none;
4     padding-left: 4px;
5     padding-right: 4px;
6     margin: 3px
7 }
8
9 a:hover {
10    color: rgb(80,80,255);
11 }
12
13 #ui {
14     margin: 10px;
15     border-radius: 10px;
16     padding: 5px;
17     background-color: rgb(200, 200, 200);
18 }

```

Fragment 97: Estil de la interfície gràfica

Per tal d'establir la funcionalitat dels botons *home*, *zoom-in*, *zoom-out* i *full-screen*, podem indicar al visor mitjançant un identificador, quins elements *HTML* prenen la funcionalitat de cada botó. En assignar els nous elements, *OpenSeaDragon* oculta automàticament els botons predeterminats. En declarar el visor, només ens cal afegir l'*id* de cada botó (vegeu Fragment 98).

```

1 var viewer = OpenSeadragon({
2     // Especificacio botons
3     zoomInButton: "zoom-in",
4     zoomOutButton: "zoom-out",
5     homeButton: "home",
6     fullPageButton: "full-page",
7     // Element que fa de visor

```

```

8     id: "openseadragon1",
9     // Mostrar imatges ja indicades
10    tileSources: sources,
11    // Mostrar navegador a la cantonada dreta superior
12    showNavigator: true,
13 });

```

Fragment 98: Establir la posició i punt objectiu dels controls

Afegir la interfície gràfica d'aquesta manera, fa sorgir el mateix problema que a l'entorn 3D amb la barra d'informació de la secció 8.6.1. Per defecte, el visor estableix la seva mida amb la de la finestra, és a dir que ocupa tota la finestra. En afegir un element addicional, el visor mantindrà la seva mida, fent que tot el contingut de la finestra sigui més alt que la finestra. Això fa que aparegui una barra de desplaçament que empitjora la usabilitat de l'aplicació. Per solucionar-ho, en aquest cas no disminuirem la mida del visor. Simplement, especificarem que la interfície sigui flotant, és a dir, que es trobi sobre del visor. Per fer-ho, agruparem en un mateix contenidor la interfície gràfica i el visor. Especificarem que la posició de la interfície sigui absoluta, i que es localitzi per sobre del visor. A continuació hi ha el Fragment *HTML* amb l'embolcall dels dos elements en un sol contenidor (Fragment 99), i més endavant, l'estil afegit (Fragment 100).

```

1 <div class="container">
2   <div id="ui">
3     <!-- Contingut especificat anteriorment -->
4   </div>
5   <!-- Visor -->
6   <div id="openseadragon1" style="width: 100vm"></div>
7 </div>

```

Fragment 99: Envolcall dels dos elements en un de sol

```

1 #ui {
2   position: absolute;
3   z-index: 1;
4 }
5
6 .container {
7   position: relative;
8 }

```

Fragment 100: Estil per a la interfície flotant

La Figura 78 conté el resultat final de la interfície al visor 2D, és el següent.

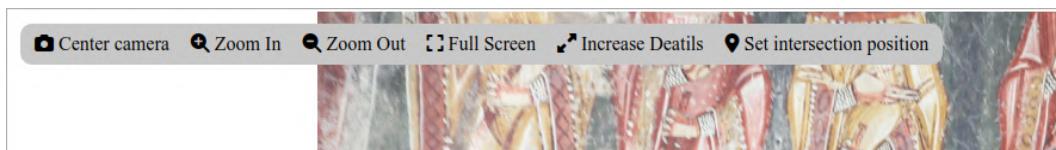


Figura 78: Interfície gràfica final
Font: Producció pròpia

10 Descripció dels resultats

En aquest capítol veurem els resultats d'una sèrie de casos d'ús. Per a cadascun d'aquest, s'especificarà els passos a seguir per tal d'obtenir-los, i es farà una petita descripció dels seus resultats.

10.1 Visualitzant una imatge

En primer lloc, tenim la visualització d'una imatge. En aquest cas, l'usuari visualitzarà una certa imatge de l'entorn 3D al visor 2D.

Per començar, l'usuari navegarà per l'entorn 3D fins a trobar la fotografia desitjada. Una vegada la trobi, situarà el ratolí a sobre d'aquesta (vegeu Figura 79). Una vegada la fotografia s'hagi ressaltat amb el color cian, l'usuari premerà el botó esquerre del ratolí.

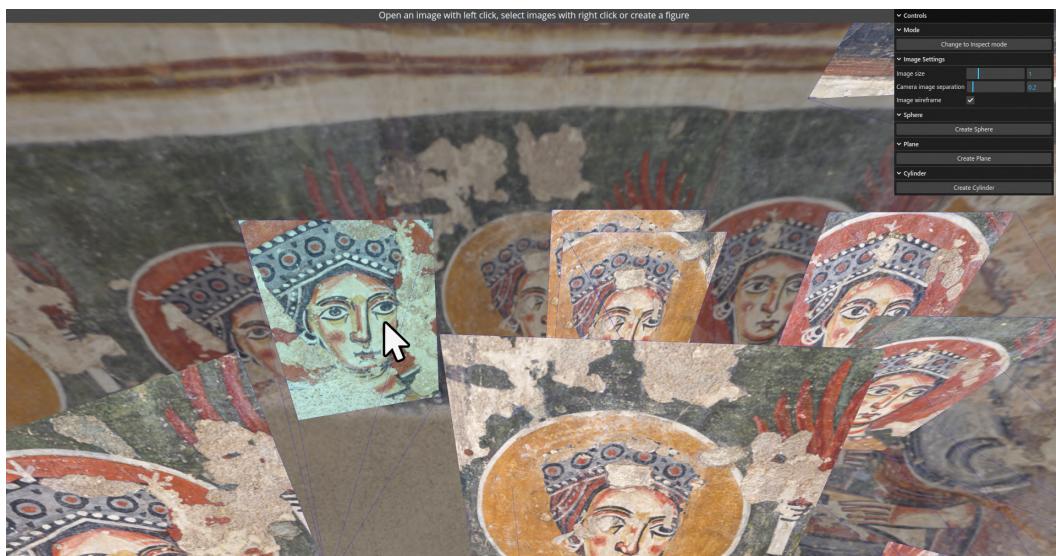


Figura 79: Imatge seleccionada a l'entorn 3D

Font: Producció pròpia

A continuació, s'obrirà una pestanya amb el visor d'imatges en 2D amb la imatge seleccionada (vegeu Figura 80). En el visor, l'usuari pot fer zoom mitjançant els botons de la interfície *Zoom In* i *Zoom Out*, o amb la rodeteta del ratolí. A més, es pot moure per la imatge arrossegant-la amb el botó esquerre del ratolí. Una altra funcionalitat permesa és mostrar el visor en pantalla completa. Per fer-ho, només cal prémer el botó *Full Screen* de la interfície. Per tal de sortir-ne, n'hi ha prou amb prémer la tecla *escape* (*esc*). Finalment, podem restablir la posició inicial de la càmera prement el botó de la interfície *Center camera*.

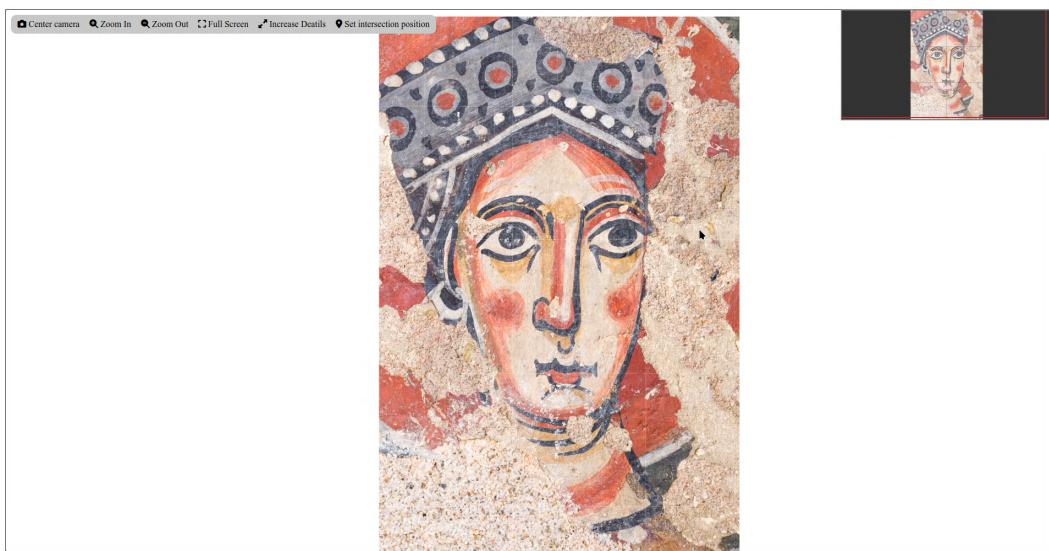


Figura 80: Imatge oberta al visor
Font: Producció pròpia

10.2 Visualitzant imatges seleccionades

El segon cas d'ús, és la visualització d'una sèrie d'imatges seleccionades per l'usuari. En aquest cas, l'usuari visualitzarà una les imatges seleccionades de l'entorn 3D al visor 2D.

Per començar, l'usuari navegarà per l'entorn 3D fins a trobar la primera fotografia desitjada. Una vegada la trobi, situarà el ratolí a sobre d'aquesta, i es ressaltarà amb el color cian. En prémer el botó dret sobre aquesta, el color de la imatge quedarà marcada amb el color magenta, indicant que ha quedat seleccionada. En prémer per segon cop a la imatge, aquesta quedarà desseleccionada, i passarà a tenir el color neutre. A continuació, l'usuari repetirà aquest procediment amb tantes imatges com desitgi. A la Figura 81 veiem un exemple d'imatges seleccionades.



