



TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN ENXEÑARÍA DE COMPUTADORES



Seguimiento y alineación 3D para la visualización de modelos anatómicos mediante realidad aumentada e impresión 3D

Estudiante: Brais Barboza Ordóñez
Dirección: José Antonio Iglesias Gutián
Francisco Javier Taibo Pena
Emilio José Padrón González

A Coruña, septiembre de 2025.

A mis padres y mi novia

Agradecimientos

Dedico este logro académico a aquellos que me han brindado su apoyo incondicional en mi camino. A mis tutores Emilio, Jose y Javi, quienes personificaron la guía necesaria para completar este proyecto con éxito. Gracias por vuestras enseñanzas valiosas y por creer en mí.

A mi familia, especialmente a mis padres, por su amor incondicional y por ser mi constante motivación. Gracias por estar a mi lado en cada decisión y por ser mi fuente de inspiración.

A mis amigos, por su amistad y apoyo en momentos difíciles. Gracias por hacer la vida más alegre e incitarme a alcanzar nuevos horizontes.

Gracias a todos por ayudarme a alcanzar este objetivo. Este trabajo es el resultado de vuestro apoyo y esfuerzo constante, y está dedicado con todo mi agradecimiento y cariño.

Resumen

La imagen médica se ha beneficiado a lo largo del tiempo de los avances en las técnicas de visualización de contenido con el fin de poder brindar una mejor comprensión de los datos mostrados que se traduzca en una mejor atención médica. Es por ello que en este trabajo se analizan métodos para combinar la imagen médica con la realidad aumentada, otro campo cuyo auge en los últimos años para aplicaciones tanto industriales como de ocio no han pasado desapercibidas. Se lleva a cabo un análisis completo para proporcionar una solución funcional, desde la segmentación de los modelos desde una Tomografía Computerizada (TC), el desarrollo de un marcador para el seguimiento y la alineación en 3D hasta la implementación de una solución para llevarlo a cabo. Con el objetivo de servir como discusión de futuros desarrollos, se concibe teniendo como piedra angular su libre disposición y el software libre como alternativa a métodos existentes tras licencias restrictivas.

Abstract

Medical imaging has benefited over time from advances in content visualization techniques in order to provide a better understanding of the displayed data, leading to improved healthcare. Therefore, this project examines methods for combining medical imaging with augmented reality, another field whose recent rise for both industrial and recreational applications has not gone unnoticed. A comprehensive analysis is carried out to provide a functional solution, including model segmentation from a computerized tomography (CT) scan, marker development for tracking and 3D alignment, and the implementation of a solution to execute the process. In order to serve as a discussion of future developments, this work is conceived with the cornerstone of open availability and open-source software as an alternative to existing methods with restrictive licenses.

Palabras clave:

- Realidad aumentada
- Impresión 3D
- Imagen médica

Keywords:

- Augmented reality
- 3D Printing
- Medical imaging

Índice general

1	Introducción	1
1.1	Contexto y motivación	1
1.2	Objetivos	2
1.3	Estructura de la memoria	3
2	Fundamentos Teóricos y Técnicos	4
2.1	Tomografía Computarizada	4
2.2	Impresión 3D	6
2.3	Realidad Extendida	7
2.4	Visión Artificial	8
3	Estado del Arte	9
3.1	Visualización	9
3.2	Realidad Extendida	9
3.3	Impresión 3D	11
4	Herramientas y Software	12
4.1	Lenguajes de programación	12
4.1.1	C++	12
4.2	Sistema Operativo	12
4.2.1	Windows 11	12
4.3	Control de versiones	12
4.3.1	GitLab	12
4.4	Entorno de desarrollo	13
4.4.1	Visual Studio Community	13
4.5	Herramientas	13
4.5.1	DICOM	13
4.5.2	3D Slicer	13

4.5.3	Meshmixer	13
4.5.4	Meshlab	13
4.5.5	Blender	14
4.5.6	UltiMaker Cura	14
4.6	Frameworks	14
4.6.1	OpenXR	14
4.6.2	OpenCV	16
4.6.3	OpenSceneGraph	16
4.6.4	Assimp	16
4.6.5	GLFW	16
4.6.6	GLEW	17
4.6.7	GLM	17
4.6.8	ArUco	17
4.6.9	Funcionamiento de ArUco	19
4.7	Hardware	22
4.7.1	Impresión 3D	22
4.7.2	Obtención de vídeo	22
5	Metodología y Gestión del proyecto	23
5.1	Metodología	23
5.2	Sprints	24
5.2.1	Extracción de un modelo a partir de un TC	24
5.2.2	Diseño del marcador fiduciario y software de tracking	24
5.2.3	Integrar solución de tracking en Exposure Render	24
5.2.4	Integrar passthrough en Exposure Render	24
5.2.5	Integrar con motor de render independiente	25
5.3	Planificación temporal	25
6	Ejecución del proyecto	26
6.1	Consideraciones previas	26
6.2	Ánalisis	27
6.3	Generación de volúmenes a partir de TC	27
6.4	Impresión 3D del volumen	28
6.5	Desarrollo del marcador fiduciario	30
6.5.1	Planteamiento del problema y selección de la solución	30
6.5.2	Fundamentos teóricos del seguimiento con ArUco	31
6.5.3	Generación de marcadores a partir de diccionarios predefinidos	33
6.5.4	Algoritmo de cálculo de pose del cubo	34

6.5.5	Diseño geométrico del marcador cúbico	34
6.5.6	Transformación de coordenadas de cara a centro del cubo	35
6.6	Implementación del Passthrough en Exposure Render	38
6.6.1	Captura de imagen	38
6.6.2	Integración del seguimiento	38
6.6.3	Consideraciones de implementación alternativa	41
6.7	Aplicación Independiente de Realidad Aumentada	41
6.7.1	Arquitectura de la aplicación	41
6.7.2	Sistema de renderizado basado en OpenGL	42
6.7.3	Sistema de carga de modelos	43
6.7.4	Flujo de ejecución de la aplicación	44
7	Conclusiones y trabajo futuro	46
7.0.1	Enriquecimiento Formativo	47
7.0.2	Trabajo futuro	47
A	Detalles de Implementación	50
A.1	Implementación del algoritmo de tracking	50
A.1.1	Función moveAxis	50
A.1.2	Aplicación de las transformaciones al centro del cubo	50
A.1.3	Función cubeCoordinates	51
A.2	Transformaciones de coordenadas entre sistemas	52
A.2.1	Conversión del espacio de cámara al espacio del cubo	52
A.2.2	Inversión para obtener la matriz modelo	52
A.2.3	Conversión entre sistemas row-major y column-major	52
A.3	Interfaz principal de tracking	52
A.3.1	Función getCubePoseMatrix	52
B	Diagramas y Gráficos Detallados	53
B.1	Diagrama de flujo completo del sistema de tracking	53
B.2	Planificación temporal detallada	53
B.3	Tabla detallada de tareas del proyecto	53
Lista de acrónimos		57
Glosario		59
Bibliografía		60

Índice de figuras

2.1	Representación del funcionamiento de un TC	5
2.2	Representación simplificada del concepto de <i>virtuality continuum</i>	7
3.1	Aproximaciones a la realidad extendida para aplicaciones biomédicas. (fuente: [1])	10
4.1	Ciclo de vida de OpenXR. (fuente: https://www.khronos.org/)	15
4.2	Explicación de la API por capas. (fuente: https://registry.khronos.org)	16
4.3	Marcador Generado con ArUco	18
4.4	Ejemplos de marcadores fiduciarios previos (fuente: [2])	18
4.5	Proceso de extracción de Bits	21
6.1	Flujo de integración del seguimiento en Exposure Render.	28
6.2	TC de partida y obtención del modelo 3D final con 3D Slicer.	29
6.3	Modelo de cráneo 3D con geometrías erróneas.	29
6.4	Proceso de recuperación de material.	30
6.5	Esquema de un marcador.	32
6.6	Esquema de una tabla de marcadores.	33
6.7	Layout de las caras del marcador fiduciario.	35
6.8	Diseño final del marcador fiduciario.	36
6.9	Imágenes sobre las que se estima la pose del cubo.	37
B.1	Diagrama de flujo detallado del sistema de tracking implementado.	55
B.2	Diagrama de Gantt completo del proyecto mostrando todos los sprints, tareas y dependencias temporales.	56

Índice de tablas

B.1 Tabla completa de tareas del proyecto con estimaciones detalladas y asignación a sprints.	54
---	----

Capítulo 1

Introducción

El constante progreso en medicina, y en particular en imagen médica, hace necesario contar con sistemas cada vez más avanzados para la representación y visualización de los datos obtenidos. El progreso en esta área ha sido patente en los últimos 30 años [3]. Más recientemente, en esta búsqueda de métodos que sean capaces de explotar las mejoras tecnológicas, aparece la aplicación de la [Realidad Aumentada \(AR\)](#) y de otras técnicas englobadas dentro de lo que se conoce como [Realidad Extendida \(XR\)](#) en la medicina [4]. Un uso efectivo de este tipo de tecnologías en el campo de la imagen médica supone un complemento de gran utilidad para muchas de las tareas del personal médico: diagnóstico, planificación preoperatoria, explicación a pacientes, cirugía guiada por imagen, formación médica, etc.

