UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Facultad de Informática de Barcelona
Grado en Ingeniería Informática
Especialidad de Computación

Estudio y evaluación de técnicas de navegación para Realidad Virtual colaborativa

Trabajo de Fin de Grado

José Luis Pontón Martínez

Directora: Nuria Pelechano Gómez
Codirector: Carlos Andújar Gran

2 de julio de 2020

Resumen

En la actualidad existe una gran variedad de técnicas para navegar por entornos virtuales con cascos de realidad virtual, los problemas más habituales en la navegación son los mareos y la desorientación espacial. Cuando se trabaja con realidad virtual colaborativa, varios usuarios comparten espacio físico y virtual, por lo tanto, la representación visual de los usuarios es importante para facilitar la interacción y evitar colisiones.

Este trabajo presenta una aplicación que permite la conexión de múltiples usuarios a un entorno virtual colaborativo, donde se han estudiado, desarrollado y evaluado diferentes técnicas de navegación y representación visual de usuarios. La aplicación permite la navegación de múltiples usuarios en un mismo entorno virtual, así como la interacción entre participantes.

Las técnicas de navegación que se presentan ofrecen un buen equilibrio entre la flexibilidad de desplazamiento, y la inducción de mareos o desorientación espacial. También, se han desarrollado técnicas que permiten conocer la posición de los demás participantes en todo momento, así como hacia dónde están mirando y apuntado, para facilitar la comunicación, evitar colisiones y conseguir interacciones más naturales.

Resum

En l'actualitat existeix una gran varietat de tècniques per a navegar per entorns virtuals amb cascos de realitat virtual, els problemes més habituals en la navegació són els marejos i la desorientació espacial. Quan es treballa amb realitat virtual col·laborativa, diversos usuaris comparteixen espai físic i virtual, per tant, la representació visual dels usuaris és important per a facilitar la interacció i evitar col·lisions.

Aquest treball presenta una aplicació que permet la connexió de múltiples usuaris a un entorn virtual col·laboratiu, on s'han estudiat, desenvolupat i avaluat diferents tècniques de navegació i representació visual d'usuaris. L'aplicació permet la navegació de múltiples usuaris en un mateix entorn virtual, així com la interacció entre participants.

Les tècniques de navegació que es presenten ofereixen un bon equilibri entre la flexibilitat de desplaçament, i la inducció de marejos o desorientació espacial. També, s'han desenvolupat tècniques que permeten conèixer la posició dels altres participants en tot moment, així com cap a on estan mirant i apuntant, per a facilitar la comunicació, evitar col·lisions i aconseguir interaccions més naturals.

Abstract

Currently there is a wide variety of techniques to navigate virtual environments with virtual reality headsets, the most common problems in navigation are cybersickness and spatial disorientation. When working with collaborative virtual reality, several users share physical and virtual space, therefore, the visual representation of users is important to facilitate interaction and avoid collisions.

This work presents an application that allows the connection of multiple users to a collaborative virtual environment, where different navigation techniques and visual representation of users have been studied, developed and evaluated. The application allows multiple users to navigate in the same virtual environment, as well as the interaction between participants.

The navigation techniques presented offer a good balance between flexibility of movement, and the induction of dizziness or spatial disorientation. Also, techniques have been developed to know the position of the other participants at all times, as well as where they are looking and pointing in order to facilitate communication, avoid collisions and achieve more natural interactions.

Índice

1. Introducción y contextualización **8**
- 1.1. Contexto del proyecto . . . 10
- 1.2. Realidad Virtual . . . 10
- 1.3. Colaborativo . . . 10
- 1.4. Identificación del problema . . . 11
  - 1.4.1. Navegación . . . 11
  - 1.4.2. Interacción . . . 12
- 1.5. Agentes implicados . . . 13

2. Justificación **14**
- 2.1. Estudio soluciones existentes . . . 14
- 2.2. Herramientas . . . 15
- 2.3. Conclusiones . . . 18

3. Alcance **19**
- 3.1. Objetivos . . . 19
- 3.2. Requerimientos . . . 19
  - 3.2.1. Requerimientos funcionales . . . 19
  - 3.2.2. Requerimientos no funcionales . . . 20

4. Navegación en realidad virtual colaborativa **21**
- 4.1. *Networking* . . . 21
  - 4.1.1. Arquitectura de comunicación . . . 21
  - 4.1.2. Implementación del código de red . . . 25
- 4.2. Técnicas de navegación . . . 27
  - 4.2.1. Espacio físico y virtual . . . 27
  - 4.2.2. Representación visual del usuario . . . 30
  - 4.2.3. Vuelo libre . . . 31
  - 4.2.4. Teletransporte . . . 33
  - 4.2.5. Representación visual doble . . . 36
  - 4.2.6. Puntos de vista . . . 38
  - 4.2.7. Videocámara . . . 39
- 4.3. Evaluación de los modos de navegación . . . 41
  - 4.3.1. Pruebas piloto . . . 41
  - 4.3.2. Preparación del estudio con usuarios . . . 44

5. Representación visual de usuarios **46**
- 5.1. *Avatar* a partir de cámaras . . . 46
- 5.2. Textura de profundidades . . . 48
  - 5.2.1. Cálculo de las profundidades . . . 49
  - 5.2.2. Segmentación mediante *thresholding* . . . 51

5.2.3. Segmentación por inundación 54
5.2.4. Otros usos 61
5.3. OpenPose 63
5.3.1. Segmentación a nivel de píxel 64
5.3.2. Segmentación con *SLIC* 67
5.4. Segmentación a partir de primitivas geométricas 73
5.4.1. Aproximación mediante primitivas geométricas 73
5.4.2. Red neuronal para aproximar el radio 75
5.5. Implementación en la *GPU* 87
5.6. Resultados y comparación 94

6. Gestión del proyecto **99**
6.1. Metodología 99
6.1.1. Modificaciones 99
6.1.2. Herramientas 100
6.2. Planificación temporal 100
6.2.1. Descripción de las tareas 101
6.2.2. Recursos humanos 104
6.2.3. Recursos materiales 105
6.2.4. Gestión del riesgo 105
6.3. Gestión económica 108
6.3.1. Costes de personal 108
6.3.2. Costes genéricos 109
6.3.3. Contingencia 109
6.3.4. Imprevistos 111
6.3.5. Coste total 112
6.3.6. Control de gestión 112
6.4. Cambios en la planificación 113
6.4.1. Cambios en las tareas 113
6.4.2. Cambios en el presupuesto 117
6.4.3. Cambios en los objetivos 117
6.5. Leyes y regulaciones 117
6.6. Informe de sostenibilidad 119
6.6.1. Dimensión ambiental 119
6.6.2. Dimensión económica 121
6.6.3. Dimensión social 122

7. Conclusiones y trabajo futuro **124**

Bibliografía **126**

Índice de figuras

1. *Grupo de usuarios usando RV en un entorno semi-inmersivo* 8
2. *Dos personas usando cascos de RV* 9
3. *Método de navegación basado en teletransporte* 15
4. *Ejemplos de dispositivos de RV* 16
5. *Arquitecturas de comunicación* 21
6. *Comunicación Cliente-Servidor* 23
7. *Unity Multiplayer API* 24
8. *Implementación código red* 25
9. *Representación habitación física con bases de HTC Vive* . . . 28
10. *Representación movimiento usuarios* 30
11. *Avatares de dos usuarios* 30
12. *Usuario desplazándose con vuelo libre* 32
13. *Usuario restaurando distancias físicas mediante vuelo libre* . . 33
14. *Usuario empleando el teletransporte* 34
15. *Teletransporte sin y con ajuste automático* 35
16. *Secuencia de un teletransporte* 36
17. *Representación visual doble de un usuario* 37
18. *Interfaz de puntos de vista* 39
19. *Usuario usando videocámara* 40
20. *HTC Vive Pro* 47
21. *Planos de la cámara vistos en la escena virtual* 48
22. *Textura de profundidades* 48
23. *Epiline de un punto* 49
24. *Representación del cálculo de la profundidad de un punto* . . 50
25. *Método para proyectar puntos 3D al plano de la cámara* . . . 51
26. *Normalización de los puntos en espacio de cámara* 53
27. *Segmentación mediante thresholding en la textura de profundidades* 54
28. *Diagrama general segmentación por inundación* 55
29. *Proceso de escalado de una textura* 56
30. *Pseudocódigo algoritmo inundación* 58
31. *Suavizado máscara después de la inundación* 61
32. *Detección de objetos con la textura de profundidades* 63
33. *Puntos de interés detectados por OpenPose* 64
34. *Píxeles iniciales para la segmentación a nivel de píxel* 65
35. *Comparación funciones de distancia de colores* 66
36. *Ejemplo algoritmo SLIC* 67
37. *Pseudocódigo algoritmo SLIC* 68
38. *Pseudocódigo algoritmo segmentación basada en grafos* 70
39. *Representación del mínimo corte de la segmentación basada en grafos* 71

40. *Resultado segmentación basada en grafos* 72
41. *Segmentación a partir de primitivas geométricas y OpenPose* 74
42. *Modelo 3D creado con SMPL* 76
43. *Rigging y asignación de pesos al modelo 3D* 77
44. *Ejemplo animación de caminar en un modelo 3D* 77
45. *División del modelo 3D en colores para su posterior segmentación* 78
46. *Pseudocódigo algoritmo generación muestras en Blender* . . . 79
47. *Imágenes resultantes del script de Blender* 79
48. *Representación visual del cálculo del error de las primitivas* . 80
49. *Radio de las primitivas en función del factor de penalización* 82
50. *Comparación error y predicciones de diferentes capas internas* 84
51. *Comparación error y predicciones de diferentes números de neuronas por capa.* 86
52. *Resultado de aplicar diferentes operaciones morfológicas* . . . 89
53. *Kernel de la operación morfológica erosión* 90
54. *Resultado de aplicar el filtro de suavizado Gaussian Blur* . . . 90
55. *Kernel del filtro Gaussian Blur* 92
56. *Rasterizado de 3 círculos y 3 rectángulos* 92
57. *Kernels para rasterizar círculos y rectángulos* 93
58. *Comparación técnicas segmentación del usuario sin suavizado* 95
59. *Comparación técnicas segmentación del usuario con suavizado* 96
60. *Diagrama de Gantt Final* 107
61. *Diagrama de Gantt Inicial* 115

Índice de tablas

1. *Comparativa cascos RV* 17
2. *Comparativa colas de prioridades para la inundación* 60
3. *Error conjunto de test en función del número de capas* 85
4. *Error conjunto de test en función del número de neuronas* . . 85
5. *Rendimiento en función de la técnica de segmentación* 95
6. *Planificación final en tareas* 106
7. *Costes personal* 109
8. *Partidas por tarea* 110
9. *Costes hardware* 111
10. *Contingencia* 111
11. *Imprevistos* 112
12. *Presupuesto total* 112
13. *Planificación inicial en tareas* 114
14. *Matriz de sostenibilidad* 119
15. *Consumo dispositivos* 120
16. *Consumo dispositivos por tarea* 120

1 Introducción y contextualización

La realidad virtual (RV) se podría definir como la representación de entornos tridimensionales (escenas) producidas por un sistema informático, que dan la sensación de su existencia real. En la actualidad, gracias al aumento de dispositivos *hardware* para RV, existen numerosas aplicaciones que permiten una alta inmersión en escenas virtuales. Los ámbitos de uso son cada vez más numerosos, algunos ejemplos son los videojuegos, el entrenamiento de profesionales en sectores como la medicina, y la visualización de modelos en la arquitectura [1].

Inicialmente, las escenas eran relativamente simples, con métodos de interacción y navegación muy básicos. No obstante, con el auge de la tecnología, surge la necesidad de representar escenas cada vez más grandes y complejas, así como métodos de navegación para recorrerlas.

En los ámbitos donde la cooperación es esencial, se están empezando a desarrollar aplicaciones donde varios usuarios pueden colaborar o coexistir en una misma escena virtual. La colaboración en RV busca la interacción natural entre los diferentes usuarios, de forma que estos interactúen como lo harían en la realidad. En la Figura 1 se muestra un grupo de usuarios haciendo uso de realidad virtual colaborativa en un entorno semi-inmersivo tipo *powerwall*. La ventaja de este sistema, es que las imágenes se proyectan en una pantalla de grandes dimensiones, y varios usuarios pueden estar observando el entorno virtual al mismo tiempo, mientras se ven entre ellos y por tanto facilita la colaboración

Figura 1: Grupo de usuarios debatiendo un diseño en un entorno semi-inmersivo de RV. Fuente: [2]

Actualmente, se están popularizando los sistemas inmersivos mediante cascos de realidad virtual. Estos sistemas ofrecen una experiencia mucho más realista, ya que el usuario se encuentra envuelto por la escena de realidad virtual en cualquier dirección de visión. El problema surge cuando varios usuarios quieren colaborar, porque la visualización virtual sustituye todo el campo de visión del usuario, y por tanto no podemos ver a los otros usuarios que comparten el entorno virtual utilizando también un casco. Esto ha supuesto nuevos retos de investigación en realidad virtual colaborativa para entornos inmersivos. Tanto las técnicas de interacción entre usuarios como la navegación cuando varios usuarios colaboran en RV presenta grandes dificultades. En RV, son habituales los mareos y la desorientación espacial durante la navegación, y estos problemas pueden empeorar todavía más en RV colaborativa, si solo uno de los usuarios tiene el control de la navegación. Además, dado que el casco de realidad virtual no nos permite ver al otro usuario, es necesario desarrollar técnicas que permitan conocer la posición del otro en todo momento, así como hacia dónde está mirando, o si está apuntando a algo para facilitar la comunicación y evitar colisiones.

En la Figura 2 se muestran dos personas haciendo uso de la RV inmersiva mediante cascos y controladores. Se puede ver que el casco les impide ver dónde se encuentran los demás participantes.

Figura 2: Dos personas colaborando en RV inmersiva mediante el uso de cascos. Fuente: [3]

Este trabajo de fin de grado (TFG) desarrolla diferentes técnicas de navegación y colaboración en RV, así como su estudio y evaluación para proponer soluciones a los problemas mencionados.

1.1 Contexto del proyecto

Dentro del Grado en Ingeniería Informática, impartido por la Facultad de Informática de Barcelona, existen diferentes menciones. Este trabajo de fin de grado pertenece a la mención de computación, concretamente, al ámbito de los gráficos por ordenador o computación gráfica. El proyecto se realiza dentro del marco de desarrollo del grupo de investigación ViRVIG [4] especializado, entre otros temas, en RV.

El proyecto parte del trabajo realizado para la Sagrada Familia [1] para la visualización de modelos tridimensionales en RV. La Sagrada Familia es una basílica de Barcelona diseñada por el arquitecto Antoni Guadí; iniciada en 1882, en la actualidad todavía se encuentra en construcción. Es uno de los máximos exponentes de la arquitectura modernista catalana y uno de los monumentos más visitados de España.

En el proyecto inicial, los arquitectos de la Sagrada Familia podían visualizar las escenas virtuales de forma individual en RV, hecho que les permitía debatir y presentar los diseños con mayor fidelidad que con herramientas clásicas. En este trabajo se amplían las soluciones existentes a varios usuarios, de forma que estos puedan interactuar con mayor naturalidad.

1.2 Realidad Virtual

Cuando se habla de Realidad Virtual (RV), comúnmente se refiere a la simulación de una realidad a través de un entorno informático. En consecuencia, mediante aplicaciones *software* y diferentes dispositivos *hardware*, el objetivo es crear una experiencia inmersiva donde el usuario tenga la sensación de estar interactuando con elementos reales.

Es habitual pensar que RV solo contempla la visualización de las escenas, no obstante, para ofrecer una experiencia inmersiva es necesario tener en cuenta otros factores, ya sean otros sentidos como el oído, o conceptos más generales como la percepción que se adquiere sobre el entorno virtual o sobre otro usuario.

1.3 Colaborativo

Aunque colaborativo es un concepto muy amplio, esta sección se limita a explicar su relación con RV.

Realidad virtual colaborativa se refiere a cuando dos o más usuarios comparten un mismo espacio virtual. Típicamente, para que dos usuarios puedan interactuar es necesario representarles virtualmente, para ello, se usan representaciones virtuales (avatares) que marcan la posición y orientación de estos. La correcta representación visual de los usuarios puede tener un gran impacto en la inmersión o en la percepción de los demás usuarios [5], o incluso, en la de uno mismo [6].

Podemos clasificar las aplicaciones colaborativas de dos formas a partir del espacio físico:

- Compartido - Se refiere a cuando todos los usuarios comparten el espacio físico, es decir, se encuentran en la misma sala. En este tipo de interacción los usuarios podrían colisionar accidentalmente.
- Separado - Por tal de compartir espacio virtual no es necesario encontrarse en el mismo lugar físicamente, en este caso cada usuario se encuentra en una sala diferente y la interacción es exclusivamente virtual.

1.4 Identificación del problema

En este trabajo se desarrolla una aplicación que permite a varios usuarios interactuar y navegar en un espacio virtual. Por ello, se divide el problema en dos bloques, por una parte, la navegación en entornos virtuales, y por otra parte, la posibilidad de permitir a varios usuarios colaborar de forma natural en dichos entornos. A continuación, se concretan los problemas a resolver de cada bloque.

1.4.1 Navegación

En [7] se discute el concepto de *cybersickness* que se puede desarrollar durante el uso de RV. Entre otros motivos, el hecho de mover virtualmente a un usuario, sin que se esté movimiento físicamente, puede producir mareos, hecho que anula por completo la inmersión y el uso práctico de la aplicación. Además, dependiendo del tipo de desplazamiento realizado, es fácil inducir desorientación espacial al usuario, es decir, que pierda el sentido de su posición y orientación dentro del entorno virtual.

En RV aparecen dos conceptos relacionados con la navegación: movimiento físico y virtual. Cada movimiento esta relacionado con su espacio tridimensional, inicialmente, y dependiendo del dispositivo de RV que se use, se define un área sobre la que los usuarios podrán moverse en el mundo real, formando así el espacio físico; el usuario podrá moverse libremente por dicha superficie.

Por otro lado, el espacio virtual es el mundo que se está representando informáticamente. En muchas ocasiones el espacio virtual será superior al físico, pues las escenas a representar serán mucho más grandes que el lugar donde se encuentre el usuario físicamente.

Por último, cabe destacar que un movimiento en el espacio físico suele implicar el mismo movimiento en el espacio virtual, no obstante, es habitual realizar movimientos en el espacio virtual sin que el usuario tenga la necesidad de moverse físicamente (por ejemplo, mediante el uso de un *joystick*).

El método de navegación natural en RV es el propio movimiento del usuario en su espacio físico, es decir, si el usuario se mueve físicamente, su posición virtual se desplaza acorde con el movimiento realizado. Este tipo de movimiento no suele presentar problemas de mareos o desorientación espacial, no obstante, usado de forma exclusiva, limita la capacidad de desplazamiento en el entorno virtual al tamaño de la habitación física en la que nos encontremos.

Una posibilidad para reducir los problemas de mareos y desorientación es permitiendo solo movimiento físico al usuario, no obstante, como se exponía anteriormente, la capacidad de movimiento sería muy limitada. Por lo tanto, se presentan dos variables opuestas: capacidad de movimiento, y *cybersickness*. Se busca desarrollar una solución flexible con un compromiso entre las dos variables, es decir, obtener la mayor capacidad de movimiento posible sin inducir *cybersickness* o desorientación.

1.4.2 Interacción

En realidad virtual colaborativa, además de los problemas de navegación, surgen otros relacionados con la percepción. Fundamentalmente, existen tres conceptos relacionados con la percepción de un usuario sobre los demás [8]: conocimiento de la posición de los otros usuarios, hacia dónde están mirando, y gestos no verbales.

En una aplicación colaborativa, ya sea con espacio físico separado o compartido, es importante conseguir los puntos anteriores para que los usuarios puedan interactuar con naturalidad. Esto les permitirá expresar intenciones y conseguir una mayor inmersión [9]. Además, si el espacio físico es compartido, es importante evitar que los usuarios puedan colisionar accidentalmente. Tal y como se muestra en [10], conocer hacia donde mira un usuario y su orientación puede ayudar a evitar colisiones.

Por lo tanto, nuestro objetivo consiste en investigar métodos para evitar colisiones entre usuarios, mejorar la inmersión y la percepción que se tiene sobre los demás usuarios, así como conseguir interacciones naturales.

1.5 Agentes implicados

El sistema va dirigido a diferentes ámbitos donde se necesite cooperación entre varios usuarios, algunos ejemplos son: el entrenamiento de profesionales de diferentes sectores como la medicina o la ingeniería, la visualización de modelos arquitectónicos, o incluso, gracias a las técnicas de navegación, también puede ser utilizado por aplicaciones que típicamente utilizan grandes espacios virtuales, como por ejemplo los videojuegos.

Concretamente, los primeros beneficiarios del proyecto son los arquitectos de la Sagrada Familia, tal y como se detalla en la sección 1.1. Actualmente, los arquitectos visualizan modelos aún por construir, con el objetivo de discutir los diseños o presentarlos a dirección para su aprobación. El problema es que sólo un usuario puede usar la aplicación mediante unas gafas o casco de RV, mientras que los demás ven lo mismo en una pantalla sin visualización 3D. En consecuencia, difícilmente pueden interactuar entre ellos, pues no pueden realizar acciones tan simples como apuntar a algún lugar para comentarlo.

A partir de este trabajo, varios usuarios podrán interaccionar a través de la escena virtual, de manera que les permitirá aumentar la productividad de dichas reuniones y mejorar la experiencia inmersiva.

2 Justificación

La realidad virtual colaborativa es un área de estudio relativamente nueva, en la que hay mucho por innovar. Por este motivo, aunque se busca desarrollar una aplicación que pueda ser usada por la Sagrada Familia, se enfocará como un trabajo de investigación. En consecuencia, dados los problemas planteados en la sección 1.4, se desarrollarán estudios sobre el trabajo realizado para determinar que alternativas son las más adecuadas y producen mejores resultados.

A continuación, se presentan las soluciones existentes para determinar si es necesario el desarrollo de la aplicación que se plantea, además, se revisan los estudios realizados sobre navegación y colaborativo en RV. Seguidamente, se analizan las herramientas necesarias para la realización del proyecto.

2.1 Estudio soluciones existentes

Dado que RV es una tecnología relativamente nueva y en constante evolución, es difícil estandarizar los métodos y dispositivos con los que se trabaja, por este motivo, la mayoría de soluciones son aplicaciones *in-house*[1], que difícilmente se pueden aplicar más allá del contexto de la empresa que las desarrolla.

A pesar de las dificultades, la empresa *TechViz* [11] desarrolla un producto de realidad virtual colaborativa, que aparentemente, se puede utilizar en un amplio número de contextos. Principalmente, se centra en la visualización en colaborativo de modelos creados con software de diseño *CAD*[2] (usado comúnmente en diseños de ingeniería o arquitectura).

No obstante, la tecnología de *TechViz* no es una alternativa viable para el sistema que se quiere desarrollar. Por un lado, no incorpora capacidades de navegación más allá del propio movimiento físico, por otro lado, no sería posible la integración del sistema con la aplicación actual que usan los arquitectos, basada en *Unity3D* [12]. Por lo tanto, es necesario desarrollar un sistema propio que permita el desarrollo e investigación de nuevas técnicas, así como la fácil integración con las herramientas actuales.

[1] Aplicaciones creadas para el uso interno de una empresa.

[2] CAD o diseño asistido por computadora se refiere al uso de programas de ordenador para crear representaciones 2D o 3D de objetos físicos.

Desde el punto de vista académico, sí existen diferentes estudios que pueden ser de utilidad para el desarrollo de las técnicas de navegación y colaboración. En la sección 1.4 se presentan diferentes estudios que exponen el problema del *cybersickness* en RV, así como métodos para minimizarlo. Otros estudios presentan técnicas de navegación, por ejemplo, en [13] se presenta una técnica basada en el teletransporte[3] tal y como se muestra en la Figura 3, el usuario apunta a la escena para elegir su nueva posición.

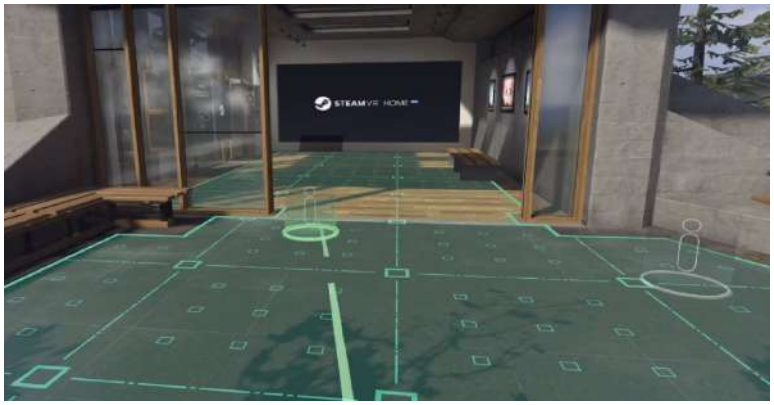

Figura 3: Método de navegación basado en teletransporte. Fuente: [14]

2.2 Herramientas

Para el desarrollo de la aplicación es necesario un conjunto de herramientas *software* y *hardware* especializadas en RV. Dado que el mercado de RV todavía está en crecimiento, las herramientas son limitadas y en constante evolución, a continuación, se realizará un estudio de las diferentes alternativas.

Dispositivos realidad virtual

Primero de todo, es necesario algún sistema de visualización de entornos tridimensionales, para ello se encuentran dos tipos de dispositivos basados en RV:

- Realidad virtual semi-inmersiva - típicamente consiste en una o áserie de pantallas que muestran el mundo virtual y permiten la interacción del usuario con elementos reales, el ejemplo más representativo es el *CAVE* [15], tal y como se puede ver en la Figura 4a, se trata de un lugar de trabajo rodeado de pantallas donde las imágenes se presentan de acuerdo con la posición y la dirección de la mirada del usuario.

[3]Teletransporte es el proceso de mover algo de un lugar a otro instantáneamente.

- Realidad virtual inmersiva - los sistemas inmersivos, a diferencia de los semi-inmersivos, aíslan completamente al usuario del exterior; se visualiza y se manipula el entorno virtual a través de dispositivos como cascos o guantes. La Figura 4b muestra un usuario utilizando un sistema inmersivo basado en un casco y unos controladores.

(a) Sistema semi-inmersivo CAVE *(b) Casco de RV inmersivo*

Figura 4: Ejemplos de dispositivos de RV. Fuentes: [4] [16]

Los sistemas de RV semi-inmersiva presentan diferentes inconvenientes para el tipo de aplicación que se quiere realizar. Por una parte, suelen ser dispositivos que requieren mucho espacio y tiempo de montaje, hecho que dificultaría su instalación y uso más allá del ámbito académico o de grandes empresas. Otro inconveniente, es la incapacidad de que cada usuario pueda tener su posición y punto de vista en la escena, en los sistemas tipo *CAVE* sólo se puede visualizar lo que ve un usuario. Además tienen un precio muy elevado que puede fácilmente superar los 50.000 euros. Es por estos motivos que, para este trabajo, se considera más adecuado el uso de técnicas de RV inmersiva.

Una vez decidido el uso de RV inmersiva mediante cascos, es necesario analizar los dispositivos que se encuentran en el mercado. Afortunadamente, gracias al auge de la tecnología, existe una gran oferta de cascos de RV con altas prestaciones y precios relativamente asequibles. Principalmente, existen los cascos mostrados en la Tabla 1. Como se puede ver, todos comparten características muy similares, a excepción del tipo de pantalla, donde las *HTC* se basan en tecnología *LED* y puede sugerir una mayor calidad.

Finalmente, dado que el desarrollo de la aplicación se realiza en el Centro de Realidad Virtual[4], se usarán los cascos ya disponibles en las instalaciones: *HTC Vive* y *HTC Vive Pro*.

[4]Sede del grupo de investigación ViRVIG [4]

Casco	Pantalla	Resolución	Refresco	Precio
Oculus Rift S	LCD	2560x1440px	80Hz	450€
HTC Vive	OLED	2160x1200px	90Hz	600€
HTC Vive Pro	AMOLED	2280x1600px	90Hz	800€

Tabla 1: Comparativa entre los cascos de RV más populares. [17] [18]

Software

Una vez seleccionados los dispositivos a utilizar, es necesario un motor gráfico que permita visualizar las escenas tridimensionales, así como la creación de *scripts*[5] que permitan desarrollar la aplicación.

Un motor gráfico es un conjunto de herramientas para desarrollar aplicaciones visuales, es decir, para renderizar[6] escenas 2D o 3D. Además, suelen contener otros sistemas útiles en el desarrollo de aplicaciones relacionadas con la computación gráfica, por ejemplo, pueden contener un motor de físicas. La interacción con el motor se suele realiza a través de una *API*.

Para el desarrollo profesional de aplicaciones de RV se usa principalmente *Unity3D* [12] y *Unreal Engine* [19]. En la actualidad, ambos motores presentan prestaciones similares, la principal diferencia es el lenguaje de programación con el que se programan los *scripts*: *C#* en *Unity3D* y *C++* en *Unreal Engine*. En cuanto a RV ofrecen características muy similares, especialmente porque ambos se basan en la librería *SteamVR* [20] para el uso de los cascos *HTC*.

Finalmente, ya que ambos motores ofrecen características similares en RV, se usará *Unity3D* para el desarrollo del trabajo por motivos de compatibilidad, ya que la aplicación que usan actualmente los arquitectos para visualizar los diseños se ha desarrollado con *Unity3D*.

[5] En motores gráficos, típicamente se introduce el código necesario para desarrollar la aplicación a través de fragmentos de código que interaccionan con el núcleo del motor.

[6] Proceso de generar imágenes a partir de modelos 2D o 3D.

2.3 Conclusiones

Al principio de esta sección, se ha visto que hay empresas que ofrecen soluciones al problema de la colaboración en RV, no obstante, dichas soluciones carecen de elementos de navegación, y no presentan la posibilidad de integración con la herramienta usada actualmente, *Unity3D*. Además, es necesario crear una aplicación propia de cara a la investigación de nuevos métodos de navegación e interacción. Por estos motivos, es conveniente desarrollar una aplicación en vez de usar una existente.

Otra decisión esencial es el uso de RV inmersiva, ya que presenta grandes mejoras respecto a la RV semi-inmersiva. Por un lado, es más adecuado para sistemas colaborativos porque cada usuario tiene su propio punto de vista y puede interactuar con el entorno individualmente, y por otro lado, es más asequible, ocupa menos espacio y reduce la complejidad.

Respecto al *hardware*, a pesar de que todas las opciones presentan características similares, se ha decidido utilizar los cascos *HTC Vive* y *HTC Vive Pro* ya que son los cascos disponibles en el Centro de Realidad Virtual.

Por último, era necesario escoger un motor gráfico como base de la aplicación que desarrollaremos. Puesto que tanto *Unreal Engine* como *Unity3D* presentan características similares, se ha seleccionado *Unity3D* ya que la aplicación usada actualmente por los arquitectos está basada en dicha tecnología.

3 Alcance

Como en todo proyecto, es importante definir el alcance de este, pues el tiempo de desarrollo es limitado y es necesario definir los objetivos y requerimientos de la aplicación. A continuación, se definen los objetivos, los requerimientos no funcionales y los obstáculos y riesgos del proyecto.

3.1 Objetivos

El objetivo principal de este trabajo es el desarrollo, implementación y evaluación de diferentes técnicas de interacción y navegación en realidad virtual colaborativa. Para poder facilitar la comunicación entre usuarios, pues necesitan poder hablar e interactuar físicamente en el mismo espacio, se desarrollarán técnicas para espacios físicos compartidos. Seguidamente, se divide el objetivo en los siguientes subobjetivos:

1. Desarrollar una aplicación que permita la conexión de varios usuarios a una misma escena virtual, además debe permitir la transmisión de información entre usuarios para tareas de sincronización e interacción.

2. Investigar y desarrollar técnicas de navegación en entornos virtuales, dichos métodos deben intentar minimizar el *cybersickness* sin comprometer la capacidad de movimiento. Un ejemplo de técnica de navegación es el presentado en [13] basado en el teletransporte.

3. Investigar y desarrollar métodos de interacción entre usuarios en espacio físico compartido, se busca mejorar la presencia e inmersión, así como reducir la posibilidad de colisión de los usuarios en la escena. Por ejemplo, se puede trabajar con la representación virtual de los usuarios.

4. Investigar y desarrollar diferentes estrategias para representar al otro usuario dentro del entorno de realidad virtual, con el objetivo de facilitar la colaboración entre ambos sin perder inmersión.

3.2 Requerimientos

3.2.1 Requerimientos funcionales

A pesar de que los principales requerimientos funcionales del sistema se definen en el objetivo, a continuación, se definen otros requisitos necesarios para el funcionamiento de la aplicación:

1. Compatibilidad dispositivos RV - Puesto que cada vez existen más dispositivos de RV, el sistema ha de tener compatibilidad con diferentes dispositivos de RV inmersiva que se encuentran en el mercado.

2. Integración aplicación inicial - Tal y como se explica en la sección 1.1, el trabajo parte de una aplicación que se desarrolló para los arquitectos de la Sagrada Familia, el objetivo es que el nuevo desarrollo sea compatible con todo el trabajo previo.

3.2.2 Requerimientos no funcionales

A continuación, se definen los requisitos no funcionales del sistema, es decir, aquellos requerimientos que no hablan directamente del funcionamiento del sistema, sin embargo, se han de tener en cuenta desde el principio para el desarrollo de la aplicación:

1. Usabilidad - Los dispositivos de RV, así como el *software* necesario para su funcionamiento, suele ser complicado de preparar y usar, por ello, si la aplicación se ha de usar en ámbitos muy diversos, es importante que sea fácil de usar para incrementar su adopción.

2. Eficiencia - Las aplicaciones gráficas en tiempo real suelen consumir muchos recursos ya que han de computar la escena entre 30 y 60 veces por segundo. Usando cascos de RV la eficiencia de la aplicación aún es más importante, pues la escena se calcula dos veces (una por cada ojo) y se ha de computar unas 90 veces por segundo para no generar mareos. Se crean y modifican *shaders*[7] para mejorar el rendimiento.