Figura 81: Imatges seleccionades a l'entorn 3D
Font: Producció pròpia

Per visualitzar les imatges seleccionades al visor 2D, cal prémer el botó *Open in 2D viewer* sota de la secció *Individual Selection* (vegeu Figura 82).

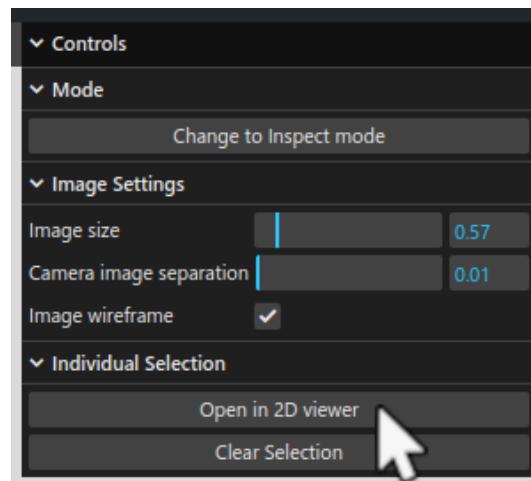


Figura 82: Botó *Open in 2d viewer*

Font: Producció pròpria

A continuació, s'obrirà una pestanya amb el visor d'imatges en 2D amb les imatges seleccionades (vegeu Figura 83). Aquestes imatges s'han desplaçat pel visor de manera que no hi hagi oclusions, i respecti la posició relativa entre elles el màxim possible. En el visor, l'usuari pot fer les accions descrites anteriorment al capítol 10.1. A més a més, pot prémer el botó de la interfície *Increase Details*, que modificarà la mida de les imatges. Els resultats d'aquesta funcionalitat queda descrita a la secció 10.4. Finalment, també pot prémer el botó *Set intersection position*, que modificarà la posició de les imatges, i quedarà vist a 10.3.



Figura 83: Imatges seleccionades obertes al visor

Font: Producció pròpria

10.3 Visualitzant imatges d'una esfera

En tercer lloc, veurem els resultats de la visualització d'imatges d'una esfera. En aquest cas, l'usuari definirà una esfera a l'entorn 3D, i visualitzarà les imatges properes a l'esfera projectades sobre aquesta. També, es descriurà els resultats de la visualització d'imatges amb la posició real, i amb la posició segons el *target*.

Per començar, l'usuari seleccionarà el botó de la interfície *Create Sphere* (vegeu Figura 84).

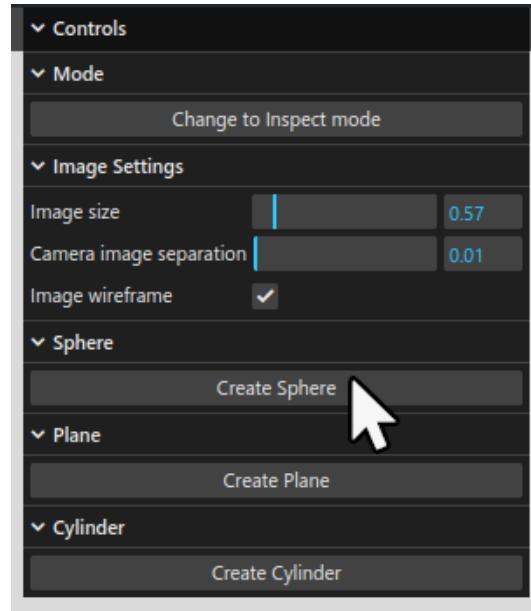


Figura 84: Botó *Create Sphere*
Font: Producció pròpia

Tot seguit, l'usuari seleccionarà amb el botó dret del ratolí, la imatge que serà el centre de l'esfera. Després de seleccionar-la, apareixerà una esfera amb filferros centrada a la imatge (vegeu Figura 85). Mitjançant el *slider Radius* (vegeu Figura 86), l'usuari podrà modificar la mida d'aquesta. La definició de la mida de l'esfera permet definir quines imatges incloure al visor 2D. Aquestes queden ressaltades de color taronja (vegeu Figura 85).

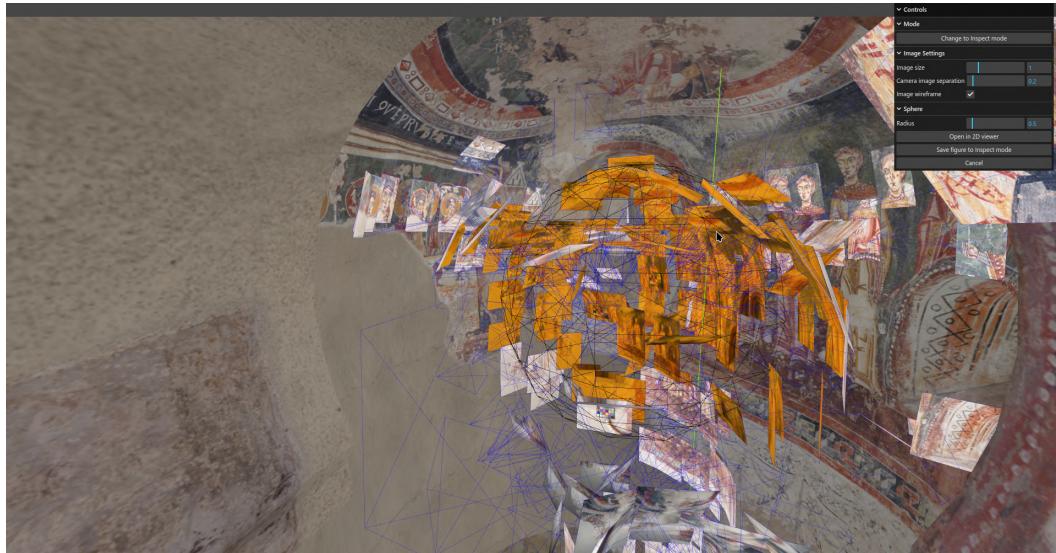


Figura 85: Esfera creada a l'entorn 3D
Font: Producció pròpia

Una vegada l'usuari està satisfet amb les imatges incloses, l'usuari les podrà obrir al visor prement el botó *Open in 2D viewer* de la secció *Sphere* (vegeu Figura 86).

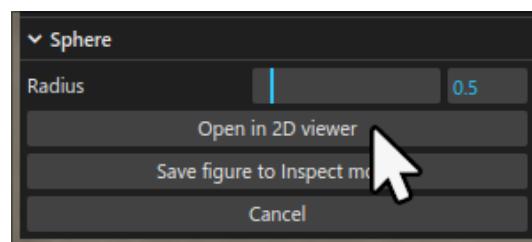


Figura 86: Botó *Open in 2d viewer*

Font: Producció pròpria

A continuació, s'obrirà una pestanya amb el visor d'imatges en 2D amb les imatges incloses a l'esfera (vegeu Figura 87). Aquestes imatges s'han desplaçat pel visor de manera que no hi hagi oclusions, i respecti la posició relativa entre elles el màxim possible. El posicionament de les imatges ve definit per la posició de la seva projecció a l'esfera. Es pot donar el cas que l'usuari tingui la necessitat de visualitzar les imatges segons la posició respecte al *target*. Per habilitar aquest posicionament, n'hi ha prou amb seleccionar el botó de la interfície *Set intersection position*.

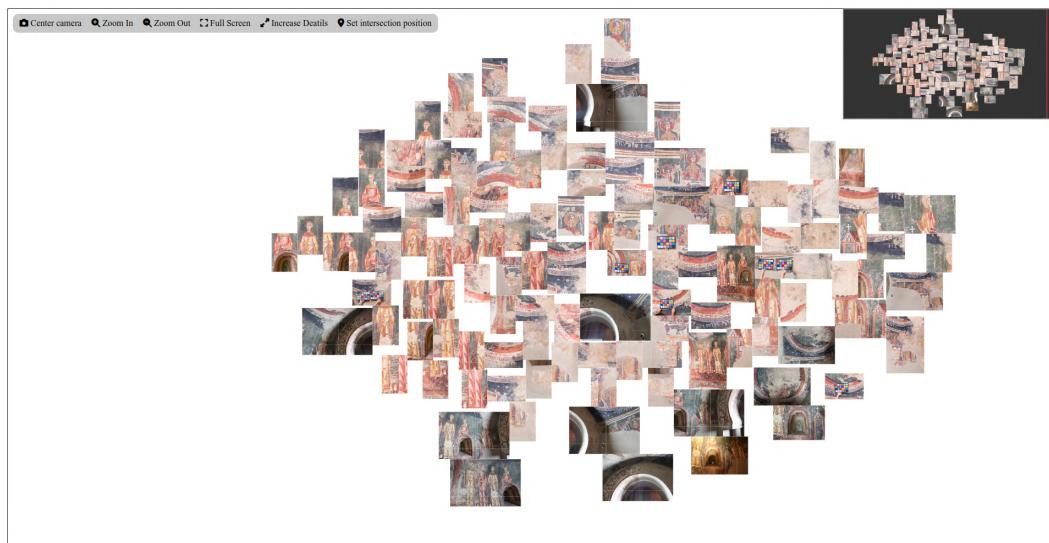


Figura 87: Imatges amb posició real

Font: Producció pròpria

A continuació, les imatges canviaran de posició, evitant l'oclusió entre elles (vegeu Figura 88). Per tal de revertir el posicionament, l'usuari farà clic sobre el botó *Set real position*, que havia canviat el nom de *Set intersection position*.