Este proyecto plantea el uso de piezas creadas con una impresora 3D a partir de imagen médica en un entorno de realidad aumentada, abordando la problemática de su detección y seguimiento para un correcto alineamiento con un modelo 3D virtual, con el objetivo de integrar una imagen sintética sobreimpresa en la imagen real capturada por la cámara.

1.1 Contexto y motivación

Varias son las razones que explican el actual auge en la aplicación de técnicas de [XR](#) en medicina. Por un lado, la superposición de información digital sobre imágenes reales facilita la visualización de datos médicos y su interpretación. Esta información extra ayuda al personal sanitario a tener una comprensión más clara y detallada de la anatomía de un paciente, lo que resulta especialmente útil tanto en las fases de diagnóstico como durante procedimientos quirúrgicos complejos, proporcionando información relevante en el campo de visión que sirve como guía en tiempo real durante una intervención. Además, la posibilidad de superponer modelos tridimensionales de estructuras anatómicas en el paciente facilita la planificación precisa de una cirugía, así como la comunicación con el paciente sobre el procedimiento. Por otro lado, la realidad extendida ofrece una herramienta efectiva para la educación y formación

médica. Los estudiantes y profesionales sanitarios pueden utilizarla para practicar y simular procedimientos médicos en un entorno virtual realista antes de realizarlos en pacientes reales. Esto brinda la oportunidad de adquirir experiencia y habilidades sin riesgo para los pacientes.

En este proyecto proponemos el uso de piezas creadas con una impresora 3D a partir de imágenes médicas en un entorno de realidad aumentada que permita superponer una imagen sintética (un *render* 3D) sobre la visualización de la pieza en la imagen real capturada por la cámara. La idea fundamental es facilitar la manipulación de lo que podría ser una prótesis médica en un entorno de realidad extendida completo. Dentro de ese entorno, la pieza de AR desarrollada en este proyecto se encargaría de la correcta detección y seguimiento de la prótesis, junto a la integración de una imagen virtual superpuesta sobre la misma.

Para el correcto seguimiento y alineación de una pieza física con su imagen virtual en un flujo de vídeo es preciso conocer los parámetros de la cámara y la posición de la misma respecto al objeto. Para la impresión puede extraerse un modelo 3D a partir de una [Tomografía Computerizada \(TC\)](#). Con el fin de facilitar el seguimiento es posible que sea preciso añadir marcadores de referencia sobre la pieza, para utilizarlos como guía. En el proyecto se cubrirá el flujo de trabajo completo, desde el análisis y la extracción de un modelo a partir de la [TC](#), pasando por el diseño de un marcador fiduciario que se usará para hacer el seguimiento del objeto, hasta la manipulación física de la pieza en un entorno de realidad aumentada en la que se proyectará un modelo virtual sobre la pieza impresa vista en imágenes reales.

1.2 Objetivos

Los objetivos principales de este proyecto son:

- Extraer un modelo 3D a partir de imágenes capturadas mediante [Tomografía Computerizada \(TC\)](#) para su impresión.
- Diseñar un marcador fiduciario que sirva como guía para facilitar el seguimiento de la pieza.
- Hacer detección, seguimiento y alineamiento 3D de la pieza impresa o guía en el flujo de vídeo capturado por un sistema de realidad aumentada, como puede ser un [HMD](#) que incorpore cámaras de vídeo.
- Integrar elementos sintéticos en la imagen real de la visualización 3D.
- El objetivo final es disponer de un software capaz de resolver el problema del seguimiento y la estimación de pose en 3D y que pueda ser fácilmente integrable en un sistema de realidad extendida completo.

1.3 Estructura de la memoria

Esta memoria se estructura en siete capítulos que abarcan desde los fundamentos teóricos hasta las conclusiones y trabajo futuro, complementados con dos apéndices técnicos.

Capítulo 1: Introducción. Presenta el contexto del proyecto, la motivación que lo impulsa, los objetivos planteados y la estructura del documento.

Capítulo 2: Fundamentos Teóricos y Tecnológicos. Establece las bases conceptuales necesarias para comprender el trabajo, cubriendo los principios de tomografía computarizada, impresión 3D, realidad extendida y visión artificial.

Capítulo 3: Estado del Arte. Realiza una revisión de las soluciones existentes en el campo del seguimiento 3D y la realidad aumentada aplicada a imagen médica, analizando sus ventajas y limitaciones.

Capítulo 4: Herramientas y Software. Describe en detalle las tecnologías, frameworks, librerías y herramientas hardware utilizadas en el desarrollo del proyecto, incluyendo OpenXR, OpenCV, ArUco, Assimp, y los equipos de impresión 3D empleados.

Capítulo 5: Metodología y Gestión del Proyecto. Explica la metodología ágil implementada, la organización en sprints, la planificación temporal y la gestión de riesgos durante el desarrollo.

Capítulo 6: Ejecución del Proyecto. Constituye el núcleo técnico del trabajo, detallando el proceso completo desde la extracción de volúmenes 3D a partir de TC, el diseño e impresión del marcador fiduciario cúbico, el desarrollo del algoritmo de seguimiento basado en ArUco, hasta la implementación de la aplicación independiente de realidad aumentada.

Capítulo 7: Conclusiones y Trabajo Futuro. Evalúa el cumplimiento de los objetivos planteados, reflexiona sobre el enriquecimiento formativo del proyecto y propone líneas de desarrollo futuro.

Apéndice A: Detalles de Implementación. Incluye el código fuente completo de las funciones clave, diagramas de flujo detallados y especificaciones técnicas de la implementación.

Apéndice B: Diagramas Detallados. Contiene la planificación temporal completa del proyecto mediante diagramas de Gantt y la descripción detallada de todas las tareas realizadas.

La memoria se complementa con un glosario de acrónimos y términos técnicos, así como una bibliografía completa que sustenta el trabajo realizado.

Capítulo 2

Fundamentos Teóricos y Técnicos

EN este capítulo se repasan los principios básicos sobre los que se establece este trabajo.

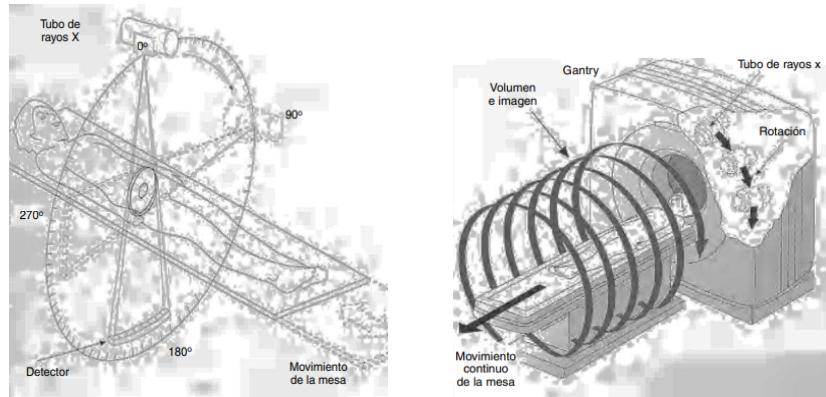
2.1 Tomografía Computarizada

Una **Tomografía Computerizada (TC)**, también conocida como escáner, es una técnica de diagnóstico médico que utiliza rayos X, detectores y un ordenador para obtener imágenes detalladas de estructuras internas del cuerpo. La **TC** combina una serie de imágenes radiográficas en secciones transversales para crear imágenes en 2D y 3D del área estudiada.

Durante una **TC**, el paciente se coloca en una mesa que se desliza dentro de un dispositivo circular llamado tomógrafo. Un tomógrafo es, en esencia, una máquina de rayos X en la cual se ha sustituido la placa por una serie de detectores [5]. La fuente de rayos X y los detectores efectúan un movimiento circular y avanzan lentamente hasta cubrir el área deseada, como se puede ver en Figura 2.1. Los detectores capturan la radiación después de que ha atravesado el cuerpo. La información recopilada se envía a un ordenador que procesa los datos y los convierte en imágenes transversales o en secciones longitudinales del área de interés.

La tomografía computarizada proporciona imágenes detalladas de tejidos blandos, huesos y órganos internos, lo que permite a los médicos diagnosticar y evaluar una amplia variedad de condiciones y enfermedades. Se utiliza comúnmente en el diagnóstico y seguimiento de enfermedades como cáncer, lesiones traumáticas, enfermedades cardiovasculares, trastornos pulmonares, afecciones cerebrales y abdominales, entre otros.

Teorema de Radon. Propuesto por el matemático austriaco Johann Radon en 1917, el Teorema de Radon es un resultado fundamental en la teoría de la tomografía computerizada. Este teorema establece que es posible reconstruir una función bidimensional a partir de sus pro-



(a) Orientación del tubo de rayos X respecto al eje corporal.

(b) Tomografía axial computerizada convencional.

Figura 2.1: Representación del funcionamiento de un TC

yecciones a lo largo de diferentes ángulos. En el contexto de la tomografía computerizada, las proyecciones se obtienen mediante la medición de la atenuación de la radiación a medida que atraviesa el cuerpo en estudio. Una vez que se han recopilado todas las proyecciones, se utiliza un algoritmo de reconstrucción para combinarlas y generar una imagen transversal detallada del área estudiada. La base matemática de ese proceso de reconstrucción de imágenes en la tomografía computerizada la proporciona el teorema de Radon y ha llevado al desarrollo de diversos algoritmos de reconstrucción.