3. Reusabilidad - Para facilitar la integración del sistema en otras aplicaciones, es interesante que las técnicas implementadas puedan integrarse en diferentes proyectos.

[7]Programa informático usado para especificar a la tarjeta gráfica como debe pintar la escena virtual.

4 Navegación en realidad virtual colaborativa

4.1 *Networking*

En este trabajo, se quiere desarrollar e investigar técnicas de navegación en RV colaborativa, por lo tanto, para permitir la interacción entre usuarios, es conveniente empezar definiendo qué tipo de arquitectura y qué tecnologías se van a utilizar a nivel de red, el objetivo es que diferentes usuarios puedan compartir un mismo espacio virtual además de interactuar entre ellos y con el entorno.

4.1.1 Arquitectura de comunicación

A lo largo de la historia de las aplicaciones en tiempo real y videojuegos, se han utilizado diferentes tipos de arquitecturas de comunicación. Para este trabajo se investigan las 3 categorías principales [21] para poder seleccionar la opción más adecuada. En la Figura 5 se pueden observar los diferentes tipos de arquitectura.

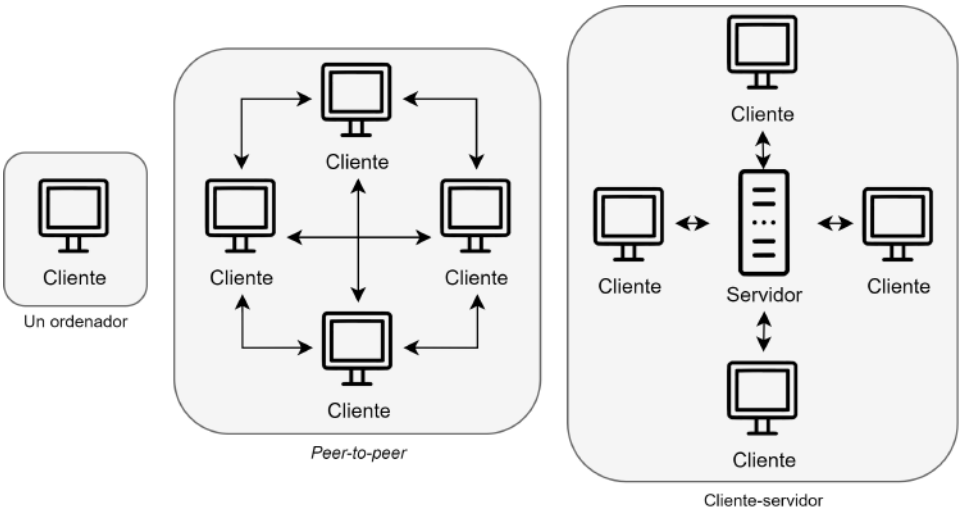

Figura 5: Diferentes arquitecturas de comunicación para aplicaciones en tiempo real. De izquierda a derecha: un ordenador, peer-to-peer*, cliente-servidor.*

Puesto que se quiere desarrollar una aplicación de RV colaborativa en espacio físico compartido, no es necesario usar varios ordenadores diferentes porque podemos simplemente conectar varios cascos de RV a un mismo ordenador. Por tanto en un primer momento, decidimos utilizar la primera opción, porque así nos evitamos la necesidad de usar una red. No obstante, esta opción se descartó rápidamente ya que las aplicaciones de RV consumen muchos recursos computacionales, y por tanto es conveniente utilizar una tarjeta gráfica dedicada para cada dispositivo de RV.

La siguiente opción es utilizar una arquitectura *peer-to-peer*; cada usuario tiene su propio ordenador ejecutando la aplicación, la comunicación se establece entre usuarios sin ningún tipo de mediador a través de una red, normalmente en una red local (LAN) aunque también es posible en redes tipo Internet (WAN).

La última opción es el modelo cliente-servidor, en este caso existe un proceso (servidor) que actúa como árbitro: se encarga de almacenar el estado de la aplicación y la comunicación siempre es a través de este proceso. El servidor se convierte en una parte crítica del sistema, cualquier cliente (usuario) para comunicarse con otro cliente debe primero comunicarse con el servidor para que éste envíe el mensaje.

Finalmente, para el tipo de aplicación que se quiere desarrollar tanto la arquitectura *peer-to-peer* como cliente-servidor podrían funcionar, no obstante, se ha decidido utilizar la arquitectura cliente-servidor porque permite mayor escalabilidad en redes tipo Internet, de esta forma, la aplicación funcionaría con cambios mínimos tanto en espacios compartidos como en separados.

Cliente-servidor

En el apartado anterior, se explica que se va a utilizar la arquitectura cliente-servidor, por lo tanto, se van a utilizar dos tipos de procesos, en la Figura 6 se detalla la interacción entre el cliente y el servidor.

Por una parte, cada usuario ejecuta un proceso cliente, que se ha de conectar con el proceso servidor, esta conexión se establece mediante el protocolo *TCP* o *UDP* [8]. Seguidamente, para cada proceso se establecen diferentes tipos de objetos para representar la escena virtual:

[8]TCP y UDP son protocolos de la capa de transporte de una red, se encargan de la transferencia de datos y la gestión de errores, además se encargan de mantener el flujo de la red. Para más información ver: [22]

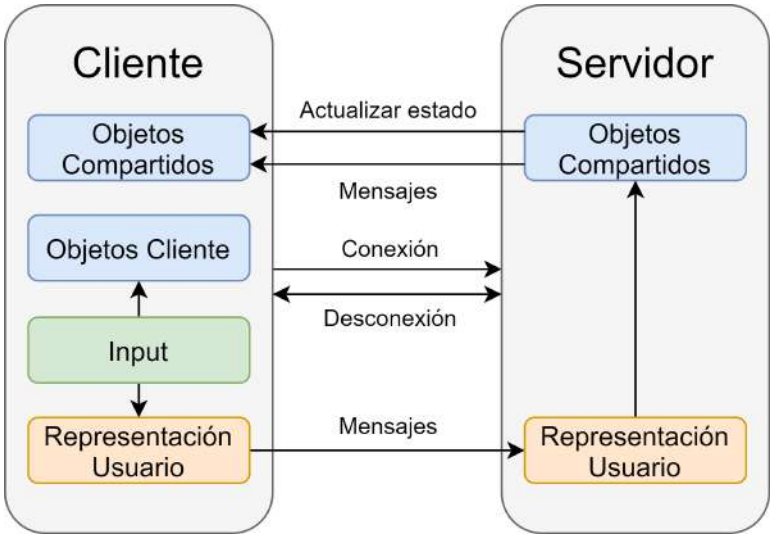

Figura 6: Diagrama de comunicación entre el cliente y el servidor.

1. Representación visual del usuario - Es el objeto que contiene la información del usuario, existe una copia tanto en el proceso cliente como en el servidor. A través del *input* se puede manipular e utilizar para enviar mensajes al servidor para modificar el estado del mundo compartido.

2. Objetos cliente - Son aquellos objetos que solo existen en el proceso cliente, no es necesaria la intervención del servidor para su manipulación, se suelen usar para aspectos visuales u objetos que no necesitan ser sincronizados entre todos los clientes.

3. Objetos compartidos - Se trata de los objetos que ven todos los clientes y se han de sincronizar, existe una copia en cada cliente y en el servidor, los usuarios no pueden modificarlos directamente.

En general, el funcionamiento es el siguiente: cada usuario es capaz de interactuar con el entorno a partir del *input* (controladores de RV), si interacciona con un objeto compartido, entonces se envía un mensaje al servidor para que actualice el estado del objeto. El servidor analiza y ejecuta las acciones que recibe de los usuarios, para ello, actualiza el estado de todos los objetos compartidos en los clientes y enruta mensajes para permitir la interacción cliente-cliente.

Por último, se ha de especificar dónde se ejecutará el proceso servidor. Para este proyecto se definen dos tipos de usuarios. Por un lado, un usuario ejecutará el proceso servidor y a continuación el proceso cliente, aunque se encuentren en el mismo ordenador, el proceso cliente se conecta al servidor con normalidad. Por otro lado, el resto de usuarios solo ejecutarán el proceso cliente y se conectarán al servidor.

Unity Multiplayer API

Para este trabajo se utiliza el motor gráfico *Unity3D*, la escena virtual está representada por un conjunto de entidades con comportamientos (*scripts*) asociados, estas entidades corresponden a los objetos que se explican en el apartado anterior, además, cada usuario estará representado también por una entidad. Para facilitar la conexión entre entidades de *Unity3D* por red, se utiliza la *Unity Multiplayer API*.

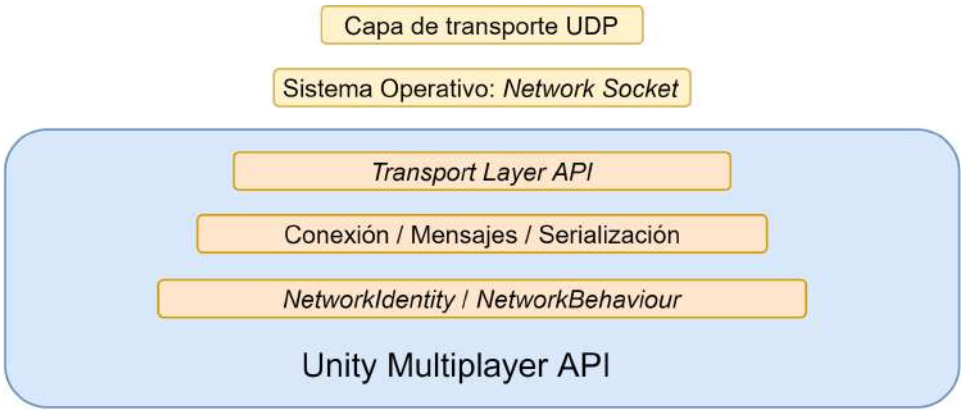

Figura 7: Diagrama de la Unity Multiplayer API.

En la Figura 7 se muestra un diagrama simplificado de las diferentes capas de la *Unity Multiplayer API*, a continuación, se definen las capas:

1. *Transport Layer API* - Actúa como capa de abstracción de los *network sockets* del sistema operativo, permite trabajar con máxima flexibilidad al tratarse de una *API* de bajo nivel, establece comunicaciones a través de la capa de transporte UDP[9].

2. Capa de conexión, mensajes y serialización - Establece funciones y métodos para enviar mensajes entre procesos a través de la *Transport Layer API*.

[9]UDP es utilizado en sistemas en tiempo real para reducir la latencia, se trata de un protocolo mucho más liviano que TCP, por otro lado, no asegura que los paquetes lleguen en orden ni implementa un control de errores de transmisión, esto lo debe realizar la aplicación [21].

3. *NetworkIdentity* y *NetworkBehaviour* - Capa de unión de las entidades de *Unity3D* con el código de red. *NetworkIdentity* sirve para definir un identificador único para un objeto compartido y *NetworkBehaviour* para implementar código de red para una entidad.

Para este trabajo, se ha hecho uso de la última capa de la *Unity Multiplayer API* para escribir el código de red que se ejecuta en los procesos cliente y servidor, y además, se ha usado la capa de conexión y mensajes para definir la comunicación entre procesos.

4.1.2 Implementación del código de red

Una vez definida la arquitectura a utilizar y como se implementa en *Unity3D*, en esta sección se detallan los procedimientos usados para implementar el código de red de las técnicas de navegación, la implementación de cada técnica por separado se detalla en la sección 4.2. En la Figura 8 se puede ver un esquema general de la implementación.

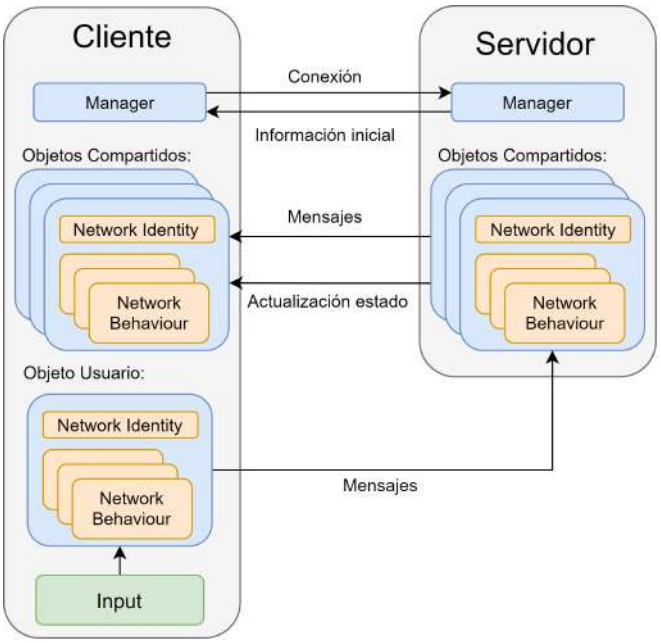

Figura 8: Diagrama de la implementación del código de red.

Primero se ha implementado la clase *Manager* que se encarga de gestionar la conexión y la inicialización, tanto el cliente como el servidor tienen una copia de la clase. En el cliente simplemente se realiza la petición de conexión con el servidor y recibe la información inicial para establecer los objetos compartidos y el objeto usuario. En el servidor se mantiene la información de todos los clientes conectados, cada vez que se conecta (o desconecta) un nuevo cliente, se actualiza el estado de los demás clientes para que puedan visualizarlo.

Todos los objetos contienen el componente *Network Identity*, de esta forma se puede reconocer un objeto entre el servidor y los diferentes clientes de forma única. Cada objeto contiene además uno o más *Network Behaviour* donde se implementa el código que podrá realizar, además se implementan los diferentes tipos de mensaje.

El objeto usuario implementa las acciones a realizar en función del *input*, si la acción no ha de ser compartida solo se ejecuta en el cliente, si la acción modifica el estado de la escena, entonces se envía un mensaje de sincronización al servidor, en este caso, el servidor se guarda el cambio y envía un mensaje de actualizar estado a los demás clientes. Por ejemplo: el cliente c aprieta el botón de moverse en el controlador, c calcula que ha de moverse a una posición p, entonces se envía un mensaje al servidor para notificar que se encuentra en la posición p, el servidor informa al resto de clientes que c está en la posición p.

Si el usuario quiere comunicarse con otro usuario para ordenarle una acción, por ejemplo, que se mueva hasta su posición, entonces el servidor solo actúa como enrutador; el usuario envía un mensaje (previamente definido) de *teletransporte* al servidor, el servidor envía ese mismo mensaje al cliente destinatario, el cual ejecuta la acción.

A modo de resumen, en el cliente los objetos implementan las acciones a realizar y contienen copias del estado de la escena almacenado en el servidor. En el servidor se definen las acciones que puede realizar cada usuario y es el que decide si se ejecutan o no, además mantiene el estado de la aplicación actualizado.

4.2 Técnicas de navegación

Las técnicas de navegación afectan considerablemente a la experiencia del usuario al usar RV. Durante la navegación, es fácil que el usuario experimente mareos o desorientación espacial, especialmente si no cuenta con experiencia previa en RV. Además, al involucrar a varios usuarios en una misma sala, es importante crear métodos de interacción que permitan a los usuarios interactuar con naturalidad, de forma que se puedan evitar colisiones y conseguir experiencias más satisfactorias. En la sección 1.4 se explica con más detalle los problemas derivados de la navegación y la interacción en RV colaborativa.

Con la aplicación preparada para soportar múltiples usuarios a través de una red, y los mecanismos necesarios para permitir la interacción compartida, en esta sección, se describe el diseño y la implementación de las diferentes técnicas de navegación que se usarán en este trabajo.

4.2.1 Espacio físico y virtual

En realidad virtual los usuarios se desplazan en entornos virtuales de diferentes tamaños, no obstante, físicamente, se encuentran en habitaciones o lugares que no suelen concordar con las dimensiones del espacio virtual.

Primero, se define el espacio físico donde los usuarios podrán moverse en el mundo real, que consiste en un área típicamente rectangular; por otro lado, se define el espacio virtual como la escena tridimensional. Normalmente el espacio virtual será superior al físico ya que las escenas a representar son mucho más grandes que el espacio físico, por ejemplo, en este proyecto se utiliza una escena virtual de una de las torres de la Sagrada Familia.

En la Figura 9 se puede ver arriba la representación del espacio físico, dos bases (sensores) de *HTC Vive* se usan para determinar la posición y orientación del casco y los controladores de RV; abajo se ve la representación visual, mediante líneas amarillas, del espacio físico en el entorno virtual.

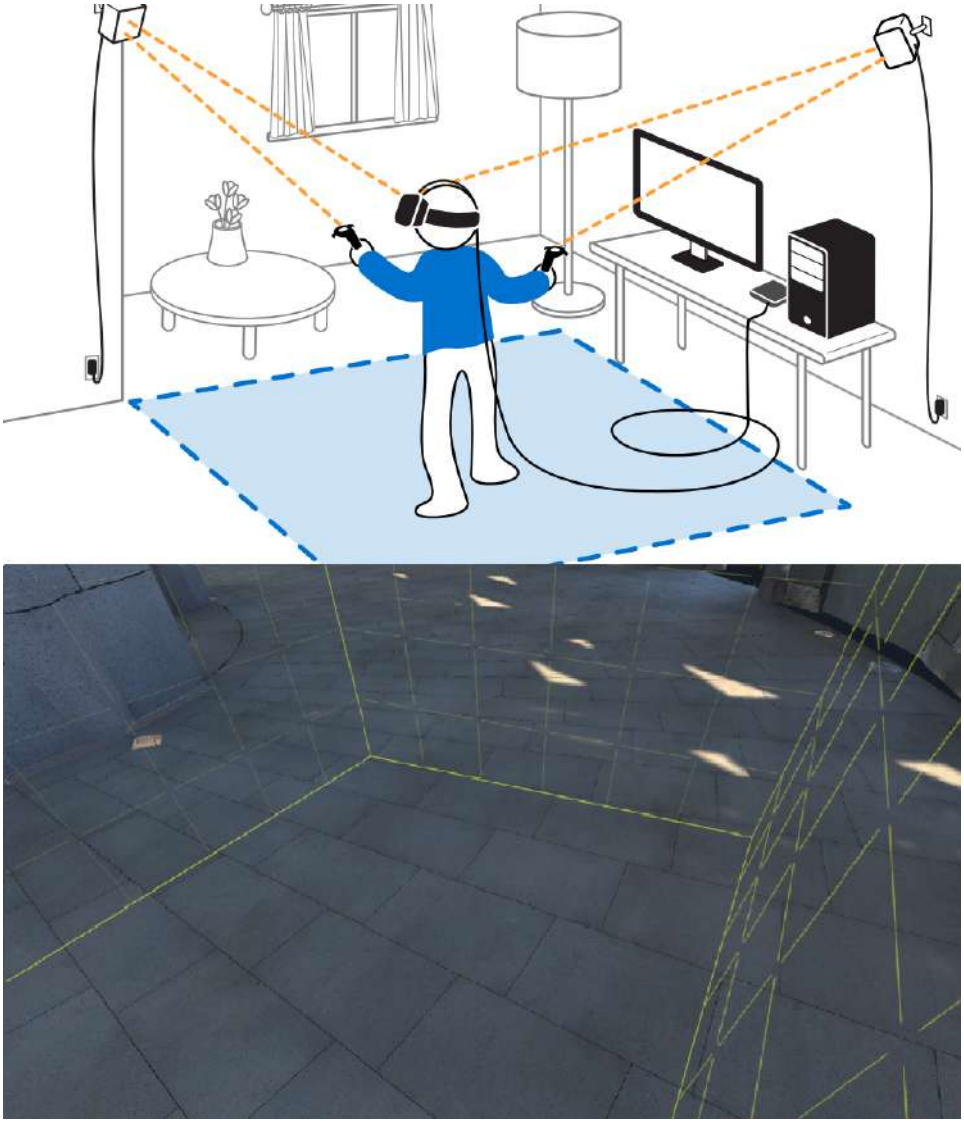

Figura 9: Arriba la representación de una habitación física con dos bases de HTC Vive para determinar la posición e orientación del casco y controladores de RV. Fuente: [23]. Abajo la representación visual, mediante líneas amarillas, de la habitación física en el entorno virtual.

Lo más adecuado en navegación virtual sería que un movimiento en el espacio físico implicase el mismo movimiento en el espacio virtual, es decir, que si el usuario camina medio metro hacia la derecha físicamente, su representación virtual también se mueva hacia la derecha medio metro en la escena virtual. No obstante, para desplazarse mayores distancias en el entorno virtual, es habitual realizar movimientos en el espacio virtual mediante un *controller* (mando de RV) sin que el usuario tenga la necesidad de moverse físicamente.

La posición y la orientación del usuario se obtiene a partir de sensores del casco de RV, esta información es la posición del usuario respecto el centro del espacio físico definido previamente. Para poder determinar la posición virtual de cada usuario, y mantener información sobre las posiciones físicas, se define, por cada usuario, una habitación del tamaño del espacio físico.

Cada usuario tiene su propia habitación y el tamaño es el mismo para todos los participantes. Inicialmente, el centro de cada habitación se encuentra en las coordenadas (0,0) del mundo virtual, cada vez que se realice un movimiento virtual, la habitación se moverá acordemente. Además, el usuario almacena su posición física respecto el centro de la habitación, es decir, la información que se obtiene de los sensores (la bases que funcionan con el HTC VIVE). Para obtener la posición física del usuario solo se ha de consultar la posición del usuario capturada por las bases dentro del entorno físico, para obtener la posición virtual se ha de sumar la posición del centro de la habitación en coordenadas de la aplicación más la posición física del usuario.

En la Figura 10 se puede ver un ejemplo de movimiento físico y otro de movimiento virtual. En ambos casos, inicialmente, la habitación del usuario 1 y 2 se encuentra en la misma posición (0,0), los usuarios se encuentran en distintas posiciones físicas. En el primer caso, el usuario 2 realiza un movimiento físico (andando en la realidad), la habitación no se mueve, simplemente se actualiza la posición del usuario 2. En el segundo caso, el usuario 2 realiza un movimiento virtual (con su controlador de RV), la habitación se desplaza, pero la posición física del usuario 2 sigue siendo la misma ya que físicamente no se ha movido, la posición virtual del usuario se calcularía como (3,2) + (1,1) = (4,3), donde (3,2) es el centro de la habitación virtual en coordenadas de la aplicación, y (1,1) es el desplazamiento del usuario entre su posición actual y el centro de la habitación real.

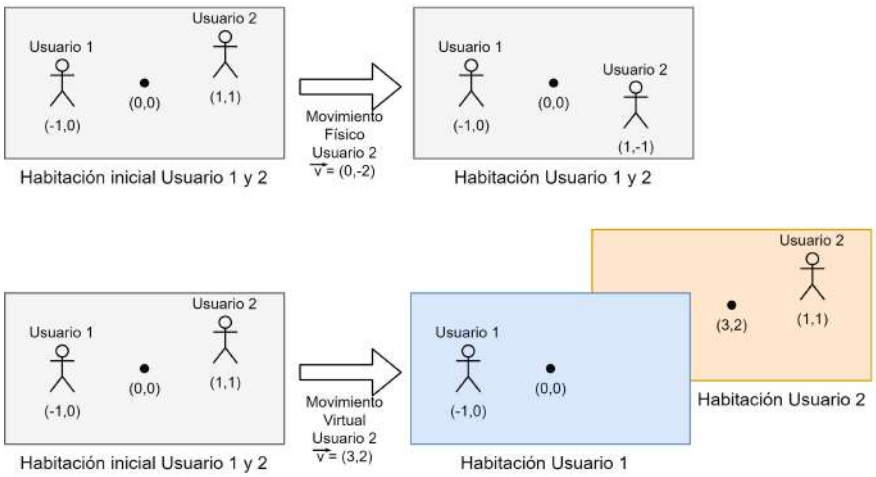

Figura 10: Representación de la posición de dos usuarios y sus habitaciones al realizar un movimiento físico y uno virtual.

4.2.2 Representación visual del usuario

Antes de empezar a desarrollar técnicas de navegación se ha creado la representación virtual (avatar) de los usuarios. En esta primera parte del trabajo, el objetivo de los avatares es que representen gráficamente la posición y orientación de la cabeza del usuario, por tanto se ha optado por una representación visual sencilla. En la Figura 11 se pueden ver dos usuarios representados con sus avatares.

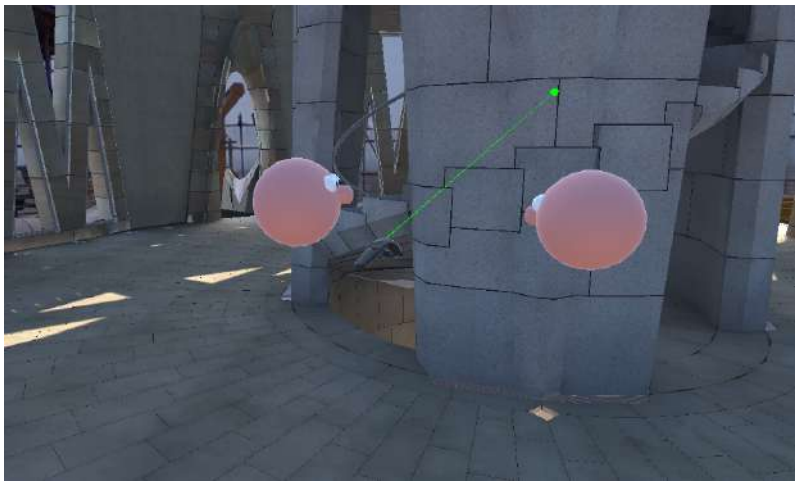

Figura 11: Avatares de dos usuarios interactuando en la misma escena virtual.

El avatar representa la posición virtual del usuario, es decir, dónde se encuentra en la escena virtual. Nótese que la posición virtual del usuario puede ser diferente a la posición física, los usuarios pueden estar a 2 metros físicamente, pero a una distancia completamente diferente en la escena virtual, en este caso el usuario no tendría conocimiento de la distancia real y podría ocasionar colisiones. Cuando los usuarios comparten el espacio físico, es preferible que la posición del avatar se corresponda con la real para evitar colisiones.

Otra característica del avatar son los ojos y la nariz, estos elementos denotan la dirección de visión del usuario. Esta información ayuda a los participantes a entenderse mejor y a predecir hacia dónde se producirán movimientos, así como entender mejor a qué hace referencia el usuario cuando habla de algún elemento del entorno virtual. Tanto la posición virtual como la orientación de todos los usuarios se almacena en el servidor, de esta forma se envía a todos los clientes para posicionar correctamente los avatares.

Para poder señalar a objetos y facilitar la interacción, se ha creado un puntero que parte del controlador de cada usuario, en la Figura 11 se puede ver como los usuarios usan el puntero para apuntar al elemento virtual del qué están hablando. El puntero consta de una línea formada por *quads*[10] y una esfera que se posiciona en el primer punto de intersección entre el rayo y la escena. El servidor guarda la orientación y la posición de todos los punteros para que los clientes puedan representarlos.

4.2.3 Vuelo libre

Una de las características necesarias en un método de navegación de RV es la rapidez y la flexibilidad para navegar en escenas virtuales grandes, y potencialmente, con múltiples alturas a las que acceder. Al trabajar con los arquitectos de la Sagrada Familia, era necesario desarrollar un método que les permitiera posicionarse en diferentes puntos y distancias de las torres.

El primer método desarrollado permite el desplazamiento por la escena virtual hacia cualquier dirección y sin restricciones, se podría decir que es asimilable a la acción de volar. En la Figura 12 se puede ver un usuario usando esta técnica para desplazarse por la escena.

[10]Un *quad* es un tipo de primitiva rectangular formado por dos triángulos, se usa para representar una superficie a pintar en la tarjeta gráfica.

Figura 12: Secuencia de un usuario desplazándose mediante vuelo libre en la dirección de su controlador.

El funcionamiento es el siguiente: se determina la dirección del desplazamiento a partir de la dirección del controlador de RV, cuando el usuario aprieta el gatillo del controlador, se aplica una aceleración a la velocidad del movimiento hasta que se alcanza la velocidad máxima definida, cuando el usuario deja de apretar el gatillo se aplica una desaceleración hasta que se para el movimiento.

La aceleración ha de ser progresiva para evitar mareos, no obstante, aunque la desaceleración rápida también puede provocar mareos, el hecho de que el avatar siga moviéndose (para suavizar la frenada) una vez el usuario ha decidido parar el vuelo, puede llegar a ser contraproducente, por este motivo la frenada es considerablemente más rápida que la aceleración.

Al iniciar la aplicación, las distancias físicas y virtuales entre los participantes se mantienen, sin embargo, en el momento en el que uno de los usuarios empieza a desplazarse con el vuelo libre, se pierde la información de las distancias físicas: los usuarios ya solo pueden saber dónde se encuentran virtualmente. Por este motivo, se ha desarrollado un forma de posicionarse virtualmente, respecto a un usuario, pero manteniendo las distancias reales.

Al apuntar al avatar de otro usuario y apretar el gatillo del controlador, se realiza automáticamente vuelo libre hasta la habitación del otro usuario, de esta forma ambas habitaciones concordaran virtualmente y la distancia virtual entre usuarios será la misma que la distancia física. Además, durante el vuelo, se marca la posición final del usuario para evitar causar desorientación espacial al tratarse de un movimiento automático. En la Figura 13 se puede ver el funcionamiento del sistema, a la izquierda el usuario selecciona con quién quiere compartir habitación virtual, a la derecha se muestra la posición final del usuario.

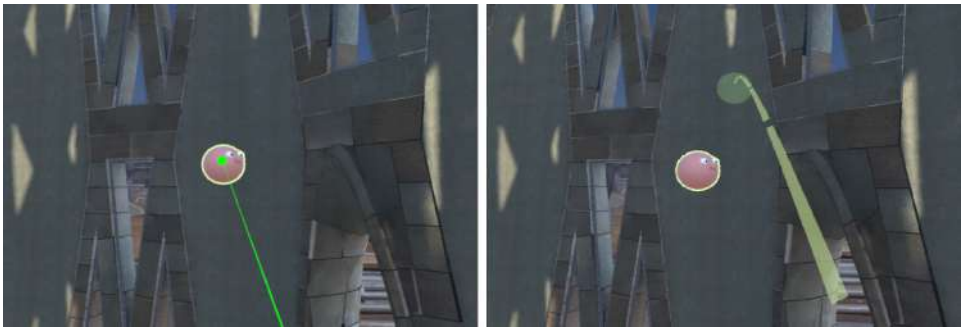

Figura 13: A la izquierda el usuario selecciona con quién quiere compartir habitación virtual, a la derecha se muestra la posición final del usuario. Técnica para restaurar la posición entre dos usuarios mediante vuelo libre.

4.2.4 Teletransporte

El segundo método de navegación está basado en el teletransporte, es decir, en mover a los usuarios de un punto a otro de forma instantánea. La ventaja de los métodos basados en la teletransportación es la baja sensación de mareo que producen, al evitar el movimiento continúo es más fácil reducir el *cybersickness*. El inconveniente es que al realizarse un cambio discreto de posición, suele ser habitual que los usuarios experimenten un cierto grado de desorientación espacial.

Se han implementado 3 modalidades de teletransporte con pequeñas variaciones:

1. En la primera modalidad, el usuario emplea el puntero para determinar a dónde quiere ir, a continuación, mantiene apretado el botón asignado al teletransporte (del controlador de RV) y aparece una área rectangular simulando la habitación virtual, el objetivo es indicar a dónde se realizará el movimiento. Al soltar el botón se efectúa el cambio de posición. Cada usuario mueve su habitación, por lo tanto, pueden moverse de forma independiente perdiendo la consistencia entre distancias físicas y virtuales. En la Figura 14 se puede ver a un usuario usando este método.

2. El segundo método es similar al primero, no obstante, en este caso sólo uno de los usuarios podrá realizar teletransportes, además, moverá a todos los usuarios al punto que escoja, de modo que se mantiene la consistencia entre distancias físicas y virtuales. Este método es útil cuando se quieren realizar demostraciones a usuarios no expertos en RV, ya que un usuario es quien decide los cambios de posición, y el otro hace de observador dentro de la habitación virtual.

3. Por último, si en vez de mantener apretado el botón se realiza una pulsación rápida, el usuario se posiciona sobre el suelo más cercano. Este método es especialmente interesante cuando los participantes usan vuelo libre, y en un momento dado necesitan posicionarse en un punto tal y como estarían en la realidad. El hecho de posicionarse con la altura real del usuario aumenta el realismo y la inmersión.

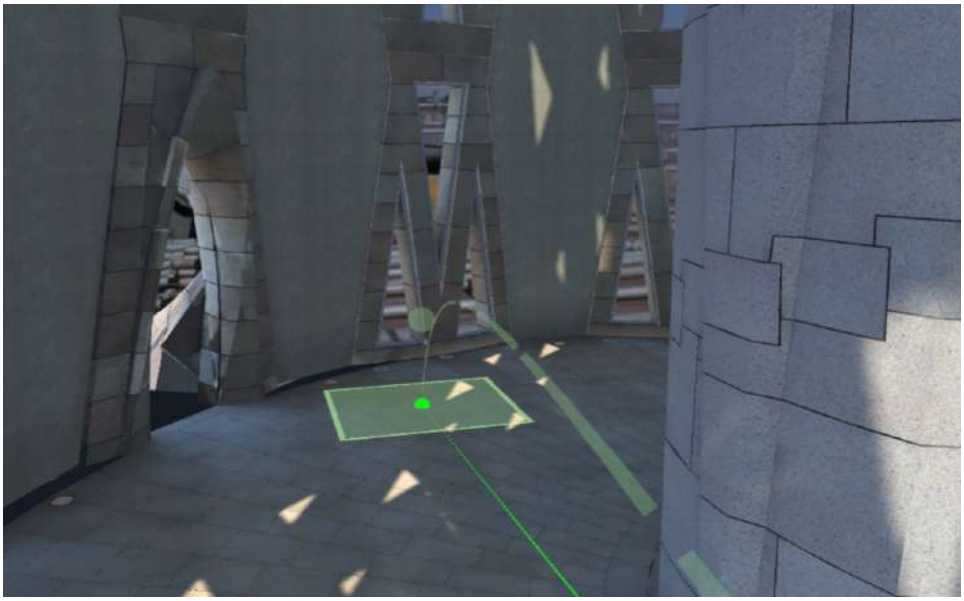

Figura 14: Usuario empleando el teletransporte. Se puede apreciar, para reducir la desorientación espacial, un arco marcando la trayectoria del teletransporte, la posición final de la habitación y del usuario.