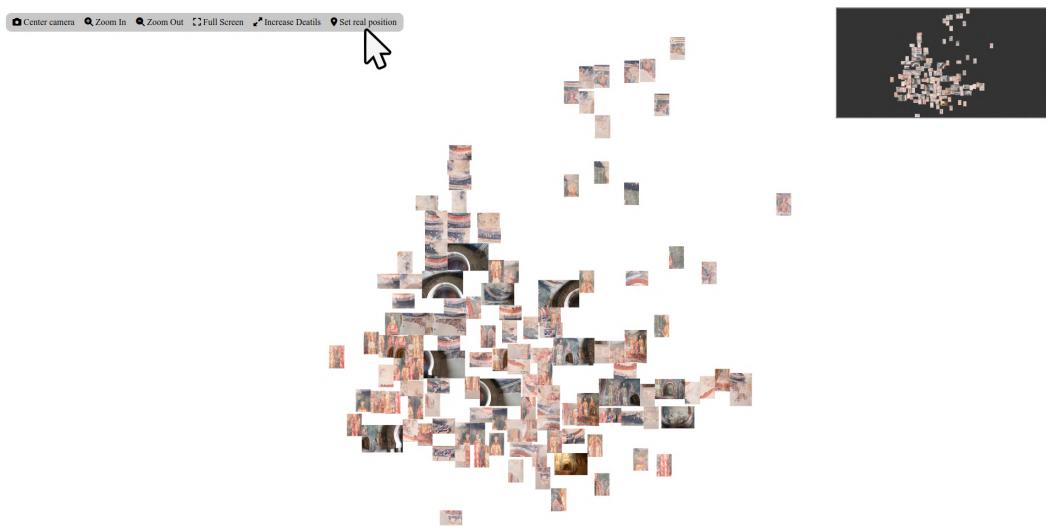


Figura 88: Imatges amb posició segons el target
Font: Producció pròpia

10.4 Visualitzant imatges d'un cilindre

En quart lloc, veurem els resultats de la visualització d'imatges d'un cilindre. En aquest cas, l'usuari definirà un cilindre a l'entorn 3D, i visualitzarà les imatges properes al cilindre projectades sobre aquest. També, es descriurà els resultats de la visualització d'imatges amb la mida d'imatges respecte al zoom, i també, la mida amb els detalls augmentats.

Per començar, l'usuari seleccionarà el botó de la interfície *Create Cylinder* (vegeu Figura 89).

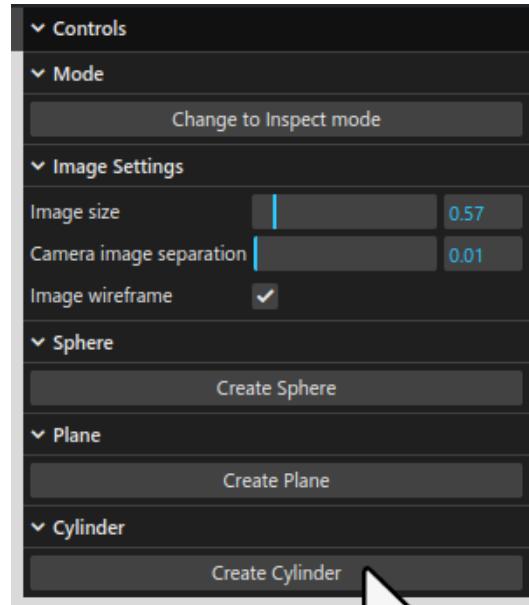


Figura 89: Botó *Create Cylinder*
Font: Producció pròpia

Tot seguit, l'usuari seleccionarà amb el botó dret del ratolí, les dues imatges que seran el centre de les dues cares del cilindre. Després de seleccionar-les, apareixerà un cilindre amb filferros posicionat entre les dues imatges (vegeu Figura 90). Mitjançant els *sliders Radius* i *Height* (vegeu Figura 91), l'usuari podrà modificar la

mida d'aquest. La definició de la mida del cilindre permet definir quines imatges incloure al visor 2D. Aquestes queden ressaltades de color taronja (vegeu Figura 90).

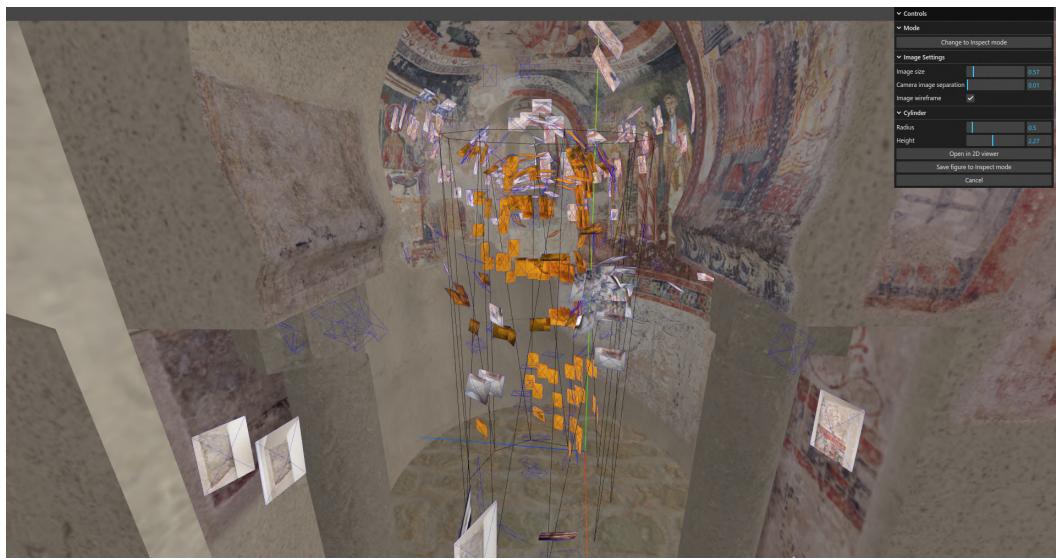


Figura 90: Cilindre creat a l'entorn 3D
Font: Producció pròpia

Una vegada l'usuari està satisfet amb les imatges incloses, l'usuari les podrà obrir al visor prement el botó *Open in 2D viewer* de la secció *Cylindrer* (vegeu Figura 91).

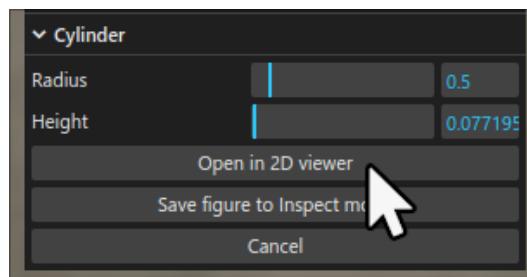


Figura 91: Botó *Open in 2d viewer*
Font: Producció pròpia

A continuació, s'obrirà una pestanya amb el visor d'imatges en 2D amb les imatges incloses al cilindre (vegeu Figura 92). Aquestes imatges prenen la mida respecte al zoom de la càmera a l'hora de fer-les. Es pot donar el cas que l'usuari no li interessi tenir una visió global de les imatges, sinó que busqui un petit detall en una imatge. Pot modificar la mida de les imatges seleccionant el botó de la interfície *Increase details*.

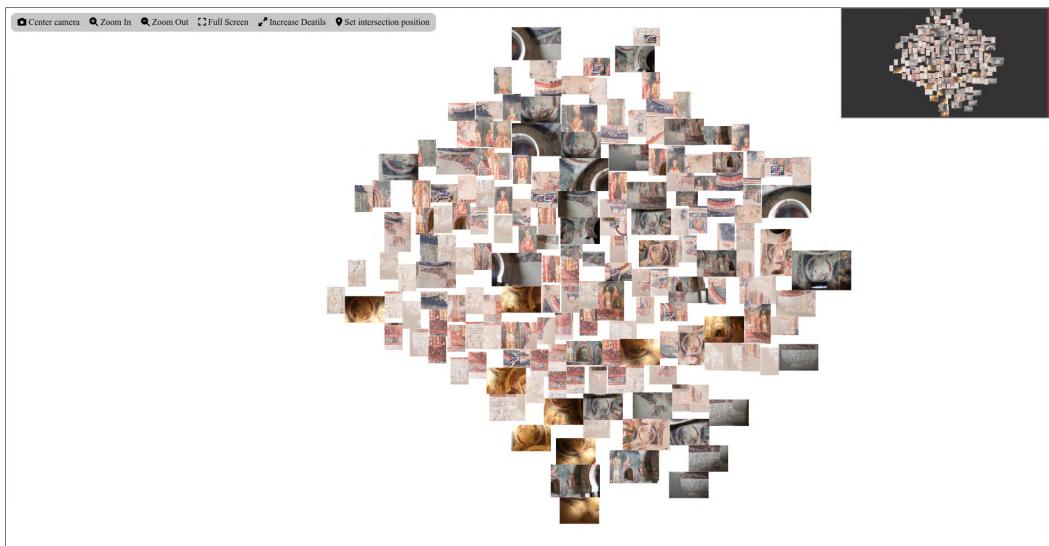


Figura 92: Imatges amb la mida segons el zoom
Font: Producció pròpia

Després de seleccionar el botó, les imatges canviaran la seva mida. Les imatges que enfoquen una part més global de l'església es reduiran, mentre les que mostren els detalls, augmentaran de mida (vegeu Figura 93). Cal dir que hi ha la possibilitat de fer qualsevol combinació de la modificació de mida i de posicionament. El botó premut haurà canviat a *Reduce details*. En cas de fer clic sobre aquest, revertirà la mida de les imatges.