Algoritmos de reconstrucción de imágenes. Los algoritmos de reconstrucción de imágenes son métodos computacionales utilizados para reconstruir imágenes bidimensionales o tridimensionales a partir de proyecciones adquiridas en la tomografía computerizada. Estos algoritmos se basan en el Teorema de Radon y pueden clasificarse en dos categorías principales: métodos analíticos [6] y métodos iterativos [7].

1. Métodos analíticos: Estos algoritmos, como la [Retroproyección Filtrada \(FBP\)](#), procesan las proyecciones de manera directa para obtener la imagen reconstruida. La [FBP](#) es el algoritmo más utilizado en la práctica clínica debido a su rapidez y eficiencia.

2. Métodos iterativos: Estos algoritmos, como el de [Máxima Verosimilitud de la Expectativa-Maximización \(MLEM\)](#) y el de [Mínimos Cuadrados Conjugados \(CGLS\)](#), utilizan un enfoque iterativo para mejorar la calidad de la imagen reconstruida. Aunque estos métodos suelen ser más lentos que los analíticos, pueden proporcionar imágenes de mayor calidad y son especialmente útiles en aplicaciones donde la cantidad de datos de proyección es limitada o ruidosa.

2.2 Impresión 3D

La impresión 3D es una tecnología de fabricación aditiva que permite crear objetos tridimensionales a partir de un modelo digital. Esta tecnología ha revolucionado la forma en que se fabrican piezas y productos, ya que permite la creación de objetos complejos con geometrías que serían difíciles o imposibles de lograr con métodos de fabricación tradicionales. La impresión 3D se utiliza en una amplia variedad de aplicaciones, desde la fabricación de piezas de repuesto hasta la creación de prótesis médicas personalizadas.

Fabricación aditiva. La impresión 3D es un ejemplo de fabricación aditiva, que se refiere a la manipulación y depósito de un material a escala micrométrica de forma muy precisa para construir un sólido. La fabricación aditiva es una alternativa a los métodos de fabricación tradicionales, como el fresado y el torneado, que implican la eliminación de material de una pieza bruta [8].

Esta técnica de fabricación presenta una serie de ventajas. La complejidad de la geometría de la figura no encarece la fabricación de la misma (a expensas de la necesidad de material como soporte de la geometría principal), sino que permite generar piezas con geometrías previamente inviables o con un alto coste. Otra de las ventajas es la posibilidad de generar prototipos de piezas cuyas versiones finales presentan un alto coste, por un precio muy reducido y una alta fidelidad, acelerando así el proceso iterativo del diseño.

Modelado. El proceso de impresión 3D comienza con el modelado de la pieza o producto que se desea imprimir. Este modelo puede crearse utilizando software [Computer-Aided Design \(CAD\)](#) o mediante la digitalización de un objeto existente utilizando un escáner 3D. En nuestro caso, el modelo tridimensional se obtiene de la reconstrucción 3D a partir de una [TC](#).

Slicing (rebanado) En impresión 3D, el proceso de slicing (o rebanado, en español) se refiere a la preparación del modelo tridimensional, típicamente en formato STL, para su impresión en capas sucesivas. Es un paso crucial que convierte el modelo en una serie de capas planas y delgadas que la impresora 3D puede imprimir una por una. El *slicer* permite además realizar ajustes en el modelo 3D, con el fin de obtener el resultado deseado, generando como resultado una serie de instrucciones precisas que la impresora entiende para elaborar capa a capa el volumen.

GCODE. Las instrucciones que se generan a partir del proceso de *slicing* son provistas en forma de GCODE. El GCODE es el lenguaje utilizado para describir paso a paso qué movimientos y acciones debe tomar la impresora en cada momento. Este lenguaje tiene distintas

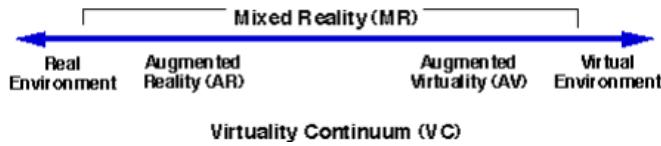


Figura 2.2: Representación simplificada del concepto de *virtuality continuum*.

implementaciones dependiendo del fabricante del equipamiento, ya que se trata de un lenguaje utilizado en múltiples aplicaciones de control numérico.

2.3 Realidad Extendida

Realidad Extendida (XR) es un término general que engloba todo el espectro de tecnologías inmersivas, incluyendo Realidad Virtual (VR), Realidad Aumentada (AR) y Realidad Mixta (MR). XR se refiere a la fusión de los mundos físico y digital, creando un entorno inmersivo que puede incluir objetos virtuales, información digital y elementos del mundo real.

En 1994, Milgram and Kishino [9] acuñan el término de *virtuality continuum*, que representa una escala que comprende desde la realidad física pura hasta la realidad virtual total, abarcando diferentes niveles de inmersión e interacción. Este *continuum* describe así la gama completa de experiencias, desde la realidad física no modificada hasta entornos completamente virtuales, como se muestra en la Figura 2.2:

- Entorno Real: consiste únicamente en objetos reales (extremo izquierdo de la figura).
- Realidad Aumentada: mundo real enriquecido con elementos digitales.
- Virtualidad Aumentada: mundo digital aumentado por objetos reales o físicos.
- Entorno Virtual: entorno puramente virtual (extremo derecho de la figura).

Realidad aumentada. Posteriormente Azuma [10] definía la Realidad Aumentada como una variación de la Realidad Virtual, en la cual el usuario es capaz de ver el mundo real, con objetos virtuales superpuestos, o compuestos por el mundo real. La posibilidad de crear objetos y prototipos de forma rápida, sobre los cuales poder iterar y componer imágenes virtuales, demuestra el potencial de estas tecnologías inmersivas trabajando a la par. En la actualidad sus principales aplicaciones se encuentran, además de en juegos y entretenimiento, en educación, medicina, arquitectura, ingeniería e interpretación del patrimonio.

2.4 Visión Artificial

Se denomina visión artificial al campo que incluye los métodos necesarios para adquirir, procesar, analizar y comprender las imágenes del mundo real con el fin de producir información procesable por un ordenador. Está fuertemente vinculado con la realidad aumentada, ya que es imprescindible para poder transmitir la información del medio al ordenador encargado de generar imágenes correspondientes. Entre los objetivos de la visión artificial se encuentra la capacidad de reconocer patrones dentro de una imagen o vídeo con el fin de poder extraer las características de los objetos dentro de dicho medio y procesarlas. Otro objetivo es la reconstrucción 3D a partir de imágenes, que pretende generar volúmenes 3D desde las imágenes obtenidas; esto es especialmente importante en la realidad aumentada porque permite una mayor percepción de la profundidad sobre el medio generado por ordenador.

Capítulo 3

Estado del Arte

Este capítulo se centra en detallar el estado actual de las tecnologías utilizadas en el proyecto así como en analizar soluciones similares al proyecto.

3.1 Visualización

Los componentes artificiales que deseamos incluir en las imágenes del mundo real proceden del framework desarrollado por Kroes et al. que aplica técnicas de Monte Carlo Ray Tracing (MCRT) sobre Direct Volume Rendering (DVR). El término DVR se utiliza para referirse a las técnicas que producen una imagen directamente a partir de datos de un volumen, sin realizar pasos intermedios. Para que esto sea posible es necesario implementar modelos físicos que indiquen cómo se genera, refleja, dispersa u oculta la luz [12]. Estos modelos con el paso del tiempo han evolucionado en modelos más y más complejos que han probado ser beneficiosos para la visualización científica de modelos 3D [13], [14], [15].

La implementación de estos modelos conlleva altos tiempos de renderizado, o en su defecto, un equipo increíblemente costoso para poder obtener una experiencia interactiva [16]. Para enfrentar esta casuística, Iglesias-Guitian et al. implementan un algoritmo de reducción de ruido basado en Recursive Least Squares (RLS) que permite una experiencia interactiva en tiempo real, sobre GPUs comerciales. Este proyecto se fundamenta en [16] para el renderizado de las imágenes que posteriormente se apliquen en realidad aumentada sobre las imágenes reales.