El principal problema del teletransporte es la desorientación especial, por este motivo, al apuntar hacia el próximo punto de teletransporte se muestra la ubicación final de la habitación, además, se muestra la posición final de los usuarios, así sabrán en todo momento dónde aparecerán y con qué orientación. Asimismo, se muestra un arco entre el controlador del usuario y la nueva posición de la habitación para facilitar la comprensión de la distancia que se está recorriendo.

Una limitación de este método es la imposibilidad de teletransportarse a cualquier posición, hay que evitar, por ejemplo, que el usuario aparezca dentro de un objeto de la escena. Cuando el usuario apunta a una posición para realizar el movimiento, primero se calcula el primer punto de intersección entre el rayo y la escena, a continuación, calculando el producto escalar entre el vector vertical[11] y la normal del punto de colisión, se puede saber si la superficie es un suelo o algún tipo de superficie no apta para el movimiento (como por ejemplo una pared).

Con la comprobación del tipo de superficie no es suficiente, en la Figura 15 se puede ver que al desplazar la habitación, debido a las distancias físicas, el usuario puede acabar dentro de una pared o fuera de la escena. Para solucionar esto se realiza un ajuste automático, desde el centro de la habitación se lanzan rayos a cada uno de los usuarios, si el rayo no colisiona con la geometría, no hay ningún problema, de lo contrario, se calcula la normal del punto de colisión para cada usuario, si las normales no son opuestas se mueve la habitación en la dirección de la media de las normales, si son opuestas no se puede efectuar el teletransporte: el rectángulo y los demás indicadores visuales se ponen en rojo para indicar al usuario que no es una posición valida para el teletransporte.

Figura 15: A la izquierda el usuario acabará dentro de la pared al realizar el movimiento, a la derecha la habitación se ajusta automáticamente.

[11] Las posiciones en la escena virtual se representan a partir de 3 ejes (X,Y,Z), sabiendo que Y representa la altura, el vector vertical sería (0,1,0).

Por último, al desplazarse de forma instantánea de una posición a otra, es posible causar mareos al usuario al tratarse de un movimiento antinatural y brusco. Por este motivo, antes de realizar el teletransporte se oscurece la pantalla gradualmente, y después del movimiento, se esclarece la visión. Para implementar este efecto de forma eficiente, al tratarse de un sistema en tiempo real, primero se pinta la escena en la tarjeta gráfica en una textura, a continuación, la textura se oscurece o esclarece gradualmente (durante la duración del teletransporte) con un *shader*. Tanto la textura como el *shader* solo existen en la tarjeta gráfica, por lo que no hay comunicación con la CPU y la operación se realiza sin alterar los fotogramas por segundo de la aplicación. En la Figura 16 se muestran unos pocos fotogramas de la secuencia de teletransporte.

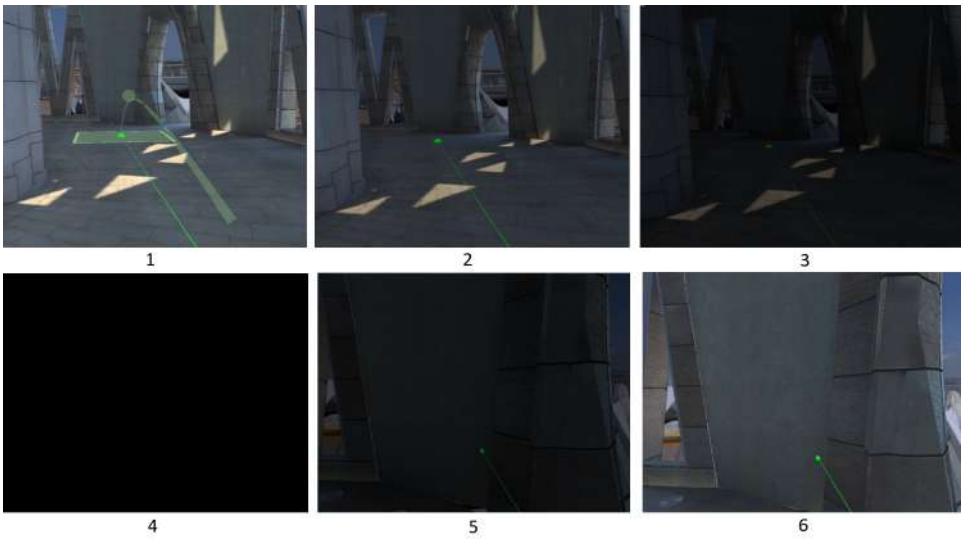

Figura 16: Secuencia de imágenes de un teletransporte, de 1 a 3 se puede apreciar como la pantalla se oscurece progresivamente, en 4 se realiza el teletransporte, de 5 a 6 se esclarece progresivamente.

4.2.5 Representación visual doble

En los apartados anteriores se han presentado métodos de desplazamiento por el mundo virtual sin necesidad de moverse físicamente. Existen dos modos de movimiento: movimiento físico hace referencia a que el usuario puede desplazarse físicamente por la habitación manteniendo la consistencia entre las distancias físicas y virtuales, por otro lado, el movimiento virtual se refiere a moverse virtualmente con el controlador sin desplazarse físicamente perdiendo la consistencia entre las distancias físicas y virtuales con los demás usuarios.

Los usuarios sólo ven la posición virtual de los demás participantes, mientras las distancias virtuales entre usuarios coinciden con las físicas no hay problemas de colisión, no obstante, en cuanto usen el controlador para moverse virtualmente, las distancias virtuales y físicas dejarán de ser las mismas, en la Figura 10 se puede ver como el usuario 2 realiza un movimiento virtual, y aunque preserva la posición física, virtualmente se encuentra mucho más lejos del usuario 1.

Aprovechando la representación por habitaciones de los usuarios, el servidor puede saber la posición física y virtual de todos los participantes (y por lo tanto puede enviar la información a los clientes); a partir de estas posiciones, además de la posición virtual, cada cliente puede representar la ubicación física de los demás usuarios.

El funcionamiento consiste en comprobar la posición de las habitaciones de los usuarios, si dos habitaciones tienen la misma posición, significa que los usuarios de dichas habitaciones se ven (virtualmente) a la misma distancia que lo harían físicamente, por lo tanto, es suficiente con mostrar el avatar virtual. De forma contraria, si la posición de las habitaciones no coincide entonces se muestra un avatar en la posición virtual, y otro avatar en la posición física. Por ejemplo, en la Figura 17 arriba se muestra un caso en el que las habitaciones virtuales coinciden, abajo uno de los usuarios se mueve virtualmente, pero no físicamente, y aparece su representación visual doble.

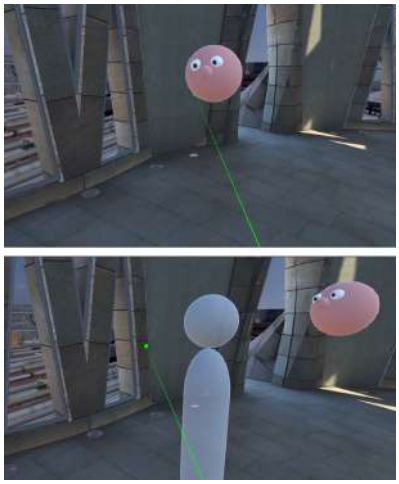

Figura 17: Arriba la habitación virtual de los dos usuarios es la misma, abajo el otro participante se mueve virtualmente hacia su izquierda, pero no se mueve físicamente, se muestra el doble avatar en su posición física real.

El objetivo de este nuevo avatar es dar a entender al usuario dónde se encuentran los demás participantes, de esta forma puede moverse libremente (físicamente) evitando colisiones. Aunque la representación visual física del usuario debería estudiarse con mayor detenimiento, de momento, se ha optado por usar una representación simplificada como con el avatar virtual. No obstante, se ha observado que como este nuevo avatar solo denota posición (simplemente sirve para evitar colisiones), es más efectivo usar una representación más *estática*. El resultado es un avatar formado por un cuerpo y una cabeza para dar el aspecto de humano, pero, la cabeza se representa simplemente con una esfera sin ojos ni nariz, pues el usuario ha de entender que este avatar no denota la dirección de visión real. A diferencia del avatar virtual, se ha añadido un cuerpo para dar una sensación más estacionaria, y por lo tanto, de obstáculo.

4.2.6 Puntos de vista

Otro método de navegación muy usado en RV son los puntos de vista, consiste en definir una serie de posiciones en el mundo virtual a las que los usuarios podrán teletransportarse. Los puntos de vista son útiles, entre otras situaciones, para establecer puntos de interés o puntos de reunión de usuarios, además, son ideales para presentaciones o para usuarios con poca experiencia, el recorrido puede estar preparado y el movimiento es muy bajo, por lo contrario, ofrecen muy poca flexibilidad de movimiento, pues se han de definir previamente.

Dado una serie de puntos de vista, se ha implementado una interfaz en RV para que el usuario pueda seleccionar a qué punto de vista desplazarse, si se dispone de la representación en miniatura de la escena virtual, también es posible mostrar y seleccionar los puntos de vista sobre un modelo 3D de la miniatura. En la Figura 18 se puede ver a la izquierda la interfaz usada por los usuarios para seleccionar los puntos de vista, y a la derecha un modelo en miniatura de la escena virtual con los puntos de vista marcados en verde.

En la interfaz, se generan imágenes en tiempo real de los puntos de vista, de esta forma el usuario puede ver dónde se va a mover, la obtención de las imágenes se explica en detalle en la sección 4.2.7. Una vez el usuario apunta y selecciona con el controlador una imagen, se teletransporta hasta ese punto de vista; simplemente es necesario almacenar la posición virtual del punto de vista (para mover la habitación del usuario), y la rotación a la que mirará la cámara para hacer la foto de la interfaz.

Figura 18: A la izquierda, interfaz dentro de RV que permite ver y seleccionar los puntos de vista en tiempo real, a la derecha, representación en miniatura de la escena, en verde los puntos de vista, en rojo el usuario.

Para usar los puntos de vista en RV colaborativa, se define un usuario que será el *master* y tendrá acceso al menú de los puntos de vista. Todos los participantes pueden ver el menú pero sólo el *master* puede seleccionar los puntos de vista. Esto permite que en un momento dado el *master* pueda reunir a todos los participantes a un punto preestablecido, pues cuando seleccione el punto de vista todos los participantes se desplazaran a esa posición.

Otra funcionalidad necesaria es la capacidad de añadir nuevos puntos de vista. Se han implementado dos métodos, el primero, consiste en una interfaz dentro de RV que se guarda la posición en la que se encuentra el usuario y la orientación de su dirección de visión, este método es de gran utilidad cuando el usuario encuentra una posición que le gustaría guardar mientras se está desplazando con vuelo libre o teletransporte. El segundo método es una interfaz que se ha creado como extensión de *Unity3D*, permite al usuario introducir una posición del mundo virtual, a continuación, puede ver como queda el punto de vista en el propio visor de *Unity3D* y finalmente añadirlo. Los puntos de vista se guardan en un fichero de texto para cada escena virtual.

4.2.7 Videocámara

Muchas veces en RV colaborativa un usuario quiere compartir lo que esta viendo, por lo tanto, el resto de usuarios han de encontrarlo en la escena virtual y, después, se han de desplazar hasta su posición virtual. Si simplemente se trata de algo puntual, puede ser más adecuado buscar una forma de compartir que evite el desplazamiento. Por este motivo, se ha implementado un visor que permite a un usuario ver lo que hacen los demás participantes.

Casa usuario tiene un visor que se activa apretando un botón del controlador de RV, al activar el visor, una especie de televisor tal y como se puede ver en la Figura 19, se empieza a retransmitir la visión de otro usuario, de esta forma, a través de esta televisión pueden ver todo lo que ve el otro usuario y comentar fácilmente cualquier elemento del entorno sin necesidad de desplazarse. Si hay más de dos participantes, pulsando de nuevo el botón se cambia de usuario y finalmente se apaga el visor.

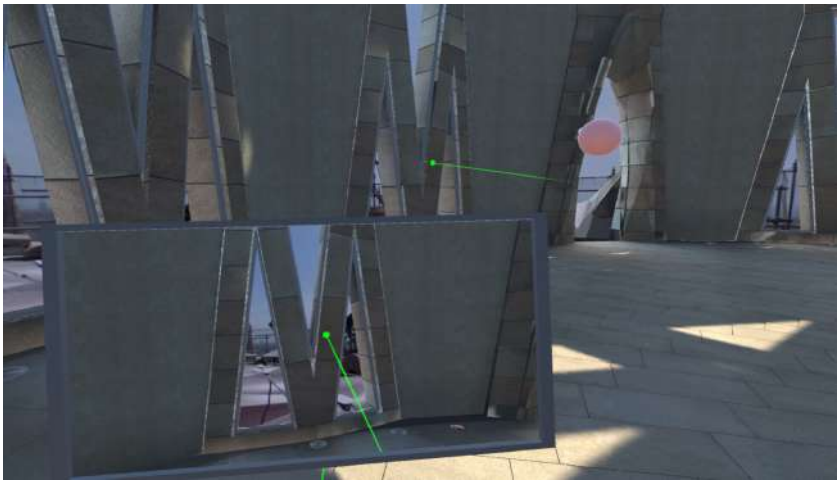

Figura 19: Un usuario usando su visor para ver la visión de otro usuario.

Para conseguir este efecto es necesario renderizar la escena desde varios puntos de vista. Dados dos usuarios 1 y 2, el usuario 1 quiere ver la visión del usuario 2; primero se renderiza la escena desde el punto de vista del usuario 2, el resultado se guarda en una textura de color en la tarjeta gráfica. A continuación, la escena virtual se renderiza con normalidad desde el usuario 1, pero, esta vez aplicando la textura creada en el paso anterior al visor (televisor) del usuario 1.

Para que el impacto de esta funcionalidad en los fotogramas por segundo de la aplicación sea mínimo, primero, la textura que se crea para el visor ha de existir únicamente en la tarjeta gráfica y no en la CPU: toda la información se genera y se usa en la tarjeta gráfica y por lo tanto no hay necesidad de transmitir la información a la CPU. Segundo, la resolución del renderizado de esta textura no es necesario que sea la misma que la del visor de RV, de hecho, como se trata de un elemento pequeño comparado con todo el campo de visión, con muy baja resolución y unos pocos fotogramas por segundo es suficiente.

4.3 Evaluación de los modos de navegación

Uno de los objetivos de este trabajo era evaluar las técnicas de navegación una vez implementadas, no obstante, debido a la declaración del estado de alarma durante la realización de este TFG (para más información ver la sección 6.1.1 y 6.4), no ha sido posible llevar a cabo el estudio con usuarios. Sin embargo, antes de estos sucesos, sí que se pudieron llevar a cabo diferentes pruebas piloto. En este apartado, se explican las primeras conclusiones de estas pruebas, así como el trabajo que se empezó a llevar a cabo para preparar el estudio final.

4.3.1 Pruebas piloto

Durante el desarrollo de los métodos de navegación se realizaron pequeñas pruebas piloto a medida que las funcionalidades del sistema empezaban a estar disponibles, en concreto, se recogieron opiniones de varios miembros del Centro de Realidad Virtual, y se presentaron una parte de las técnicas de navegación a los arquitectos de la Sagrada Familia. Aunque no es un estudio completo, se exponen las ventajas e inconvenientes de cada técnica en función de estas pruebas piloto, el objetivo es ver si reducen el *cybersickness* y la desorientación espacial sin limitar en exceso la libertad de movimiento.

El conjunto de técnicas de navegación se ha dividido en tres para poder comparar la eficacia de los métodos, algunas de las técnicas están disponibles en los tres modos de navegación porque se han considerado complementarias, por lo tanto, se evalúan por separado. A continuación, se listan los diferentes grupos:

- Técnicas disponibles en los tres modos - Este grupo representa las técnicas compartidas entre los tres modos de navegación: Puntos de vista, posicionamiento a nivel del suelo y videocámara.
- Modo de navegación 1 - Este modo representa la libertad máxima, cada usuario puede moverse con vuelo libre y teletransporte, una vez las distancias virtuales y las físicas se han descoordinado, los usuarios no saben la posición física de los demás participantes.
- Modo de navegación 2 - Este es el modo más restrictivo, se define a un usuario como el *master* y será el encargado de realizar todos los movimientos. Es decir, solo se permite el uso de teletransporte y puntos de vista, cuando el *master* realice una de estas acciones, todos los participantes también la harán, de esta forma el grupo siempre se mantiene unido y con las distancias físicas y virtuales en coherencia.

- Modo de navegación 3 - Tiene las mismas funcionalidades que el modo 1, no obstante, se añade la representación visual doble para que los usuarios sepan en todo momento dónde se encuentran físicamente los demás participantes.

Modo de navegación 1

En el primer modo de navegación se han podido observar diferentes comportamientos, el primero está relacionado con las técnicas de teletransporte y vuelo libre. Por un lado, las personas menos acostumbradas al uso de RV han preferido el uso del teletransporte porque genera menos mareos y desorientación espacial, esto se debe seguramente a que el teletransporte te obliga siempre a estar sobre una superficie y son movimientos discretos muy controlados. Por otro lado, las personas más acostumbradas a usar la herramienta prefieren el vuelo libre, pues restringe menos la capacidad de movimiento. En especial en este trabajo, al tratar con modelos arquitectónicos, es de especial interés ya que permite ver cualquier detalle del modelo aunque sea necesario ir volando.

Aunque inicialmente no se había planteado esta idea, se ha observado el uso del teletransporte como método de viaje rápido, aunque en general los usuarios experimentados han usado el vuelo libre, para realizar desplazamientos rápidos entre puntos muy distantes del escenario virtual usaban el teletransporte. Por lo tanto, parece adecuado permitir al usuario usar las dos opciones de desplazamiento.

Por último, el movimiento más natural es el propio movimiento físico, por lo tanto, uno de nuestros objetivos era que el usuario priorizara este tipo de desplazamiento. Con este modo de navegación los usuarios prácticamente no se mueven físicamente, con vuelo libre pueden ir a cualquier posición y al no saber dónde se encuentran los demás participantes no se mueven por miedo a colisionar.

Modo de navegación 2

El segundo modo se caracteriza por ser muy restrictivo, pero ha resultado ser de utilidad para el primer contacto con la aplicación. Al basarse solamente en la teletransportación, se reduce al mínimo la sensación de mareo o desorientación, además, como todos los participantes se mueven de forma conjunta, en todo momento se mantienen las distancias físicas y las virtuales, cada usuario sabe en todo momento dónde están los demás participantes, el resultado es que se aumenta considerablemente el uso del movimiento físico.

Para ejemplificar los casos de uso de este método, en una presentación de la aplicación en la Sagrada Familia se empezó usando este método ya que los usuarios no tenían experiencia previa en RV, el *master* (el que controla el movimiento) era un usuario experimentado y con conocimientos del escenario. El resultado fue muy positivo, los participantes se llevaron una buena primera impresión al no sentir mareos y al reducirse la complejidad del sistema (no controlan el movimiento virtual), y sirvió como fase de acostumbramiento para que posteriormente pudieran probar los otros modos con menos restricciones.

Otra ventaja de este método, es que al respetarse la distancia física y virtual entre participantes, así como la orientación entre ellos, facilita la comunicación. En este método al hablar con el otro usuario, el origen y dirección del sonido coincide con lo que estamos observando en realidad virtual. En cambio, en los otros métodos donde el usuario virtual no está solapado con el físico, puede desorientar ver a tu compañero hablándote desde la distancia y delante de ti, mientras que su voz te llega por detrás y a menos de un metro de distancia.

Modo de navegación 3

El principal cambio que incorpora este modo de navegación es la representación visual doble del usuario, en cuanto la distancia virtual y la física dejan de ser la misma, se activa el doble *avatar* para que los usuarios sepan en todo momento dónde se encuentran los demás usuarios físicamente, pues virtualmente ya lo saben por el *avatar* básico.

Aunque la explicación teórica de los dos *avatares* puede ser confusa, a la práctica los usuarios se han adaptado muy rápidamente al uso de la doble representación, algunos participantes usan la doble representación simplemente como una estatua para evitar colisiones, y otros además para saber a quién dirigirse al hablar dando lugar a interacciones más naturales. La principal ventaja es que los usuarios se sienten más cómodos para realizar movimientos físicos y no colisionar, por lo tanto se fomentan los movimientos naturales en combinación con los virtuales.

El principal inconveniente es que disminuye la sensación de inmersión, el segundo *avatar* recuerda a los participantes que se encuentran en un entorno físico compartido, hecho que es necesario para evitar colisiones, por lo tanto es un compromiso entre inmersión y mejora de la capacidad de movimiento.

Técnicas compartidas

Las técnicas compartidas están disponibles en los tres modos de navegación, el objetivo no es compararlas sino ver sus ventajas e inconvenientes. Los puntos de vista son una herramienta de especial interés en escenarios grandes, el principal uso que han tenido durante las pruebas ha sido el de punto de reunión. A veces es difícil encontrarse en la escena virtual, por lo tanto los usuarios podían usar un punto de vista para encontrarse en el mismo punto. En conjunto con el segundo modo de navegación han sido útiles para planificar la demostración, el *master* puede crear puntos de vista antes de empezar la demostración, y luego usarlos como recorrido con el resto de participantes.

El posicionamiento a nivel de suelo es una funcionalidad muy usada por los arquitectos de la Sagrada Familia, una vez se usa el vuelo libre es difícil posicionarse al nivel del suelo, sin embargo, es importante mantener la altura real del usuario cuando anda sobre una superficie para no romper la inmersión.

Por último, la videocámara es una de las funcionalidades que menos se pudo probar con las pruebas piloto, no obstante, como se exponía antes, cuando dos participantes se han alejado mucho en la escena es bastante difícil que se encuentren, aunque pueden usar los puntos de vista, si simplemente se quiere echar un vistazo rápido a lo que ve el otro usuario, es mucho más rápido activar la videocámara.

4.3.2 Preparación del estudio con usuarios

Antes de la declaración del estado de alarma se empezaron los preparativos del estudio, la idea era analizar el tipo de movimiento que hacían los usuarios en cada modo de navegación y qué técnicas se usaban más. Se preparó un *script* que recogía datos en el servidor de todos los usuarios participantes, los datos se volcaban en formato *CSV* para facilitar su posterior interpretación con *Excel*.

Uno de los datos que se recogían eran las coordenadas virtuales de cada usuario, con esto se podía saber qué distancia había recorrido cada uno de los usuarios dentro de la escena, esta métrica serviría de indicador para analizar la libertad y flexibilidad de movimiento de cada técnica.

También se almacenaban las coordenadas físicas de cada usuario, esta métrica es de especial interés, tiene dos usos principales:

1. Uno de los objetivos es potenciar el movimiento físico de los participantes, con esta métrica se puede medir si han usado este tipo de movimiento o se han quedado quietos.

2. Otro objetivo es la evasión de colisiones, si los usuarios se mueven más cerca entre ellos significa que están más seguros y son más conscientes de la posición de los demás participantes, con las coordenadas físicas es posible calcular la distancia entre usuarios en todo momento.

Por último, también se almacenaban otras métricas que pudieran ser de interés, por ejemplo la orientación de todos los participantes, y otras características relacionadas con técnicas de navegación en concreto, por ejemplo, el número de teletransportes de un usuario o las veces que ha usado los puntos de vista.

5 Representación visual de usuarios

5.1 *Avatar* a partir de cámaras

Uno de los retos en realidad virtual colaborativa, es facilitar la comunicación y la interacción entre los usuarios, de modo que el trabajo en realidad virtual se pueda llevar a cabo de forma similar a como se haría en el mundo real. Cuando la realidad colaborativa se lleva a cabo compartiendo el espacio físico, ofrece la ventaja de que los usuarios pueden utilizar la comunicación verbal de un modo natural, simplemente hablando con el otro usuario. Pero existe otra parte muy importante de la comunicación que es la no verbal, y que consiste en gestos que hacemos mientras hablamos. En el capítulo anterior ya hemos indicado que permitimos que los usuarios puedan apuntar a objetos virtuales mediante un rayo que sale del controlador, lo cual permite un cierto grado de comunicación no verbal. El hecho de tener representaciones virtuales sencillas con ojos y nariz también ayuda a entender a dónde está mirando el otro usuario. Pero esta sencilla representación tienen todavía muchas limitaciones de cara a comunicación no verbal.

Por tanto, tras ver en la sección 4.2.2 una representación virtual simple utilizada para evaluar las técnicas de navegación estudiadas, en las siguientes secciones se estudia una representación virtual alternativa mediante cámaras, con el objetivo de facilitar la interacción y la comunicación entre usuarios.

Uno de los visores de RV que se usan en este trabajo es el *HTC Vive Pro*[18], este visor cuenta con dos cámaras frontales tal y como se puede ver en la Figura 20, dichas cámaras se sitúan a la altura de los ojos para crear aplicaciones de *Mixed Reality* o *Augmented Reality*[12]. Por lo tanto, se pueden obtener imágenes en tiempo real en cada una de las cámaras, como el visor de RV cuenta con una pantalla para cada ojo, en cada pantalla se puede mostrar la imagen de la cámara correspondiente.

Típicamente para visualizar la escena virtual se establecen dos cámaras virtuales en la posición del visor de RV, las cámaras se separan una cierta distancia en función de la separación de los ojos, y seguidamente se renderiza la escena desde cada cámara virtual, cada resultado se envía a la pantalla correspondiente del visor de RV.

[12] A diferencia de *RV* donde el mundo se representa en su totalidad de forma virtual, en MR y AR se mezclan elementos reales con elementos digitales. En concreto, AR superpone información virtual sobre el mundo real, MR mezcla elementos digitales con elementos reales de forma que se interactúa con los dos tipos de objetos indistinguiblemente.

Figura 20: HTC Vive Pro*, delante se pueden ver las dos cámaras frontales a la altura de los ojos.*

Para poder mostrar elementos del mundo real se ha de añadir un paso extra al final: se define un plano para cada cámara, cada plano ocupará todo el campo de visión de su cámara, y cada cámara solo verá su plano (ignorará el otro al renderizar). Primero se renderiza la escena como de costumbre pero ignorando los planos, a continuación, se proyectan, en los planos definidos, las imágenes obtenidas de las cámaras físicas del visor, finalmente, se renderizan los planos encima de la escena.

En la Figura 21 se muestra un ejemplo con un plano y una cámara visualizando la imagen de una cámara del visor de RV, en este caso el usuario solo vería el plano ya que se está renderizando por completo, además, solo puede ver la parte de la imagen que no está distorsionada, por lo tanto no verá la parte de arriba ni los bordes.

Los planos se mueven de forma coherente con las cámaras y el visor, si se renderizan por completo, actúan como una cámara estéreo *see-through*, es decir, el usuario simplemente ve el mundo real. Sin embargo, se pueden visualizar partes concretas del plano y por lo tanto se mezcla el mundo virtual con el real.

Desde el punto de vista de un usuario es posible detectar la posición de los demás usuarios y segmentarlos en el plano. De este modo se vería el mundo virtual combinado con la imagen del otro usuario, que actuaría como su *avatar*. A lo largo de las siguientes secciones se presentan diferentes técnicas para segmentar a los usuarios respecto al entorno físico capturado por las cámaras. Una vez segmentados, se puede crear una máscara que permita renderizar al otro usuario combinado con el entorno virtual.

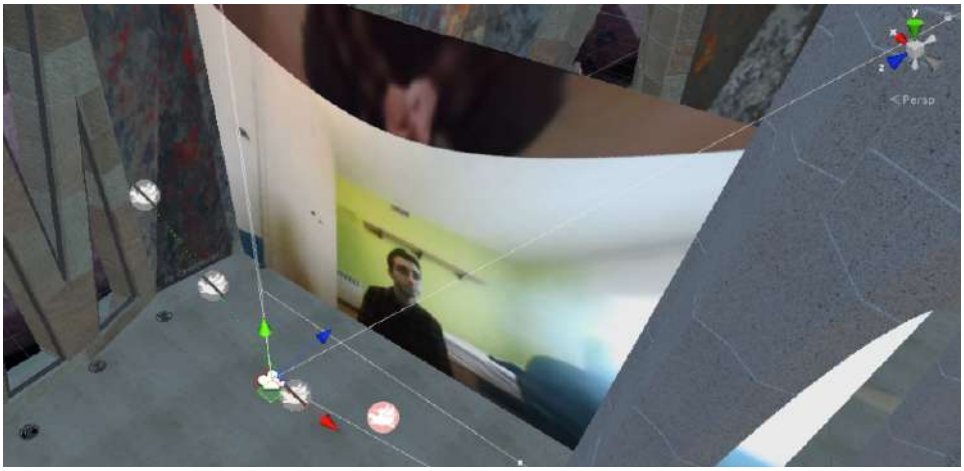

Figura 21: Plano utilizado para renderizar la imagen de una de las cámaras del visor de RV, el usuario estaría posicionado virtualmente en la posición de la cámara. Se repite el mismo proceso para la cámara del otro ojo. El usuario solo puede ver la parte de la imagen rectificada (no ve las distorsiones de arriba y abajo).

5.2 Textura de profundidades

El primer método de segmentación está basado en utilizar texturas de profundidades. Una textura de profundidades almacena información sobre la distancia de los objetos a un determinado punto de vista, es decir, la imagen de profundidades desde el punto de vista del usuario contiene la distancia de los objetos hasta el usuario. En la Figura 22 se puede ver que los puntos más cercanos son más oscuros que los lejanos, no obstante, algunos puntos en negro son indeterminados: es decir, que no se ha podido calcular su profundidad.

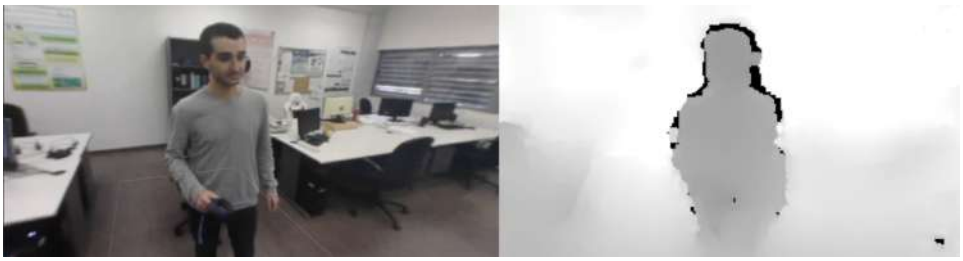

Figura 22: A la izquierda una imagen extraída del visor de RV, a la derecha la textura de profundidades desde el mismo punto de vista.

5.2.1 Cálculo de las profundidades

Cuando se usa una sola cámara se pierde la información 3D sobre la escena, al estar proyectando los puntos a una imagen 2D no es posible recuperar la profundidad de los mismos. No obstante, cuando se usa más de una cámara, es posible triangular la profundidad de los puntos de una imagen.

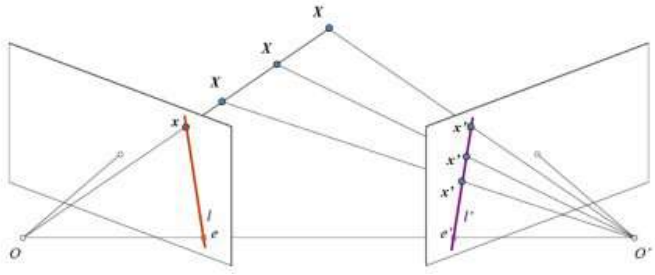

Figura 23: Representación de la epiline del punto x para encontrar su profundidad. Fuente: [24].

Dado un punto x de la imagen de la izquierda de la Figura 23 es imposible calcular su profundidad si solo usamos un plano, dado que todos los puntos de la línea formada por O (posición de la primera cámara) y x se proyectan en x. Sin embargo, los puntos de la línea, l, forman otra línea l' en la imagen de la segunda cámara O', a l' se le llama *epiline* correspondiente al punto x. Para encontrar un punto de la primera imagen sólo se ha de encontrar su epiline en la segunda imagen, por lo tanto, se está simplificando la búsqueda a un espacio 1D[13], hecho que aumenta la precisión y el rendimiento, a esta propiedad se le llama *Epipolar Constraint*.

Para encontrar las *epilines* se han de calcular la *Fundamental Matrix* y la *Essential Matrix*. La *Essential Matrix* contiene la información de la traslación y rotación relativa de la segunda cámara respecto la primera. La *Fundamental Matrix* es como la anterior pero añadiendo información de los parámetros intrínsecos de las cámaras, de esta forma se pueden relacionar las cámaras a nivel de píxel. No obstante, si se usan imágenes rectificadas y se normaliza el punto dividendo las distancias focales, las matrices son iguales. Como las distancias focales son una propiedad de la cámara que se puede conocer, simplemente se calcula la *Fundamental Matrix* que se usará para conseguir las *epilines* en la segunda imagen, de los puntos de la primera imagen.

[13]Una vez encontrada la *epiline* se ha de realizar una búsqueda 1D del píxel a partir del color o similitud.

Para calcular la *Fundamental Matrix* se han de buscar puntos de interés y descriptores en las dos imágenes (por ejemplo, mediante los descriptores *SIFT*[25]), a continuación, con algún algoritmo de *matching* se emparejan los puntos de interés que se usarán para calcular la matriz.

Teniendo en cuenta que las cámaras del visor de RV son paralelas, y que dado un punto de la primera imagen se puede obtener el mismo punto en la segunda imagen con el procedimiento explicado, se obtiene un escenario similar a la Figura 24, f es la distancia focal (que es conocida), B la separación entre cámaras (también conocida), Z la profundidad que se quiere encontrar, x y x' es la distancia del punto al centro de la imagen de su cámara.

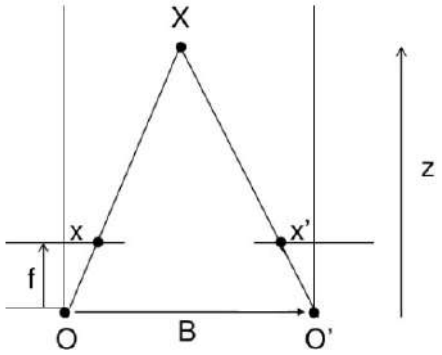

Figura 24: Representación del cálculo de la profundidad de un punto x.