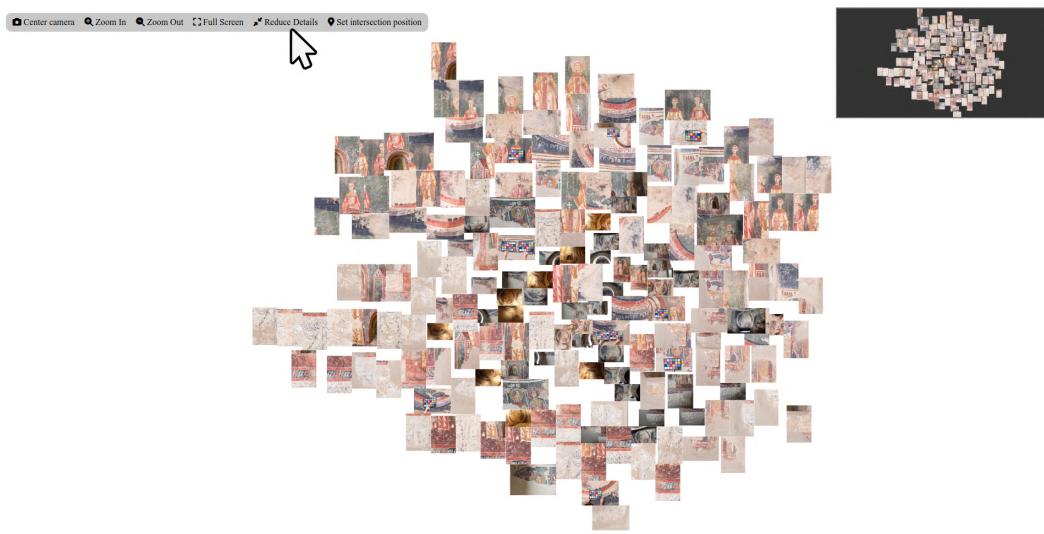


Figura 93: Imatges amb la mida augmentar detalls
Font: Producció pròpia

10.5 Visualitzant imatges d'un pla

En cinquè lloc, veurem els resultats de la visualització d'imatges d'un pla. En aquest cas, l'usuari definirà un pla a l'entorn 3D, i visualitzarà les imatges properes al pla projectades sobre aquest. També, farà la navegació a l'entorn 3D d'una imatge seleccionada al visor 2D.

Per començar, l'usuari seleccionarà el botó de la interfície *Create Plane* (vegeu Figura 94).

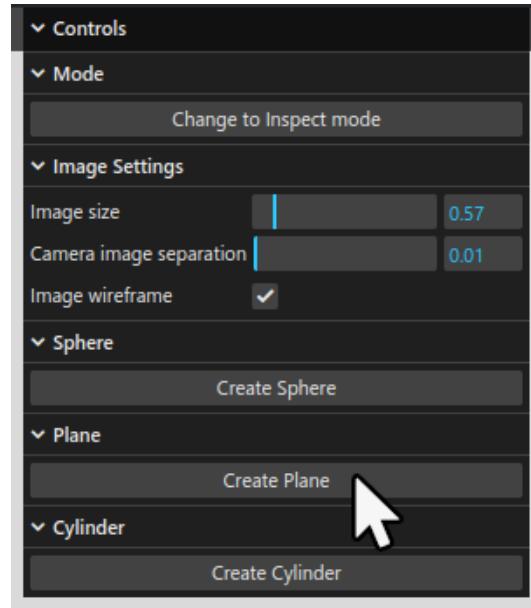


Figura 94: Botó *Create Plane*

Font: Producció pròpia

Tot seguit, l'usuari seleccionarà amb el botó dret del ratolí, les tres imatges coplanares en el pla, és a dir, les 3 imatges que defineixen el pla. Després de seleccionar-les, apareixerà un prisma amb filferros posicionat de manera que inclou les 3 imatges (vegeu Figura 95). Mitjançant els *sliders Width*, *Height* i *Max distance* (vegeu Figura 96, l'usuari podrà modificar la mida d'aquest. La definició de la mida del cilindre permet definir quines imatges incloure al visor 2D. Aquestes queden ressaltades de color taronja (vegeu Figura 95).



Figura 95: Pla creat a l'entorn 3D

Font: Producció pròpia

Una vegada l'usuari està satisfet amb les imatges incloses, l'usuari les podrà obrir al visor prement el botó *Open in 2D viewer* de la secció *Plane* (vegeu Figura 96).

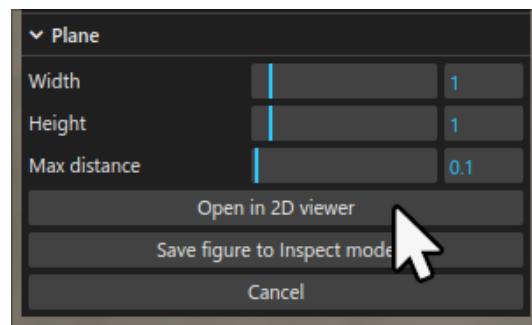


Figura 96: Botó *Open in 2d viewer*
Font: Producció pròpia

A continuació, s'obrirà una pestanya amb el visor d'imatges en 2D amb les imatges seleccionades (vegeu Figura 97). Aquestes imatges s'han desplaçat pel visor de manera que no hi hagi oclusions, i respecti la posició relativa entre elles el màxim possible.

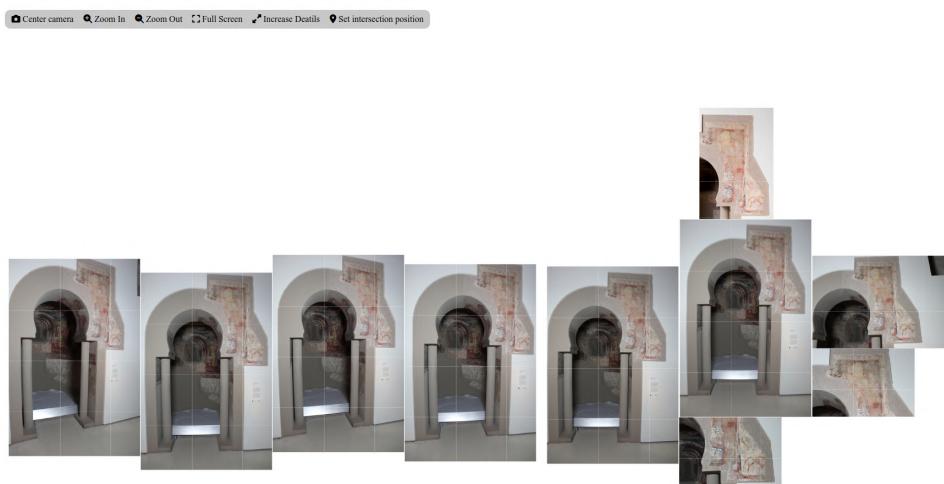


Figura 97: Imatges al visor
Font: Producció pròpia

A questa vegada, l'usuari decideix buscar una imatge que no trobava a l'entorn 3D. Una vegada la troba, fa clic a sobre d'aquesta (vegeu Figura 98).

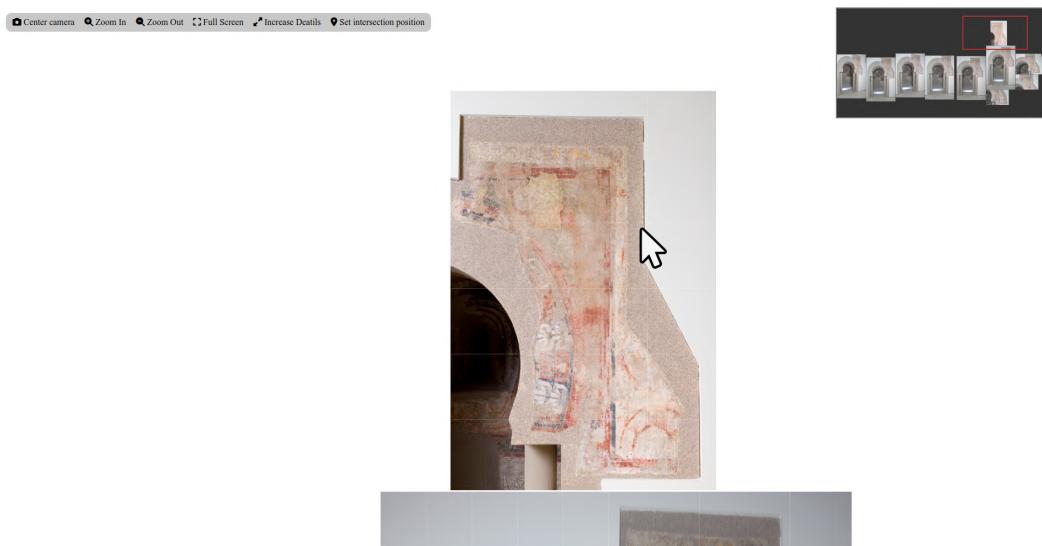


Figura 98: Imatge premuda per a navegar
Font: Producció pròpia

Tot seguit, a l'entorn 3D, la càmera enfoca la imatge seleccionada (vegeu Figura 99). D'aquesta manera, l'usuari pot trobar imatges de l'entorn 3D, trobant-les al visor 2D.