3.2 Realidad Extendida

Para llevar a cabo el proyecto existen varias aproximaciones en el estado del arte [1]. Se seleccionaron aquellas que más se ajustaban al proyecto. Uno de los objetivos principales es el seguimiento de piezas extraídas de un TC y diseñar marcadores que facilitasen el registro

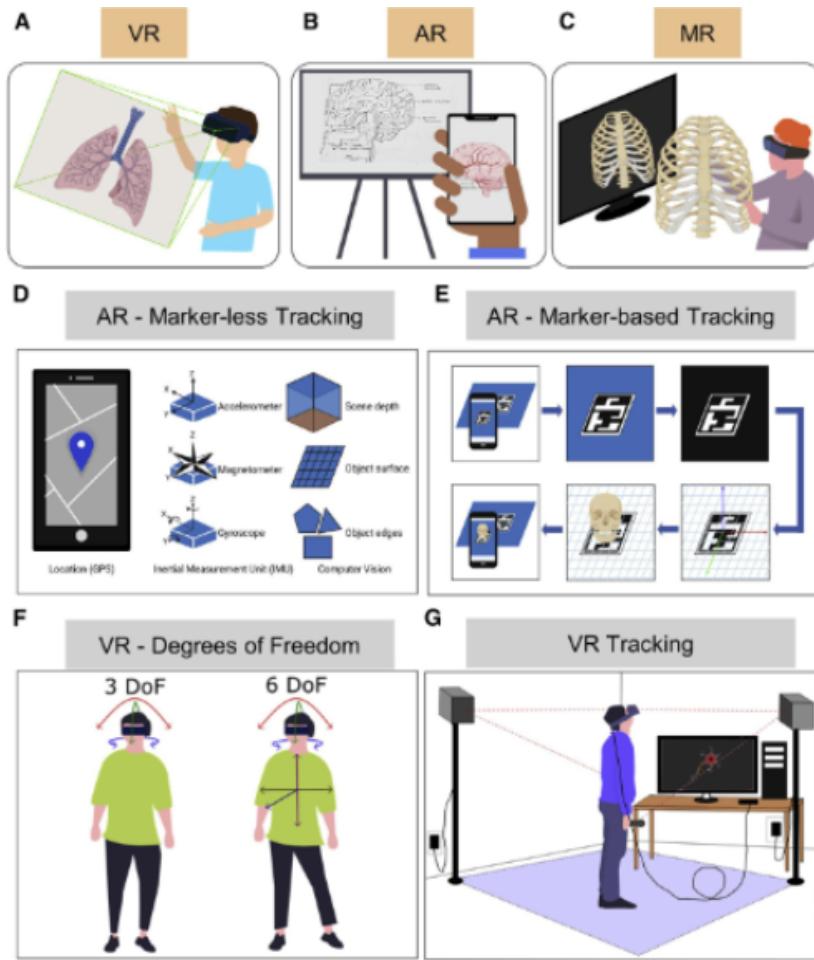


Figura 3.1: Aproximaciones a la realidad extendida para aplicaciones biomédicas. (fuente: [1])

de las mismas, por lo que se optó por un seguimiento basado en marcadores. No obstante, en lo que a la visualización se refiere, es necesario un equipo con gran potencia computacional, o en su defecto, un visor que permita la reproducción de vídeo renderizado por un tercer equipo. Dadas estas restricciones se decidió por utilizar un **Head Mounted Display (HMD)** HTC VIVE PRO. Utilizando la Figura 3.1 [1] como referencia, el sistema implementado se compondría de un seguimiento basado en marcadores (E) visualizando estos marcadores en realidad aumentada (B). Para este proyecto se escogieron las tecnologías E y G que se observan en la Figura 3.1.

Destacar también la solución presentada por [17]. Este proyecto presenta un método para diseñar aplicaciones de realidad aumentada para la visualización de modelos anatómicos en 3 dimensiones mediante el uso de un marcador fiduciario. En él se explica un método para extraer una figura a partir de una **TC**. Posteriormente, provee instrucciones detalladas para la impresión 3D del marcador fiduciario. Finalmente, utilizando la extensión de Vuforia para

Unity crea una aplicación móvil que permite la visualización en realidad aumentada de la pieza.

3.3 Impresión 3D

En el estado actual de la tecnología de impresión 3D, se observan avances significativos en materiales biocompatibles y técnicas de impresión de alta resolución, que permiten la creación de modelos anatómicos precisos para aplicaciones médicas. Tecnologías emergentes como la impresión 3D de metales y resinas fotopolimerizables han revolucionado la producción de prótesis personalizadas y herramientas quirúrgicas, facilitando intervenciones más seguras y eficientes [18]. Además, la integración con [Computer-Aided Design \(CAD\)](#) y escaneo 3D ha permitido la fabricación de marcadores fiduciarios personalizados, esenciales para el registro en realidad aumentada, como se detalla en proyectos similares [17]. Estos desarrollos no solo han reducido costos y tiempos de producción, sino que también han expandido el acceso a tecnologías de vanguardia en entornos clínicos.

Capítulo 4

Herramientas y Software

En este capítulo se explican las herramientas y librerías utilizadas para llevar a cabo el proyecto.

4.1 Lenguajes de programación

4.1.1 C++

El proyecto se desarrolló en su totalidad en C++. Esto se debe a que, como se menciona previamente, este trabajo forma parte del esfuerzo académico de Iglesias-Guitian et al. y por coherencia se decidió seguir la línea de trabajo. C++ es un lenguaje de programación que se beneficia de programación orientada a objetos sobre la sintaxis de C y se ha utilizado para implementar librerías gráficas intrínsecas en el proyecto.

4.2 Sistema Operativo

4.2.1 Windows 11

Windows 11 es un sistema operativo desarrollado por Microsoft. Se utilizó debido a la familiaridad del proyecto con el mismo.

4.3 Control de versiones

4.3.1 GitLab

Para llevar a cabo el control de versiones se utilizó GitLab ya que el código implementado formaba parte del proyecto previamente mencionado, y este se almacena en GitLab.

4.4 Entorno de desarrollo

4.4.1 Visual Studio Community

Es el Entorno de desarrollo de C++ por excelencia en Windows.

4.5 Herramientas

4.5.1 DICOM

DICOM es la denominación de un estándar utilizado principalmente para la visualización, impresión, almacenamiento y transmisión de imágenes y datos de propósito médico. Los ficheros **DICOM** consisten en una cabecera con campos estandarizados y de forma libre, y un cuerpo con datos de imagen. Un objeto **DICOM** simple puede contener solamente una imagen, pero esta imagen puede tener múltiples fotogramas, permitiendo el almacenamiento de bloques de datos con varios fotogramas.

4.5.2 3D Slicer

3D Slicer es un programa de software libre diseñado para solventar los problemas más avanzados de la computación de imagen relacionados con las aplicaciones clínicas y biométricas. Entre las capacidades del mismo se encuentra la posibilidad de implementar scripts de Python, segmentación de imágenes y volúmenes, la posibilidad de añadir extensiones para aumentar su funcionalidad y la interoperabilidad del estándar **DICOM**, entre otras.

4.5.3 Meshmixer

Al procurar generar un modelo a partir de una nube de puntos de un TAC, es común encontrarse con que los modelos exportados en STL contienen errores y no pueden ser impresos directamente. Se probaron varias herramientas para solventar estos errores en la estructura de los modelos 3D, pero finalmente **Meshmixer** resultó dar los mejores resultados a la hora de arreglar las geometrías con esta casuística tan específica.

4.5.4 Meshlab

Una alternativa de software libre para la reparación de mallas 3D es **Meshlab**, que ofrece herramientas similares para limpiar y corregir geometrías de modelos STL.

4.5.5 Blender

Blender es un potente software de modelado, animación, renderizado y edición 3D de código abierto y gratuito. Ofrece una suite completa de herramientas para la creación y manipulación de modelos 3D, incluyendo texturizado, iluminación, simulación física y composición. En este proyecto, se utilizó para procesar y refinar modelos 3D generados a partir de imágenes médicas, aprovechando su flexibilidad para trabajar con formatos como STL y OBJ, y su capacidad para integrar scripts en Python para automatizar tareas.

4.5.6 UltiMaker Cura

UltiMaker Cura es el programa desarrollado por UltiMaker para generar el GCODE necesario para imprimir modelos en una impresora de dicha marca. Se utilizó ya que permite importar ajustes específicos de la impresora sobre la que se trabajó de forma sencilla.

4.6 Frameworks

4.6.1 OpenXR

OpenXR es una [Interfaz de programación de aplicaciones \(API\)](#) multiplataforma que permite el desarrollo de medios en el *virtual continuum* mediante ordenadores a través de interacción humano-máquina. Esta [API](#) es la interfaz con un runtime para llevar a cabo operaciones comunes como puede ser acceder al estado de un mando o periférico, obtener o predecir la posición del sistema o enviar frames para ser renderizados.

Ciclo de Vida

En la Figura 4.1 se muestra la máquina de estados de la sesión:

1. La aplicación crea una sesión escogiendo un sistema y una API gráfica. En un primer momento esta se encuentra en estado IDLE.
2. Se monitorea la sesión en busca de cambios de estados mediante eventos.
3. Cuando el runtime determina que el sistema está listo para empezar con el contenido [XR](#) de la sesión, se recibe un cambio de estado a READY.
4. Mientras que la sesión está corriendo, se espera que la aplicación ejecute continuamente el frame loop, estableciendo así sincronización con el runtime, lo que provoca un cambio de estado a SYNCHRONIZED.