De la Figura 24 se obtiene que:

$$\frac{B}{Z} = \frac{B + x' - x}{Z - f}$$

Por lo tanto, la profundidad Z del punto x se puede calcular con la siguiente fórmula:

$$Z = \frac{Bf}{x - x'}$$

Para emparejar los puntos de ambas imágenes y calcular la profundidad, se ha utilizado la librería de visión por computador *OpenCV*[24] que contiene funciones para realizar dicha implementación. Inicialmente, se programó un código para obtener la textura de profundidades con *OpenCV*, no obstante, el fabricante de *HTC Vive Pro* proporciona una librería (*SRWorks*[26]) para aprovechar las cámaras estéreo del visor. Dicha librería ofrece, entre otras funcionalidades, la posibilidad de obtener la textura de profundidades. Por tanto, finalmente se escogió la librería *SRWorks* ya que obtenía mejores resultados gracias al conocimiento que tiene de las cámaras del visor de RV.

5.2.2 Segmentación mediante *thresholding*

Con la textura de profundidades disponible se han desarrollado diferentes técnicas para segmentar a los usuarios, en esta sección se explica una técnica basada en *thresholding*. Este primer método es rápido de implementar, la dificultad radica en encontrar la profundidad del usuario que se quiere segmentar.

El objetivo es segmentar los usuarios que participan en la aplicación de RV, por lo tanto, todos llevarán un visor de RV, y quizás, uno o dos controladores. Teniendo en cuenta que la posición tridimensional real del visor y de los controladores se puede obtener a partir de las bases del sistema HTC VIVE, es posible proyectar la posición 3D del visor sobre el plano de la cámara, y a partir de esta posición se puede estimar la profundidad del usuario a segmentar.

Para proyectar la posición 3D (del visor o del controlador) a la cámara, se utiliza un método similar al usado al renderizar los modelos virtuales al plano de la cámara (ver Figura 25).

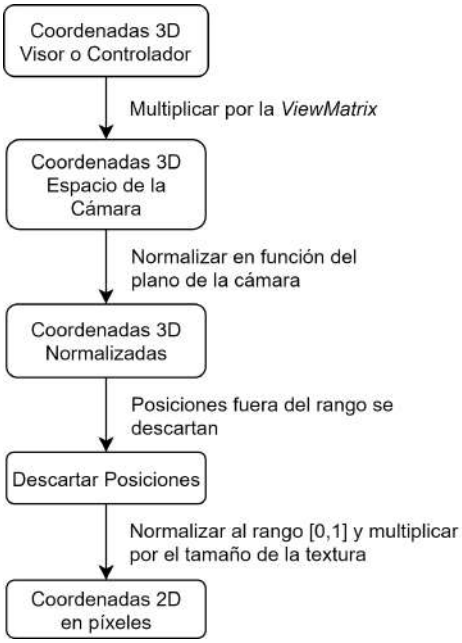

Figura 25: Método para proyectar puntos 3D al plano de la cámara.

El primer paso es poner las coordenadas del punto p del visor en función de la cámara, es decir, se considera un espacio en el que la cámara se encuentra en las coordenadas (0,0,0) y mirando hacia la dirección negativa del eje Z[14]. Para conseguir esto se crea la *View Matrix*, se trata de una matriz que multiplicada a un vector (que representa un punto) y que realiza las siguientes operaciones: traslada el punto restándole la posición de la cámara, aplica al punto la rotación inversa de la cámara y niega el componente del eje Z. El punto p pasará a estar en espacio de cámara, este paso simplifica la normalización de las coordenadas.

El siguiente paso es normalizar p, el objetivo es que los componentes del eje X e Y de p (en el espacio de la cámara) vayan de -1 a 1. Se quiere saber p', que representa la proyección de perspectiva de p en el plano (respecto la cámara), a partir de la Figura 26 se puede ver que la siguiente igualdad se cumple:

$$\frac{p'_{\mathrm{x}}}{Z} = \frac{p_{\mathrm{x}}}{p_{\mathrm{z}}} \qquad\qquad p'_{\mathrm{x}} = \frac{p_{\mathrm{x}}}{p_{\mathrm{z}}} Z$$

p'_{x} y p_{x} representan el componente del eje X de p' y p respectivamente, el cálculo para el eje Y se haría de forma equivalente. Finalmente, para normalizar p'_{x} entre -1 y 1, se ha de dividir por la mitad del ancho (alto en el eje Y) del plano, por lo tanto p' y p normalizadas se calcularían de la siguiente manera:

$$p'_{\mathrm{norm}} = (\frac{\frac{p_{\mathrm{x}}}{p_{\mathrm{z}}} Z}{ancho}, \frac{\frac{p_{\mathrm{y}}}{p_{\mathrm{z}}} Z}{alto}, p_{\mathrm{z}})$$

Todos los elementos se conocen: p es el punto del visor o del controlador en el espacio de la cámara, Z es la distancia del plano a la cámara y *ancho* y *alto* es el ancho y alto del plano.

A continuación, si los componentes del eje X o Y de p'_{norm} se encuentran fuera del rango -1 y 1, o el componente del eje Z es positivo (la cámara mira hacia el eje Z negativo), quiere decir que p no se proyecta en el plano, y por lo tanto, se descarta. Finalmente, se transforma el componente del eje X e Y de p'_{norm} al rango [0,1] y se multiplica por el tamaño de la textura del plano para tener la posición en píxeles.

[14]No es necesario que la cámara esté mirando hacia la dirección negativa del eje Z, podría ser hacia cualquier otro eje o hacia la parte positiva, no obstante, es una convención comúnmente usada.

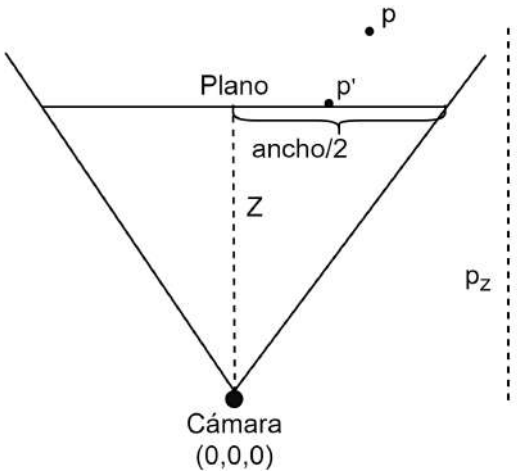

Figura 26: Diagrama de los componentes necesarios para normalizar un punto en coordenadas de cámara, p_z denota la componente del eje Z del punto p.

Después de todo el proceso, se obtiene la posición del visor y de los controladores en la imagen de la cámara, seguidamente, se comprueba la profundidad en estos puntos mediante la textura de profundidades. Si la textura de profundidades es correcta en esa posición, es decir, se ha podido calcular una profundidad para esa posición (si la profundidad es incorrecta la textura de profundidades almacena un 0 para esa posición), la profundidad es la del usuario a segmentar, por lo tanto se aplica *thresholding*.

Se define una profundidad mínima y una máxima a partir de la profundidad obtenida y un porcentaje, seguidamente se compara cada píxel de la textura de profundidades con dichos valores, si están dentro del rango, se aceptan como usuario. Al tratarse de una operación completamente paralela, se realiza con *compute shaders* en la tarjeta gráfica; los *compute shaders* se explican en detalle en la sección 5.5.

Debido a que la aplicación es en tiempo real, esta técnica es útil por su alto rendimiento, simplemente se ha de calcular la posición del usuario y realizar una operación sobre la textura de profundidades, que además, se aplica en paralelo en la *GPU*. Por otro lado, presenta diversos problemas, por ejemplo, el usuario no tiene por qué estar a la misma profundidad en su totalidad, o incluso, puede haber otros objetos a la misma profundidad. En la Figura 27 se puede ver el resultado de aplicar *thresholding*, al no tener en cuenta la conexión entre los píxeles y haber objetos a una distancia parecida a la del usuario (a la izquierda se detecta un mueble a la distancia del usuario), detecta zonas que no corresponden al usuario.

Figura 27: Segmentación mediante thresholding en la textura de profundidades.

5.2.3 Segmentación por inundación

En el apartado anterior se explica una técnica de segmentación basada en *thresholding*, aunque puede dar buenos resultados en entornos abiertos, una de las principales deficiencias es la incapacidad de diferenciar dos objetos si se encuentran a la misma distancia, además, es necesario ajustar el *threshold* en función de las condiciones de iluminación, posición de la cámara, etc. En esta sección, se explica la implementación de un algoritmo de inundación para solucionar los problemas planteados.

La idea principal del algoritmo de inundación es tener dos regiones representando al usuario y al fondo, de forma iterativa se expanden dichas regiones hasta que colisionan los límites entre el usuario y el fondo, y de este modo se determina la segmentación basada en dichas intersecciones. En la Figura 28 se presenta un diagrama del método, seguidamente, se detallan las distintas fases.

Preproceso textura de profundidades

Inicialmente la textura de profundidades se encuentra en la tarjeta gráfica en el formato *Float*, es decir, cada píxel contiene un valor de coma flotante de 32 bits indicando la profundidad en centímetros. El primer paso es convertir esta textura al formato *RGBA*, el cual contiene 4 canales de 8 bits cada uno para cada píxel. La profundidad se almacenará en el primer canal (R), el rango de valores será entre 0 y 255 (se escalan los valores a este rango), aunque se pierde la profundidad exacta en cada píxel, el objetivo es saber qué píxeles están más o menos lejos sin importar el valor exacto, esta discretización se usa más adelante para acelerar el algoritmo de inundación. En el segundo canal (G) se marcan los píxeles con información de profundidad incorrecta: este canal contendrá 0 para los píxeles correctos y 1 para los incorrectos. La información del segundo canal (G) se utiliza más adelante para rectificar la textura de profundidades.

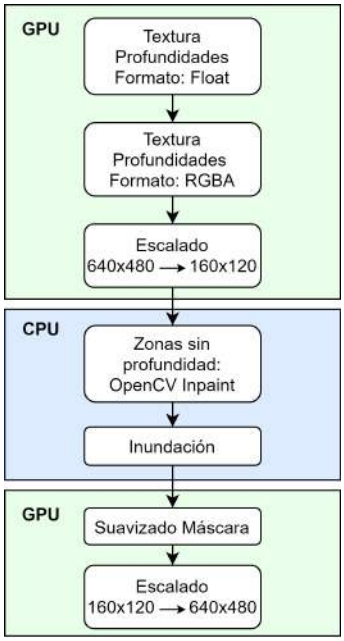

Figura 28: Diagrama general de la segmentación por inundación. En verde las partes que se ejecutan en la tarjeta gráfica, y en azul las que se ejecutan en el procesador.

El siguiente paso es reducir el tamaño de la textura, ya que las dimensiones de la textura son divisibles entre dos (la cámara del *HTC Vive Pro* ofrece imágenes de 640x480 píxeles), para aplicar una reducción se itera sobre bloques de 2x2 píxeles, por cada bloque se calcula el valor medio y se almacena como un solo píxel en otra textura. Si se aplica dos veces este procedimiento, se reduce el tamaño total de la imagen por 16. Este procedimiento se aplica en la tarjeta gráfica para aprovechar el paralelismo (en la Figura 29 se muestra un ejemplo).

El principal problema del escalado es la pérdida de información al reducir el tamaño de la imagen por 16, aun así, para la inundación, una resolución de 160x120 es suficiente para obtener la silueta de la persona. Sin embargo, el escalado ofrece varias ventajas que son importantes para nuestro algoritmo:

1. Eliminación de ruido - El escalado actúa como un filtro de suavizado, esto permite reducir el ruido.

2. Reducir carga *CPU* - Al tratarse de un proceso iterativo, el principal problema de la inundación es que se ha de ejecutar en el procesador, por lo tanto, para un sistema en tiempo real no es factible ejecutar el algoritmo en texturas de 640x480 píxeles.

3. Reducir transferencia *GPU-CPU* - El paso de información entre el procesador y la tarjeta gráfica se ha de reducir al máximo al tratarse de un bus de transmisión relativamente limitado[15].

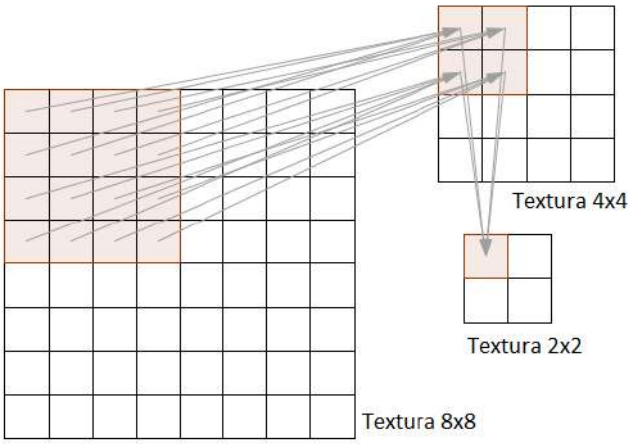

Figura 29: Proceso de escalado de una textura en la tarjeta gráfica. Fuente [27]

El último paso antes de ejecutar la inundación es aproximar la profundidad de las zonas donde no se ha podido calcular, a veces no es posible emparejar algunos puntos de las imágenes para calcular la profundidad, por ejemplo, en un plano con poca variación de color o si el objeto se encuentra muy cerca de la cámara, otras veces simplemente una cámara ve un punto que la otra cámara no es capaz de ver. Para empezar la inundación es conveniente que todos los píxeles tengan una profundidad asociada, por lo tanto es necesario algún tipo de aproximación.

Para la reconstrucción de estas zonas se utiliza el método *Inpaint* de la librería de procesado de imágenes *OpenCV*. Una de las entradas de la función es una máscara con todas las zonas a reconstruir, tal y como se ha visto dos pasos atrás al convertir la imagen a RGBA, en el canal G se guarda un 1 en aquellos píxeles donde la profundidad está mal calculada, por lo tanto la máscara se puede obtener fácilmente a partir del canal G de la imagen. *Inpaint* aproximará el valor de estas zonas a partir de los píxeles vecinos.

[15]En el ordenador utilizado para este trabajo, el bus que comunica la tarjeta gráfica con el procesador es el *PCI Express 3.0 x16*, consta de una velocidad de transmisión de aproximadamente 16GB/s (en cada dirección), y la tarjeta gráfica una NVIDIA 1070 con una velocidad de transmisión a memoria de 256GB/s, en comparación el bus es muy limitado y se ha de reducir su uso al máximo.

Inundación

En esta sección, se explica el algoritmo de inundación a partir del pseudocódigo de la Figura 30. El primer paso es obtener los marcadores iniciales del usuario y del fondo, para los marcadores del usuario se usará el mismo método que en la sección 5.2.2, se obtienen a partir del visor y de los controladores. Para el fondo se utilizan los bordes de la imagen ya que normalmente no pertenecen a la persona, no obstante, solo se utilizan aquellos píxeles que tienen una profundidad muy diferente a la de los marcadores del usuario, inicialmente solo se buscan marcadores que con total seguridad son del fondo, por ejemplo, si el marcador inicial del visor indica que el usuario está a 1 metro, es seguro escoger píxeles del borde que se encuentran a 3 metros.

Entre las líneas 2 y 4 (Figura 30) se inicializa una textura donde se almacenará un 1 en los píxeles del usuario y un 0 en los del fondo, una array para marcar los píxeles ya visitados, y una cola de prioridades donde se almacenan los píxeles vecinos de los píxeles ya procesados, inicialmente los vecinos de los marcadores. Contra menor es la distancia entre dos píxeles (más similar es el valor de la profundidad) antes se procesan, de esta forma las dos regiones crecen hasta encontrarse y crear una frontera. Entre las líneas 6 y 12 se prepara el estado inicial de dichas estructuras a partir de los marcadores iniciales.

El bucle principal transcurre entre las líneas 14 y 21, mientras la cola de prioridades no esté vacía se van obteniendo píxeles sin región asignada, seguidamente, se itera sobre los píxeles vecinos, si el píxel vecino no ha sido visitado quiere decir que no pertenece a ninguna región, se calcula su prioridad y se pone en cola. Antes de acabar la iteración, se comprueba la región mayoritaria entre los píxeles vecinos (con región asignada) y se asigna dicha región al píxel; al menos uno de los píxeles vecinos forma parte de una región, pues para que un píxel esté en la cola de prioridades lo ha tenido que poner en cola uno de sus vecinos con región asignada. Al acabar el bucle la máscara contendrá todos los píxeles pertenecientes al usuario con valor 1.

Para ejecutar este algoritmo en tiempo real se ha de minimizar el trabajo a realizar en cada iteración, a continuación, se analiza brevemente una solución para aumentar sustancialmente la efectividad del bucle principal.

Considerando que n es el número de píxeles a procesar (el tamaño de la textura), se sabe que cada píxel se va a procesar solo 1 vez, pues cuando un píxel es visitado se marca y no se vuelve a insertar en la cola de prioridades. Por cada píxel en el bucle principal (líneas 14 a 21) se realizan 4 operaciones:

```
   Input: Depth : Texture
   Input: Markers : Array Pixels
 1 // Inicialización
 2 mask ⟵ Empty Texture;
 3 visited ⟵ Empty Boolean Array;
 4 priorityQueue ⟵ Empty PriorityQueue;
 5 // Inserta los marcadores del usuario y del fondo
 6 foreach m in Markers do
 7     mask[m] ⟵ m.IsUser() ? 1 : 0;
 8     visited[m] ⟵ True;
 9     foreach n in m.Neighbours() do
10         visited[n] ⟵ True;
11         // Enqueue(priority, pixel)
12         priorityQueue.Enqueue(abs(Depth[n] - Depth[m]), n);
13 // Incrementa las regiones iterativamente
14 while not priorityQueue.Empty() do
15     pixel ⟵ priorityQueue.Dequeue();
16     foreach n in pixel.Neighbours() do
17         if not visited[n] then
18             visited[n] ⟵ True;
19             distance ⟵ abs(Depth[n] - Depth[pixel]);
20             priorityQueue.Enqueue(distance, n);
21     mask[pixel] ⟵ pixel.NeighboursRegion();
22 return mask;
```

Figura 30: Pseudocódigo algoritmo inundación, la función IsUser() devuelve True si el marcador es de usuario, False si es de fondo, Neighbours() devuelve todos los píxeles vecinos de un píxel, priorityQueue.Dequeue() devuelve primero los valores de priority *más bajos, NeighboursRegion() retorna la región mayoritaria entre los píxeles vecinos.*

1. Comprobar si la cola de prioridades está vacía - Se puede conseguir en tiempo $O(1)$ simplemente incrementando un contador cada vez que se inserta un elemento, y decrementándolo cuando se elimina. Por tanto, para saber si la pila está vacía, simplemente hay que consultar si este contador es igual a 0.

2. Obtener el elemento con mayor prioridad - Si se implementa la cola de prioridades con un *binary heap* esta operación tiene un coste de $O(log(n))$.

3. Poner en cola hasta 4 elementos (bucle líneas 16 a 20) - El número de iteraciones de esta bucle es constante ya que cada píxel tiene hasta 4 vecinos, sin embargo, una de las operaciones que se realizan es insertar un elemento a la cola de prioridades, que puede tener un coste de $O(log(n))$ como en el caso anterior.

4. Comprobar la región de los píxeles vecinos - Coste $O(1)$, consiste en un acceso a textura por cada píxel vecino (número constante de vecinos).

Se puede ver que el coste del bucle es $O(nlog(n))$ respecto el número de píxeles, no obstante, en uno de los pasos iniciales se discretiza la profundidad a valores entre 0 y 255, y la función que determina la prioridad es el valor absoluto de la resta de dos profundidades. Por este motivo la cola de prioridades se puede implementar como un vector de 256 colas convencionales[16], cubriendo así todos los posibles valores que puede tomar la prioridad: para insertar un píxel simplemente se accede a la cola con índice el valor de la prioridad, y se inserta el píxel; para eliminar y obtener el píxel con máxima prioridad se iteran las colas de mayor a menor prioridad hasta encontrar una no vacía, a continuación se elimina un píxel. Ambas operaciones tienen coste $O(1)$, pues insertar y eliminar en una cola convencional tiene coste constante. Finalmente, se reduce el coste total a $O(n)$, en la Tabla 2 se presenta una comparación de tiempos entre la cola de prioridades presentada y una implementación con *binary heap*.

[16]El vector de 256 colas se crea al principio de la aplicación, en cada fotograma se reutiliza, pues el coste de creación de dicha estructura de datos podría ser demasiado elevado.

	n = 160x120	n = 320x240	n = 640x480
Binary heap	23ms	89ms	365ms
Vector de colas	3ms	14ms	51ms

Tabla 2: Comparativa colas de prioridades para la inundación, se han medido solo las operaciones de enqueue *y* dequeue*. Los tiempos son un promedio de valores obtenidos en 60 fotogramas.*

Tratamiento de la máscara

Finalmente, una vez obtenida la máscara se envía a la *GPU* para acabar de refinar el resultado. El primer refinado consiste en la interpolación de la máscara entre múltiples fotogramas: se guarda la máscara obtenida y se comprueban las máscaras de un número fijo de fotogramas pasados, por ejemplo, suponiendo que se comprueban 3 fotogramas, se cogería la máscara actual y la máscara de los 2 fotogramas anteriores, si algún píxel no es de usuario en cualquiera de las 3 máscaras se descarta. Este paso añade latencia y por lo tanto es opcional y se puede desactivar.

El siguiente paso consiste en suavizar la silueta de la persona tal y como se puede ver en la Figura 31. Al operar a nivel de píxel, la inundación suele generar una segmentación imperfecta y ruidosa. Se aplica un filtro de desenfoque gaussiano (*gaussian blur*) para suavizar las fronteras de la silueta, previamente se aplica la operación morfológica de dilatación para evitar que el filtro de desenfoque recorte la silueta de la persona[17]. La implementación con *compute shaders* y definición de estas operaciones se explica en la sección 5.5.

Para acabar se escala la máscara al tamaño original, por lo tanto, la imagen final será el resultado de aplicar la máscara a la imagen obtenida de la cámara. Solo se pintarán aquellos píxeles que se encuentren a 1 en la máscara.

[17]Al introducir elementos del mundo real en la simulación virtual es preferible que se vea un poco del fondo, pues aquello que estamos segmentando no puede aparecer cortado o se rompería la inmersión.

Figura 31: A la izquierda la máscara obtenida directamente de la inundación, a la derecha se le ha aplicado interpolación, Gaussian Blur *y la operación morfológica de dilatación.*

Tal y como se puede ver en la Figura 31, el algoritmo de inundación consigue una segmentación más ajustada al usuario que el método de *thresholding*, además, al hacer crecer las dos regiones iterativamente, se consigue que cada región sea completamente conexa, y por tanto, los objetos que están a la misma distancia que el usuario no formen parte de la segmentación. Sin embargo, ninguno de los dos métodos es capaz de ajustar con precisión la silueta del usuario, esto se debe a la baja precisión de la textura de profundidades, seguramente, con cámaras de mayor calidad o *hardware* específico para detectar profundidades sería posible conseguir una segmentación más ajustada.

5.2.4 Otros usos

El objetivo principal de la textura de profundidades desarrollada en estas secciones es segmentar al usuario, no obstante, puede tener otros usos interesantes relacionados con la evasión de colisiones. Cuando un usuario se encuentra en el mundo virtual, con el visor de RV puesto, no tiene información visual sobre lo que ocurre a su alrededor en el mundo real, solo tiene información de los demás participantes gracias a los *avatares*. Aunque el espacio físico se ha de mantener libre de obstáculos, no siempre es posible, una de las posibilidades es detectar posibles colisiones a partir de la textura de profundidades.

La idea es que con la textura de profundidades se puede saber a qué distancia están los objetos, si se detecta que un objeto está muy cerca se podría mostrar para que el usuario no colisione. Para mostrar el objeto se aprovecha el mismo sistema de los planos usado para representar visualmente a los demás participantes (sección 5.1).

Para detectar objetos cercanos primero se establece a partir de qué distancia se empiezan a mostrar objetos, por ejemplo 30 centímetros, seguidamente, a partir de la textura de profundidades se crea una máscara con todos aquellos píxeles a 30 centímetros, posteriormente esta máscara se puede procesar con un filtro de suavizado, finalmente se aplica la máscara a la imagen en color extraída de la cámara.

En función de la dirección de visión del usuario es posible que 30 centímetros sea muy poco, por ejemplo, si el usuario está mirando recto esta distancia será suficiente, sin embargo, si está mirando hacia abajo, se deberían mostrar los obstáculos que se encuentran en el suelo, por lo tanto, la distancia debería ser la altura del usuario aproximadamente. Para tener en cuenta hacia dónde mira el usuario, se establece un mínimo y un máximo y se interpolan estos valores en función de la dirección de visión del visor de RV. Para calcular si la distancia está más cerca del mínimo o del máximo, se usa el producto escalar entre la dirección de visión normalizada y el vector (0,-1,0) (que representa mirando hacia abajo): si el resultado es 1 quiere decir que los vectores son paralelos y por lo tanto se utiliza la máxima distancia, si es 0 entonces se usa la mínima distancia porque los vectores son perpendiculares y el usuario está mirando recto, los valores entre 0 y 1 se usan para interpolar el resultado entre el mínimo y el máximo.

Por último, mostrar de repente el objeto a partir de la imagen de color puede ser demasiado intrusivo, ya que al reemplazar parte del entorno virtual por un objeto real disminuirá la sensación de inmersión del usuario. En la práctica, no es necesario tener tanta información sobre el objeto, es suficiente con mostrar la forma del objeto, por lo tanto, se aplica un filtro de detección de contornos y sólo se muestra el contorno del objeto con el que se va a colisionar. En la Figura 32 se muestra un obstáculo con el que podría encontrarse un usuario, una silla, el sistema detecta que el objeto está demasiado cerca a partir de la textura de profundidades, y lo muestra con un filtro de detección de contornos en la escena virtual.

Figura 32: A la izquierda se muestra una silla que actúa de obstáculo para el usuario, a la derecha, el sistema detecta que está demasiado cerca y la muestra con un filtro de detección de contornos.

5.3 OpenPose

En la sección anterior se presentaban una serie de técnicas para segmentar a los usuarios a partir de la textura de profundidades. A pesar de que el método presentado nos permite detectar al usuario, no permite obtener un ajuste perfecto de la silueta del usuario. Es por ello que dentro de la segmentación del usuario se incluye una parte importante del fondo de la escena. En RV, sería deseable minimizar la visualización de la escena real, y limitarse de forma más exacta a dejar pasar exclusivamente la silueta del usuario. Por tanto, es necesario mejorar la extracción de información exclusiva del usuario. Para ello, empezamos por analizar información obtenida directamente de las imágenes que se obtienen de las cámaras del visor de RV. En esta sección se analizan diferentes técnicas relacionadas con la segmentación a partir de las imágenes en color.

Para detectar la posición del usuario de cara a poder segmentarlo, utilizamos la librería de código abierto *OpenPose* [28]. Se trata de un sistema de detección de personas en tiempo real capaz de extraer la siguiente información:

1. Articulaciones (*joints*) del cuerpo - Detecta la posición de las diferentes articulaciones y partes de una persona, por ejemplo, codos, hombros, etc. Reconoce hasta 25 puntos de interés.

2. Información sobre las manos - Detecta hasta 21 puntos de interés por cada mano para reconocer gestos y la posición exacta de los dedos.

3. Información sobre la cara - Detección de hasta 70 puntos de interés de la cara para reconocer expresiones faciales.

Para este trabajo, por motivos de rendimiento, sólo se extraen los 25 puntos de interés de la detección de articulaciones del cuerpo, es decir, la información relacionada con las manos y la cara no se utiliza.

OpenPose recibe como entrada una imagen y devuelve la posición, en coordenadas de la imagen, de las diferentes articulaciones detectadas. En la Figura 33 se pueden ver los diferentes puntos de interés detectados por *OpenPose*, estos puntos se usarán como marcadores para segmentar al usuario.

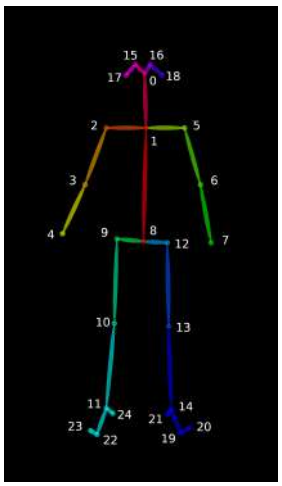

Figura 33: Puntos de interés detectados por OpenPose. Fuente: [28]

5.3.1 Segmentación a nivel de píxel

El primer método de segmentación está basado en el algoritmo de *Region Growing*, la idea es seleccionar una serie de píxeles iniciales que pertenecen al usuario, de forma iterativa se comprueban los píxeles contiguos y se añaden a la región o se descartan, al final de la segmentación se obtendrán una o más regiones pertenecientes al usuario.

El primer paso es reducir el tamaño de la imagen de color, tal y como se explica en la sección 5.2.3, para permitir que la aplicación funcione en tiempo real. Seguidamente, se seleccionan los píxeles iniciales a partir del conjunto de puntos que devuelve *OpenPose* así como los píxeles de los segmentos que unen las diferentes articulaciones del usuario, en la Figura 34 se puede ver a la izquierda el conjunto de píxeles iniciales obtenidos a partir de la información de *OpenPose* y a la derecha el resultado de añadir los segmentos que unen las articulaciones.

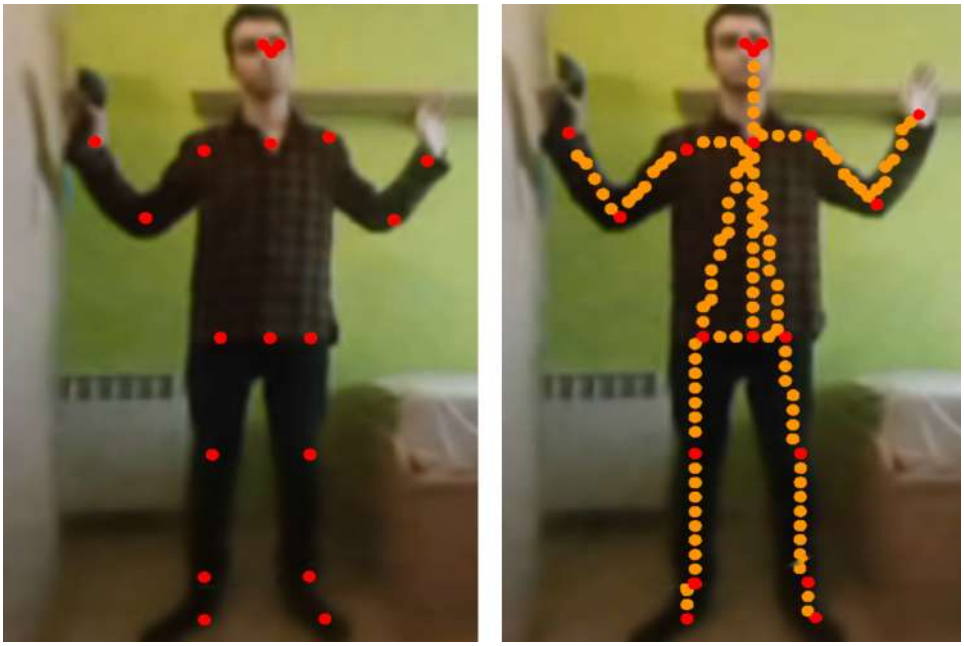

Figura 34: Píxeles iniciales para la segmentación a nivel de píxel. A la izquierda solo se muestran los píxeles obtenidos directamente de OpenPose*, a la derecha se añaden los segmentos que unen dichos píxeles.*

Con los píxeles iniciales definidos, se insertan los píxeles vecinos en una pila. Mientras la pila no se vacíe se van extrayendo píxeles y se determina si son parte de la región o no, si son parte de la región se insertan sus vecinos a la pila, de lo contrario, se descartan. La parte importante de este algoritmo es la elección de los píxeles a comparar y la función de comparación para determinar la distancia entre dos colores.

Para los píxeles a comparar existen dos opciones, cada vez que se extrae un píxel de la pila se puede comparar con el píxel vecino que lo ha introducido, o con el píxel inicial de la región. Para esta aplicación, como se obtienen muchos píxeles iniciales formando el esqueleto del usuario, comparar con el píxel inicial de la región es suficiente, pues las regiones no necesitan ser muy grandes. Por otro lado, si la comparación se realiza entre píxeles vecinos, si hay un cambio gradual de color (por ejemplo por la iluminación), se incluyen en la región píxeles que no se parecen en nada con el color inicial de la región.

Para comparar dos píxeles existen múltiples opciones, se han diseñado tres funciones de distancia (ver Figura 35), cada una con más componentes que la anterior para aprovechar toda la información disponible para segmentar al usuario:

1. Distancia entre colores - La primera opción es simplemente calcular la distancia euclidiana entre los colores de los píxeles, dados los componentes R (rojo), G (verde), B (azul) de un color, se define la distancia euclidiana como:

$$d = \sqrt{(R_2 - R_1)^2 + (G_2 - G_1)^2 + (B_2 - B_1)^2}$$

2. Penalización por distancia - A la ecuación anterior se le puede añadir una penalización en función de lo lejos que se encuentre el nuevo píxel respecto el píxel inicial de la región. Como los píxeles iniciales componen el esqueleto del usuario, tiene sentido que cuanto más lejos se encuentre el nuevo píxel, menos posibilidades tenga de pertenecer al usuario.

3. Profundidad - Por último, y para aprovechar la textura de profundidades desarrollada en el apartado anterior, a la ecuación anterior se le puede añadir la diferencia de las profundidades de los píxeles a comparar, esta diferencia se puede multiplicar por un factor para regular la importancia que tiene.

Figura 35: A la izquierda, se calcula la distancia entre colores con la distancia euclidiana, a la derecha, se añade la penalización por distancia y la profundidad.