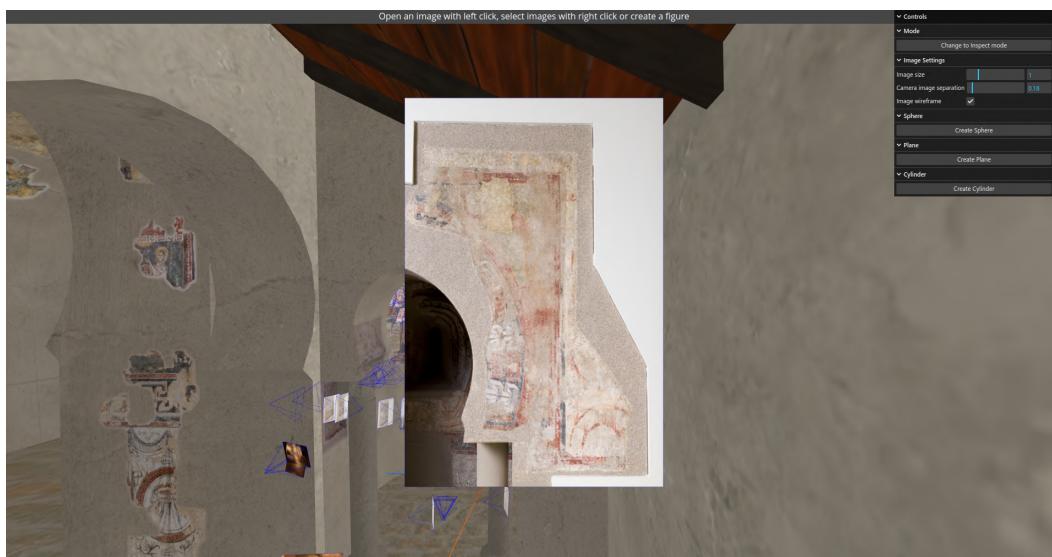


Figura 99: Navegació a la imatge seleccionada
Font: Producció pròpia

10.6 Visualitzant imatges amb el mode Inspecció

En tercer lloc, veurem els resultats de la visualització d'imatges en mode Inspecció. En aquest cas, l'usuari definirà una esfera a l'entorn 3D utilitzant el mode Autor. Un usuari en mode Inspecció, visualitzarà les imatges de l'esfera definida. També, farà la navegació a l'entorn 3D d'una imatge seleccionada al visor 2D.

Per començar, l'usuari seleccionarà el botó de la interfície *Create Sphere* (vegeu Figura 100).

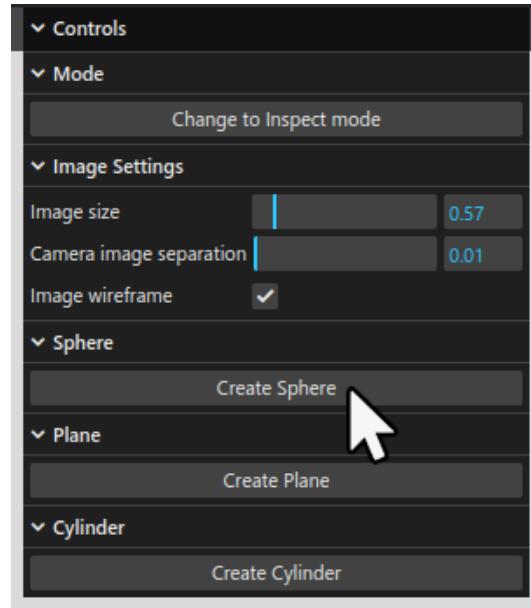


Figura 100: Botó *Create Sphere*
Font: Producció pròpia

Tot seguit, l'usuari seleccionarà amb el botó dret del ratolí, la imatge que serà el centre de l'esfera. Després de seleccionar-la, apareixerà una esfera amb filferros centrada a la imatge (vegeu Figura 101). Mitjançant el *slider Radius* (vegeu Figura 102), l'usuari podrà modificar la mida d'aquesta. La definició de la mida de l'esfera permet definir quines imatges incloure al visor 2D. Aquestes queden ressaltades de color taronja (vegeu Figura 101).



Figura 101: Esfera creada a l'entorn 3D
Font: Producció pròpia

Una vegada l'usuari està satisfet amb les imatges incloses, l'usuari podrà desar la figura pel mode Inspecció prement el botó de la interfície *Save figure to Inspect mode* (vegeu Figura 102).

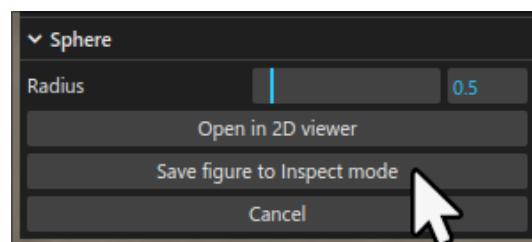


Figura 102: Botó *Save figure to Inspect mode*
Font: Producció pròpia

Un usuari que vulgui utilitzar el mode inspecció clicarà sobre el botó *Change to Inspect mode* (vegeu Figura 103). El botó canviarà a *Change to Authoring mode*, per tornar a canviar de mode.

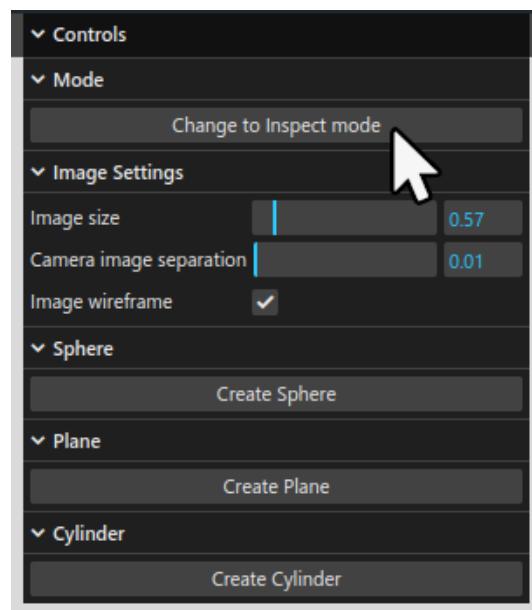


Figura 103: Botó *Change to Inspect mode*
Font: Producció pròpia

Una vegada l'usuari se situa al mode Inspecció, l'usuari visualitzarà el model amb les figures definides. En aquest cas, a la Figura 104, apareix una esfera petita, representant l'esfera generada per l'usuari en mode Autor. L'usuari en mode inspecció pot navegar pel model, i en cas de voler visualitzar les imatges d'una figura, n'hi ha prou en prémer el botó esquerre del ratolí sobre una d'aquestes (vegeu Figura 104).

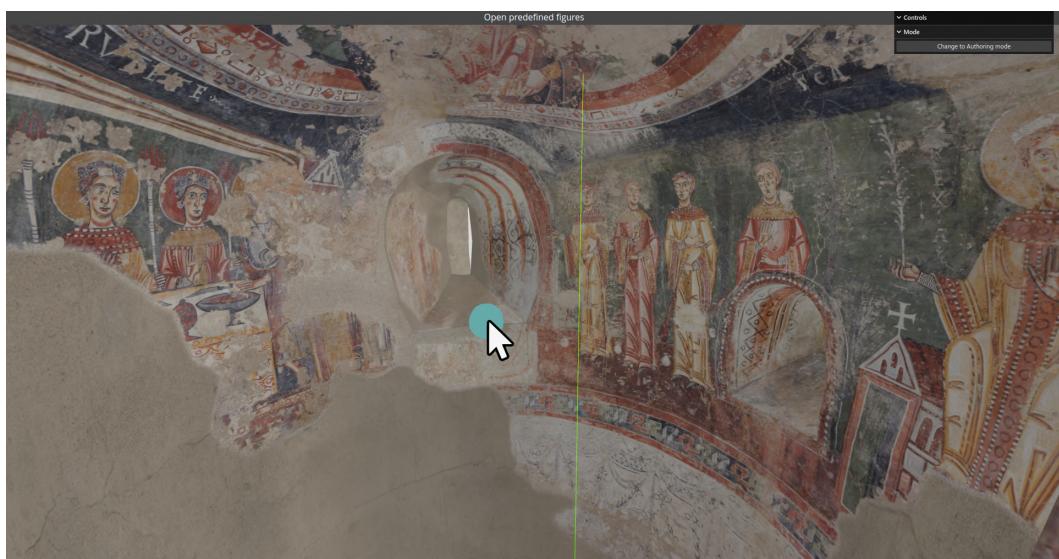


Figura 104: Esfera generada al mode Inspecció
Font: Producció pròpia

A continuació, s'obrirà una pestanya amb el visor d'imatges en 2D amb les imatges incloses a l'esfera (vegeu Figura 105). Aquestes imatges s'han desplaçat pel visor de manera que no hi hagi oclusions, i respecti la posició relativa entre elles el màxim possible.

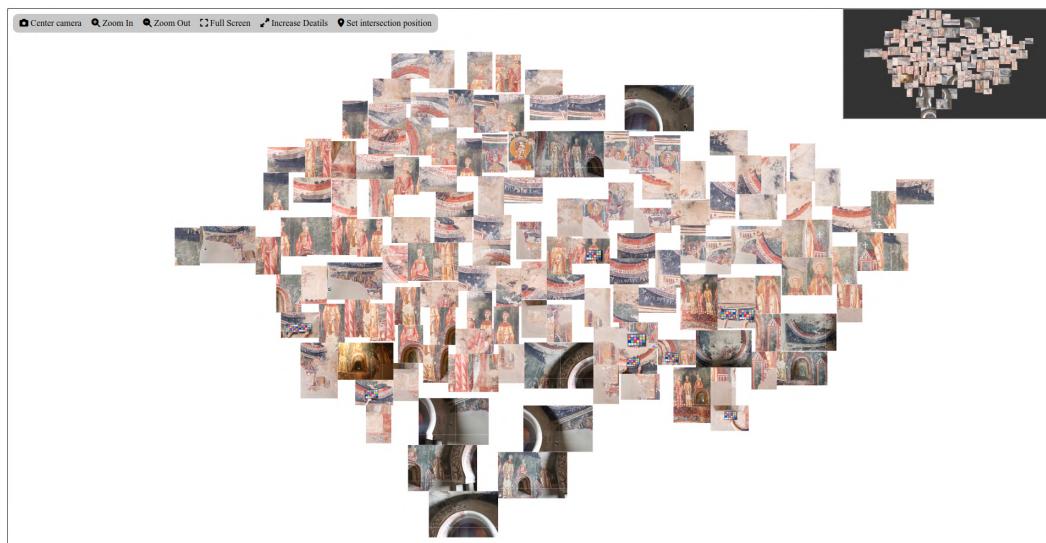


Figura 105: Imatges al visor
Font: Producció pròpia

Similar al cas anterior, l'usuari buscava una pintura en concret del model, que no ha trobat. A les imatges del visor, troba una imatge que correspon als detalls que volia visualitzar en el model, i fa clic sobre d'aquesta (vegeu Figura 106).