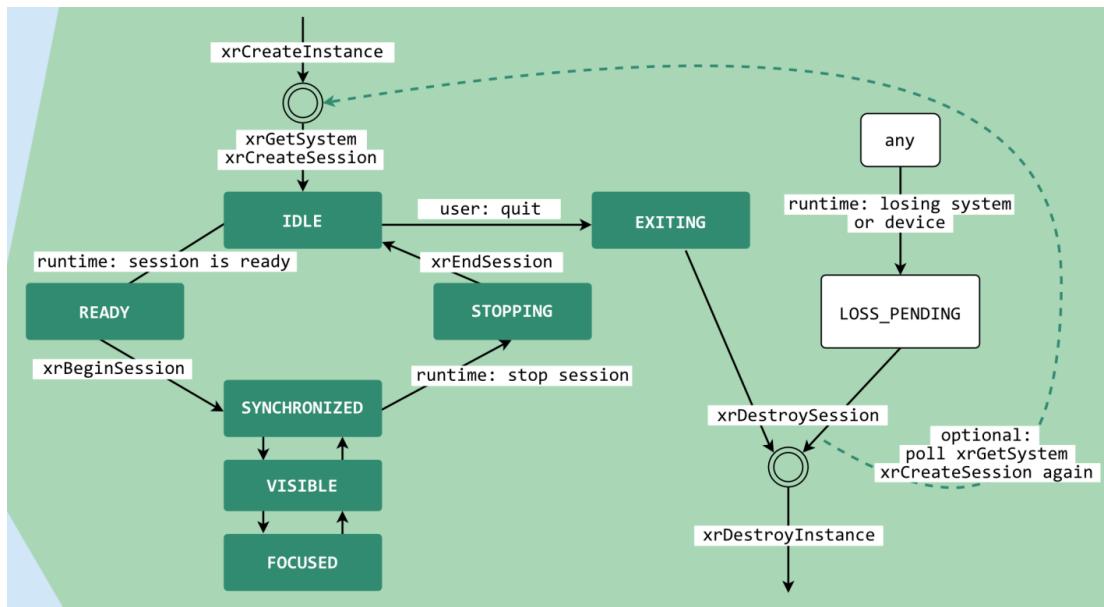


Figura 4.1: Ciclo de vida de OpenXR. (fuente: <https://www.khronos.org/>)

5. Una vez que el runtime esté listo para mostrar frames de la aplicación, se notifica con el estado **VISIBLE**.
6. Si el runtime detecta que es posible recibir entradas desde un mando, reconocimiento facial o demás, notifica con un estado **FOCUSSED**.
7. Estos estados, como se ve en la Figura 4.1, también tienen carácter retroactivo, de forma que cuando las características dejan de estar disponibles se va cambiando de estado, hasta que se desee parar o cerrar la aplicación.

API Layers

OpenXR está diseñado como una API por capas, lo que quiere decir que una aplicación puede insertar más o menos capas entre la aplicación y la implementación del runtime seleccionada. Estas capas proveen de funcionalidades adicionales interceptando las funciones de OpenXR de la capa superior, y posteriormente llevando a cabo operaciones distintas a las que se llevarían a cabo en caso de que no estuviese presente la capa. En el más sencillo de los casos una capa simplemente llama a la inferior con los mismos argumentos, pero en casos más elaborados se pueden implementar funcionalidades no disponibles en las capas o incluso runtime inferiores (Figura 4.2).

Esta arquitectura permite el desarrollo multiplataforma con mayor simplicidad, pero es dependiente de que los vendedores implementen sus propias capas API de OpenXR, lo que limita en cierta medida las posibilidades de desarrollo.

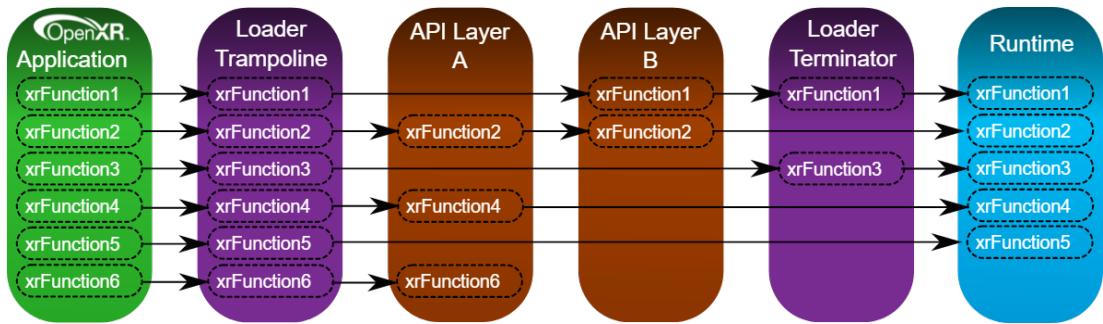


Figura 4.2: Explicación de la API por capas. (fuente: <https://registry.khronos.org>)

4.6.2 OpenCV

[Open Source Computer Vision Library \(OpenCV\)](#) es una librería de código abierto que implementa principalmente funciones de visión artificial en tiempo real. Se utilizó para la generación y el seguimiento de los marcadores ArUco que forma parte de los paquetes adicionales de la librería.

4.6.3 OpenSceneGraph

[OpenSceneGraph \(OSG\)](#) es una librería gráfica 3D de código abierto escrita en C++ que encapsula la funcionalidad de OpenGL mediante una arquitectura basada en grafos de escena. Esta estructura permite organizar eficientemente objetos 3D, transformaciones y materiales en aplicaciones de renderizado en tiempo real.

En este proyecto, [OSG](#) se utiliza para renderizar la interfaz de usuario en modo VR, proporcionando las herramientas necesarias para interactuar con la visualización en el entorno de realidad virtual.

4.6.4 Assimp

[Asset Import Library \(ASSIMP\)](#) es una librería portátil de código abierto diseñada para importar múltiples formatos de modelos 3D. La librería soporta decenas de formatos de archivo diferentes incluyendo FBX, OBJ, 3DS, DAE, entre otros. En este proyecto se utiliza para cargar modelos 3D anatómicos generados a partir de imágenes de [TC](#), proporcionando una interfaz consistente para acceder a geometrías, texturas y materiales independientemente del formato de archivo original.

4.6.5 GLFW

[Graphics Library Framework \(GLFW\)](#) es una librería multiplataforma de código abierto para desarrollo de aplicaciones OpenGL, OpenGL ES y Vulkan. Proporciona una [API](#) simple

para crear ventanas, contextos y superficies, recibir entradas y eventos. En este proyecto se utiliza para la gestión de ventanas de renderizado y la creación de contextos OpenGL necesarios para la visualización de modelos 3D en tiempo real.

4.6.6 GLEW

OpenGL Extension Wrangler ([GLEW](#)) es una librería multiplataforma de código abierto que ayuda en la consulta y carga de extensiones OpenGL. GLEW proporciona mecanismos para determinar qué extensiones OpenGL están soportadas en la plataforma objetivo, simplificando el uso de OpenGL. En este proyecto se utiliza para acceder a características de OpenGL necesarias para el renderizado de modelos 3D con shaders programables.

4.6.7 GLM

OpenGL Mathematics ([GLM](#)) es una librería de matemáticas C++ diseñada específicamente para gráficos por computador basada en las especificaciones OpenGL Shading Language (GLSL). Proporciona clases y funciones para operaciones con vectores, matrices y transformaciones 3D. En este proyecto se utiliza para realizar cálculos de transformaciones geométricas, proyecciones de cámara y manipulación de matrices necesarias para el correcto posicionamiento de modelos 3D en el espacio de realidad aumentada.

4.6.8 ArUco

Los últimos años, los desarrollos de nuevos marcadores han tendido a un cuadrado negro con distintos patrones interiores, como en el ejemplo de la Figura 4.4, ya que permiten extraer la pose de la cámara a partir de sus 4 esquinas, asumiendo que esta esté adecuadamente calibrada [2]. Esencialmente estos marcadores comparten ciertas características comunes en cuanto a su funcionamiento, entre todas las opciones disponibles se escogió ArUco como solución a nuestro proyecto por varios motivos:

- Diccionarios generados dinámicamente.
- Posibilidad de crear tablas de marcadores lo que incrementa la resistencia a las occlusiones.
- Software para la calibración de cámara: De forma sencilla se puede calibrar cualquier cámara.
- La librería ha soportado el paso de los años sin problema, existiendo ejemplos y documentación extensa sobre el funcionamiento de la misma, facilitando así el desarrollo.

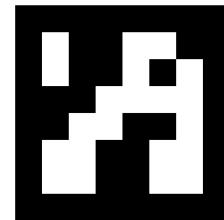


Figura 4.3: Marcador Generado con ArUco

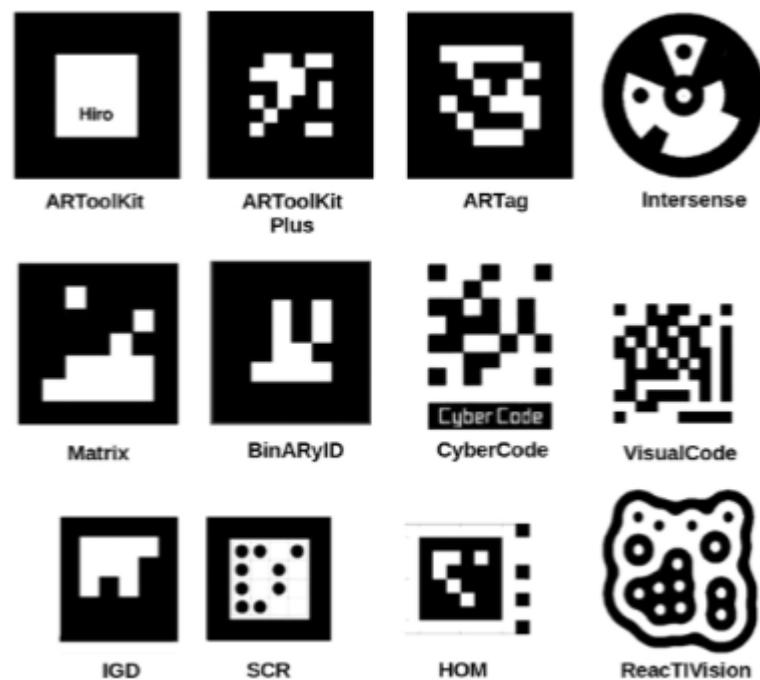


Figura 4.4: Ejemplos de marcadores fiduciarios previos (fuente: [2])

4.6.9 Funcionamiento de ArUco

Captura de imágenes o videos

La captura de imágenes o vídeos se realiza mediante un dispositivo específico, como una cámara digital o un dispositivo móvil con cámara incorporada. En este paso se espera obtener una serie de imágenes o un vídeo sobre el que se espera encontrar uno o varios marcadores ArUco.