Finalmente, se obtiene la región del usuario, se pueden aplicar filtros de suavizado o interpolar el resultado entre diferentes fotogramas, el método sería el mismo que se usaba en la inundación (sección 5.2.3). Por último, se escala la máscara al tamaño original de la imagen y se aplica a la imagen de color obtenida de la cámara.

5.3.2 Segmentación con *SLIC*

En el apartado anterior se presenta una técnica de segmentación a partir de la información de los píxeles, al tratar los píxeles uno a uno, muchas veces estas soluciones no consiguen definir bien la silueta del usuario y es necesario hacer un análisis por grupos de píxeles. Con el objetivo de reducir las imperfecciones del análisis píxel a píxel, en esta sección se explica la implementación que se ha realizado de *SLIC* y un algoritmo de segmentación basado en grafos para segmentar a los usuarios, además, se han incluido las modificaciones necesarias para conseguir la segmentación en tiempo real.

Superpixels y *SLIC*

Un *superpixel* se puede definir como un grupo de píxeles que comparten características similares, por ejemplo, la intensidad o el color. La imagen se divide en un número fijo de *superpixels* en vez de trabajar directamente con píxeles, principalmente, hay dos ventajas:

1. Los *superpixels* tienen más significado que un píxel por si solo, por lo tanto, tiene más sentido segmentar a partir de estas regiones.

2. Los siguientes algoritmos que se usen para segmentar la imagen tendrán una entrada mucho más reducida que si operasen directamente con todos los píxeles de la imagen.

En la Figura 36 se muestra una imagen dividida en *superpixels*, los píxeles se representan con el color del *superpixel* al que pertenecen.

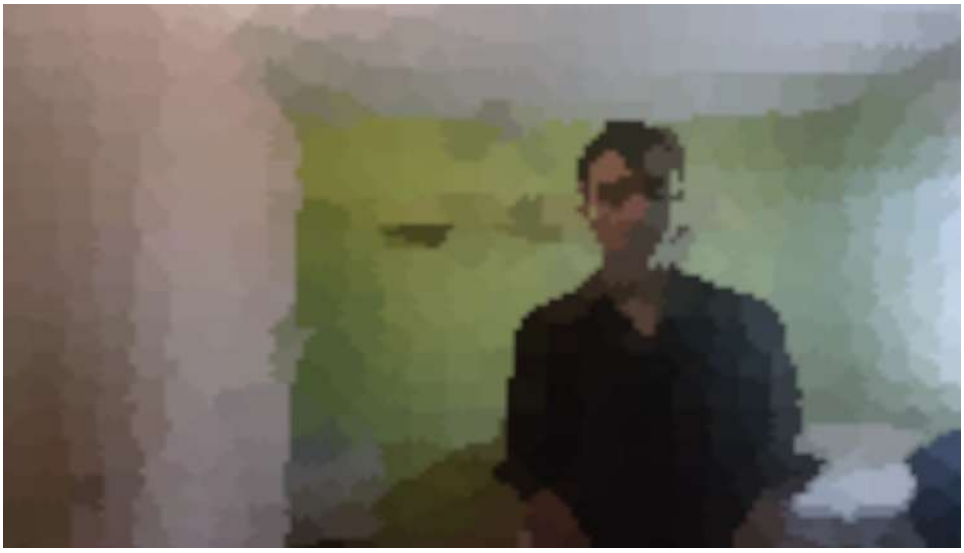

Figura 36: Representación de los superpixels en una imagen. Cada grupo de píxeles del mismo color representa un superpixel.

Algoritmo

Para generar los *superpixels* se ha implementado el algoritmo *Simple linear iterative clustering (SLIC)* [29] que se encarga de agrupar píxeles en función de su color y proximidad. Típicamente, primero se convierte la imagen del formato de color *RGB* a *CIELAB*, pues esta última representación separa la luminancia de la crominancia y por lo tanto es más robusto a los cambios de iluminación, no obstante, la conversión es demasiado compleja y penaliza mucho la eficiencia del algoritmo, por este motivo se utiliza el formato *RGB*.

El algoritmo se describe a partir de la Figura 37, el primer argumento de entrada es K que define el número aproximado de *superpixels* de tamaño similar que se van a generar. El primer paso es inicializar los centros de los *superpixels* muestreando la imagen cada S píxels, para producir tamaños de *superpixel* similares se define $S = \sqrt{N/K}$ donde N es el número de píxeles de la imagen. Por cada *superpixel* se almacenan las tres componentes del color y la posición en la imagen.

A continuación, para evitar que los centros caigan en un píxel con ruido o en un contorno de la imagen, se mueven los centros a la posición con menos gradiente dentro de una región de 3x3 centrada en el *superpixel*.

```
   Input: K : Número aproximado de superpixels
 1 // Inicialización
 2 Inicializar centros de los superpixels C_k = (r_k, g_k, b_k, x_k, y_k)
 3 Mover C_k a la posición con menos gradiente en una región de 3x3
 4 label ⟵ Empty Array;
 5 distance ⟵ Empty Array;
 6 while iterations < MAX_ITERATIONS do
 7     // Asignación
 8     foreach C_k do
 9         foreach píxel p en una región 2Sx2S alrededor de C_k do
10             d = getDistance(C_k, p);
11             if d < distance[p] then
12                 distance[p] ⟵ d;
13                 label[p] ⟵ k;
14     // Actualización
15     Recalcular los nuevos centros
```

Figura 37: Pseudocódigo algoritmo SLIC.

Seguidamente, empieza la fase de asignación, por cada *superpixel* se define una región de búsqueda de tamaño $2Sx2S$, pues el área esperada de cada *superpixel* será SxS y, por lo tanto, es suficiente con buscar en los píxeles cercanos, esta región de búsqueda acelera considerablemente la asignación de píxeles a *superpixels*. Por cada píxel de la región, se compara la distancia con el *superpixel* que se esta procesando, si la distancia es menor a la del *superpixel* asignado actualmente, se reemplaza.

En la fase de actualización, se recalculan los nuevos centros de los *superpixels*, para maximizar el rendimiento, este paso es mejor calcularlo incrementalmente en la fase de asignación y al final simplemente actualizar los valores. Finalmente, una opción es calcular el movimiento de los centros y a partir de un *threshold* decidir si la solución converge o se repite el proceso, no obstante, para limitar el tiempo de ejecución se ha decidido establecer un número máximo de iteraciones.

Distancia

Cada *superpixel* tiene tres componentes de color y dos componentes indicando la posición; es necesario identificar una medida de distancia para poder asignar correctamente los píxeles. No se puede utilizar directamente la distancia euclidiana porque diferentes tamaños de *superpixel* variarían la importancia de la posición en función del color, por lo tanto, se han de normalizar los componentes para poder compararlos adecuadamente. Dado un *superpixel* c y un píxel p Se define d_c como la distancia entre los colores de c y p, y d_s como la distancia entre las posiciones de c y p:

$$dc = \sqrt{(r_c - r_p)^2 + (g_c - g_p)^2 + (b_c - b_p)^2}$$

$$ds = \sqrt{(x_c - x_p)^2 + (y_c - y_p)^2}$$

Para normalizar ds se puede usar $S = \sqrt{N/K}$ ya que representa la máxima distancia esperada en un *superpixel*, para dc se crea un nuevo parámetro m que regula el peso del color respecto la posición, regulando m se pueden tener *superpixels* más o menos compactos. La ecuación final de la distancia es:

$$D = \sqrt{\left(\frac{d_c}{m}\right)^2 + \left(\frac{d_s}{S}\right)^2}$$

Segmentación basada en grafos

Con la imagen dividida en *superpixels* se ha de hacer una segmentación del usuario, para ello se ha implementado un algoritmo basado en grafos. El algoritmo que se presenta a continuación está basado en *GrabCut* [30], pero para poder ejecutarlo en tiempo real se han realizado unas pequeñas modificaciones que se irán presentando en esta sección.

El primer paso del algoritmo (ver Figura 38) es determinar qué zonas seguro que son parte del usuario y cuales del fondo. Para ello se utiliza *OpenPose*, todos los *superpixels* que contengan píxeles del esqueleto extraído de *OpenPose* (la extracción del esqueleto se explica con detalle en la sección 5.3.1) se consideran parte del usuario, a continuación, se calcula el punto máximo y mínimo de los marcadores obtenidos de *OpenPose* para obtener la caja que engloba al usuario, todos los *superpixels* que estén fuera de esta caja (dejando un margen) se consideran parte del fondo.

A continuación, se usa un algoritmo de *clustering* para modelar el usuario y el fondo a partir de los *superpixels*, *GrabCut* utiliza el algoritmo *Gaussian Mixture Model* [31], no obstante, dada su complejidad no es adecuado para un sistema en tiempo real, por eso se ha implementado *k-means*. El objetivo es generar dos clústers: uno para el usuario y otro para el fondo. Típicamente, *k-means* obtiene diferentes resultados en función del valor inicial de los centros de los clusters, para evitar esta situación se inicializan los centros con la media de los *superpixels* que se saben que son de usuario y de fondo. Después de aplicar *k-means* se obtienen los centros de los clústers, que actuarán como representantes de la clase usuario y fondo.

Input: C_k : Superpixels
1 Obtener superpixels del usuario a partir de OpenPose
2 Obtener superpixels del fondo a partir de la caja contenedora del usuario
3 // R_u representantes del usuario
4 // R_f representantes del fondo
5 R_u, $R_f \longleftarrow$ KMeans(C_k);
6 $G \longleftarrow$ buildGraph(C_k, R_u, R_f);
7 $residualGraph \longleftarrow$ ford_fulkerson(G);
8 **return** minCut($residualGraph$);

Figura 38: Pseudocódigo algoritmo segmentación basada en grafos.

Posteriormente se crea un grafo donde los nodos son los *superpixels* y cada nodo tiene 4 aristas conectando sus 4 *superpixels* vecinos, cada arista tiene un peso asociado en función de la similitud de los *superpixels* que conecta, la similitud se calcula con la distancia euclidiana de los colores, cuanto mayor es la distancia, menor peso se asigna a la arista. Además, se añaden dos nodos llamados *source* y *sink*, cuyos valores son los representantes del usuario y del fondo (calculados con *k-means*) respectivamente. Todos los nodos de los *superpixels* se conectan a estos dos nodos, el peso de las aristas también se calcula con la distancia euclidiana.

Si el grafo se interpreta como una red de flujo donde el peso de las aristas es la cantidad de flujo que pueden soportar, y el nodo *source* tiene una capacidad infinita de generar flujo; se busca el mínimo corte *source-sink*, es decir, minimizar el peso de las aristas que hay que quitar por tal de que el grafo se divida en dos conjuntos, un conjunto contiene el nodo *source* y el otro el nodo *sink*, el primer conjunto contendrá los *superpixels* clasificados como de usuario, y el segundo conjunto los clasificados como fondo. En la Figura 39 representa visualmente una segmentación basada en el mínimo corte.

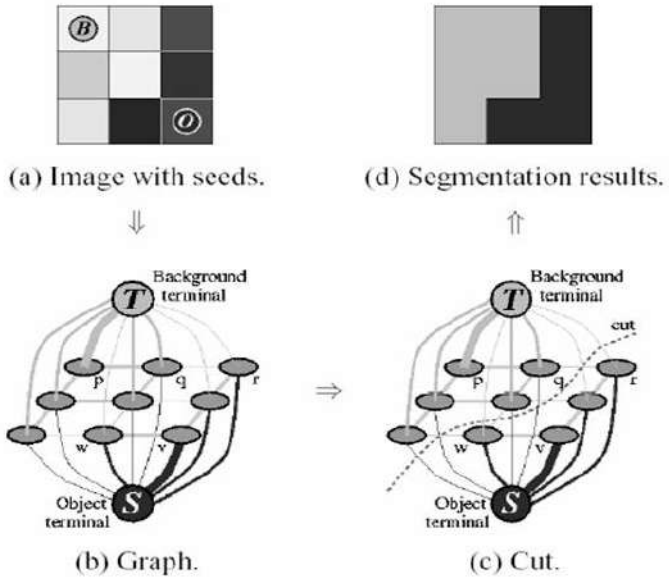

Figura 39: Representación del mínimo corte de la segmentación basada en grafos. S representa el nodo source y T el nodo sink. Fuente: [32]

Para encontrar las aristas que generan el mínimo corte se ha implementado el algoritmo de *Ford-Fulkerson*, que dado un grafo representando una red de flujo genera el grafo residual. El grafo residual indica por qué aristas aún se puede enviar flujo del nodo *source* a *sink*, por lo tanto, si se envía flujo infinito por el nodo *source*, eventualmente, se saturarán las aristas y no habrá más caminos de *source* a *sink*, por lo tanto, se habrá dividido el grafo en dos, las aristas que no permiten ir de un conjunto al otro forman el corte mínimo, pues forman el cuello de botella que no ha permitido seguir aumentando el flujo de la red. Finalmente, se comprueban todos los nodos o *superpixels* conectados a *source* en el grafo residual para segmentar al usuario.

En la Figura 40 a la izquierda se muestra el resultado de la segmentación basada en grafos, a la derecha se aplica un filtro de suavizado para mejorar el resultado final. Con la segmentación basada en la imagen de color se han conseguido mejores resultados que usando la textura de profundidades, no obstante, estos métodos son más costosos y afectan considerablemente al rendimiento de la aplicación.

Figura 40: Resultado de la segmentación basada en grafos. A la izquierda el resultado de la segmentación sin suavizado, a la derecha se aplica suavizado.

5.4 Segmentación a partir de primitivas geométricas

En los anteriores apartados se han trabajado diferentes técnicas de segmentación basadas en la información a nivel de píxel, o se ha trabajado con clústers para conseguir zonas con mayor significado, no obstante, la iluminación, el ruido, y otros aspectos externos modifican la segmentación basada en color y texturas de profundidad haciendo que sea muy difícil conseguir una buena silueta del usuario.

A todo esto se añade la necesidad de computar la segmentación en tiempo real, además los recursos (CPU y GPU) se han de compartir con el resto de la aplicación, pues el renderizado de escenas virtuales en RV ocupa gran parte de los recursos disponibles.

Teniendo en cuenta lo expuesto, a modo de resumen, se busca un método que ha de ser capaz de:

1. Reducir el coste computacional de las técnicas presentadas anteriormente.

2. Conseguir una silueta bien definida del usuario al segmentarlo; aunque deje ver un poco del fondo, es preferible a tener artefactos que rompan con la típica silueta de una persona.

5.4.1 Aproximación mediante primitivas geométricas

Para conseguir los dos puntos expuestos anteriormente se utiliza la aproximación mediante primitivas geométricas simples, se trata de definir una serie de primitivas simples como rectángulos y círculos, y utilizarlos para representar la silueta de un usuario.

Para ello se utiliza la librería *OpenPose* que permitirá conseguir diferentes puntos de interés que servirán de referencia para posicionar las primitivas. En la Figura 41 a la izquierda se muestran los puntos de interés devueltos por *OpenPose* en una imagen, algunos de estos puntos servirán para determinar el centro de un circulo, y otros se agruparán de dos en dos para determinar la orientación de los rectángulos, por ejemplo, para determinar la orientación de la parte alta del brazo se usará el punto del hombro y el del codo, en la Figura 41 a la derecha se puede ver a una persona segmentada completamente a partir de rectángulos y círculos.

Figura 41: Segmentación a partir de primitivas geométricas y OpenPose. A la izquierda se muestran los puntos de interés devueltos por OpenPose, a la derecha una posible segmentación a partir de rectángulos y círculos.

Con *OpenPose* se obtienen los centros de los círculos y las orientaciones y posiciones de los rectángulos, pero, aún falta otro parámetro por calcular: el radio. El significado de radio para los círculos es el habitual, para los rectángulos hace referencia a la anchura. Para calcular el radio se utiliza una red neuronal que se presenta en la sección 5.4.2.

Otro paso importante es la creación de las primitivas geométricas, para ello es necesario rasterizar las funciones que definen los círculos y los rectángulos, es decir, determinar qué píxeles están dentro de las primitivas geométricas. Teniendo en cuenta que la máscara se utilizará en la *GPU* para aplicarse en el *shader* encargado de renderizar la imagen del usuario, no hay necesidad de crear la máscara en la *CPU*, de esta forma se evita la transferencia de datos y se maximiza la eficiencia. La *CPU* solo tendrá que enviar a la *GPU* un punto y un radio por cada círculo y cuatro puntos por cada rectángulo.

Para rasterizar se usan *compute shaders*, los detalles de la implementación se explican en la sección 5.5, a continuación, se explica qué cálculo se ha de realizar en cada píxel[18] para determinar si es parte de alguna primitiva o no.

Para los círculos es suficiente con calcular la distancia del píxel al centro del círculo, si es más pequeña que el radio se acepta, si es más grande se

[18] Aunque hay técnicas más avanzadas para rasterizar en la *CPU*, en la *GPU*, al tratarse de un entorno masivamente paralelo, es muy rápido ejecutar una pequeña operación por cada píxel de la imagen.

rechaza. En el caso de los rectángulos, como pueden tener cualquier orientación, se hace uso de la fórmula:

$$orientación = det(p, q, r) = \begin{vmatrix} q_x - p_x & r_x - p_x \\ q_y - p_y & r_y - p_y \end{vmatrix}$$

p, q y r son tres puntos de la imagen, se quiere saber si r está a la derecha o a la izquierda del vector formado por p y q. La anterior fórmula calcula el área formada por el paralelogramo formado por los vectores pq y qr, este cálculo da positivo si p, q y r están en orden contrario al sentido del reloj, de lo contrario da negativo, o lo que es lo mismo, da positivo si r está a la izquierda del vector pq y negativo si está a la derecha.

Por lo tanto, se forman los cuatro vectores que envuelven el rectángulo a partir de los cuatro puntos que definen al rectángulo, dado un píxel, se computa la fórmula anterior una vez por cada vector (el vector representa pq y el píxel es r), si el píxel tiene la misma orientación en los 4 vectores, quiere decir que se encuentra dentro del rectángulo.

5.4.2 Red neuronal para aproximar el radio

Una vez determinada la posición y orientación de las primitivas geométricas, falta determinar el radio (o el ancho en el caso de los rectángulos). Se ha creado una red neuronal profunda que es capaz de estimar el radio de las diferentes primitivas geométricas que forman la silueta del usuario.

Teniendo en cuenta que ya se utiliza *OpenPose* para determinar la posición y la orientación de las primitivas geométricas, los puntos de interés (que corresponden a diferentes partes del cuerpo y articulaciones) que devuelve *OpenPose* se usarán para la entrada de la red neuronal, el objetivo es que a partir de esta información la red infiera los radios aunque el usuario se encuentre a diferentes distancias o posiciones.

Entrenamiento

El primer paso para crear la red neuronal es preparar el conjunto de entrenamiento, para poder conseguir un conjunto de muestras lo suficientemente grande, se ha optado por crear una serie de scripts con *Blender* y *OpenCV* para generar imágenes de personas artificiales, calcular las posiciones que devolvería *OpenPose* y calcular los radios para cada primitiva.

Primero se usa un modelo 3D realista de un cuerpo humano generado con la librería SMPL [33], se trata de un modelo que se puede cargar directamente en cualquier aplicación de modelado y animación, para este trabajo se usa el *software* de código libre *Blender*. En la Figura 42 se puede ver el modelo 3D importado en *Blender*.

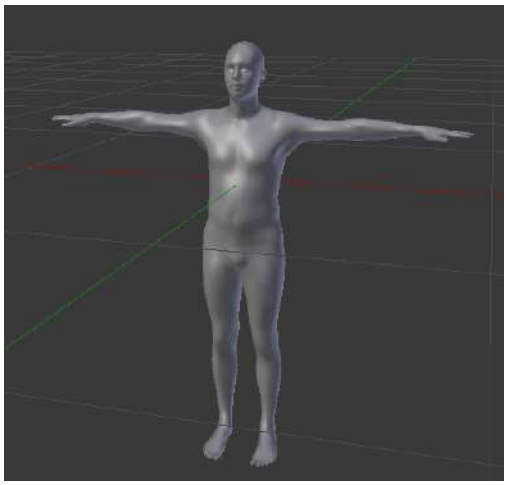

Figura 42: Modelo 3D de una persona creado con SMPL y visualizado con Blender.

De entrada el modelo está en una posición estándar que no es muy útil para generar el conjunto de muestras, para poder cambiarlo de posición se usará la animación mediante huesos, comúnmente llamada *rigging*. La técnica consiste en representar al objeto en dos partes: por una parte está la superficie del objeto a renderizar, y por otra, una serie de componentes ordenados jerárquicamente que forman los huesos del esqueleto. Las diferentes partes de la superficie a renderizar se asignan a los huesos, por ejemplo, la mano estará asignada al hueso de la mano, por lo que al mover el hueso de la mano, se moverá la mano. Como están ordenados jerárquicamente, al mover el hueso del brazo también se moverá el de la mano. Al final, cada hueso representa una transformación geométrica que se aplica a la superficie que se le ha asignado junto a todas las transformaciones de los huesos anteriores en la jerarquía. En la Figura 43 a la izquierda se muestra el esqueleto del modelo 3D y a la derecha en rojo[19] la parte del modelo que se controla con el hueso de la cabeza.

[19]Existen zonas con influencias de varios huesos, por este motivo en la imagen se reduce gradualmente el rojo, porque la influencia del hueso de la cabeza en esas zonas es más baja.

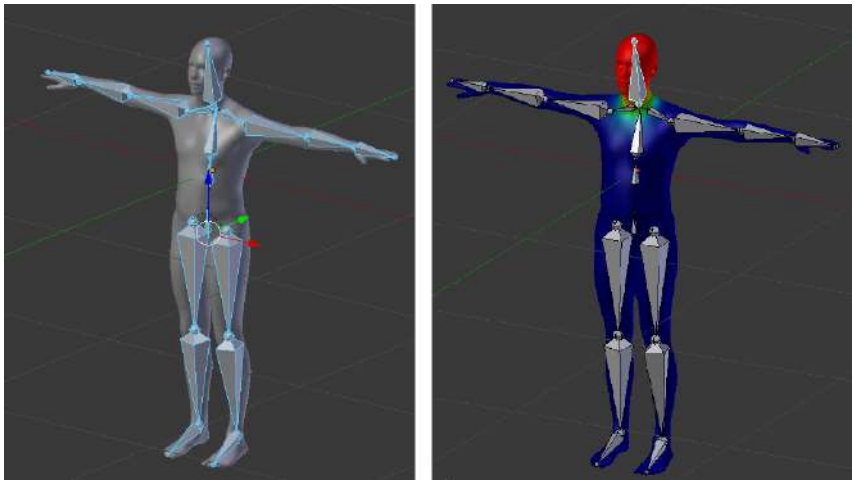

Figura 43: Rigging y asignación de pesos al modelo 3D, a la izquierda el esqueleto del modelo, a la derecha en rojo la zona en la que el hueso de la cabeza tiene efecto, en azul la zona sin influencia.

Una vez hecho el *rigging* del modelo 3D, se pueden aplicar animaciones para modificar la pose del modelo. Las animaciones tienen una duración fija, por cada fotograma asignan la posición y rotación de los huesos del modelo, por ejemplo en la Figura 44 se muestran tres fotogramas de una animación de caminar.

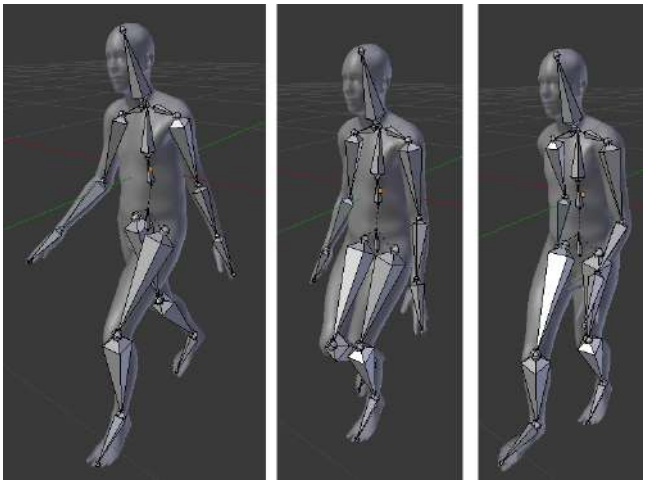

Figura 44: Tres fotogramas del resultado de aplicar la animación de caminar al modelo 3D con huesos.

Se han escogido una serie de animaciones de personas de la base de datos de CMU [34], a partir de estas animaciones y de diferentes posiciones de la cámara, se pueden simular las poses que tendrán los usuarios durante una sesión de RV colaborativa. Además, a partir de la posición de los huesos en cada fotograma, se puede extraer la posición de los puntos de interés que extrae *OpenPose*.

Para poder calcular posteriormente el radio de las primitivas geométricas, se segmenta el modelo por colores para identificar más fácilmente cada parte por separado tal y como se muestra en la Figura 45.

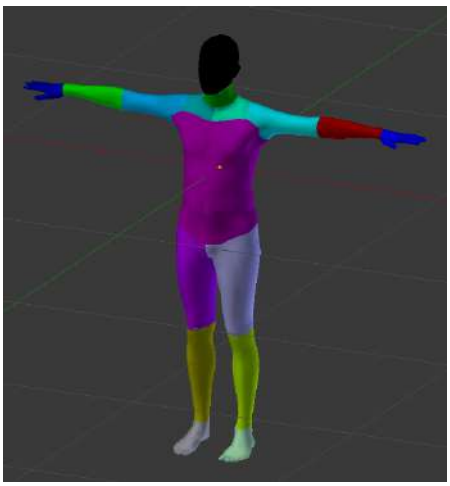

Figura 45: División del modelo 3D en colores para su posterior segmentación.

Por último, se genera una imagen, que se usará más adelante para calcular los radios por cada una de las poses generadas, además, también se guarda un fichero JSON por cada imagen donde se almacena la posición de los huesos. Se ha creado un *script* de *Blender* con *Python* para automatizar todos los pasos descritos, el pseudocódigo se muestra en la Figura 46.

```
Importar modelo 3D
Asignar color a las diferentes partes del cuerpo
Rigging
foreach animación do
    Cargar y aplicar animación al esqueleto
    foreach posición cámara do
        Mover cámara a la posición y mirando hacia el modelo
        // No es necesario coger todos los fotogramas de la
           animación, por ejemplo, con coger un fotograma cada 20
           es suficiente
        foreach fotograma en animación do
            Renderizar imagen a partir de la cámara
            Guardar en un JSON la posición de los huesos
```

Figura 46: Pseudocódigo algoritmo generación muestras en Blender.

Al finalizar el script de *Python* en *Blender*, se obtiene una serie de imágenes, como se muestra en la Figura 47, y ficheros JSON que se han de procesar para obtener los radios de las diferentes primitivas que aproximaran a la persona, para ello, se ha creado otro *script* con *Python* y *OpenCV*.

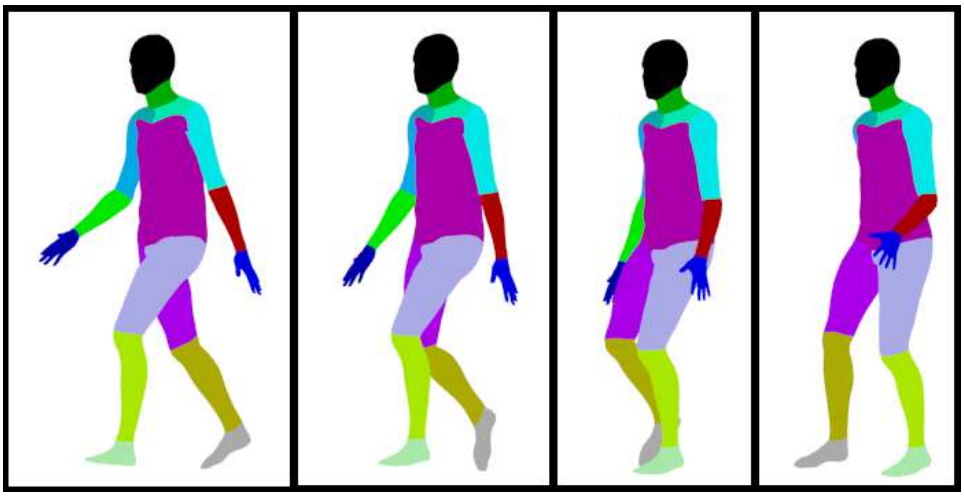

Figura 47: 4 imágenes resultantes del script de Blender.

A continuación, por cada imagen se lee el JSON asociado donde se almacena la posición, en coordenadas de textura, de los diferentes huesos del esqueleto definido anteriormente, con la posición de los huesos se simula el resultado que daría *OpenPose*. Seguidamente, se trata cada parte del cuerpo de forma independiente.

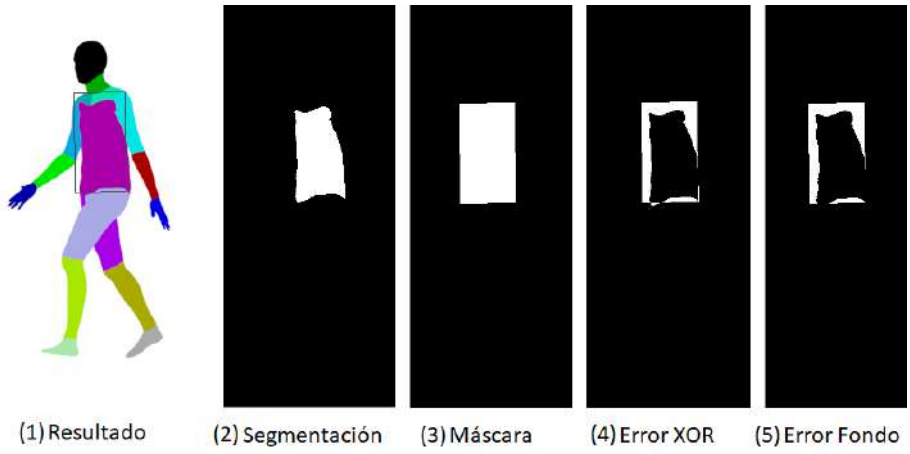

Figura 48: Representación visual del cálculo del error de la primitiva del torso. De izquierda a derecha, (1) resultado del rectángulo que aproxima el torso, (2) segmentación del torso, (3) máscara de la superficie que ocupa la primitiva, (4) primer error realizando la XOR entre (2) y (3), (5) segundo error para penalizar solo los píxeles que se han marcado como de usuario pero son de fondo (para está parte del cuerpo en concreto).

Por cada parte, primero se crea una máscara dónde los píxeles de esa parte del cuerpo estarán a 1 y el resto de píxeles a 0 (segunda imagen Figura 48), esto se puede conseguir fácilmente ya que se ha asignado un color diferente a cada región. Seguidamente, se seleccionan del JSON los puntos de interés necesarios para crear el círculo o el rectángulo que aproximará dicha zona, por ejemplo, para la parte alta del brazo derecho, como se usa un rectángulo, se coge el punto inicial (hombro) y final (codo) del hueso que se encuentra en esta parte.

Con la posición de la primitiva fijada, se crea una nueva imagen, inicialmente con todo a 0, donde se incrementará iterativamente el radio de la primitiva. El proceso es el siguiente: se incrementa el radio de la primitiva, se rasteriza en la imagen y se calcula el error, si el error es menor al de la anterior iteración se sigue iterando, si es mayor entonces se escoge el radio de la anterior iteración y se almacena en el JSON.

Para calcular el error se utiliza el operador lógico XOR píxel a píxel, se compara la máscara de la parte del cuerpo y la imagen con la primitiva rasterizada. Dado un píxel, si el valor en las dos imágenes es el mismo, quiere decir que las dos imágenes lo están marcando como fondo o como usuario, por lo tanto no hay error y la XOR devuelve 0. En cambio, si una imagen determina que un píxel es fondo y la otra que es usuario, o a la inversa, hay un error y la XOR devuelve un 1. Se cuentan los números de 1 para obtener el error, en la cuarta imagen de la Figura 48 se ve el resultado de aplicar la XOR para la primitiva del torso. A este tipo de cálculo del error se le suele llamar *intersection over union*, dados dos conjuntos, se calcula el error como la intersección de los conjuntos dividido por su unión, que es lo que se consigue con el operador XOR.

Uno de los problemas de este error es que muchas veces se prefiere aumentar el radio de la primitiva para cubrir totalmente la zona, y otras veces se quiere priorizar no abarcar ningún píxel del fondo. Por este motivo, se crea un segundo error que cuenta sólo los píxeles de la primitiva rasterizada que no se ajustan a la zona que han de cubrir, es decir, aquellos píxeles que cubren el fondo, para ello, se hace una XOR sólo en el área de la primitiva. En la Figura 48 se puede ver en la última imagen este segundo error, la diferencia entre esta imagen y la cuarta es que sólo se marcan los píxeles que se segmentan como de usuario pero realmente son de fondo.

Este segundo error se multiplica por un factor y se suma al error total, con el factor se puede ajustar si se permiten más o menos píxeles de fondo. En el caso de los círculos, como cubren áreas de especial interés (cabeza y manos) se ha decidido intentar cubrir siempre el máximo de la zona, por este motivo modificar el factor no modifica el resultado en los círculos, en los rectángulos se puede ver el cambio en la Figura 49 con 3 valores diferentes para el factor de penalización.

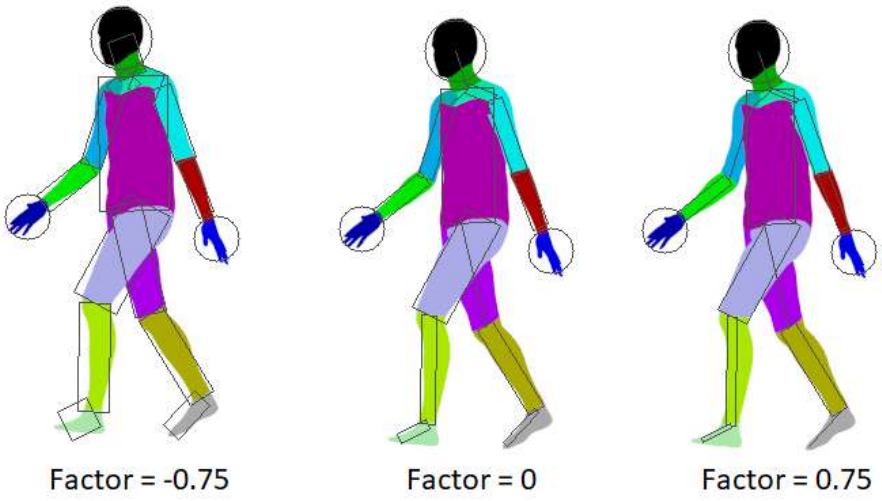

Figura 49: Radio de las primitivas en función del factor de penalización, cuanto mayor es el factor de penalización, más restrictivo es al escoger píxeles del fondo.