Figura 106: Imatge premuda per a navegar
Font: Producció pròpia

Tot seguit, a l'entorn 3D, la càmera enfoca la part del model que correspon a la imatge seleccionada (vegeu Figura 107). D'aquesta manera, l'usuari pot trobar parts del model de l'entorn 3D, trobant les imatges corresponents al visor 2D.



Figura 107: Navegació a la part del model corresponent
Font: Producció pròpria

11 Conclusions

11.1 Valoració personal

L'elaboració d'aquest projecte m'ha aportat satisfaccions i coneixements sobre àrees que no dominava. He pogut aprendre les relacions que existeixen entre les imatges i els models 3D. Amb aquests coneixements apresos a l'inici del projecte, he pogut enllaçar un entorn 3D amb un 2D, de manera que vagin donats per la mà, i no són simplement dos programes per separat. El resultat obtingut té una qualitat superior a la que m'esperava, ja que als inicis no em veia capaç de desenvolupar una aplicació amb aquesta complexitat. Tenia algunes nocions prèvies sobre motors gràfics com *Unity*, però moltes vegades, amb aquest últim passem per alt unes bases necessàries.

També, estic satisfet d'aportar al grup de recerca *ViRVIG* un projecte que els és d'utilitat. A més a més, historiadors amb màquines poc potents poden fer ús de l'aplicació desenvolupada. És a dir, que sento que no he fet un treball només per fer-lo, sinó que serà d'utilitat.

A més a més, cal comentar que aquest projecte ha sigut les bases d'un article sobre gràfics en el patrimoni cultural. Juntament amb el director del treball Imanol Munoz-Pandiella, i el codirector Carlos Andujar estem preparant un article sobre aquest treball a una conferència de prestigi internacional.

A continuació, volia mencionar que estic satisfet per tal de poder aplicar els coneixements de la menció de *Computació*. Conceptes d'assignatures com *Gràfics i Interacció i Disseny d'Interfícies* han sigut claus, sobretot els sistemes de coordenades ensenyats han sigut de gran utilitat. L'assignatura de *Projectes de Programació*, tot i no ser de l'especialitat, ha aportat coneixements per a planificar, gestionar i elaborar un projecte de certes característiques.

Finalment, voldria descriure que durant l'elaboració del projecte he obtingut una sèrie de coneixements. Primer, he après les bases del llenguatge *JavaScript*, juntament amb els frameworks *ThreeJS* i *OpenSeaDragon*. També, he après què són i com implementar la projecció de punts sobre una superfície, i a obtenir les coordenades en 2D d'aquesta superfície. A més a més, amb aquest projecte, he obtingut coneixements sobre la definició de figures, i quins són els paràmetres clau d'aquestes. Per últim, m'he emportat una lliçó. No podem passar per alt l'eficiència d'un programa. Moltes vegades no li donem la importància que realment té, i sense aplicar tècniques que ens permeten optimitzar-la, molts programes no serien capaços de ser executats per la majoria de màquines.

11.2 Reptes

Descripció de reptes durant el transcurs del treball Durant el transcurs del treball he trobat moltes dificultats i reptes que he hagut de solucionar. Algunes d'aquestes dificultats han sigut majors que altres, i a continuació llisto els 4 reptes més grans de tot el projecte.

El primer gran repte que vaig trobar va ser la càrrega d'imatges a l'entorn 3D.

Calia complir un gran nombre de tasques com la lectura de fitxers, la creació d'un pla, la lectura de textures, etc. Però d'aquestes, va ser el posicionament i rotació d'imatges el que em va donar més mals de cap. La *dessincronització* de les imatges respecte al model, juntament amb els càlculs de la rotació i la poca experiència amb el framework *ThreeJS* van fer que aquesta tasca fos més desafiant que la resta.

Un segon gran repte va ser la incorporació de les figures del cilindre i el pla. Aquestes figures necessitaven la implementació de funcions on calculava quines imatges incloure, la conversió de coordenades 3D a 2D, i la representació de la figura a l'entorn. Individualment, cadascuna d'aquestes tasques portaven la seva dificultat, i un petit error en un càlcul, feia que no funcionés res de la figura.

Dels quatre grans reptes, la resolució d'occlusions de les imatges al visor 2D va ser la que menys mals de cap em va donar. El càlcul en si no va ser el problemàtic. En provar l'algorisme, aquest m'indicava que no hi havia més occlusions, però podies observar al visor que encara se'n produïen. Fent els càlculs manuals també m'indicava que les occlusions estaven resoltes. Després de tornar-me a llegir la documentació d'*OpenSeaDragon*, vaig trobar que l'origen de coordenades de la imatge era la cantonada superior esquerra, en contres del centre de la imatge, havent de refer tots els càlculs.

L'últim repte que descric és el més gran. Aquest és l'eficiència de l'aplicació. En tractar amb imatges de tanta qualitat, en tractar unes poques, l'entorn 3D i el visor 2D quedaven bloquejats. Cada problema individual no era difícil de solucionar, però és la quantitat de problemes d'eficiència el que fa que aquest sigui el repte més gran.

11.3 Assoliment d'objectius

Per a dur a terme aquest projecte, es van proposar una sèrie d'objectius, subobjectius i requeriments. Aquests estan definits als capítols 3.1 i 3.2. A continuació es farà una valoració de l'assoliment de cadascun d'aquests.

1. **Estudiar la relació entre imatges 2D i models 3D.** Aquest objectiu s'ha assolit correctament com a pas previ al desenvolupament de l'entorn 3D. Tot i que no figuri cap estudi formal a la memòria, és un estudi que s'ha fet.
 - (a) **Estudiar la relació entre les imatges 2D amb l'espai 3D que es van captar.** Aquest objectiu s'ha assolit correctament com a pas previ al desenvolupament de l'entorn 3D. Tot i que no figuri cap estudi formal a la memòria, és un estudi que s'ha fet.
 - (b) **Estudiar la relació entre els espais de captació físics i els models 3D.** De la mateixa forma que el subobjectiu anterior, també s'ha assolit correctament com a pas previ al desenvolupament de l'entorn 3D. Tot i que no figuri cap estudi formal a la memòria, és un estudi que s'ha fet.
2. **Visualitzar les imatges 2D tenint en compte l'espai 3D.** Aquest objectiu s'ha assolit correctament, com es pot observar durant el transcurs dels capítols 8 i 9.
 - (a) **Visualitzar les imatges en un entorn 3D.** Aquest objectiu també s'ha

assolit correctament, com es pot observar durant el transcurs del capítol 8.

- (b) **Visualitzar les imatges amb els models 3D.** De la mateixa forma que el subobjectiu anterior, també s'ha assolit correctament, com es pot observar durant el transcurs del capítol 8.

A més a més de descriure l'assoliment dels objectius i subobjectius, també es valoren els requeriments, començant pels funcionals.

1. **Format d'imatges.** Aquest objectiu s'ha assolit correctament, tot i que totes les imatges estiguin en format *JPG*. La classe *textureLoader* [42] suporta els principals formats d'imatges, fent que no hi hagi complicacions en la lectura de formats com el *PNG*.
2. **Format de models.** Aquest objectiu s'ha assolit parcialment. Tal com s'ha especificat al capítol 8.1, es va llegir models de diferents formats. Tal com està l'aplicació, només permet llegir models en format *glTF*. En el cas que es volgués llegir altres formats, caldria fer una petita adaptació al codi.

A continuació es descriu l'assoliment dels requeriments no funcionals.

1. **Eficiència.** Aquest objectiu s'ha assolit correctament, tot i que és millorable. L'aplicació s'executa de manera fluida en la majoria d'accions, excepte el càlcul d'occlusions, que en grans quantitats d'imatges, pot trigar uns pocs segons. També, en carregar l'entorn 3D per primera vegada, pot trigar alguns segons.
2. **Usabilitat.** De la mateixa manera que a l'eficiència, aquest objectiu s'ha assolit correctament, però hi ha algunes millores. En general, la interfície et guia amb les funcionalitats permeses, però en algun cas no és així. Per exemple, no hi ha cap indicació sobre la possibilitat de navegar a una imatge a l'entorn 3D, fent clic a la imatge del visor.

11.4 Treballs futurs

Aquest projecte ha tingut l'objectiu d'abrir el màxim possible els objectius principals marcats. Així i tot, amb el termini establert no és possible incloure totes les funcionalitats desitjades. Dit això, a continuació es defineixen una sèrie de millors possibles a fer en un futur.

La primera millora és millorar la definició d'una esfera. En el treball, es crea una esfera mitjançant el punt central. Hi ha casos on t'interessa crear una esfera, però no trobes cap imatge que faci de centre. Per això, es podria definir una esfera mitjançant 4 punts de la seva superfície, o amb 2 punts i la seva normal [96].

Seguint el mateix article [96], podríem definir automàticament la figura que inclou les imatges de la forma més pròxima possible utilitzant *RANSAC*. Per exemple, en el cas de definició d'una esfera a partir de 2 punts, podríem iterar parells de punts utilitzant *RANSAC*, i agafar aquells 2 punts que aproximi millor l'esfera.

Una altra possible millora és no limitar-nos a l'esfera, el cilindre i el pla, i donar suport a qualsevol figura. En aquest cas, a partir de la parametrització d'aquesta,

implementar automàticament la funció que aplica la projecció de coordenades 3D a coordenades 2D.

A més a més, podríem afegir que l'usuari pogués visualitzar les imatges més properes després de fer clic sobre qualsevol part del model. Es podria implementar aplicant el *ray casting* per a obtenir la posició del punt en el model, i mostrar les imatges inferiors a una certa distància.