Conversión a escala de grises

La conversión a escala de grises se realiza mediante el algoritmo de promedio ponderado de los canales RGB. Esta conversión reduce la cantidad de información a procesar y mejora la velocidad de procesamiento al trabajar con un único canal de información.

Aplicación de un filtro de bordes

Para resaltar los bordes de los marcadores en la imagen se aplica un filtro de bordes. Un ejemplo común es el algoritmo de Canny, que se basa en la detección de gradientes y utiliza un umbral para determinar qué bordes son relevantes y cuáles no.

Detección de contornos

Se utiliza un algoritmo de detección de contornos adaptativo para encontrar los contornos de los marcadores en la imagen. Un algoritmo común es el Transformada de Hough que permite detectar contornos circulares y lineales en la imagen. Este algoritmo busca patrones en la imagen que se correspondan con los contornos de los marcadores ArUco. En un sistema de thresholding tradicional, se elige un umbral global para toda la imagen. Cualquier píxel con un valor de brillo superior al umbral se considera activo (p.ej. negro) y cualquier píxel con un valor de brillo inferior al umbral se considera inactivo (p.ej. blanco). Sin embargo, en muchas imágenes, el nivel óptimo de umbral puede variar entre diferentes partes de la imagen. El thresholding adaptativo se utiliza para solucionar este problema.

El thresholding adaptativo se divide en dos pasos:

- Selección de una región de interés (ROI) en la imagen. Esta región puede ser de cualquier tamaño y forma.
- Selección del umbral para cada pixel dentro de la ROI. El umbral se calcula a partir de la distribución de los niveles de gris dentro de la ROI.

Existen varios métodos para calcular el umbral adaptativo, algunos de los más conocidos son:

- Método de media global
- Método de la desviación estándar
- Método de Otsu

Cada uno de estos métodos tiene sus propios pros y contras y en función de la aplicación y el tipo de imagen, se puede elegir uno u otro. Además es posible modificar los siguientes parámetros para adecuar la librería a nuestro caso de uso, los más importantes son:

- **markerBorderBits:** El número de bits que se utilizan para representar el borde de un marcador. El borde de un marcador es el área blanca que rodea el patrón de código de barras en un marcador ArUco. El valor predeterminado es 4.
- **adaptiveThreshWinSizeMin and adaptiveThreshWinSizeMax:** El tamaño mínimo y máximo de la ventana utilizada para la umbralización adaptativa. La umbralización adaptativa es un método para determinar automáticamente el valor de umbral óptimo para una imagen. Estos parámetros se utilizan para especificar el tamaño de la ventana en píxeles que se utilizará para la umbralización adaptativa.
- **adaptiveThreshWinSizeMax:** Especifica el paso o incremento con el cual se variará el tamaño de la ventana utilizada en la umbralización adaptativa.
- **adaptiveThreshConstant:** Especifica una constante que se utilizará en el cálculo del valor de umbral para cada subregión de la imagen.
- **minMarkerPerimeterRate and maxMarkerPerimeterRate:** El porcentaje mínimo y máximo del perímetro de un marcador en relación con su área. Estos parámetros se utilizan para especificar el tamaño mínimo y máximo de los marcadores que se detectarán en la imagen.
- **minCornerDistanceRate:** La relación entre la distancia entre las esquinas de un marcador y su longitud de lado. Esto es utilizado para ignorar marcadores que tengan esquinas muy cercanas entre sí.
- **minDistanceToBorder:** La distancia mínima desde el borde de la imagen hasta el borde de un marcador. Esto se utiliza para ignorar marcadores que estén demasiado cerca del borde de la imagen.
- **minMarkerDistanceRate:** La relación entre la distancia entre los marcadores y su longitud de lado. Esto se utiliza para ignorar marcadores que estén demasiado cerca entre sí.

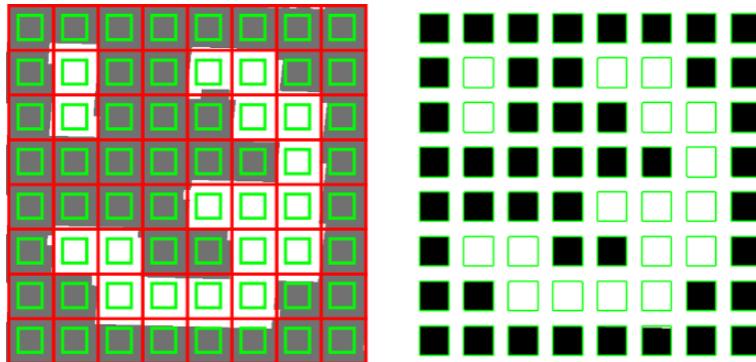


Figura 4.5: Proceso de extracción de Bits

Extracción de Bits

Los bits extraídos de cada candidato se analizan para determinar si corresponden a marcadores válidos. Para ello se somete cada sección de la imagen a una corrección de perspectiva. A continuación se subdivide el candidato en la cantidad previamente establecida en el diccionario de bits que componen cada marcador. La imagen corregida se binariza utilizando el [Método de Otsu](#) para separar píxeles blancos y negros. Posteriormente, la imagen se divide en una cuadrícula con el mismo número de celdas que bits tiene el marcador. En cada celda se cuenta el número de píxeles blancos y negros para determinar el valor del bit asignado a dicha celda (mediante el valor mayoritario), como en el ejemplo de la Figura 4.5.

Identificación de Marcadores

Se verifica si el código extraído pertenece al diccionario y, en caso necesario, se aplican algoritmos de corrección de errores. La primera operación consiste en determinar la cantidad de bits erróneos permitidos en el borde de un marcador, ya que todos los marcadores ArUco cuentan al menos con un bit de borde. En lo que a corrección de errores se refiere, cada diccionario cuenta con un límite teórico de bits que pueden ser corregidos.

Refinado de Esquinas

Se puede realizar un refinado a nivel de subpíxel de las esquinas para mejorar la precisión del sistema. Este último paso implica una alta carga computacional, pero se recomienda en aplicaciones en las que prima la precisión como es el caso.

Esta librería trabaja con un [Pinhole camera model](#) lo que quiere decir que se considera como el origen de coordenadas el punto en el que todos los rayos de luz convergerían en una supuesta cámara estenopeica ideal.

Las coordenadas se definen de la siguiente forma: Z crece frente a la cámara mientras que

X e Y se encuentran en el plano ortogonal de Z. X aumenta de izquierda a derecha e Y de abajo a arriba.

4.7 Hardware

4.7.1 Impresión 3D

Se utilizaron dos impresoras 3D a lo largo del proyecto puesto que eran necesarios distintos requisitos para cada pieza. Los modelos anatómicos debido a su complejidad se imprimieron en una impresora Fuse 1+ 30W que utiliza polvo de nylon para llevar a cabo las piezas. Para los marcadores fiduciarios, se utilizó la Ultimaker 3, ya que se trata de figuras más simples en las que la posibilidad de imprimir en distintos colores era especialmente importante.

4.7.2 Obtención de vídeo

Inicialmente se intentaron utilizar las cámaras frontales integradas del [HMD HTC Vive Pro 2](#) para esta función, pero debido a limitaciones técnicas en el acceso nativo a las cámaras (como se detalla en la Sección [6.6.1](#)), se optó por utilizar una cámara externa.

Logitech Brio 500

La Logitech Brio 500 es una cámara web de alta calidad que ofrece grabación en HD a 60 fps. Esta cámara fue utilizada para la captura de imágenes y video en el desarrollo del sistema de realidad aumentada, proporcionando la entrada visual necesaria para el seguimiento de marcadores ArUco y la calibración del sistema.

Capítulo 5

Metodología y Gestión del proyecto

Este capítulo se centra en la organización del proyecto, la metodología utilizada y todo lo que a gestión de proyecto se refiere.

5.1 Metodología

Para el desarrollo del proyecto se decidió implementar una metodología *Ágil*. Este término se refiere a una metodología regida por una serie de principios que permiten ajustar la forma de trabajo a las condiciones del proyecto. Se debe priorizar a los individuos e interacciones sobre procesos y herramientas, asegurando no perder el valor humano de la comunicación no verbal en el proceso. También se debe atender antes a soluciones funcionales sobre la documentación exhaustiva. Lejos de procurar dejar de lado la documentación, es necesario tener en cuenta que el valor del desarrollo está en la funcionalidad del mismo, por lo que sacrificar tiempo de trabajo por documentar en exceso un proyecto puede acarrear resultados no deseados.