Al final del proceso se obtiene una serie de ficheros JSON con las posiciones de los huesos, que serán la entrada de la red neuronal emulando el resultado de *OpenPose*, y los radios calculados para cada primitiva, que serán las salidas de la red neuronal, los radios se usarán para calcular el error de la red neuronal durante el entrenamiento comparándolos con las predicciones.

Modelo

Con el conjunto de entrenamiento preparado, el siguiente paso es crear el modelo para aproximar los radios de las primitivas geométricas, para ello se utiliza una red neuronal profunda creada con *TensorFlow* y *Python*.

El vector de características está formado por los 19 puntos que se utilizan para posicionar las primitivas, que son las posiciones de los huesos o la salida de *OpenPose*, cada punto está formado por 2 elementos, la coordenada horizontal y la vertical, por lo tanto el vector de entrada de la red neuronal está formado por 38 elementos. La capa de la salida es de 17 elementos, que corresponden a los radios de las 17 primitivas necesarias para aproximar la silueta del usuario.

El conjunto de entrenamiento está formado por 104.400 muestras, un 80 % de las muestras se utilizan para el entrenamiento y un 20 % se utilizan como conjunto de *test* para poder evaluar el error una vez entrenado el modelo. Para calcular el error, al tratarse de una red neuronal de regresión, se utiliza una variación del error cuadrático medio: la salida de la red neuronal son radios, pero como representan superficies (porque se utilizan para las primitivas), se ha decidido hacer la resta de los radios al cuadrado, por lo tanto, siendo $\hat{Y}$ un vector de n predicciones y Y el vector con los valores de verdad, la fórmula del error es:

$$error = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y_i}^2 - Y_i^2)^2$$

Como la entrada y la salida son capas fijas, se analiza cuál es el mejor tamaño para las capas densas que se encuentran entre la capa de entrada y la de salida. A mayor cantidad de neuronas y capas más funciones puede representar la red, sin embargo, es más fácil caer en el *overfitting*. Además, es conveniente mantener el número de capas y neuronas al mínimo para poder ejecutar la red en tiempo real.

Primero se fija el número de neuronas a 64 y se experimenta con el número de capas internas, empezando desde una capa hasta cuatro capas. Para determinar el mejor número de capas se analiza el error durante el entrenamiento, así como la gráfica de las predicciones contra los valores de verdad y el error en el conjunto de test, no obstante, como el error en general se mantiene muy bajo en todos los casos, una parte de la decisión recae en la experimentación en la propia aplicación.

En la Figura 50 se pueden ver las gráficas del error durante el entrenamiento, y de los valores predichos contra los de verdad. Se puede observar que el comportamiento en el entrenamiento es similar en todos los casos, aunque a mayor número de capas, antes decrece el error. Respecto los valores predichos[20], el objetivo es que los valores de verdad y los predichos sean los mismos, por lo que en el mejor de los casos todos los puntos estarían en la línea azul. Por un lado, se puede observar que al modelo le cuesta predecir los valores de verdad cercanos a cero, esto seguramente implica que no es capaz de detectar las oclusiones, por otro lado, parece que la gráfica mejora modestamente hasta en la capa 3, especialmente para valores de radio grandes.

[20]En la gráfica de los valores predichos contra los de verdad, se forman franjas verticales porque los valores de verdad antes de normalizarse se representaban en píxeles, al tratarse de valores discretos, al pasarlo a una gráfica continua se forman estas franjas verticales.

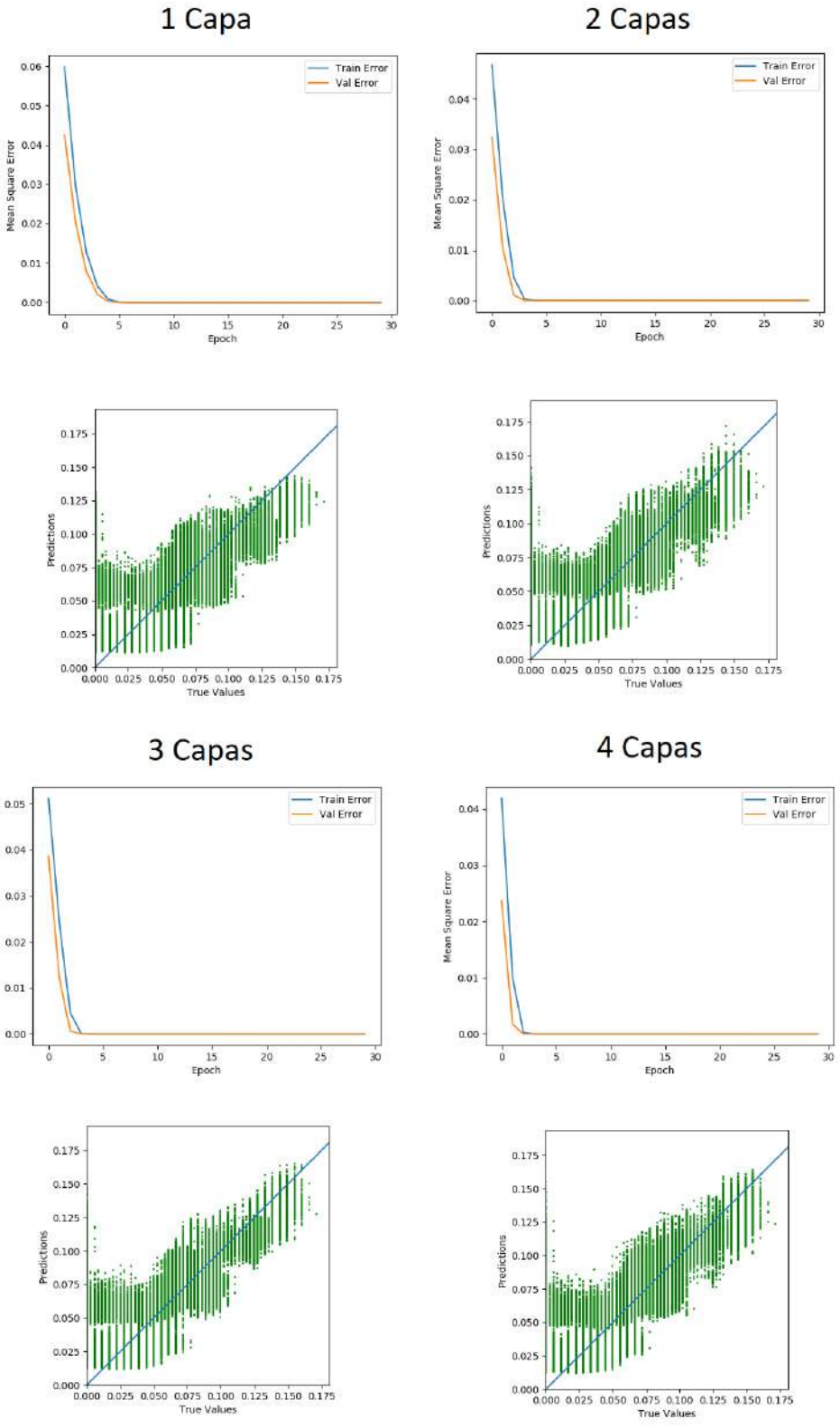

Figura 50: De izquierda a derecha y de arriba a abajo, se comparan de 1 a 4 capas con 64 neuronas cada una. Por cada capa, arriba las gráficas del error durante el entrenamiento, abajo los valores predichos contra los de verdad.

Por último, en la Tabla 3 se compara el error en el conjunto de test. El error es muy bajo en todos los casos porque los radios se han normalizado en función del tamaño de la imagen, hecho que ocasiona valores muy pequeños en la mayoría de los casos. Se puede ver que una capa genera más error, y por lo tanto se deberá escoger un número de capas superior a 1.

Finalmente, se ha decidido utilizar 3 capas porque la gráfica de valores predichos contra los de verdad es ligeramente mejor, el error es el mejor obtenido, y experimentalmente los resultados eran mejores.

Número de capas	Error conjunto de test
1	3.42e-06
2	2.88e-06
3	2.79e-06
4	2.93e-06

Tabla 3: Error conjunto de test en función del número de capas.

Para seleccionar el número de neuronas por capa se ha seguido el mismo procedimiento fijando el número de capas en 3. Se ha probado desde 16 hasta 128 neuronas por capa, a partir de 128 la red empieza a ser demasiado grande y a afectar negativamente en el rendimiento de la aplicación.

En la Figura 51 se puede ver que a mayor número de neuronas, antes converge la red neuronal, esto podría indicar *overfitting*, por lo tanto, hay que evitar un gran número de neuronas. En las gráficas de los valores predichos se puede observar que ligeramente mejora la predicción a mayor número de neuronas, hecho que se refleja también en la Tabla 4.

Número de neuronas por capa	Error conjunto de test
16	3.53e-06
32	3.42e-06
64	2.79e-06
128	2.64e-06

Tabla 4: Error conjunto de test en función del número de neuronas.

Finalmente, se ha decidido utilizar 64 neuronas por capa ya que el error mejora sustancialmente de 32 a 64 neuronas, y experimentalmente se han obtenido buenos resultados con buenos tiempos de ejecución.

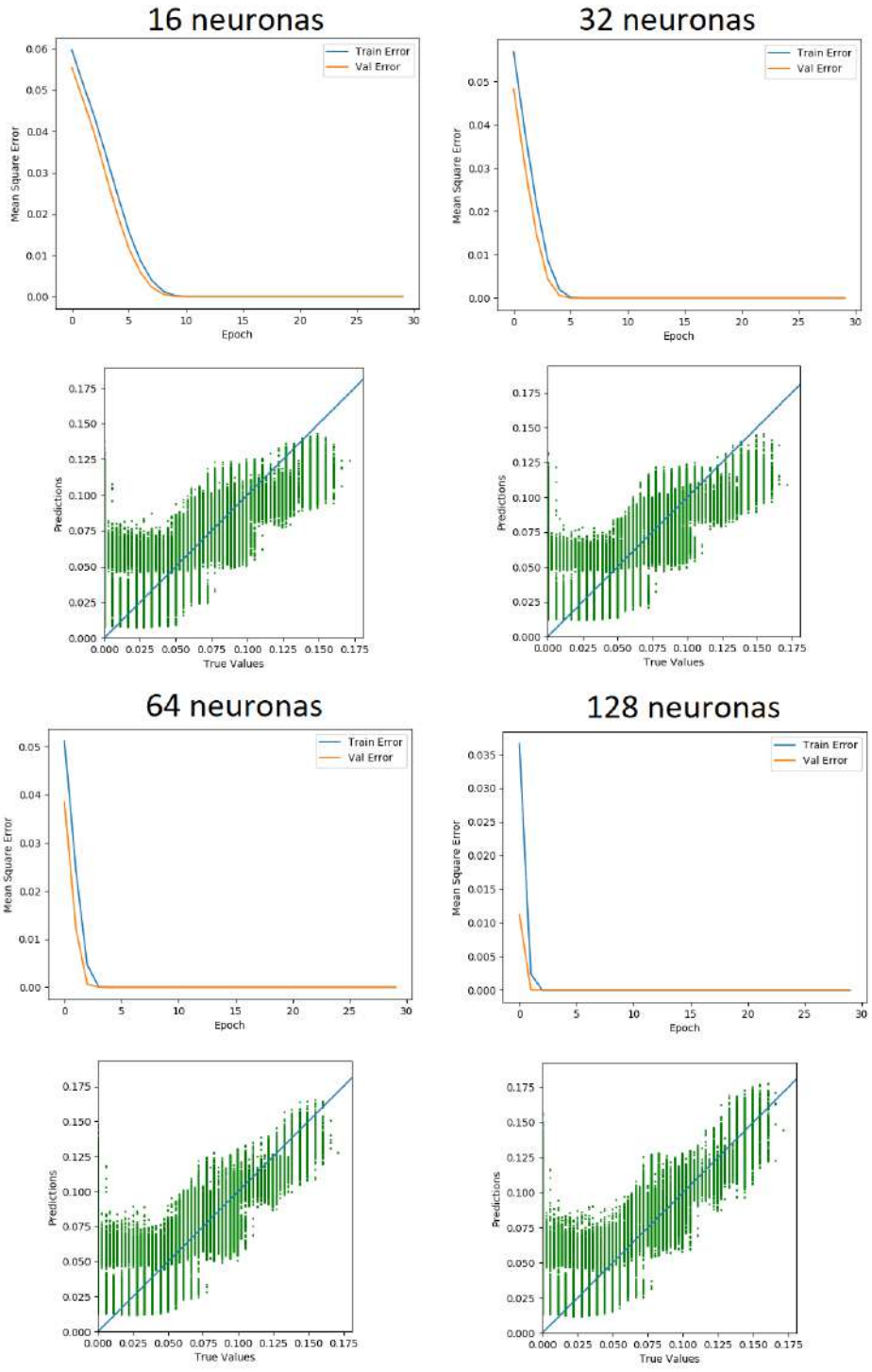

Figura 51: De izquierda a derecha y de arriba a abajo, se comparan de 16 a 128 neuronas en cada capa (3 capas). Por cada grupo, arriba las gráficas del error durante el entrenamiento, abajo los valores predichos contra los de verdad.

La red neuronal final consiste en una capa de 38 neuronas de entrada, 3 capas densas de 64 neuronas cada una, y una capa final, también densa, de 17 neuronas (1 neurona por cada radio de salida). Una vez entrenada, en la aplicación se introducirán los datos extraídos de *OpenPose* y la red predecirá los radios de las primitivas geométricas.

5.5 Implementación en la *GPU*

En muchas de las técnicas desarrolladas en este trabajo ha sido necesario implementar partes del código en la tarjeta gráfica, al tratarse de un *hardware* orientado a ejecutar código en paralelo, el modelo de programación es ligeramente diferente al de una *CPU*. Por este motivo, en esta sección se muestra cómo se ha programado en la tarjeta gráfica, así como algunos fragmentos de código.

La tarjeta gráfica se programa a través de *shaders* que definen cómo se pintará una escena virtual, recientemente, gracias a herramientas como *CUDA* o los *compute shaders* es posible utilizar la *GPU* para tareas genéricas, es decir, no necesariamente relacionadas con la renderización de escenas virtuales. En este trabajo se utilizan *compute shaders* porque se integran de manera natural con aplicaciones gráficas como la de este TFG.

La principal característica de los *compute shaders* es que se pueden definir entradas y salidas arbitrariamente, es posible enviar y recibir variables de tipos simples, estructuras, vectores y texturas. La información se almacena en la memoria global de la tarjeta gráfica y se opera a través de los *kernels*. Un *kernel* es una función que se ejecuta en la tarjeta gráfica.

De la misma forma que en los procesadores multinúcleo, el trabajo se divide en *threads* o tareas. Cada *thread* representa una porción de trabajo que se ha de ejecutar en una de las unidades de cálculo. En una *CPU* al disponer de pocos núcleos, cada tarea representa una parte importante de trabajo, en una *GPU* ocurre todo lo contrario, se dispone de miles de núcleos que realizan tareas muy concretas.

La porción de código que ejecuta cada *thread* se específica a través de los *kernels*. Todos los *threads* ejecutan el mismo *kernel*, por ejemplo, si se quiere pintar de color rojo una textura de 640×480 píxeles, se invocan 307.200 *threads*, cada uno ejecuta un *kernel* que cambia el color de un sólo píxel, cada *thread* se asigna a un píxel, por lo tanto la textura se pinta completamente de rojo.

A continuación, se presentan los principales *compute shaders* implementados en este trabajo.

Operaciones morfológicas

En visión por computador existen una serie de filtros que se usan para modificar las propiedades y estructura de una imagen. Las operaciones morfológicas se encargan de modificar la estructura geométrica de una imagen a partir de una elemento estructurante. Los elementos estructurantes pueden tener diferentes formas, para simplificar se ha usado un cuadrado, es decir, por cada píxel p de la imagen se inspeccionan los píxeles vecinos en un cuadrado de radio n centrado en p, el cuadrado tiene tamaño $(2n+1) \times (2n+1)$. Se han implementado 4 tipos de operaciones morfológicas para imágenes binarias:

1. Erosión - Si algún píxel del elemento estructurante es 0, se cambia el valor de p a 0.

2. Dilatación - Si algún píxel del elemento estructurante es 1, se cambia el valor de p a 1.

3. Apertura - Primero se erosiona la imagen y seguidamente se dilata, con esto se consigue eliminar pequeñas zonas a 1, mientras que se mantiene la estructura de las zonas más grandes que el elemento estructurante.

4. Cierre - Lo contrario a la apertura, primero se aplica la dilatación y seguidamente la erosión.

En la Figura 52 se puede ver el resultado de aplicar las diferentes operaciones morfológicas implementadas sobre una máscara de un engranaje. Se ha usado un elemento estructurante de $n = 5$, por lo tanto, en la dilatación se han agrandado los bordes por 5 píxeles, en la erosión se han disminuido, y en el cierre se han tapado aquellas partes más pequeñas que el elemento estructurante (un cuadrado de 11×11 píxeles).

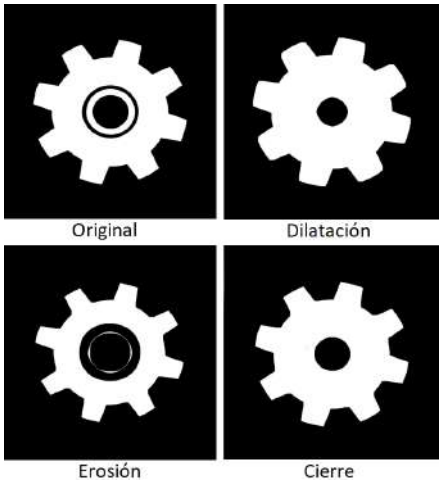

Figura 52: Resultado de aplicar diferentes operaciones morfológicas sobre la máscara de un engranaje, se ha usado un elemento estructurante de $n = 5$. La dilatación amplia la superficie, la erosión disminuye la superficie, el cierre simplemente tapa aquellos agujeros que son más pequeños que el elemento estructurante.

Para implementar estas operaciones con *compute shaders*, se genera una matriz de *threads* del tamaño de la imagen, pues así cada *thread* tiene un identificador horizontal y otro vertical, de esta forma cada *thread* corresponde a un píxel de la imagen. Una opción es aplicar el elemento estructurante en cada píxel, no obstante, para $n = 5$ se han de comprobar $11 \times 11 = 121$ píxeles. Estos filtros son separables, lo que quiere decir que se puede aplicar primero la componente horizontal del filtro, es decir, usar un filtro de 11×1, y luego sobre la imagen resultante aplicar el filtro vertical 1×11, de esta forma se reducen las operaciones por píxel a 22.

En la Figura 53 se puede ver el *kernel* utilizado para implementar la componente horizontal de la erosión. El parámetro *id* almacena el identificador del *thread* actual, como los *threads* se inician en forma de matriz, *id* tiene dos componentes x e y. A partir de esta información, se determina el primer y el último píxel a procesar a partir de la variable *Radius* (simboliza n). Con un bucle se recorren los $2n+1$ píxeles en horizontal y se coge el valor mínimo, el valor de cada píxel se extrae de la textura $Input$. Finalmente, en la textura *Output* se almacena el valor mínimo, es decir, si había algún 0, se guarda un 0. Para la componente vertical del filtro se haría el mismo proceso pero iterando en vertical. Para el filtro de dilatación se usa el máximo en vez del mínimo.

```
void HorizontalErode(uint3 id : SV_DispatchThreadID)
{
    uint start = max(id.x - Radius, 0);
    uint end   = min(id.x + Radius, Resolution.x - 1);

    float value = 1;
    for (uint x = start; x <= end; x++)
    {
        value = min(value, Input[uint2(x, id.y)].r);
    }
    Output[id.xy] = float4(value, 0, 0, 1);
}
```

Figura 53: Kernel de la operación morfológica erosión, corresponde sólo al filtro horizontal, este código se ejecuta una vez por cada píxel de la imagen.

Filtro de suavizado *Gaussian Blur*

Otro de los filtros de procesado de imagen que se han implementado es el filtro de suavizado *Gaussian Blur*, este filtro se ha utilizado principalmente para eliminar las imperfecciones en los bordes de las máscaras al segmentar a un usuario, pues suaviza la imagen consiguiendo transiciones y formas más continuas. De la misma manera que las operaciones morfológicas, *Guassian Blur* es un filtro que se aplica a cada píxel de la imagen, y por lo tanto, es conveniente usar *compute shaders* para aprovechar al máximo el paralelismo.

Para implementar *Gaussian Blur* se fija un filtro de 9×9 píxeles, para conseguir más o menos suavizado simplemente se aplica el filtro repetidamente, por ejemplo, en la Figura 54 se puede ver el resultado de aplicar el filtro una y nueve veces a la misma imagen.

Figura 54: De izquierda a derecha, imagen original, filtro Gaussian Blur aplicado 1 vez, filtro Gaussian Blur aplicado 9 veces.

Se tiene un filtro de 9×9 píxeles que se aplica centrado a cada píxel p de la imagen, a diferencia de las operaciones morfológicas donde se buscaba el máximo o el mínimo, en este caso el filtro almacena una ponderación para cada píxel en función de su distancia al píxel central, el resultado es la suma del color de todos los píxeles ponderados. Para calcular los factores de ponderación del filtro se utiliza la función gaussiana de dos dimensiones:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

La función gaussiana de dos dimensiones se puede calcular multiplicando dos funciones gaussianas de una dimensión, por lo tanto, se pueden separar los filtros tal y como se hizo en las operaciones morfológicas, reduciendo las operaciones por píxel de $9 \times 9 = 81$ a 2 filtros de 9×1 y 1×9, por lo tanto 18 operaciones por píxel. Por último, a efectos prácticos, como la función gaussiana es la función de distribución de la distribución normal, y la distribución normal discreta es equivalente a la distribución binomial, se usan los coeficientes binomiales para determinar los factores de ponderación del filtro.

Aunque la implementación con filtros separables es muy eficiente, aún es posible reducir el coste de aplicar este filtro. La operación más costosa del *kernel* son los accesos a textura, se han de realizar 9 accesos a textura por cada aplicación del filtro horizontal, y otros 9 para el filtro vertical. No obstante, las tarjetas gráficas tienen *hardware* de función fija para realizar múltiples consultas a una textura con un sólo acceso.

Para ello se utiliza el filtro bilineal, con las coordenadas apropiadas devuelve la interpolación de dos píxeles contiguos en la textura. Para aprovechar este filtro es necesario recalcular los factores de ponderación del filtro, y las coordenadas en las que se hará el acceso a textura, de forma que interpolen los colores de los dos píxeles en función de la ponderación obtenida por los coeficientes binomiales. Si $offset$ es la distancia del píxel central al centro de otro píxel, $weight$ es el factor de ponderación de un píxel, y p_1 y p_2 son dos píxeles contiguos, se obtienen las dos siguientes formulas:

$$weight(p_1, p_2) = weight(p_1) + weight(p_2)$$

$$offset(p_1, p_2) = \frac{offset(p_1) \times weight(p_1) + offset(p_2) \times weight(p_2)}{weight(p_1, p_2)}$$

Por lo tanto, los 9 accesos a textura de cada filtro se pueden reducir a 5 accesos a textura: el píxel central tiene su propio acceso a textura, los otros 8 se han agrupado en grupos de 2. En la Figura 55 se puede ver el *kernel* final utilizado para implementar el filtro horizontal del *Gaussian Blur*.

```
void HorizontalBlur(uint3 id : SV_DispatchThreadID)
{
    float2 uv = id.xy / Resolution + TexelSize * 0.5;

    float4 c = Input.SampleLevel(LinearClampSampler, uv.xy, 0) * 0.227027027;

    float2 stride = float2(TexelSize.x, 0.0);

    float2 d1 = stride * 1.3846153846;
    c += Input.SampleLevel(LinearClampSampler, uv.xy + d1, 0) * 0.3162162162;
    c += Input.SampleLevel(LinearClampSampler, uv.xy - d1, 0) * 0.3162162162;

    float2 d2 = stride * 3.2307692308;
    c += Input.SampleLevel(LinearClampSampler, uv.xy + d2, 0) * 0.0702702703;
    c += Input.SampleLevel(LinearClampSampler, uv.xy - d2, 0) * 0.0702702703;

    Output[id.xy] = c;
}
```

Figura 55: Kernel del componente horizontal del filtro Gaussian Blur, se hacen 5 accesos a textura aprovechando la interpolación bilineal del hardware específico de la GPU.

Rasterización de círculos y rectángulos

Por último, para generar máscara del usuario a partir de primitivas geométricas se han usado *compute shaders*. Una ventaja es que al igual que en los otros filtros, al hacer operaciones independientes en cada píxel es posible aprovechar la arquitectura de la tarjeta gráfica. Además, en este caso, al generar la textura directamente en la *GPU* no es necesario enviarla a la *CPU*, pues el pintado final de la escena se realiza en la misma tarjeta gráfica. En la Figura 56 se puede ver el resultado de rasterizar 3 círculos y 3 rectángulos de diferentes tamaños y posiciones.

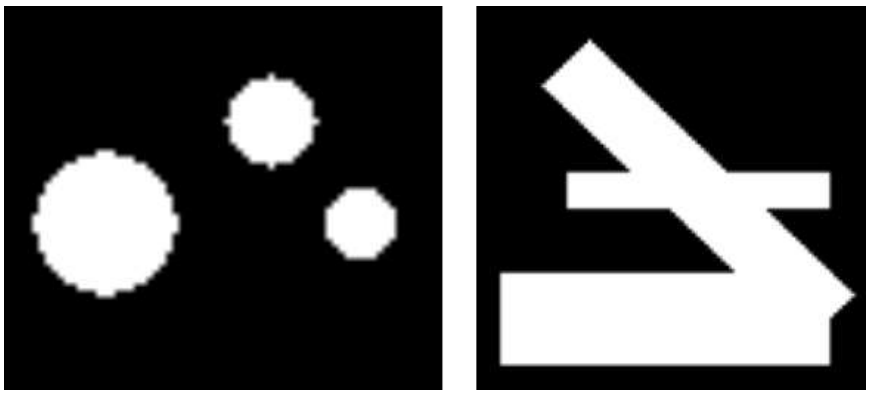

Figura 56: A la izquierda el resultado de rasterizar 3 círculos de diferentes tamaños, a la derecha se han rasterizado 3 rectángulos diferentes.

Se han creado dos *kernels*, uno para rasterizar los círculos y otro para los rectángulos. Básicamente, cada *kernel* procesa un píxel y decide si el píxel se encuentra dentro de algún círculo o algún rectángulo. La explicación teórica sobre como decidir si un píxel pertenece a una primitiva se encuentra en la sección 5.4.1. En la Figura 57 se muestran los *kernels* usados para rasterizar círculos y rectángulos, en ambos casos se recorren todas las primitivas y se comprueba si son parte del interior de alguna primitiva. Los círculos se representan con un centro y un radio, los rectángulos con las coordenadas de las cuatro esquinas.

```
void RasterCircle (uint3 id : SV_DispatchThreadID)
{
    uint white = 0;
    for (uint i = 0; i < C_SizeArrays; i++)
    {
        float2 v = id.xy - C_Points[i];
        white = white | dot(v, v) < C_SqrRadius[i];
    }
    Result[id.xy] = float4(white, 0.0, 0.0, 1.0);
}
void RasterQuad(uint3 id : SV_DispatchThreadID)
{
    float2 a, b, c, d;
    float white = 0;
    for (uint i = 0; i < Q_SizeArrays; i++)
    {
        a = Q_QuadsA[i];
        b = Q_QuadsB[i];
        c = Q_QuadsC[i];
        d = Q_QuadsD[i];
        float2 r = id.xy;
        // a-b
        int d1 = ((b.x - a.x) * (r.y - a.y)) - ((r.x - a.x) * (b.y - a.y));
        // b-c
        int d2 = ((c.x - b.x) * (r.y - b.y)) - ((r.x - b.x) * (c.y - b.y));
        // c-d
        int d3 = ((d.x - c.x) * (r.y - c.y)) - ((r.x - c.x) * (d.y - c.y));
        // d-a
        int d4 = ((a.x - d.x) * (r.y - d.y)) - ((r.x - d.x) * (a.y - d.y));

        white = max(white, min(min(min(sign(d1),sign(d2)),sign(d3)),sign(d4)));
    }
    Result[id.xy] = float4(max(Result[id.xy].r, white), 0.0, 0.0, 1.0);
}
```

Figura 57: Arriba kernel para rasterizar círculos representados por un centro y un radio, abajo kernel para rasterizar rectángulos representados con las coordenadas de las cuatro esquinas.

5.6 Resultados y comparación

En este trabajo se han implementado y probado múltiples técnicas para segmentar a los usuarios en RV colaborativa, en todos los casos se usan las cámaras disponibles en el visor de RV *HTC Vive Pro*. Por último, se hace una comparación final de las técnicas, se tiene en cuenta tanto el resultado visual como el rendimiento, pues al tratarse de una aplicación en tiempo real, es muy importante utilizar una técnica de bajo coste.

En la Figura 58 se muestra una comparación visual de las 5 técnicas de segmentación implementadas, no se ha aplicado ningún filtro de suavizado. En la Figura 59 se muestra una comparación entre la segmentación por inundación sobre la textura de profundidades, y la segmentación con primitivas geométricas, en esta figura sí se aplica un filtro de suavizado sobre la máscara final para evitar las imperfecciones en la segmentación del usuario. Se puede ver como la segmentación por primitivas geométricas consigue aproximar la silueta del usuario con mayor precisión.

En la Tabla 5 se compara el rendimiento de las diferentes técnicas en fotogramas por segundo (FPS). Los FPS son una medida de rendimiento común en aplicaciones en tiempo real, se trata de la cantidad de imágenes o fotogramas que puede generar la aplicación por segundo, contra mayor sea este número mayor será la fluidez y la capacidad de respuesta del sistema. Para tomar las medidas se ha usado una escena virtual con 2,5 millones de triángulos, el visor de RV *HTC Vive Pro*, la *CPU Intel i7 8700k*, y la *GPU Nvidia GeForce GTX 1070*; los fotogramas por segundo se han capturado a lo largo de un minuto y se presenta la media redondeada al entero más cercano.

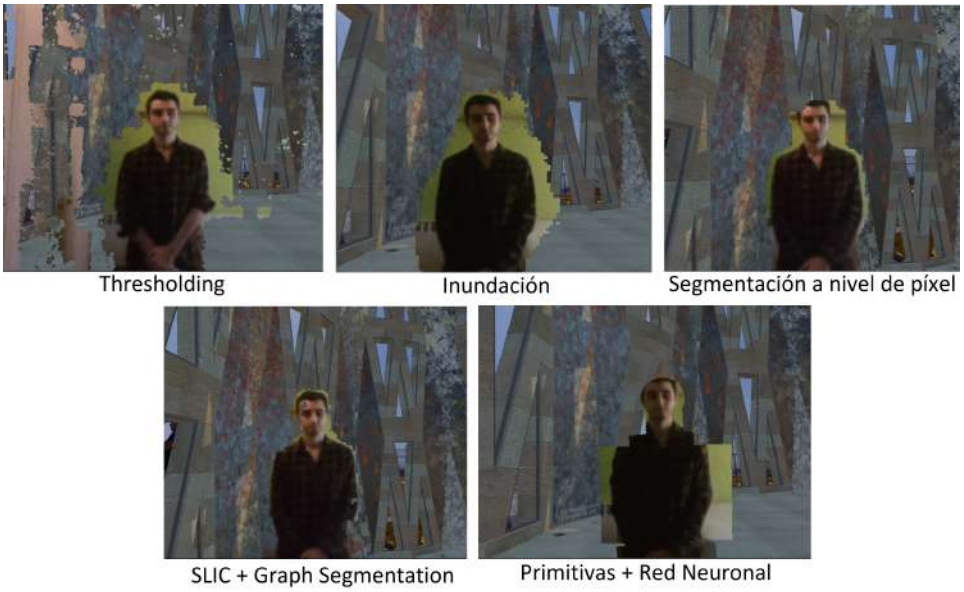

Figura 58: Comparación técnicas segmentación del usuario sin suavizado.

Técnica	Rendimiento aplicación (FPS)
Sin Segmentación	90
Thresholding	47
Inundación	21
Segmentación a nivel de píxel	33
SLIC + Graph Segmentation	17
Primitivas + Red Neuronal	45

Tabla 5: Rendimiento de la aplicación en función de la técnica de segmentación. El rendimiento se mide en fotogramas por segundo.

Segmentación basada en la textura de profundidades

Thresholding e inundación hacen uso de la textura de profundidades para segmentar al usuario. En el rendimiento se puede observar un comportamiento que se repite en las demás técnicas: si el principal algoritmo de la técnica se ejecuta en la *CPU* el rendimiento empeora sustancialmente. *Thresholding* se ejecuta exclusivamente en la tarjeta gráfica mediante *compute shaders*, en cambio, el algoritmo de inundación obtiene peores rendimientos porque se ejecuta en la *CPU*.

Figura 59: Comparación técnicas segmentación del usuario con suavizado. En la primera columna segmentación por inundación en la textura de profundidades. En la segunda columna segmentación con primitivas geométricas.

Visualmente *thresholding* no consigue buenos resultados ya que no tiene en cuenta la conectividad de los píxeles, no obstante, dado el buen rendimiento, podría funcionar en entornos muy abiertos donde el sistema no confundiera al usuario con otros objetos a la misma distancia.

La inundación consigue buenos resultados ya que sí tiene en cuenta la conectividad de los píxeles, a cambio tiene un bajo rendimiento. En general, no se ajusta tan bien al usuario como otras técnicas por la falta de resolución en la textura de profundidades.