Finalment, m'hauria agradat poder escalar l'aplicació per tal que funcionés amb qualsevol monument. En aquest cas, l'usuari podria indicar un model, un fitxer *Bundler* i el seu llistat d'imatges, i representar l'escena segons aquests.

Referències

- [1] Google LLC. (n.d.). Google Photos. Google Photos. Retrieved February 21, 2024, from <https://www.google.com/photos/about/>
- [2] Universitat Politècnica de Catalunya & Universitat de Girona. (n.d.). ViRVIG. VirVIG. Retrieved February 22, 2024, from <https://www.virvig.eu/>
- [3] Brivio, P., Benedetti, L., Tarini, M., & ISTI-CNR Visual Computing Lab. (2013). PhotoCloud: interactive remote exploration of joint 2D and 3D datasets. ResearchGate. Retreived February 22, 2024, from https://www.researchgate.net/profile/Paolo-Cignoni/publication/260584610_PhotoCloud_Interactive_Remote_Exploration_of_Joint_2D_and_3D_Datasets/links/54e45e2d0cf2b2314f605a33/PhotoCloud-Interactive-Remote-Exploration-of-Joint-2D-and-3D-Datasets.pdf
- [4] Cignoni, P., Muntoni, A., Visual Computing Lab, & ISTI-CNR. (n.d.). MeshLab. MeshLab. Retrieved February 23, 2024, from <https://www.meshlab.net/>
- [5] WebGL Public Wiki. (2011, April 10). Getting started. Retrieved February 23, 2024, from https://www.khronos.org/webgl/wiki/Getting_Started
- [6] JavaScript 3D library. (n.d.). Three.js. Retrieved February 23, 2024, from <https://threejs.org/>
- [7] Unity Technologies. (n.d.). Unity Real-Time Development Platform — 3D, 2D, VR & AR Engine. Unity. Retrieved February 23, 2024, from <https://unity.com/>
- [8] Fengyuan, C. (n.d.). JavaScript Image Viewer. Viewer.js. Retrieved February 23, 2024, from <https://fengyuanchen.github.io/viewerjs/>
- [9] Semenov, D. (n.d.). JavaScript Image Gallery and Lightbox. PhotoSwipe. Retrieved February 23, 2024, from <https://photoswipe.com/>
- [10] OpenSeadragon. (n.d.). OpenSeaDragon. Retrieved February 23, 2024, from <https://openseadragon.github.io/>
- [11] GitHub, Inc. (2022, August). GitHub: Let's build from here. GitHub. Retrieved February 4, 2024, from <https://github.com/>
- [12] Overleaf, online LaTeX editor. (n.d.). Overleaf. Retrieved February 4, 2024, from <https://overleaf.com/>
- [13] The home of Scrum. (n.d.). Scrum.org. Retrieved February 25, 2024, from <https://scrum.org/>
- [14] Git. (n.d.). Git. Retrieved February 25, 2024, from <https://git-scm.com/>
- [15] GitHub, Inc. (n.d.). About projects. GitHub Docs. Retrieved February 25, 2024, from <https://docs.github.com/en/issues/planning-and-tracking-with-projects/learning-about-projects/about-projects>
- [16] Facultat d'Informàtica de Barcelona, FIB, UPC. (2024). Normativa del Treball de Fi de Grau del grau en Enginyeria Informàtica de la FIB. Retrieved February 25, 2024, from

https://www.fib.upc.edu/sites/fib/files/documents/actes/normativatfg-gei_document_final.pdf

- [17] Endesa Energía. (n.d.). Tarifa one luz. Endesa. Retrieved March 6, 2024, from <https://www.endesa.com/es/luz-y-gas/luz/one/tarifa-one-luz>
- [18] PCComponentes. (n.d.). Medion Erazer P6705. PC Componentes. Retrieved March 6, 2024, from <https://www.pccomponentes.com/medion-erazer-p6705-intel-core-i7-8750h-16gb-256gb-ssd-gtx-1050ti-156>
- [19] O2 España. (n.d.). Fibra 300Mb. O2. Retrieved March 7, 2024, from https://dlvt1u0vrr0ux.cloudfront.net/media/public/resumen_contrato_solo_fibra_300mb_zl_1.pdf
- [20] Idealista. (n.d.). Evolución del precio de la vivienda en venta en Gràcia. Retrieved March 7, 2024, from <https://www.idealista.com/sala-de-prensa/informes-precio-vivienda/venta/cataluna/barcelona-provincia/sabadell/gracia/>
- [21] IKEA. (n.d.). LAGKAPTEN. Retrieved March 8, 2024, from <https://www.ikea.com/es/es/p/lagkapten-alex-escritorio-efecto-roble-tinte-blanco-blanco-s09432014/>
- [22] IKEA. (n.d.). FLINTAN. Retrieved March 8, 2024, from <https://www.ikea.com/es/es/p/flintan-silla-trabajo-negro-10489028/>
- [23] Adventia Spain. (n.d.). Cuánto cobra un/a Desarrollador de software/Desarrolladora de software en Barcelona. InfoJobs Salarios. Retrieved March 9, 2024, from <https://salarios.infojobs.net/desarrollador-de-software-desarrolladora-de-software/barcelona>
- [24] Adventia Spain. (n.d.). Cuánto cobra un Gestor/a de proyectos de TICs en Barcelona. InfoJobs Salarios. Retrieved March 9, 2024, from <https://salarios.infojobs.net/gestor-a-de-proyectos-de-tics/barcelona>
- [25] Adventia Spain. (n.d.). Cuánto cobra un Analista de programación en Barcelona. InfoJobs Salarios. Retrieved March 9, 2024, from <https://salarios.infojobs.net/analista-de-programacion/barcelona>
- [26] Adventia Spain. (n.d.). Cuánto cobra un Redactor de periódico/Redactora de periódico en Barcelona. InfoJobs Salarios. Retrieved March 9, 2024, from <https://salarios.infojobs.net/redactor-de-periodico-redactora-de-periodico/barcelona>
- [27] Redactor articulos salarios en Barcelona. (2024). Jooble. Retrieved March 9, 2024, from <https://es.jooble.org/salary/redactor-articulos/Barcelona#yearly>
- [28] Personio. (2022, April 23). El coste de los trabajadores para tu empresa: ¿qué es y cómo calcularlo? Retrieved March 9, 2024, from <https://www.personio.es/glosario/coste-de-trabajador-para-empresa/>
- [29] Ministerio de la presidencia, justicia y relaciones con las cortes, Gobierno de España. (2018, March 6). Resolución de 22 de febrero de 2018, de la Dirección General de Empleo, por la que se registra y publica el XVII Convenio colectivo estatal de empresas de consultoría y estudios de mercado y de

la opinión pública. Boletín Oficial Del Estado. Retrieved March 9, 2024, from https://www.boe.es/diario_boe/txt.php?id=BOE-A-2018-3156

- [30] Generalitat de Catalunya. (2023, May 4). Factor d'emissió de l'energia elèctrica: el mix elèctric. Canvi Climàtic. Retrieved March 9, 2024, from https://canviclimatic.gencat.cat/ca/actua/factors_demissio_associats_a_lenergia/index.html
- [31] Wikipedia contributors. (2024, April 17). Wavefront .obj file. Wikipedia. Retrieved April 27, 2024, from https://en.wikipedia.org/wiki/Wavefront_.obj_file
- [32] ThreeJS. (n.d.). OBJLoader. Docs. Retrieved February 26, 2024, from <https://threejs.org/docs/#examples/en/loaders/OBJLoader>
- [33] ThreeJS. (n.d.). MTLLoader. Docs. Retrieved February 26, 2024, from <https://threejs.org/docs/#examples/en/loaders/MTLLoader>
- [34] Wikipedia contributors. (2024, March 27). FBX. Wikipedia. Retrieved April 17, 2024, from <https://en.wikipedia.org/wiki/FBX>
- [35] ThreeJS. (n.d.). FBXLoader. Docs. Retrieved February 26, 2024, from <https://threejs.org/docs/#examples/en/loaders/FBXLoader>
- [36] Blender Foundation. (n.d.). Home of the Blender project. blender.org. Retrieved February 26, 2024, from <https://www.blender.org/>
- [37] ThreeJS. (n.d.). Loading 3D models. Docs. Retrieved February 26, 2024, from <https://threejs.org/docs/#manual/en/introduction>Loading-3D-models>
- [38] ThreeJS. (n.d.). GLTFLoader. Docs. Retrieved February 26, 2024, from <https://threejs.org/docs/#examples/en/loaders/GLTFLoader>
- [39] Bundler User's Manual. (2008). Cornell University. Retrieved February 28, 2024, from <https://www.cs.cornell.edu/~snavely/bundler/bundler-v0.4-manual.html>
- [40] ThreeJS. (n.d.). FileLoader. Docs. Retrieved February 26, 2024, from <https://threejs.org/docs/#api/en/loaders/FileLoader>
- [41] ThreeJS. (n.d.). PlaneGeometry. Docs. Retrieved February 26, 2024, from <https://threejs.org/docs/#api/en/geometries/PlaneGeometry>
- [42] ThreeJS. (n.d.). TextureLoader. Docs. Retrieved February 26, 2024, from <https://threejs.org/docs/#api/en/loaders/TextureLoader>
- [43] ImageMagick Studio LLC. (n.d.). ImageMagick – Mastering Digital Image Alchemy. ImageMagick. Retrieved April 28, 2024, from <https://imagemagick.org/index.php>
- [44] De Jong, J. (n.d.). math.js — an extensive math library for JavaScript and Node.js. math.js. Retrieved March 1, 2024, from <https://mathjs.org/>
- [45] ThreeJS. (n.d.). Object3D.lookAt. Docs. Retrieved March 1, 2024, from <https://threejs.org/docs/#api/en/core/Object3D.lookAt>