Otro pilar en el que se basa esta metodología es la respuesta al cambio sobre los planes preestablecidos, lo que asegura la versatilidad del proyecto aumentando las posibilidades de éxito.

Dada la naturaleza incremental e iterativa de la metodología *Ágil*, es posible dividir el ciclo de vida del proyecto en tareas para facilitar el desarrollo. Esta aproximación permite aplicar el principio de «Divide y Vencerás» fragmentando la estimación de las tareas y su desarrollo con el fin de reducir la dificultad en el desarrollo.

El proyecto se organizó en 21 tareas principales, agrupadas en 6 sprints, con una estimación total de 584 horas de trabajo. Destacar que dada la naturaleza del trabajo de fin de grado, se asume un único recurso humano, y por lo tanto no nos referiremos a las horas como horas por hombre o h × h. La tabla detallada de tareas con las estimaciones específicas se puede consultar en el Apéndice B.

Para llevar a cabo las tareas se optó por un desarrollo en *sprints*. Los sprints son los períodos de tiempo en los que se llevan a cabo las tareas. Estos se acotaron en el tiempo, dándoles siempre una longitud determinada, y a su vez se acotaron en funcionalidad procurando siempre que finalicen alcanzando algún hito del proyecto. Estos bloques o *sprints* se fueron definiendo a lo largo del proyecto, adaptándose a las necesidades y riesgos del mismo.

5.2 Sprints

Tras introducir la metodología utilizada, vamos a detallar los sprints que tuvieron lugar en el proyecto, las tareas, los riesgos y recursos disponibles para cada uno.

5.2.1 Extracción de un modelo a partir de un TC

En este sprint, se tuvo como objetivo el desarrollo de las tareas 1, 2, 3 y 4. Dado que en el estudio previo se encontraron varios caminos disponibles para alcanzar las tareas objetivo, el sprint se consideró viable en todo momento. Este sprint se llevó a cabo dentro del tiempo estimado de 2 semanas. El principal riesgo fue la complejidad de la pieza, que en una impresora 3D de filamento común podría haber alargado la impresión, o incluso fallar en el proceso. Esto se minimizó utilizando la impresora Fuse 1+ 30W capaz de imprimir modelos más complejos a pesar de tener una superficie de impresión más reducida.

5.2.2 Diseño del marcador fiduciario y software de tracking

Durante este sprint, se llevaron a cabo las tareas de la 5 a la 10. En este caso el mayor riesgo fue el tiempo de prototipado, ya que para una única pieza a imprimir, el tiempo de impresión oscila alrededor de las 15 horas. Para minimizar este riesgo se optó por realizar prototipos sobre papel como se comenta en la ejecución del proyecto. El tiempo estimado para este sprint fue de 3 semanas y media, pero debido a la dificultad de las tareas 7 y 8 se alargó a 5 semanas.

5.2.3 Integrar solución de tracking en Exposure Render

Las tareas de la 11 a la 14 se llevaron a cabo en este sprint. La compilación e instalación de todo el software necesario para ejecutar Exposure Render fue el riesgo por excelencia de este sprint. A pesar de esto se realizó dentro del tiempo estimado para ello ya que se conocía en el momento de la planificación.

5.2.4 Integrar passthrough en Exposure Render

Se llevaron a cabo las tareas de la 15 a la 18. En este sprint, nos encontramos con el mayor imprevisto del proyecto. El **SDK** del **HMD** no funcionaba correctamente, lo que imposibilitaba

la reconstrucción de imágenes y causó un gran retraso en el proyecto respecto a la planificación inicial.

5.2.5 Integrar con motor de render independiente

En este sprint se ejecutaron las tareas 19 a 21, centradas en el desarrollo de una alternativa a Exposure Render. Después de varios intentos fallidos de integrar el sistema de seguimiento en Exposure Render durante varias semanas, se decidió descartar esta opción debido a incompatibilidades técnicas y limitaciones en la integración. En su lugar, se diseñó e implementó una solución utilizando un motor de render independiente basado en OpenGL. Se llevaron a cabo pruebas exhaustivas para validar el funcionamiento del sistema y se ajustaron los parámetros necesarios para asegurar una ejecución óptima y estable.

5.3 Planificación temporal

La planificación temporal del proyecto se organizó en sprints de duración variable, adaptándose a la complejidad de las tareas y los riesgos identificados. El diagrama de Gantt completo con la distribución temporal detallada de todas las tareas se puede consultar en el Apéndice B.

Capítulo 6

Ejecución del proyecto

Este capítulo tiene como objetivo tratar el desarrollo del proyecto en sí mismo, así como discutir las opciones disponibles durante el progreso y las decisiones tomadas para llevarlo a cabo.

6.1 Consideraciones previas

Este traLa función `getCubePoseMatrix` implementa una serie de transformaciones para que la pose del cubo detectada por ArUco sea directamente utilizable por el motor de renderizado de Exposure Render. El proceso incluye:

1. **Construcción de la matriz de transformación:** Conversión de vectores de rotación y traslación de ArUco a matriz de transformación 4×4 que representa la pose del marcador respecto a la cámara.
2. **Inversión para obtener la matriz modelo:** La matriz construida representa la transformación del cubo hacia la cámara (view matrix). Se invierte para obtener la matriz modelo del cubo en el espacio de coordenadas de la cámara, necesaria para el posicionamiento de objetos virtuales.
3. **Conversión de sistemas de coordenadas:** Compatibilización entre el orden row-major de OpenCV y column-major de OSG mediante transposición matricial.

ce como un desarrollo del proyecto troncal de [Iglesias-Guitian et al.](#) y como tal se debe ceñir a las condiciones que acarrea dicho proyecto. Todo el equipo utilizado durante el desarrollo fue provisto por parte del mismo, o en su defecto por parte del [Centro de Investigación en Tecnologías de la información y Comunicaciones \(CITIC\)](#).

6.2 Análisis

Se llevó a cabo un estudio para definir la hoja de ruta del proyecto. Dada la problemática a solventar, este trabajo alcanza a tocar áreas bien diferenciadas entre si que se pueden destacar como los pasos a seguir del mismo:

- Extracción de volúmenes 3D a partir de un [TC](#) válidos para su impresión.
- Diseñar un marcador fiduciario que permita el seguimiento de una pieza en 3 dimensiones y un método de acople al volumen previamente impreso.
- Implementar una solución que permita el seguimiento de dicho marcador.
- Integrar la solución sobre un [HMD](#).

Fruto de la investigación surge el artículo de [Moreta-Martinez et al.](#), que expone una solución existente a los objetivos de este trabajo mediante el uso de software bajo licencia o de pago. Por este motivo, se adoptó una aproximación similar al problema, especialmente en las fases iniciales relacionadas con la generación de volúmenes, aunque se desarrolló una solución propia para el sistema de seguimiento. Este estudio también permitió especificar los requisitos necesarios para el software de tracking.

Uno de los principales requisitos debe ser la robustez del sistema frente a las oclusiones del marcador. Es necesario que el seguimiento sea posible a pesar de la oclusión parcial del marcador. Además, es preciso que a partir de una fuente de vídeo se puedan extraer las coordenadas del marcador, así como su rotación en el espacio. Destacar también que el seguimiento debe ocurrir en un segundo plano, entorpeciendo lo menos posible las operaciones del hilo principal de ejecución, ya que parte de estas operaciones tiene una latencia crítica.

Exposure Render cuenta con un módulo de realidad virtual, en el que se integrará la solución de tracking para lograr un control más natural sobre el modelo a tratar, moviéndose en las imágenes renderizadas a la par que se mueve en la realidad, como se muestra en el diagrama de secuencia de la Figura 6.1.

6.3 Generación de volúmenes a partir de TC

Con el fin de facilitar la validación del progreso del proyecto, se utilizó una [TC](#) de pruebas. Dichos datos contienen la sección superior (hombros y cabeza) de un sujeto, mostrado en la Figura 6.2a. Durante el desarrollo se sugirió como posible caso práctico seleccionar el cráneo del sujeto en los datos de prueba y trabajar en la alineación 3D sobre el mismo.