Aunque inicialmente se pensaba que sería suficiente con la segmentación de usuarios mediante la textura de profundidades, la baja resolución y calidad de las cámaras del visor de RV no ha permitido obtener texturas de profundidades de suficiente precisión, con *hardware* específico para generar las profundidades se podrían obtener resultados muy superiores. Por este motivo, se decidió estudiar las siguientes técnicas.

Segmentación por color

En la segmentación por color se ha implementado una técnica basada en *region growing* y una técnica basada en *SLIC* y *GrabCut*. El rendimiento de ambas técnicas es bajo, pues hacen un gran uso de la *CPU*, e incluso, en la segunda técnica es necesario recorrer la imagen varias veces.

Otra problema es la iluminación, este método de segmentación varía mucho el resultado en función de la iluminación de la habitación. Si la iluminación es constante y en todas las direcciones se pueden obtener buenos resultados, por lo tanto, en entornos controlados no sería un problema. En general esta premisa no se cumple, la iluminación genera grandes cambios de color a los píxeles y se altera la segmentación.

Por otro lado, tal y como se ve en la Figura 58, si se controla la iluminación pueden segmentar al usuario con gran precisión, especialmente al utilizar *SLIC* se consiguen bordes muy definidos.

Segmentación con primitivas geométricas

Por último, para intentar superar los puntos débiles de las anteriores técnicas se diseñó un método de segmentación a partir de primitivas geométricas. La principal ventaja de esta técnica es la robustez, pues no depende directamente de la iluminación de la escena y no usa textura de profundidades.

La técnica es capaz de conseguir la silueta del usuario de forma uniforme, no obstante, acepta bastantes píxeles del fondo y no consigue ajustarse perfectamente. Por otro lado, no se considera tan grave mostrar un poco de fondo mientras la silueta sea consistente y no se deje partes del usuario sin segmentar.

En cuanto al rendimiento, la rasterización de las primitivas se hace exclusivamente en la *GPU*, mientras que la red neuronal se ejecuta en la *CPU*, se trata de una red pequeña por lo que no penaliza sustancialmente al rendimiento. Por lo tanto, se obtienen buenos rendimientos adecuados para una aplicación en tiempo real.

6 Gestión del proyecto

6.1 Metodología

El desarrollo de este trabajo está ligado con la investigación de las técnicas de navegación y colaboración, por lo tanto, es muy importante utilizar una metodología muy flexible que permita implementar funcionalidades, probarlas, y determinar si se han de descartar, dar por terminadas, o abren una nueva vía de investigación. La metodología ágil de desarrollo de software cumple con los requisitos expuestos, la idea es definir ciclos de desarrollo cortos (alrededor de una semana) donde se diseñe, implemente y pruebe una funcionalidad.

Concretamente, siguiendo con la metodología ágil, se han realizado reuniones semanales con los directores del proyecto, en las que se han comprobado los objetivos semanales, y en función del estado, se han realizado modificaciones en la planificación o se han establecido las nuevas tareas a realizar.

El desarrollo del proyecto se iba a llevar a cabo en el Centro de Realidad Virtual, donde se tiene acceso a los cascos *HTC*. También se dispondría de una sala preparada para el uso de cascos de RV para realizar los estudios y las pruebas necesarias.

6.1.1 Modificaciones

Con la declaración del estado de alarma [35] por la enfermedad COVID-19 (generada por el virus SARS-CoV-2) se ha tenido que modificar parte de la metodología inicialmente planificada.

El primer cambio fue el lugar de trabajo, el Centro de Realidad Virtual tuvo que cerrar y por lo tanto el desarrollo se ha llevado a cabo en la casa del autor de este TFG. Este hecho tiene implicaciones directas en la falta de material y de espacios para desarrollar el trabajo, una parte del trabajo se ha podido realizar al poder transportar un visor de RV junto a sus controladores y sensores. Los cambios en las tareas y objetivos del trabajo por falta de material y espacios se presentan en la sección 6.4

Por otra parte, las reuniones semanales con los directores del trabajo se han podido realizar a través de la aplicación *Skype*, la cual permite la comunicación mediante texto, voz y vídeo entre varias personas.

6.1.2 Herramientas

Para facilitar el seguimiento de los objetivos se ha usado *Trello*, se trata de una herramienta *online* que permite establecer tareas en forma de tarjetas, listas y tableros virtuales. El objetivo es usar un tablero para el proyecto donde una lista representa el grupo de tareas a realizar en una semana, y cada tarjeta es una tarea a realizar.

El control de versiones es una herramienta importante en cualquier proyecto informático, permite hacer un seguimiento de los cambios del proyecto y sirve como copia de seguridad. Para ello se ha usado *GitLab* ya que el grupo de investigación ViRVIG cuenta con su propio servidor de *GitLab* y se abrió un repositorio para el proyecto. Se han ido subiendo los cambios al repositorio al acabar cada tarea, de esta forma ha sido fácil realizar un seguimiento de los cambios.

Por último, se utilizó la herramienta *Gantter* para la planificación de las tareas. A través del diagrama de Gantt, se realizó un control del tiempo dedicado a cada tarea para asegurar la finalización del proyecto en el plazo establecido. En las reuniones semanales, se validó el tiempo emprado en cada tarea y se actualizó el diagrama en consecuencia.

6.2 Planificación temporal

En esta sección se presenta la planificación temporal final dividida por tareas, para ver los cambios entre la planificación final y la inicial ver la sección 6.4.

El trabajo empezó el día 3 de febrero de 2020 y su finalización está prevista para el día 29 de junio de 2020. En total, el desarrollo del proyecto se ha llevado a cabo a lo largo de 137 días aproximadamente y con una duración estimada de 545 horas.

La dedicación diaria ha sido de 4 horas aproximadamente, desde el 3 de febrero y hasta el 13 de marzo de lunes a viernes se realizaron tareas de desarrollo en el Centro de Realidad Virtual. Los fines de semana se dedicaron a la documentación o tareas en las que no sea necesario el uso de equipo de RV, por lo tanto, se pudo trabajar desde casa. A partir del 13 de marzo todas las tareas se realizan desde casa para cumplir con el estado de alarma tal y como se indica en la sección 6.1.1.

6.2.1 Descripción de las tareas

A continuación, se detallan las tareas realizadas de forma individual, pero se agrupan por bloques para distinguir con mayor facilidad las distintas fases del proyecto. En la Tabla 6 se listan todas las tareas con la duración, dependencias y recursos necesarios, en la Figura 60 se muestra la planificación final del proyecto mediante un diagrama de Gantt.

GP - Gestión del proyecto

La gestión del proyecto es esencial para planificar, definir y documentar el trabajo a realizar, además, engloba las reuniones para la validación y propuesta de objetivos semanales. El grupo de gestión ha tenido una duración de 145 horas.

GP.1 - Alcance

Antes de empezar con el trabajo se acotó el desarrollo, por este motivo, se dedicó tiempo inicial a definir qué se quiere conseguir con el trabajo, qué se va a desarrollar, y qué medios serán necesarios. La duración ha sido de 25 horas. Previamente al alcance, es necesario haber investigado sobre el estado del arte de las técnicas de navegación e interacción, pues es necesario tenerlo en cuenta antes de definir qué técnicas se desarrollarán.

GP.2 - Planificación

Para cumplir con los objetivos propuestos en la definición del alcance del proyecto, se realiza una planificación temporal, así como de recursos y requerimientos asociados a cada tarea. Además, se definen los riesgos y obstáculos, y se plantean tareas alternativas para solventarlos. La duración de esta fase ha sido de 15 horas.

GP.3 - Presupuesto

Se realiza un presupuesto para cuantificar el coste del proyecto, para ello, se preparan partidas por cada tarea teniendo en cuenta los costes de personal y equipo, además, se cuantifican los costes genéricos y partidas de imprevistos. Dado que los dispositivos de RV son costosos, es necesario definir adecuadamente el presupuesto, por ello se han dedicado 10 horas.

GP.4 - Informe de sostenibilidad

Se analiza a partir de un informe el impacto medioambiental, económico y social del proyecto, en concreto, se analiza la planificación, desarrollo, vida útil y riesgos. El tiempo necesario para realizar el informe ha sido de 5 horas.

GP.5 - Reuniones

Se han realizado reuniones frecuentemente para analizar los resultados obtenidos según los objetivos semanales. Se han realizado reuniones semanales de 1 hora con los directores del proyecto. En total, una duración de 20 horas.

GP.6 - Documentación

Una parte importante del TFG es la memoria final, por lo tanto, a lo largo del desarrollo del proyecto se han documentado las distintas fases. La documentación se ha realizado de forma paralela al resto del proyecto con más énfasis en la parte final. La duración ha sido de 60 horas aproximadamente.

GP.7 - Presentación

Por último, una vez finalizada la documentación, se prepara la presentación para el tribunal que evalúa el TFG. Se crea material de soporte para la presentación, así como el guión, y se realizan ensayos. En total la duración estimada es de 10 horas.

TP - Trabajo previo

En este apartado se especifican las tareas realizadas antes del desarrollo del trabajo, es decir, tareas de preparación y estudio previo. Puesto que se trata de una fase de preparación, se puede realizar paralelamente a la mayoría de las tareas de gestión del proyecto. La duración ha sido de 25 horas.

TP.1 - Estudio del estado del arte

Considerando que se busca innovar sobre las técnicas de navegación e interacción en RV colaborativa, se han investigado las técnicas punteras actuales para analizar lo que ya está hecho y para determinar qué se puede aportar con este TFG. Principalmente se han analizado artículos científicos. La duración ha sido de 15 horas.

TP.2 - Preparación del entorno de trabajo

Teniendo en cuenta que la preparación de un entorno de trabajo de RV implica muchos dispositivos y aplicaciones diferentes, se ha establecido una tarea solo para la preparación del entorno. Concretamente, el equipo puesto en marcha consiste en: dos ordenadores de altas prestaciones, un casco de RV *HTC Vive* y uno *HTC Vive Pro* [18], y el motor gráfico *Unity3D* [12]. Puesto que el proyecto parte de una aplicación de RV, el tiempo de puesta en marcha se ha reducido, por lo tanto, la duración ha sido de 10 horas.

DC - Desarrollo aplicación colaborativa

Primero era necesario desarrollar una aplicación que permita la conexión de varios usuarios a una misma escena virtual, de forma que todos los usuarios estén sincronizados y puedan interactuar. Se ha dedicado alrededor de 55 horas. Se definen 3 subtareas:

1. DC.1 - Conexión y sincronización - Primero se ha establecido una plataforma de conexión y envío de mensajes por la red. Duración: 30 horas.

2. DC.2 - Integración RV - A continuación, se integra la tecnología de RV con la plataforma de conexión, de forma que cada usuario tenga información sobre los dispositivos de RV de los demás. Duración 15 horas.

3. DC.3 - Pruebas de validación - Se comprueba el correcto funcionamiento del sistema. Duración 10 horas.

DN - Desarrollo de técnicas de navegación

Una vez desarrollada la aplicación para la conexión de varios usuarios, se han desarrollado las diferentes técnicas de navegación. La duración ha sido de 150 horas. Aunque se han desarrollado diferentes elementos para la navegación, se pueden agrupar en tres grandes grupos:

1. DN.1 - Puntos de vista - Implementación de un sistema que permita predefinir posiciones del mundo virtual, de esta forma, el usuario pueda viajar de uno a otro a través de una interfaz tridimensional.

2. DN.2 - Vuelo libre - Investigación y desarrollo del conjunto de técnica de navegación basadas en el vuelo libre por la escena virtual.

3. DN.3 - Teletransporte[21] - Investigación y desarrollo de técnicas de teletransporte de usuarios en la escena virtual.

[21] Teletransporte es el proceso de mover algo de un lugar a otro instantáneamente.

Cada una de las técnicas consta de las tareas de diseño, desarrollo y pruebas. Las pruebas son para determinar el buen funcionamiento de la técnica. La duración para cada técnica ha sido de 15 horas para el diseño, 25 horas para el desarrollo y 10 horas para las pruebas, para más información consultar la Tabla 6.

DI - Desarrollo técnicas de interacción (representación visual del usuario)

Las técnicas de interacción se pueden desarrollar de forma paralela con las de navegación al tratarse de métodos independientes, sin embargo, en este proyecto como solo hay un programador se realizarán secuencialmente. La duración ha sido de 170 horas. Se ha investigado acerca de la representación visual de usuarios mediante las cámaras del visor *HTC Vive Pro*, los métodos de segmentación usados se pueden dividir en tres grupos:

1. DI.1 - Textura de profundidades - Investigación y desarrollo de la segmentación de usuarios mediante la textura de profundidades obtenida de las dos cámaras del visor de RV.

2. DI.2 - *OpenPose* - Investigación y desarrollo de la segmentación de usuarios a partir de la información obtenida por la librería *OpenPose* y técnicas de segmentación por color.

3. DI.3 - Red neuronal - Investigación y desarrollo de una red neuronal para segmentar al usuario a partir de la información obtenida de *OpenPose*.

De igual forma que en el desarrollo de técnicas de navegación, se crean 3 tareas por cada área: diseño (15 horas), desarrollo (25 horas) y pruebas (10 horas). Además en DI.3 se añade una tarea de entrenamiento (20 horas) para realizar todo el trabajo relacionado con el entrenamiento de la red neuronal.

6.2.2 Recursos humanos

En este proyecto se encuentran cuatro roles diferentes: jefe de proyecto, investigador, programador y *tester*. No obstante, teniendo en cuenta que este TFG se realiza por una persona, ha sido el autor el encargado de asumir los diferentes roles en función de la tarea a realizar. En la Tabla 6 se encuentra la asignación de tareas a cada rol.

1. Jefe de proyecto - Se encarga de la planificación del proyecto, liderar las reuniones con el equipo y escribir la memoria final del proyecto.

2. Investigador - Se trata de la persona responsable de la investigación del proyecto: diseña las técnicas y realiza las pruebas con los usuarios, evalúa los resultados obtenidos y contribuye en la documentación.

3. Programador - Es la persona encargada de implementar el sistema y preparar la instalación del equipo de RV.

4. *Tester* - Se ocupa de realizar las pruebas de validez del sistema, debe diseñar las pruebas, ejecutarlas y presentar un informe para poder arreglar los errores encontrados.

6.2.3 Recursos materiales

A continuación, se exponen los recursos materiales necesarios para la realización del proyecto. En la Tabla 6 se encuentra la asignación de recursos materiales a cada tarea.

1. Portátil - Ordenador portátil para realizar las tareas de gestión del proyecto y de investigación o evaluación de los estudios.

2. Dos ordenadores de altas prestaciones - Ordenadores de altas prestaciones, en especial, con tarjeta gráfica dedicada[22].

3. *HTC Vive* o *HTC Vive Pro* - En función de la disponibilidad, se necesitan dos cascos de RV inmersivos.

6.2.4 Gestión del riesgo

Toda planificación puede sufrir desviaciones, por este motivo se preparó un plan para los riesgos y obstáculos que pudieran surgir durante el desarrollo. A continuación, se describen los posibles obstáculos y los planes alternativos para solventarlos, en el apartado 6.3.4 se especifican los tiempos y recursos necesarios.

1. Dificultades imprevistas - Teniendo en cuenta el nivel de madurez de las tecnologías usadas en el proyecto, era posible no poder cumplir con los plazos especificados anteriormente, por este motivo se reservó alrededor de una semana al final del proyecto para poder alargar el desarrollo. Si aun así no fuera suficiente, ya que el proyecto se basa en la composición de diferentes técnicas, se podría haber eliminar el desarrollo de una de las técnicas.

[22]Componente *hardware* encargado del renderizado de escenas 3D con memoria especializada.

Planificación Final					
Id.	**Tarea**	**Tiempo**	**Dependencia**	**Recursos**	**Roles**
GP	**Gestión del proyecto**	**145h**	**-**	**-**	**-**
GP.1	Alcance	25h	TP.1	Portátil	JP
GP.2	Planificación	15h	GP.1	Portátil	JP
GP.3	Presupuesto	10h	GP.1	Portátil	JP
GP.4	Informe de sostenibilidad	5h	GP.1	Portátil	JP
GP.5	Reuniones	20h	-	Portátil	JP, I, P, T
GP.6	Documentación	60h	-	Portátil	JP, I
GP.7	Presentación	10h	GP.6	Portátil	JP
TP	**Trabajo previo**	**25h**	**-**	**-**	**-**
TP.1	Estudio del estado del arte	15h	-	Portátil	I
TP.2	Preparación del entorno de trabajo	10h	GP.2	2xPC, 2xCascosRV	P
DC	**Desarrollo aplicación colaborativa**	**55h**	**-**	**-**	**-**
DC.1	Conexión y sincronización	30h	TP.2	2xPC, 2xCascosRV	P
DC.2	Integración RV	15h	DC.1	2xPC, 2xCascosRV	P
DC.3	Pruebas de validación	10h	DC.2	2xPC, 2xCascosRV	T
DN	**Desarrollo de técnicas de navegación**	**150h**	**-**	**-**	**-**
DN.1	Puntos de vista	50h	-	-	-
DN.1.1	Diseño	15h	DC.3	Portátil	I
DN.1.2	Desarrollo	25h	DN.1.1	2xPC, 2xCascosRV	P
DN.1.3	Pruebas	10h	DN.1.2	2xPC, 2xCascosRV	I, T
DN.2	Vuelo libre	50h	-	-	-
DN.2.1	Diseño	15h	DC.3	Portátil	I
DN.2.2	Desarrollo	25h	DN.2.1	2xPC, 2xCascosRV	P
DN.2.3	Pruebas	10h	DN.2.2	2xPC, 2xCascosRV	I, T
DN.3	Teletransporte	50h	-	-	-
DN.3.1	Diseño	15h	DC.3	Portátil	I
DN.3.2	Desarrollo	25h	DN.3.1	2xPC, 2xCascosRV	P
DN.3.3	Pruebas	10h	DN.3.2	2xPC, 2xCascosRV	I, T
DI	**Desarrollo de técnicas de interacción**	**170h**	**-**	**-**	**-**
DI.1	Textura de profundidades	50h	-	-	-
DI.1.1	Diseño	15h	DC.3	Portátil	I
DI.1.2	Desarrollo	25h	DI.1.1	2xPC, 2xCascosRV	P
DI.1.3	Pruebas	10h	DI.1.2	2xPC, 2xCascosRV	I, T
DI.2	*OpenPose*	50h	-	-	-
DI.2.1	Diseño	15h	DC.3	Portátil	I
DI.2.2	Desarrollo	25h	DI.2.1	2xPC, 2xCascosRV	P
DI.2.3	Pruebas	10h	DI.2.2	2xPC, 2xCascosRV	I, T
DI.3	Red neuronal	70h	-	-	-
DI.3.1	Diseño	15h	DC.3	Portátil	I
DI.3.2	Desarrollo	25h	DI.3.1	2xPC, 2xCascosRV	P
DI.3.3	Entrenamiento	20h	DI.3.2	PC	I
DI.3.4	Pruebas	10h	DI.3.3	2xPC, 2xCascosRV	I, T
-	**Total**	**545h**	**-**	**-**	**-**

Tabla 6: Tabla de tareas (planificación final) con la duración, dependencias y recursos necesarios. Roles: JP - jefe de proyecto, I - investigador, P - programador, T - tester.

Planificación final

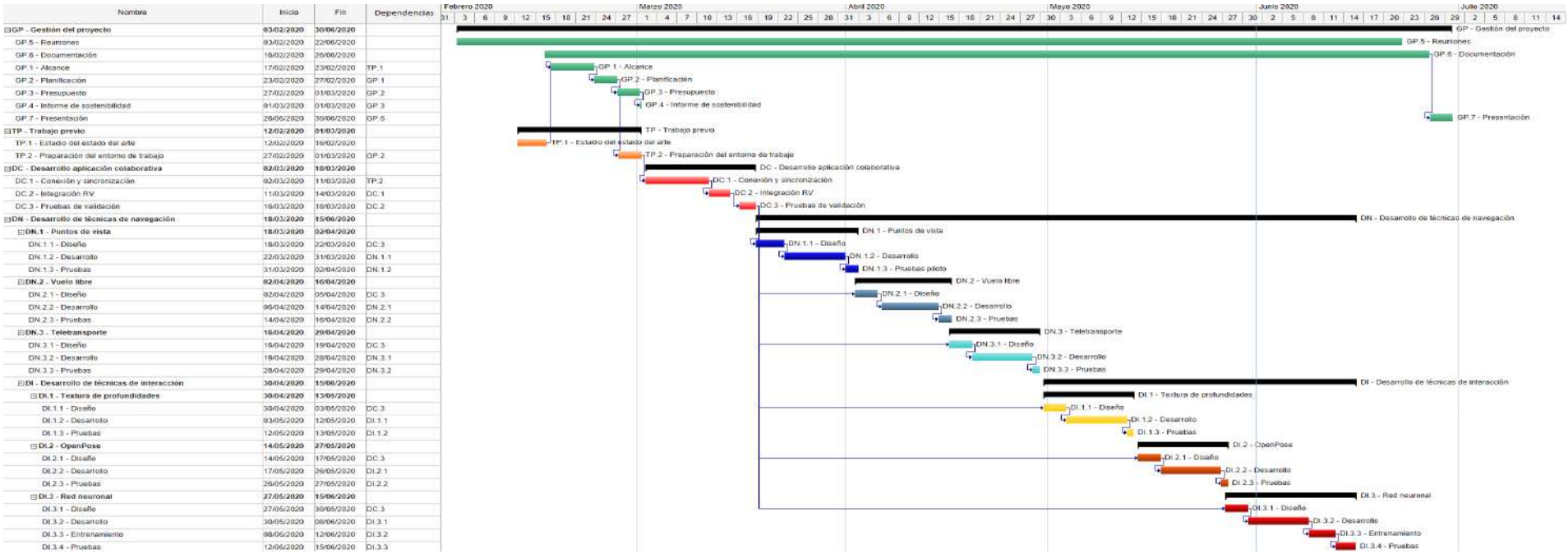

Figura 60: Diagrama de Gantt *de la planificación final. Creado con la herramienta* Gantter.

2. Recursos limitados - El Centro de Realidad Virtual, donde se realiza el proyecto, dispone de un conjunto reducido de cascos de RV. No obstante, teniendo en cuenta que el desarrollo de las diferentes técnicas se puede realizar en paralelo, en caso de no disponer de cascos se podría haber empezado otra tarea de diseño o estudio en la que los dispositivos de RV no sean necesarios, de esta forma no sería necesario modificar la planificación temporal.

3. Errores en las librerías de RV - A pesar de que *Unity3D* es usado por miles de usuarios, las librerías de RV necesarias para el funcionamiento de los dispositivos tienen algunos errores y no han sido suficientemente probadas. Si apareciera un error que bloqueara el desarrollo del proyecto, se tendría que añadir una tarea a la planificación para desarrollar una solución alternativa a la proporcionada por la librería. Se estima que podría añadir entre 20 y 40 horas de desarrollo.

4. Rendimiento - En las tareas de desarrollo ya se ha tenido en cuenta el tiempo necesario para la optimización de los métodos, no obstante, dados los requerimientos de la visualización en RV, si la aplicación no funcionara bien, se simplificaría la escena virtual para reducir la carga de trabajo de la *CPU* y de la tarjeta gráfica; el impacto sobre la duración del proyecto sería mínimo.

6.3 Gestión económica

En esta sección se calculan los costes necesarios para el desarrollo de este proyecto. Se identifican diferentes tipos de costes asociados al personal, espacio de trabajo, y a las herramientas y dispositivos usados. Además, para superar los obstáculos que aparezcan y asumir los costes no programados, se ha realizado un plan de contingencia, una partida de imprevistos y se exponen mecanismos para controlar el presupuesto.

6.3.1 Costes de personal

A partir de la planificación por tareas se calcula el coste de personal, se tienen en cuenta los 4 roles definidos anteriormente: jefe de proyecto, investigador, programador y *tester*. En la Tabla 7 se ve el coste por hora de cada puesto, los datos han sido obtenidos de la empresa de reclutamiento *Hays* [36].

En la Tabla 8 se detallan las partidas por tarea a partir de los costes de personal de la Tabla 7, y se estima el coste de la seguridad social multiplicando el coste por 1,3. En total el coste de personal del proyecto es de 19.149€.

Rol	Coste por hora
Jefe de proyecto	30€/h
Investigador	20€/h
Programador	16€/h
Tester	16€/h

Tabla 7: Costes de personal a partir de la guía de mercado laboral de Hays.

6.3.2 Costes genéricos

En la planificación temporal se especifica que se trabajó de lunes a viernes en el Centro de Realidad Virtual, y los fines de semana desde casa hasta el 13 de marzo, a partir de entonces se trabajó desde casa. Por lo tanto, como ambos espacios son compartidos y están situados en Barcelona, se estima el coste en función de la tarifa de un espacio de *coworking*[23] en Barcelona, con mesa individual y acceso todos los días de la semana. El coste es de 300 euros al mes [37], incluye los gastos de internet, agua, electricidad. Teniendo en cuenta que el proyecto se desarrolla a lo largo de 5 meses, el coste total del espacio será de 1.500 euros.

A continuación, se calculan los costes de los dispositivos *Hardware*. Para calcular las amortizaciones, se ha calculado el coste por hora teniendo en cuenta que un año tiene 220 días hábiles y 8 horas laborables al día, el coste por hora por tanto es $Coste\_Dispositivo/(Vida\_Util * 220 * 8)$. Se estima una vida útil de 4 años a los dispositivos, no obstante, para los cascos de RV se calculan 2 años de vida útil, pues rápidamente se vuelven obsoletos por su constante evolución. En la Tabla 9 se detallan las amortizaciones, las horas de uso de cada dispositivo se presentan en la planificación temporal.

6.3.3 Contingencia

Como en todo proyecto, es importante añadir un sobrecoste para cubrir obstáculos e imprevistos. En este caso, al tratarse de un trabajo de investigación con tecnologías innovadoras, la probabilidad de encontrar problemas durante el desarrollo es considerable, por lo tanto se decidió fijar un 15 % de sobrecoste. En la Tabla 10 se detalla la contingencia total del proyecto.

[23]Espacios compartidos donde profesionales independientes desarrollan sus proyectos.

Id.	Tarea	Tiempo	Roles	Coste	Coste SS
GP	**Gestión del proyecto**	**145h**	**-**	**6.590€**	**8.567€**
GP.1	Alcance	25h	JP	750€	975€
GP.2	Planificación	15h	JP	450€	585€
GP.3	Presupuesto	10h	JP	300€	390€
GP.4	Informe de sostenibilidad	5h	JP	150€	195€
GP.5	Reuniones	20h	JP, I, P, T	1.640€	2.132€
GP.6	Documentación	60h	JP, I	3.000€	3.900€
GP.7	Presentación	10h	JP	300€	390€
TP	**Trabajo previo**	**25h**	**-**	**500€**	**650€**
TP.1	Estudio del estado del arte	15h	I	300€	390€
TP.2	Preparación del entorno de trabajo	10h	P	200€	260€
DC	**Desarrollo aplicación colaborativa**	**55h**	**-**	**880€**	**1.144€**
DC.1	Conexión y sincronización	30h	P	480€	624€
DC.2	Integración RV	15h	P	240€	312€
DC.3	Pruebas de validación	10h	T	160€	208€
DN	**Desarrollo de técnicas de navegación**	**150h**	**-**	**3.180€**	**4.134€**
DN.1	Puntos de vista	50h	-	-	-
DN.1.1	Diseño	15h	I	300€	390€
DN.1.2	Desarrollo	25h	P	400€	520€
DN.1.3	Pruebas	10h	I, T	360€	468€
DN.2	Vuelo libre	50h	-	-	-
DN.2.1	Diseño	15h	I	300€	390€
DN.2.2	Desarrollo	25h	P	400€	520€
DN.2.3	Pruebas	10h	I, T	360€	468€
DN.3	Teletransporte	50h	-	-	-
DN.3.1	Diseño	15h	I	300€	390€
DN.3.2	Desarrollo	25h	P	400€	520€
DN.3.3	Pruebas	10h	I, T	360€	468€
DI	**Desarrollo de técnicas de interacción**	**170h**	**-**	**3.580€**	**4.654€**
DI.1	Textura de profundidades	50h	-	-	-
DI.1.1	Diseño	15h	I	300€	390€
DI.1.2	Desarrollo	25h	P	400€	520€
DI.1.3	Pruebas	10h	I, T	360€	468€
DI.2	*OpenPose*	50h	-	-	-
DI.2.1	Diseño	15h	I	300€	390€
DI.2.2	Desarrollo	25h	P	400€	520€
DI.2.3	Pruebas	10h	I, T	360€	468€
DI.3	Red neuronal	70h	-	-	-
DI.3.1	Diseño	15h	I	300€	390€
DI.3.2	Desarrollo	25h	P	400€	520€
DI.3.3	Entrenamiento	20h	I	400€	520€
DI.3.4	Pruebas	10h	I, T	360€	468€
-	**Total**	**545h**	**-**	**14.730€**	**19.149€**

Tabla 8: Tabla de partidas por tarea. Coste SS es el coste teniendo en cuenta la seguridad social. Roles: JP - jefe de proyecto, I - investigador, P - programador, T - tester.

Hardware	Precio	Unidades	Vida útil	Horas	Amortización
Portátil	800€	1	4 años	255h	29€
Ordenador	2.000€	2	4 años	290h	165€
HTC Vive [18]	800€	2	2 años	290h	132€
Total	**-**	**-**	**-**	**-**	**326€**

Tabla 9: Costes de los recursos hardware.

Tipo	Coste	Contingencia
Espacio	1.500€	225€
Hardware	326€	49€
Personal	19.149€	2.872€
Total	**21.880€**	**3.146€**

Tabla 10: Tabla contingencia del 15 % por tipo de gasto.

6.3.4 Imprevistos

Por último, se realizó una partida de imprevistos para los costes de los obstáculos que puedan surgir durante el desarrollo del proyecto. Los imprevistos se presentan en la planificación temporal, a continuación, sólo se cuantifica el riesgo y el coste que pueden causar, en la Tabla 11 se detalla el coste.

1. Aumento tiempo de desarrollo - En caso de necesitar más tiempo de desarrollo, se añadirían 25 horas de desarrollo a la planificación y 10 horas de *testing*. El coste total sería de 25 horas de programador y 10 horas de *tester*, por lo tanto, 560 euros. El riesgo es elevado debido al uso de nuevas tecnologías, por lo que se estima un 20 % de probabilidades.

2. Fallo dispositivo - En caso de que algún dispositivo falle sería necesario comprar uno nuevo. Los costes son los expuestos en la Tabla 9, se estima un riesgo del 5 % por cada ordenador o portátil, y un 10 % por cada casco de RV ya que son más frágiles y están en constante movimiento.

3. Error librerías RV - Tal y como se comentaba en la planificación temporal, en caso de que una librería tuviera un error, se ha de implementar la función equivalente, por lo tanto se añadirían 30 horas de trabajo de programador y 10 horas de *tester*, en total 640 euros. Se estima un riesgo del 5 % debido a que son librerías usadas por muchos usuarios.

Imprevisto	Coste	Riesgo	Coste total
Aumento tiempo desarrollo	560€	20 %	112€
Portátil	800€	5 %	40€
Ordenador 1	2000€	5 %	100€
Ordenador 2	2000€	5 %	100€
HTC Vive 1	800€	10 %	80€
HTC Vive 2	800€	10 %	80€
Error librerías RV	640€	5 %	32€
Total	**7.600€**	**-**	**544€**

Tabla 11: Tabla del sobrecoste añadido por imprevistos.

6.3.5 Coste total

Una vez presentados todos los costes del proyecto, en la Tabla 12 se presenta el presupuesto final del trabajo. El coste total del proyecto es de 24.665 euros.

Tipo	Coste
Espacio	1.500€
Hardware	326€
Personal	19.149€
Contingencia	3.146€
Imprevistos	544€
Coste total	**24.665€**

Tabla 12: Tabla del presupuesto final del proyecto.

6.3.6 Control de gestión

Una vez definido el presupuesto inicial, se definen los mecanismos de control necesarios para evitar desviaciones, así como indicadores numéricos que ayuden al control. En las reuniones semanales, cada vez que se acababa una tarea, se actualizaba el presupuesto con las horas reales y se comparaba con las horas estimadas.

Para controlar los imprevistos, al finalizar una tarea también se apuntaban los gastos extra que se hubieran producido, y se comparaban con la previsión de imprevistos y contingencia. De esta forma, rápidamente se podía detectar cualquier desviación y predecir si era necesario recortar alguna tarea o aumentar el presupuesto.

A continuación, se presentan los descriptores numéricos para el control,:

1. Desviación coste personal por tarea:
   $(coste\_estimado - coste\_real) * horas\_reales$
2. Desviación realización tareas:
   $(horas\_estimadas - horas\_reales) * coste\_real$
3. Desviación total en la realización de tareas:
   $coste\_estimado\_total - coste\_real\_total$
4. Desviación total de recursos (*software*, *hardware*, espacio o personal):
   $coste\_estimado\_total - coste\_real\_total$
5. Desviación total coste de imprevistos:
   $coste\_estimado\_imprevistos - coste\_real\_imprevistos$
6. Desviación total de horas:
   $horas\_estimadas - horas\_reales$

6.4 Cambios en la planificación

En la sección 6.1.1 se comenta que ha sido necesario hacer cambios en el proyecto debido al estado de alarma: el 14 de marzo se anunciaba la declaración del estado de alarma en todo el estado español [35], desde entonces ha sido imposible realizar el proyecto tal y como se había planificado inicialmente. A continuación, se enumeran y se explican los cambios realizados sobre el trabajo, en la Tabla 13 se puede ver el listado inicial de tareas, en azul las tareas que han sido modificadas o eliminadas, en la Figura 61 se puede ver el diagrama de Gantt de la planificación inicial.

6.4.1 Cambios en las tareas

Se han tenido que modificar muchos aspectos del trabajo, pues este TFG se basa en la creación de una aplicación para realidad virtual colaborativa, la imposibilidad de llevar a cabo pruebas con múltiples usuarios conlleva muchas restricciones. Dado el avanzado estado de desarrollo de las técnicas de navegación (tareas DN) se consideró seguir con el desarrollo. La implementación de la plataforma de *networking* (tareas DC) y gran parte del trabajo base de las técnicas de navegación se pudieron hacer en el Centro de Realidad Virtual antes del 14 de marzo. A partir del 14 de marzo sólo se dispuso de un visor de RV y un ordenador para hacer las pruebas, para poder progresar con el desarrollo se usó un simulador del visor de RV para hacer las pruebas con múltiples participantes.