- [46] ThreeJS. (n.d.). Vector3.setFromMatrixPosition. Docs. Retrieved May 19, 2024, from <https://threejs.org/docs/#api/en/math/Vector3.setFromMatrixPosition>
- [47] ThreeJS. (n.d.). Vector3.setFromMatrixScale. Docs. Retrieved May 19, 2024, from <https://threejs.org/docs/index.html#api/en/math/Vector3.setFromMatrixScale>
- [48] ThreeJS. (n.d.). Euler.setFromRotationMatrix. Docs. Retrieved May 19, 2024, from <https://threejs.org/docs/#api/en/math/Euler.setFromRotationMatrix>
- [49] lil-gui 0.19.2. (n.d.). Lil-gui. Retrieved March 16, 2024, from <https://lil-gui.georgealways.com/>
- [50] Wikipedia contributors. (2024, February 21). Ray casting. Wikipedia. Retrieved May 1, 2024, from https://en.wikipedia.org/wiki/Ray_casting
- [51] ThreeJS. (n.d.). Raycaster. Docs. Retrieved March 4, 2024, from <https://threejs.org/docs/#api/en/core/Raycaster>
- [52] Mozilla Foundation. (2024, March 4). Element.tagName property. MDN Web Docs. Retrieved May 1, 2024, from <https://developer.mozilla.org/en-US/docs/Web/API/Element/tagName>
- [53] Wikipedia contributors. (2023, November 4). Mouseover. Wikipedia. Retrieved May 2, 2024, from <https://en.wikipedia.org/wiki/Mouseover>
- [54] ThreeJS. (n.d.). Line. Docs. Retrieved April 26, 2024, from <https://threejs.org/docs/#api/en/objects/Line>
- [55] ThreeJS. (n.d.). BufferGeometry. Docs. Retrieved April 26, 2024, from <https://threejs.org/docs/index.html#api/en/core/BufferGeometry>
- [56] ThreeJS. (n.d.). LineBasicMaterial. Docs. Retrieved April 26, 2024, from <https://threejs.org/docs/#api/en/materials/LineBasicMaterial>
- [57] Understanding focal length. (n.d.). Canon Georgia. Retrieved May 25, 2024, from <https://www.canon.ge/pro/infobank/understanding-focal-length/>
- [58] Wikipedia contributors. (2024, May 17). Spherical coordinate system. Wikipedia. Retrieved May 25, 2024, from https://en.wikipedia.org/wiki/Spherical_coordinate_system
- [59] Wikipedia contributors. (2024, April 17). Plane (mathematics). Wikipedia. Retrieved May 25, 2024, from [https://en.wikipedia.org/wiki/Plane_\(mathematics\)](https://en.wikipedia.org/wiki/Plane_(mathematics))
- [60] Wikipedia contributors. (2024, April 17). Euclidean planes in three-dimensional space. Wikipedia. Retrieved May 25, 2024, from https://en.wikipedia.org/wiki/Euclidean_planes_in_three-dimensional_space
- [61] ThreeJS. (n.d.). Plane. Docs. Retrieved March 7, 2024, from <https://threejs.org/docs/#api/en/math/Plane>
- [62] ThreeJS. (n.d.). Plane SetFromCoplanarPoints. Docs. Retrieved March 7, 2024, from <https://threejs.org/docs/#api/en/math/Plane.setFromCoplanarPoints>

- [63] ThreeJS. (n.d.). BoxGeometry. Docs. Retrieved March 28, 2024, from <https://threejs.org/docs/#api/en/geometries/BoxGeometry>
- [64] Wikipedia contributors. (2023, March 5). Planar projection. Wikipedia. Retrieved May 25, 2024, from https://en.wikipedia.org/wiki/Planar_projection
- [65] Wikipedia contributors. (2024, March 11). Orthographic projection. Wikipedia. Retrieved May 25, 2024, from https://en.wikipedia.org/wiki/Orthographic_projection
- [66] ThreeJS. (n.d.). Plane ProjectPoint. Docs. Retrieved March 7, 2024, from <https://threejs.org/docs/#api/en/math/Plane.projectPoint>
- [67] Wikipedia contributors. (2024, May 15). Sphere. Wikipedia. Retrieved May 25, 2024, from <https://en.wikipedia.org/wiki/Sphere>
- [68] ThreeJS. (n.d.). SphereGeometry. Docs. Retrieved March 28, 2024, from <https://threejs.org/docs/#api/en/geometries/SphereGeometry>
- [69] Wikipedia contributors. (2024, May 13). Cylinder. Wikipedia. Retrieved May 26, 2024, from <https://en.wikipedia.org/wiki/Cylinder>
- [70] ThreeJS. (n.d.). CylinderGeometry. Docs. Retrieved March 29, 2024, from <https://threejs.org/docs/#api/en/geometries/CylinderGeometry>
- [71] Wikipedia contributors. (2024, May 23). Cylindrical coordinate system. Wikipedia. Retrieved May 26, 2024, from https://en.wikipedia.org/wiki/Cylindrical_coordinate_system
- [72] ThreeJS. (n.d.). Line3 ClosestPointToPoint. Docs. Retrieved March 9, 2024, from <https://threejs.org/docs/#api/en/math/Line3.closestPointToPoint>
- [73] ThreeJS. (n.d.). Vector3 AngleTo. Docs. Retrieved March 9, 2024, from <https://threejs.org/docs/#api/en/math/Vector3.angleTo>
- [74] OpenSeadragon. (n.d.). Image Tile Source. Retrieved March 4, 2024, from <https://openseadragon.github.io/examples/tilesource-image/>
- [75] GeeksforGeeks. (2022, June 7). Maximum length of a URL in different browsers. GeeksforGeeks. Retrieved May 30, 2024, from <https://www.geeksforgeeks.org/maximum-length-of-a-url-in-different-browsers/>
- [76] Mozilla Corporation. (2023, April 8). Window: localStorage property. MDN Web Docs. Retrieved May 30, 2024, from <https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage>
- [77] Mozilla Corporation. (2023, April 8). Window: sessionStorage property. MDN Web Docs. Retrieved May 30, 2024, from <https://developer.mozilla.org/en-US/docs/Web/API/Window/sessionStorage>
- [78] Wikipedia contributors. (2024, May 13). Deep Zoom. Wikipedia. Retrieved June 1, 2024, from https://en.wikipedia.org/wiki/Deep_Zoom

- [79] Wikipedia contributors. (2024, May 7). Level of detail (computer graphics). Wikipedia. Retrieved June 1, 2024, from [https://en.wikipedia.org/wiki/Level_of_detail_\(computer_graphics\)](https://en.wikipedia.org/wiki/Level_of_detail_(computer_graphics))
- [80] OpenSeadragon. (n.d.). Creating Zooming Images. Retrieved March 7, 2024, from <https://openseadragon.github.io/examples/creating-zooming-images/>
- [81] VoidVolker. (2023, September 27). MagickSlicer: Map tiles generator. GitHub. Retrieved March 7, 2024, from <https://github.com/VoidVolker/MagickSlicer>
- [82] Daniel Gasienica. (2022, February 5). deepzoom.py: Python Deep Zoom Tools. GitHub. Retrieved March 9, 2024, from <https://github.com/openzoom/deepzoom.py>
- [83] Wikipedia contributors. (2024a, May 7). Exif. Wikipedia. Retrieved June 1, 2024, from <https://en.wikipedia.org/wiki/Exif>
- [84] OpenSeadragon. (n.d.). Multi-Image. Retrieved March 19, 2024, from <https://openseadragon.github.io/examples/multi-image/>
- [85] OpenSeadragon Class: World. (n.d.). Retrieved May 15, 2024, from <https://openseadragon.github.io/docs/OpenSeadragon.World.html# getItemAt>
- [86] OpenSeadragon Class: World. (n.d.). Retrieved May 15, 2024, from <https://openseadragon.github.io/docs/OpenSeadragon.World.html>
- [87] OpenSeadragon Class: TiledImage. (n.d.). Retrieved May 15, 2024, from <https://openseadragon.github.io/docs/OpenSeadragon.TiledImage.html# setHeight>
- [88] OpenSeadragon Class: TiledImage. (n.d.). Retrieved May 18, 2024, from <https://openseadragon.github.io/docs/OpenSeadragon.TiledImage.html# setPosition>
- [89] OpenSeadragon Class: EventSource. (n.d.). Retrieved May 20, 2024, from <https://openseadragon.github.io/docs/OpenSeadragon.EventSource.html>
- [90] OpenSeadragon Class: Viewer. (n.d.). Retrieved May 20, 2024, from <https://openseadragon.github.io/docs/OpenSeadragon.Viewer.html#.event:open>
- [91] Viewport Coordinates — OpenSeadragon. (n.d.). Retrieved May 23, 2024, from <https://openseadragon.github.io/examples/viewport-coordinates/>
- [92] OpenSeadragon Class: Viewer. (n.d.). Retrieved May 29, 2024, from <https://openseadragon.github.io/docs/OpenSeadragon.Viewer.html#.event:canvas-click>
- [93] OpenSeadragon Class: Vieweport. (n.d.). Retrieved May 29, 2024, from <https://openseadragon.github.io/docs/OpenSeadragon.Viewport.html#pointFromPixel>
- [94] Mozilla Corporation. (2024, March 13). Window: storage event. Retrieved May 29, 2024, from https://developer.mozilla.org/en-US/docs/Web/API/Window/storage_event
- [95] Fonticons, Inc. (n.d.). Icons. Font Awesome. Retrieved June 8, 2024, from <https://fontawesome.com/icons>

- [96] Universität Bonn, Computer Graphics Group. (1981). Efficient RAN-SAC for Point-Cloud Shape Detection. Retrieved April 16, 2024, from <https://www.hinkali.com/Education/PointCloud.pdf>