Planificación Inicial		
Id.	**Tarea**	**Tiempo**
GP	**Gestión del proyecto**	**145h**
GP.1	Alcance	25h
GP.2	Planificación	15h
GP.3	Presupuesto	10h
GP.4	Informe de sostenibilidad	5h
GP.5	Reuniones	20h
GP.6	Documentación	60h
GP.7	Presentación	10h
TP	**Trabajo previo**	**25h**
TP.1	Estudio del estado del arte	15h
TP.2	Preparación del entorno de trabajo	10h
DC	**Desarrollo aplicación colaborativa**	**55h**
DC.1	Conexión y sincronización	30h
DC.2	Integración RV	15h
DC.3	Pruebas de validación	10h
DN	**Desarrollo de técnicas de navegación**	**150h**
DN.1	Puntos de vista	50h
DN.1.1	Diseño	15h
DN.1.2	Desarrollo	25h
DN.1.3'	Pruebas piloto	10h
DN.2	Vuelo libre	50h
DN.2.1	Diseño	15h
DN.2.2	Desarrollo	25h
DN.2.3'	Pruebas piloto	10h
DN.3	Teletransporte	50h
DN.3.1	Diseño	15h
DN.3.2	Desarrollo	25h
DN.3.3'	Pruebas piloto	10h
DI	**Desarrollo de técnicas de interacción**	**100h**
DI.1'	*Avatares*	50h
DI.1.1'	Diseño	15h
DI.1.2'	Desarrollo	25h
DI.1.3'	Pruebas piloto	10h
DI.2'	Evasión colisión	50h
DI.2.1'	Diseño	15h
DI.2.'2	Desarrollo	25h
DI.2.3'	Pruebas piloto	10h
EU'	**Estudio con usuarios**	**70h**
EU.1'	Estudio navegación	35h
EU.1.1'	Preparación	5h
EU.1.2'	Estudio	20h
EU.1.3'	Evaluación	10h
EU.2'	Estudio interacción y colisiones	35h
EU.2.1'	Preparación	5h
EU.2.2'	Estudio	20h
EU.2.3'	Evaluación	10h
-	**Total**	**545h**

Tabla 13: Tabla de tareas (planificación inicial) con la duración. En azul se indican las tareas que han sido modificadas.

Planificación inicial

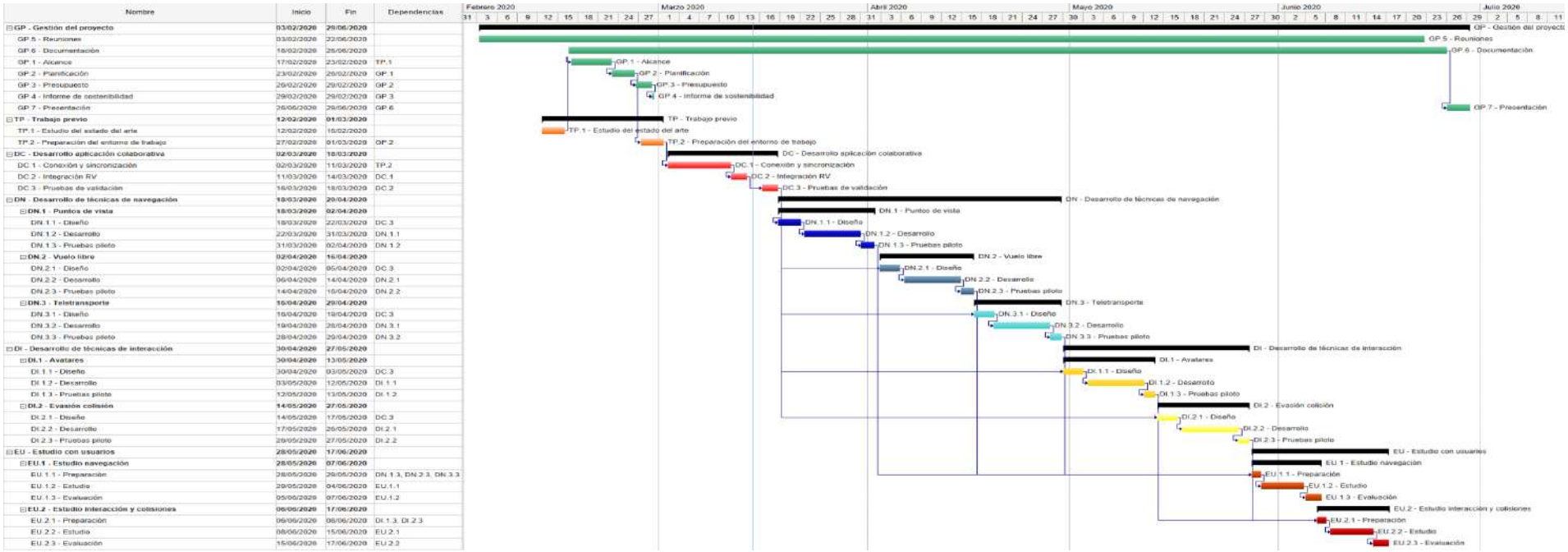

Figura 61: Diagrama de Gantt *de la planificación inicial. Creado con la herramienta* Gantter.

A raíz de esta situación, las primeras tareas modificadas fueron DN.1.3', DN.2.3' y DN.3.3'; las tres tareas hacen referencia a pruebas piloto que se iban a realizar con el personal de Centro de Realidad Virtual. En cambio, se han añadido las tres tareas DN.1.3, DN.2.3 y DN.3.3 con la misma duración, estas tres tareas son pruebas que realizará el autor del trabajo e intentarán simular las pruebas piloto planificadas. Ver la Tabla 6 para el listado de las nuevas tareas.

Seguidamente, se cancelaron todos los estudios con usuarios que se querían realizar para estudiar el comportamiento de los usuarios con las distintas técnicas de navegación e interacción (grupo de tareas EU'), la imposibilidad de reunión y el cierre del Centro de Realidad Virtual no han permitido realizar los estudios. No obstante, antes de la declaración del estado de alarma se pudieron hacer algunas pruebas piloto con los arquitectos de la Sagrada Familia, esto permitió sacar unas primeras conclusiones de las técnicas implementadas.

Teniendo en cuenta que sólo se disponía de un visor de RV (*HTC Vive Pro* con dos cámaras incorporadas), y que se tenía que dedicar el tiempo de los estudios en otras tareas, se decidió reorganizar y ampliar las tareas de desarrollo de técnicas de interacción (grupo de tareas DI). Estaba previsto investigar con diferentes *avatares* y la evasión de colisión entre usuarios, por lo tanto, se optó por investigar la representación visual de usuarios a través de las cámaras del visor de RV, de esta forma se estaba investigando con *avatares*, interacción y mejorando la evasión de colisión al mejorar la comunicación no verbal entre usuarios.

Por lo tanto, se desglosa la tarea de técnicas de interacción en función de la técnica usada para segmentar al usuario, se crean tres grupos de tareas (DI.1, DI.2, DI.3) haciendo referencia a las texturas de profundidades, *OpenPose* y el desarrollo de una red neuronal.

En resumen, el desarrollo de las técnicas de navegación se ha visto poco afectado, algunas pruebas piloto se han podido realizar, los estudios han sido cancelados, las técnicas de interacción se han reorganizado y ampliado para investigar acerca de la representación visual de los usuarios con las cámaras del visor de RV. En general, el tiempo total del trabajo no se ha visto afectado, 545h tanto en la planificación inicial como en la final.

6.4.2 Cambios en el presupuesto

El principal cambio en el presupuesto ha sido en el coste de personal debido a la reorganización de las tareas. En la Tabla 8 se pueden ver las partidas finales para las tareas definidas en la Tabla 6, en total se establece un gasto de 19.149 euros en personal, el gasto planificado inicialmente en personal era de 19.279 euros. Por lo tanto, al haber disminuido el coste, no ha habido problemas de presupuesto ni se han tenido que hacer grandes cambios.

6.4.3 Cambios en los objetivos

Inicialmente se definía un objetivo y seis subobjetivos, tanto el objetivo principal como los cuatro primeros subobjetivos no se han modificado: desarrollar una aplicación que permita la conexión de varios usuarios, investigar y desarrollar técnicas de navegación y de interacción, e investigar con la representación visual de los usuarios. Por los motivos expuestos anteriormente, se han suprimido los dos últimos subobjetivos que hacían referencia al estudio con usuarios de las técnicas de navegación e interacción.

6.5 Leyes y regulaciones

La ley orgánica 3/2018, del 5 de diciembre de 2018, sobre la protección de datos personales y garantía de los derechos digitales [38] no aplica a este proyecto: para las técnicas de navegación no se usa información de los usuarios, para la representación visual de los usuarios, de momento, el proyecto se limita a obtener imágenes y en el mismo ordenador realizar una segmentación del usuario, a continuación, se muestra la segmentación para utilizarla como representación virtual, en ningún momento se almacena la imagen ni se transfiere. En posibles ampliaciones de este trabajo, si fuera necesario enviar las imágenes a través de la red, se debería revisar el correcto tratamiento de los datos.

Por otro lado, el proyecto utiliza multitud de librerías, seguidamente se revisan las licencias para garantizar su cumplimiento.

Unity3D

Para el desarrollo del proyecto se utiliza el motor gráfico *Unity3D*, como los fines son de investigación se utiliza la licencia de educación de *Unity* [39]. Para poder obtener una licencia de educación es suficiente con ser estudiante de una universidad acreditada por el gobierno.

SRWorks

La textura de profundidades, en el apartado de representación visual del usuario, se obtiene a través de la librería *SRWorks*, la licencia completa se puede ver en [40]. La licencia permite el uso de la librería para desarrollar aplicaciones, e incluso modificar el código de ejemplo que proveen, no obstante, no se puede modificar el código fuente de la librería. El código fuente de la librería no se modifica, solo se utiliza para adquirir la textura de profundidades.

OpenPose

Para estimar la posición de los usuarios se utiliza la librería de código abierto *OpenPose*, la licencia [28] permite el uso de la librería para propósitos académicos o sin fines comerciales, por lo tanto, para investigación se podría usar, si la parte de la aplicación que usa *OpenPose* se quisiera comercializar, se debería buscar otra alternativa.

OpenCV

Para crear el conjunto de entrenamiento de la red neuronal para segmentar usuarios, se utiliza la librería de código abierto *OpenCV* con licencia *BSD License* [41]. Dicha licencia permite el uso de la librería tanto para usos académicos como comerciales, algunos algoritmos están patentados (*SIFT* y *SURF* por ejemplo) y no se rigen por la misma licencia, sin embargo, no se usan en este proyecto.

Tensorflow

La red neuronal para segmentar usuarios se ha desarrollado con *Tensorflow*, se trata de una librería de código abierto que se rige por la licencia *Apache License 2.0* [42], permite tanto el uso académico como el uso comercial.

Blender

Por último, para crear el conjunto de entrenamiento también se utiliza el *software* gratuito *Blender*. La licencia utilizada es *GNU General Public License* (GPL) [43], permite usar y modificar blender para cualquier propósito.

6.6 Informe de sostenibilidad

En esta sección se presenta el informe de sostenibilidad del proyecto, a partir de la matriz de sostenibilidad (Tabla 14) se analiza el impacto de este TFG en las tres dimensiones: ambiental, económico y social. Además, cada dimensión se analiza desde tres puntos de vista diferentes:

1. Proyecto puesto en producción (PPP) - se analiza la planificación, desarrollo e implantación del proyecto.
2. Vida útil - comporta la sostenibilidad del proyecto desde su implantación hasta su desmantelamiento.
3. Riesgos - se analizan los riesgos que pueden comprometer la sostenibilidad del proyecto en cualquiera de sus etapas.

	PPP	**Vida útil**	**Riesgos**
Ambiental	Consumo de diseño	Huella ecológica	Ambientales
Económico	Factura	Plan de viabilidad	Económicos
Social	Impacto personal	Impacto social	Sociales

Tabla 14: Matriz de sostenibilidad. Fuente: [44]

6.6.1 Dimensión ambiental

En la dimensión ambiental se estiman los recursos que consume el proyecto y el impacto que tienen sobre el medio ambiente, además, se analiza si el proyecto conlleva mejoras respecto a soluciones actuales del problema que se quiere resolver.

PPP

Primero se cuantifica el impacto ambiental de la realización del proyecto, en la Tabla 15 se especifica el consumo del *hardware* usado para implementar el trabajo.

Para realizar una estimación del consumo energético del trabajo se usa la Tabla 6 para determinar que dispositivos usa cada tarea, a partir del número de horas de cada tarea y el equipo necesario se puede estimar el consumo energético por tareas tal y como se muestra en la Tabla 16.

El consumo total de la realización del trabajo es de $173,4kWh$, se considera un consumo aceptable para realizar un trabajo de estas características. Para intentar reducir el impacto se ha minimizado el uso de los ordenadores más potentes, pues se ha usado un portátil de bajo consumo para tareas de documentación o gestión.

Dispositivo	Consumo
Portátil	60W
PC	273W
Casco RV	5W

Tabla 15: Consumo dispositivos. Para el PC se ha tenido en cuenta el consumo de los principales componentes del ordenador, CPU Intel i7-8700k (95W), GPU Nvidia 1700 (150W), monitor LG 24 pulgadas (28W).

Tarea	Energía
GP	8,7kWh
TP	6,5kWh
DC	30,6kWh
DN	61,1kWh
DI	66,5kWh
Total	**173,4kWh**

Tabla 16: Consumo dispositivos por grupos de tareas, ver Tabla 6 para el listado de las tareas.

Vida útil

Los recursos necesarios durante la vida útil del proyecto son los mismos que durante el desarrollo, es decir, dos ordenadores y dos visores de realidad virtual (o más dependiendo del número de usuarios), por lo tanto, parece un consumo aceptable teniendo en cuenta que la mayoría de trabajos necesitan ordenadores funcionando.

Uno de los puntos fuertes de este trabajo es la baja emisión en comparación con las alternativas actuales, a continuación, se presentan tres casos donde la huella ecológica se mejora considerablemente.

1. Sistemas semiinmersivos - Actualmente se utilizan sistemas semiinmersivos como el *CAVE* [15] para visualizar escenas virtuales de forma colaborativa. Por ejemplo, el *CAVE* disponible en el Centro de Realidad Virtual opera con 12 ordenadores y 40 proyectores; por otro lado en este trabajo se usa un ordenador y un visor de RV por cada participante, desde el punto de vista medio ambiental se reduce considerablemente el consumo.

2. Maquetas y prototipos - En proyectos de ingeniería y arquitectura es común crear maquetas y prototipos para visualizar los diseños, gracias a las posibilidades de interacción y visión tridimensional de los visores de RV, se minimiza considerablemente la necesidad de crear prototipos reales.

3. Vuelos - Aunque este trabajo es una aplicación colaborativa en un espacio físico compartido, fácilmente es ampliable a espacios físicos no compartidos, en especial la parte de navegación. Esta ampliación permitiría a equipos de diferentes partes del mundo tener reuniones y discutir los diseños sin necesidad de coger ningún transporte.

Riesgos

Principalmente existen dos riesgos para cualquier usuario que quiera usar el sistema. El primer riesgo está relacionado con los visores de RV, aunque se trata de dispositivos de bajo consumo durante su uso, su proceso de fabricación requiere mucha energía como en cualquier componente electrónico, al tratarse de dispositivos muy nuevos y poco probados son relativamente frágiles, por lo tanto la rotura prematura de un visor de RV podría impactar negativamente al medio ambiente.

El otro riesgo es el *cybersickness* de los usuarios, aunque sería recomendable probar la tecnología antes de adquirir el equipo, es posible que con el tiempo los usuarios no se acostumbren al cambio y dejen de usar los visores de RV, por lo que se habría fabricado innecesariamente equipo de RV que no se va a usar.

6.6.2 Dimensión económica

La dimensión económica analiza el coste y la viabilidad del proyecto, se estima a partir del presupuesto y de los costes de mantenimiento a lo largo de la vida útil del proyecto.

PPP

En relación con la dimensión económica se ha realizado un presupuesto del TFG en la sección 6.3. En total el coste del proyecto se cuantifica en 24.665 euros, la mayor parte del presupuesto está destinado al personal. Los costes de *software* y amortización del *hardware* son considerablemente bajos, pues no es necesario un gran desembolso para adquirir equipo de RV inmersiva.

Para reducir el presupuesto se ha usado *software* de código libre siempre que ha sido posible. De cara al ajuste final del presupuesto, la desviación ha sido mínima, en la sección 6.4.2 se explican los cambios del presupuesto respecto la planificación inicial, al tener que modificar las tareas se han modificado los costes de personal de 19.279 euros (inicial) a 19.149 euros (final).

Vida útil

Una vez comprado el equipo necesario, el coste durante la vida útil del proyecto es mínimo. Al igual que en la dimensión ambiental, este proyecto permitirá reducir costes respecto las soluciones actuales. Por un lado los sistemas semiinmersivos son mucho más costosos, tanto el equipo, como el hecho de necesitar un espacio relativamente grande reservado solo para el sistema semiinmersivo, en contraposición, el sistema inmersivo se puede montar y desmontar con relativa facilidad: no es necesario un espacio dedicado.

Riesgos

El principal riesgo económico del proyecto es la ruptura o la necesidad de actualización de alguno de los visores de RV, como se comentaba antes, son dispositivos muy nuevos con relativa fragilidad. Además, frecuentemente salen nuevos dispositivos que pueden dejar obsoletos a dispositivos de RV inmersiva antiguos, por lo que puede ser necesario cambiar el equipo con frecuencia.

6.6.3 Dimensión social

Por último, la dimensión social analiza el impacto del proyecto sobre las personas, es decir, sobre el autor del proyecto, los usuarios y otros afectados.

PPP

A nivel personal, este proyecto me ha permitido ampliar mis conocimientos en el ámbito de los gráficos por ordenador, visión por computador, aprendizaje automático, redes, y otras áreas. Se trata de un proyecto que engloba muchas disciplinas de la informática y me ha servido para consolidar todos los conocimientos adquiridos durante el Grado en Ingeniería Informática.

En el marco del grupo de investigación ViRVIG en el que se desarrolla este proyecto, se están realizando múltiples proyectos relacionados con la realidad virtual colaborativa, por tanto, este proyecto ha permitido progresar en la creación de técnicas de navegación e interacción que se podrán usar en otros proyectos.

Vida útil

En relación con los beneficiarios, el proyecto puede mejorar en gran medida los ciclos de desarrollo de equipos de arquitectos, ingenieros, sanitarios, etc. En cuanto se acaba un nuevo diseño, no es necesario esperar a hacer un prototipo o maqueta, gracias a este trabajo rápidamente múltiples personas pueden entrar en un entorno colaborativo y visualizar el diseño en una escena tridimensional. Incluso, si se amplía el trabajo a entornos no compartidos, y el equipo tiene personal en múltiples partes del mundo, se minimiza la necesidad de transporte dada el alto grado de naturalidad y realismo de los sistemas inmersivos.

En concreto se beneficiarán los arquitectos de la Sagrada Familia, a partir de este trabajo podrán colaborar múltiples arquitectos en una misma escena virtual, esta aplicación les permitirá visualizar diferentes diseños con un alto grado de realismo e inmersión.

El progreso en la RV colaborativa creemos que es importante para la sociedad, especialmente en la situación actual, muchos equipos se ven obligados a trabajar telemáticamente, tener herramientas para que estas interacciones sean eficientes y naturales es esencial.

Riesgos

Aunque los métodos para mejorar el *cybersickness* y la desorientación espacial están mejorando rápidamente, de hecho, en este trabajo se trata de solucionar estos problemas, hay personas más sensibles a este tipo de efectos. Si el equipo de trabajo de dichas personas adopta un sistema de RV inmersiva, podría empeorar la calidad de vida de las personas sensibles a estos efectos, por este motivo, se trabaja en mitigar los efectos mediante *hardware* y *software*.

7 Conclusiones y trabajo futuro

En este trabajo se ha desarrollado una aplicación que permite la conexión de usuarios a un entorno virtual colaborativo en tiempo real, es decir, múltiples usuarios, cada uno con su casco de realidad virtual y ordenador, se conectan mediante una red a la aplicación servidor que permite a los participantes compartir escena virtual, interaccionar, y navegar libremente.

Para permitir la libre navegación por el escenario virtual, se han estudiado, desarrollado y evaluado diferentes técnicas de navegación. Las técnicas se han agrupado en tres modos, que permiten al usuario final decidir qué tipo de navegación quiere en función de su preferencia. Algunos modos ofrecen mucha libertad de movimiento para los usuarios más experimentados y pueden inducir mareos o desorientación espacial, otros modos son más restrictivos y reducen al mínimo los problemas mencionados.

La evaluación de las técnicas se ha hecho mediante pruebas piloto, entre ellas, la aplicación se presentó a los arquitectos de la Sagrada Familia. Esto ha permitido una primera evaluación de la efectividad de las técnicas sobre diferentes tipos de usuarios, desde usuarios más experimentados con la realidad virtual, hasta participantes que nunca la habían probado.

Seguidamente, con el fin de mejorar la interacción entre usuarios, se ha trabajado con las cámaras del visor de RV para segmentar a los participantes para su representación visual en la escena virtual. Para ello, se han investigado e implementado diferentes técnicas basadas en las texturas de profundidades, la segmentación por color, y en la librería de detección de poses *OpenPose*.

La segmentación por texturas de profundidades o por color ha resultado ser poco efectiva, ya sea por la calidad de la segmentación o por los altos costes computacionales de los métodos. Por este motivo, se ha desarrollado una segmentación del usuario a partir de primitivas geométricas simples (círculos y rectángulos). Para aproximar la posición y tamaño de estas primitivas se ha usado *OpenPose* y se ha creado una red neuronal que ha sido entrenada a partir de datos generados artificialmente mediante *scripts*.

Todos los métodos se han desarrollado para ser utilizados en tiempo real, por lo tanto, se ha hecho un gran uso de la capacidad de cómputo de la tarjeta gráfica a través de *compute shaders*, y modificando los *shaders* encargados del renderizado de todo aquello relacionado con este trabajo.

En conclusión, este trabajo ha cumplido con los objetivos fijados inicialmente al estudiar, desarrollar y evaluar una aplicación colaborativa funcional en tiempo real, técnicas de navegación e interacción, y métodos de representación visual de los usuarios. El resultado es una aplicación que será utilizada por los arquitectos de la Sagrada Familia, pero que puede ser utilizada por cualquier grupo de usuarios que necesiten un sistema de colaboración en entornos virtuales inmersivos.

A continuación, se presentan aquellos aspectos que se quieren investigar en un futuro para ampliar el trabajo hecho en este proyecto.

Estudio con usuarios

Una vez desarrolladas las técnicas de navegación e interacción se han hecho pruebas piloto para determinar la efectividad de los métodos, sin embargo, para poder analizar con más profundidad estas técnicas, se quiere realizar un estudio con usuarios donde se comparen los diferentes modos de navegación. La idea es recoger datos de cómo interaccionan los usuarios, por ejemplo, se recogería información sobre la posición física y virtual de los participantes, la orientación, qué técnicas se han usado más, etc. Además, se quiere experimentar con usuarios con diferentes grados de experiencia en realidad virtual, pues el tipo de técnica a utilizar en un entorno colaborativo puede variar mucho en función de la experiencia previa de los participantes.

Colaborativo remoto

Este trabajo se centra en la aplicación de la RV colaborativa en espacios físicos compartidos, no obstante, con la creciente popularidad del trabajo remoto, con pocas modificaciones, sería posible que los participantes se conectaran a la aplicación desde diferentes espacios físicos, por ejemplo, cada usuario se podría conectar desde su casa. Tanto la aplicación como gran parte de las técnicas de navegación se pueden utilizar para esta ampliación, la representación visual mediante cámaras se tendría que plantear de manera diferente al no compartir espacio físico.

Otros métodos de segmentación

Recientemente, se está popularizando la segmentación de personas mediante redes neuronales profundas, a diferencia de este trabajo donde se usa *OpenPose* y a partir de la pose se crea una segmentación, estas redes serían capaces de segmentar directamente al usuario a partir de una imagen. Por ejemplo, *DensePose* [45] presenta una red neuronal profunda capaz de segmentar múltiples usuarios en una imagen. Como futura ampliación, se podría investigar la viabilidad de estas técnicas para ser ejecutadas en tiempo real, así como su integración en la aplicación.

Bibliografía

[1] C. Andujar, P. Brunet, J. Buxareu, J. Fons, N. Laguarda, J. Pascual y N. Pelechano, "VR-assisted Architectural Design in a Heritage Site: the Sagrada Família Case Study", en *European Association for Computer Graphics (Eurographics)*, European Association for Computer Graphics (Eurographics), 2018, págs. 47-56. DOI: 10.2312/gch.20181340.

[2] *Imagen realidad virtual colaborativa*. dirección: https://www.engineering.com/DesignSoftware/DesignSoftwareArticles/ArticleID/9484 (visitado 24-02-2020).

[3] *Virtual Reality, Augmented Reality, Mixed Reality and the Academic Library — Library Information Technology Association (LITA)*. dirección: http://www.ala.org/lita/virtual-reality-augmented-reality-mixed-reality-and-academic-library (visitado 19-06-2020).

[4] *Grupo de investigación ViRVIG*. dirección: https://www.virvig.eu/ (visitado 20-02-2020).

[5] K. Zibrek, E. Kokkinara y R. McDonnell, "Don't stand so close to me: Investigating the effect of control on the appeal of virtual humans using immersion and a proximity-based behavioral task", en *Proceedings - SAP 2017, ACM Symposium on Applied Perception*, New York, New York, USA: Association for Computing Machinery, Inc, 2017, págs. 1-11, ISBN: 9781450351485. DOI: 10.1145/3119881.3119887.

[6] B. Spanlang, J.-M. Normand, D. Borland, K. Kilteni, E. Giannopoulos, A. Pomés, M. González-Franco, D. Perez-Marcos, J. Arroyo-Palacios, X. N. Muncunill y M. Slater, "How to Build an Embodiment Lab: Achieving Body Representation Illusions in Virtual Reality", *Frontiers in Robotics and AI*, vol. 1, 2014, ISSN: 2296-9144. DOI: 10.3389/frobt.2014.00009.

[7] J. J. LaViola, "A discussion of cybersickness in virtual environments", *ACM SIGCHI Bulletin*, vol. 32, n.º 1, págs. 47-56, 2000, ISSN: 07366906. DOI: 10.1145/333329.333344.

[8] A. Ríos, M. Palomar y N. Pelechano, "Users' locomotor behavior in collaborative virtual reality", en *Proceedings - MIG 2018: ACM SIGGRAPH Conference on Motion, Interaction, and Games*, Association for Computing Machinery, Inc, 2018, ISBN: 9781450360159. DOI: 10.1145/3274247.3274513.

[9] T. Randhavane, A. Bera y D. Manocha, "F2Fcrowds: Planning agent movements to enable face-to-face interactions", *Presence: Teleoperators and Virtual Environments*, vol. 26, n.º 2, págs. 228-246, 2017, ISSN: 15313263. DOI: 10.1162/PRES_a_00294.

[10] K. Varma, S. J. Guy y V. Interrante, "Assessing the Relevance of Eye Gaze Patterns During Collision Avoidance in Virtual Reality", *ICAT-EGVE 2017 - International Conference on Artificial Reality and Telexistence and Eurographics Symposium on Virtual Environments*, 2017. DOI: 10.2312/egve.20171352.

[11] *Realidad virtual colaborativa - TechViz*. dirección: https://www.techviz.net/es/vr-collaboration/ (visitado 23-02-2020).

[12] *Unity3D - Plataforma de desarrollo en tiempo real*. dirección: https://unity.com/es (visitado 23-02-2020).

[13] E. Bozgeyikli, A. Raij, S. Katkoori y R. Dubey, "Point and Teleport locomotion technique for virtual reality", en *CHI PLAY 2016 - Proceedings of the 2016 Annual Symposium on Computer-Human Interaction in Play*, New York, New York, USA: Association for Computing Machinery, Inc, 2016, págs. 205-216, ISBN: 9781450344562. DOI: 10.1145/2967934.2968105.

[14] *Example SteamVR Teleporting*. dirección: https://threesixtyreality.co.uk/blog/interaction-patterns/virtual-reality-patterns/teleport-lock-target/ (visitado 24-02-2020).

[15] C. Cruz-Neira, D. J. Sandin, T. A. DeFanti, R. V. Kenyon y J. C. Hart, "The CAVE: Audio Visual Experience Automatic Virtual Environment", *Communications of the ACM*, vol. 35, n.º 6, págs. 64-72, 1992, ISSN: 15577317. DOI: 10.1145/129888.129892.

[16] *Virtual Reality Image Wikimedia Commons*. dirección: https://commons.wikimedia.org/wiki/File:Reality_check_ESA384313.jpg (visitado 24-02-2020).

[17] *Oculus Rift S: visor de realidad virtual para PC*. dirección: https://www.oculus.com/rift-s/ (visitado 23-02-2020).

[18] *HTC Vive: visor de realidad virtual para PC*. dirección: https://www.vive.com/eu/product/#viveseries (visitado 23-02-2020).

[19] *Unreal Engine - Plataforma de desarrollo en tiempo real*. dirección: https://www.unrealengine.com/en-US/ (visitado 23-02-2020).

[20] *SteamVR - Valve Developer Community*. dirección: https://developer.valvesoftware.com/wiki/SteamVR (visitado 19-06-2020).

[21] G. Armitage, M. Claypool y P. Branch, *Networking and Online Games: Understanding and Engineering Multiplayer Internet Games*. Chichester: Wiley, 2006, ISBN: 0470018577, 9780470018576.

[22] E. R. Braden, "Requirements for Internet Hosts - Communication Layers", inf. téc., 1989. DOI: 10.17487/rfc1122.

[23] *Imagen Setup Bases HTC Vive*. dirección: https://www.octopusrift.com/setup-your-room-for-vr/ (visitado 22-06-2020).

[24] *OpenCV: OpenCV-Python*. dirección: https://docs.opencv.org/master/d6/d00/tutorial_py_root.html (visitado 18-05-2020).

[25] D. G. Lowe, "Object recognition from local scale-invariant features", en *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2, IEEE, 1999, págs. 1150-1157. DOI: 10.1109/iccv.1999.790410.

[26] *VIVE SRWorks SDK*. dirección: https://developer.vive.com/resources/knowledgebase/intro-vive-srworks-sdk/ (visitado 18-05-2020).

[27] *Computer Graphics Learning - Textures and Sampling*. dirección: https://cglearn.eu/pub/computer-graphics/textures-and-sampling (visitado 24-05-2020).

[28] G. Hidalgo, Z. Cao, T. Simon, S.-E. Wei, H. Joo e Y. Sheikh, *OpenPose: Real-time multi-person keypoint detection library for body, face, hands, and foot estimation*. dirección: https://github.com/CMU-Perceptual-Computing-Lab/openpose (visitado 04-06-2020).

[29] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua y S. Süsstrunk, "SLIC superpixels compared to state-of-the-art superpixel methods", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, n.º 11, págs. 2274-2281, 2012, ISSN: 01628828. DOI: 10.1109/TPAMI.2012.120.

[30] C. Rother, V. Kolmogorov y A. Blake, ""GrabCut Interactive foreground extraction using iterated graph cuts", en *ACM Transactions on Graphics*, vol. 23, 2004, págs. 309-314. DOI: 10.1145/1015706.1015720.

[31] A. Dempster, N. Laird y D. Rubin, "Maximum Likelihood from Incomplete Data via the EM Algorithm", *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, págs. 1-38, 1977. DOI: 10.2307/2984875.

[32] *GrabCut Graph Image*. dirección: https://www.cs.ru.ac.za/research/g02m1682/ (visitado 07-06-2020).

[33] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll y M. J. Black, "SMPL: A skinned multi-person linear model", en *ACM Transactions on Graphics*, vol. 34, Association for Computing Machinery, 2015. DOI: 10.1145/2816795.2818013.

[34] *CMU's motion capture database*. dirección: https://sites.google.com/a/cgspeed.com/cgspeed/motion-capture/daz-friendly-release (visitado 12-06-2020).

[35] Ministerio de La Presidencia, *Real Decreto 463/2020, estado de alarma por el COVID-19*, 2020. dirección: https://www.boe.es/buscar/pdf/2020/BOE-A-2020-3692-consolidado.pdf.

[36] *Hays - Sueldos informática*. dirección: https://www.hays.es/ (visitado 07-03-2020).

[37] *Precios coworking Barcelona — Coworkidea*. dirección: https://coworkidea.com/tarifas/ (visitado 05-03-2020).

[38] J. del Estado, *Ley Orgánica 3/2018 de Protección de Datos Personales y garantía de los derechos digitales*, 2018. dirección: https://www.boe.es/buscar/pdf/2018/BOE-A-2018-16673-consolidado.pdf.

[39] *Licencia Unity3D para fines académicos*. dirección: https://unity.com/es/education/license-grant-program (visitado 31-05-2020).

[40] *Licencia SRWorks*. dirección: https://developer.vive.com/resources/knowledgebase/sdk-license-agreement-english-version/ (visitado 31-05-2020).

[41] *Licencia OpenCV*. dirección: https://opencv.org/license/ (visitado 31-05-2020).

[42] *Licencia Tensorflow*. dirección: https://github.com/tensorflow/tensorflow/blob/master/LICENSE (visitado 31-05-2020).

[43] *Licencia Blender*. dirección: https://www.blender.org/about/license/ (visitado 31-05-2020).

[44] Facultad de Informática de Barcelona, *Informe de sostenibilidad TFG*, 2018. dirección: https://www.fib.upc.edu/sites/fib/files/documents/estudis/tfg-informe-sostenibilitat-2018.pdf (visitado 01-06-2020).

[45] R. A. Güler, N. Neverova e I. Kokkinos, "DensePose: Dense Human Pose Estimation", *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, págs. 7297-7306, 2018. arXiv: 1802.00434.