Con el fin de seleccionar una sección concreta para exportar, se utilizaron las herramientas para segmentar volúmenes de 3D Slicer. Al abrir el programa se pueden ver las vistas, en

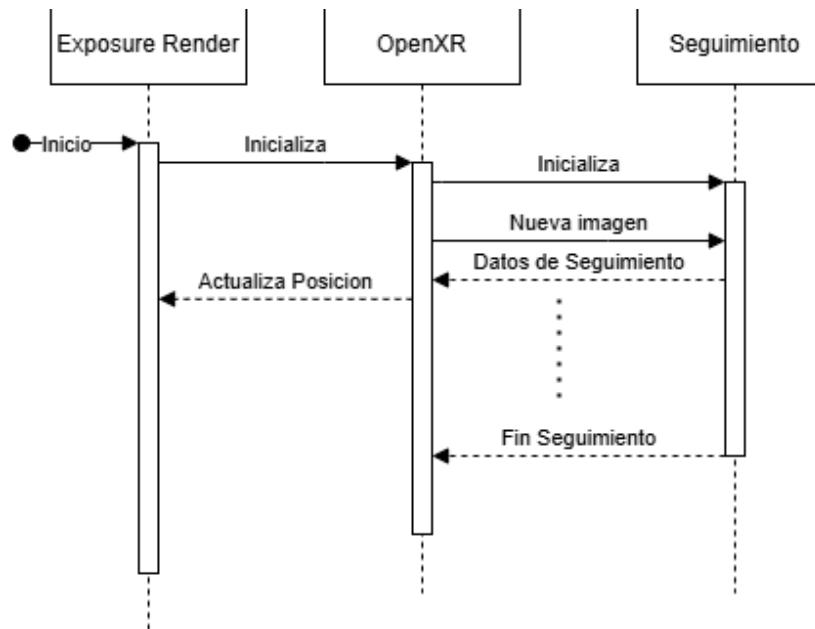


Figura 6.1: Flujo de integración del seguimiento en Exposure Render.

las que se representará el TC una vez cargado, como se muestra en la captura de pantalla de la Figura 6.2b. Se utilizó principalmente la herramienta de «Thresholding» que permite seleccionar partes del modelo cuyas intensidades se comprenden en un intervalo o «threshold» (ver Figura 6.2c). Posteriormente, para la eliminación de las partes del modelo no deseadas, se utilizó la herramienta de borrado hasta alcanzar el volumen deseado.

6.4 Impresión 3D del volumen

Durante el proceso de segmentación y exportación del modelo, pueden surgir inconsistencias geométricas debido a que el modelo se genera directamente de los datos volumétricos del TC, en lugar de construirse a partir de primitivas geométricas predefinidas. Estas irregularidades, que incluyen superficies no continuas o vértices mal conectados, impedirían un correcto proceso de Slicing y, por consiguiente, la generación adecuada del archivo GCODE necesario para la impresión 3D. Para solucionar este problema, se utilizó el software Meshmixer, una herramienta especializada en la reparación y optimización de mallas 3D. Como se muestra en la Figura 6.3, el modelo exportado presenta varios puntos problemáticos, señalados por marcadores, que corresponden a errores geométricos derivados del proceso de exportación. Una vez corregidos estos defectos mediante las herramientas de reparación de Meshmixer, el modelo quedó preparado para su impresión.

Como se comenta en el Capítulo 4, para una pieza con una geometría tan compleja, se



Figura 6.2: TC de partida y obtención del modelo 3D final con 3D Slicer.

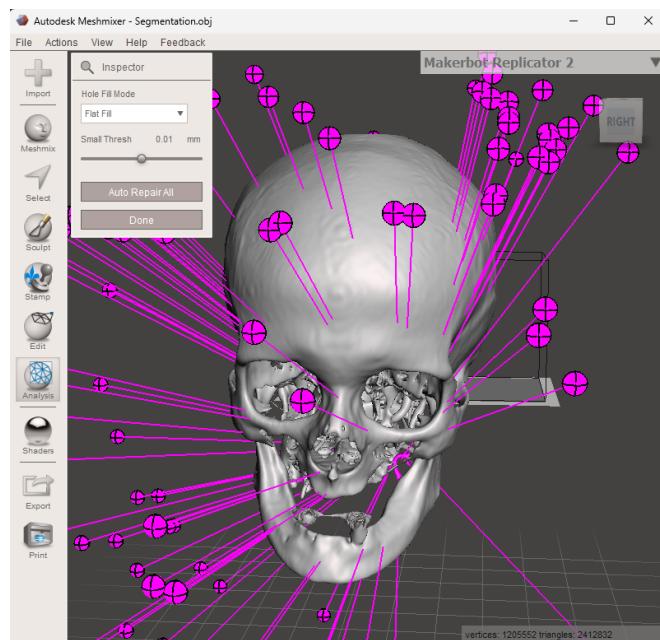


Figura 6.3: Modelo de cráneo 3D con geometrías erróneas.



(a) Pieza en el proceso de recuperación del material.

(b) Pieza final.

Figura 6.4: Proceso de recuperación de material.

requeriría una gran cantidad de soportes. Debido a esto, se optó por la impresora Fuse 1 para la impresión de este modelo. A diferencia de una impresora 3D al uso, esta impresora utiliza un láser para fijar capa a capa el polvo de nylon, lo que garantiza una gran resolución en la pieza final y una gran durabilidad de la misma.

Posterior al proceso de impresión es necesario retirar el material sobrante en la cámara de recuperación que cuenta con distintos utensilios para evitar malgastar el material sobrante ya que puede ser reutilizado.

6.5 Desarrollo del marcador fiduciario

6.5.1 Planteamiento del problema y selección de la solución

Obtener la posición y rotación de una figura desconocida en el espacio es uno de los problemas principales a la hora de implementar soluciones de realidad virtual o aumentada, ya que requiere encontrar correspondencias entre objetos conocidos en el espacio y sus proyec-

ciones en el vídeo.

Si bien existen aproximaciones que buscan puntos claves de las figuras o reconocen sus geometrías mediante técnicas de visión artificial e inteligencia artificial, se optó por el uso de marcadores fiduciarios por varios motivos:

- **Independencia del hardware:** Permite replicar el seguimiento del objeto independientemente del hardware utilizado, ya que una vez calibrada la cámara no se requiere ningún otro tipo de ajuste en el sistema.
- **Robustez frente a occlusiones:** Permite mantener el seguimiento a pesar de que parte del marcador se encuentre ocluido o no esté completamente en el campo de visión de la cámara.
- **Simplicidad de implementación:** Dados los recursos disponibles, se optó por utilizar la librería ArUco para generar y seguir el marcador, aprovechando su madurez y estabilidad.

6.5.2 Fundamentos teóricos del seguimiento con ArUco

ArUco a la hora de detectar la posición de un marcador, trabaja con un modelo de coordenadas pin-hole, donde las coordenadas y rotación de los objetos detectados se expresan en función de la posición de la cámara. El calibrado de la cámara permite determinar la proyección de cualquier punto en las 3 dimensiones del espacio en el sensor de la cámara. En una cámara ideal, un punto 3D (X, Y, Z) en el espacio se proyectaría en el píxel:

$$x_{ideal} = \frac{X \cdot fx}{Z} + cx \quad y_{ideal} = \frac{Y \cdot fy}{Z} + cy$$

Sin embargo, las lentes reales introducen distorsiones que deben corregirse. Las coordenadas finales del píxel se calculan aplicando las correcciones de distorsión radial y tangencial:

$$\begin{aligned} x_{distorsionada} &= x_{ideal} \cdot (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_1 x_{ideal} y_{ideal} + p_2 (r^2 + 2x_{ideal}^2) \\ y_{distorsionada} &= y_{ideal} \cdot (1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + p_1 (r^2 + 2y_{ideal}^2) + 2p_2 x_{ideal} y_{ideal} \end{aligned}$$

donde $r^2 = x_{ideal}^2 + y_{ideal}^2$ es la distancia radial al centro de la imagen.

Donde:

- fx, fy : Es la longitud focal de la lente de la cámara en ambos ejes (píxeles).
- cx, cy : Es el centro óptico del sensor (expresado en píxeles).
- $k1, k2, k3$: Son los coeficientes de distorsión radial que corrigen la curvatura de la lente.

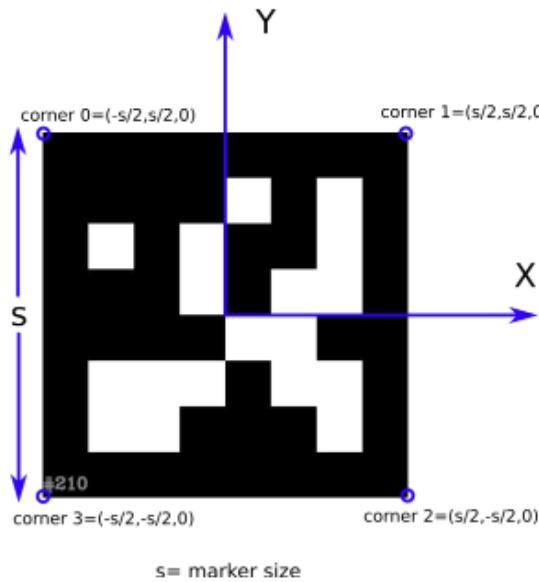


Figura 6.5: Esquema de un marcador.

- p_1, p_2 : Son los coeficientes de distorsión tangencial que corrigen la desalineación de la lente.

Asumiendo que la ubicación tridimensional del punto con respecto al sistema de referencia de la cámara es conocida. Si se desea conocer la proyección de un punto referido a un sistema de referencia arbitrario, entonces deben mencionarse parámetros extrínsecos. Los parámetros extrínsecos consisten básicamente en las rotaciones tridimensionales ($Rvec = Rx, Ry, Rz$) y las traslaciones tridimensionales ($Tvec = Tx, Ty, Tz$) requeridas para trasladar el sistema de referencia de la cámara al sistema arbitrario. Los elementos de rotación se expresan mediante la fórmula de Rodrigues [19], por lo que es posible obtener la matriz de rotación equivalente de 3×3 utilizando la función `cv::Rodrigues()` de OpenCV.

Cada marcador detectado devuelve como coordenadas la esquina superior izquierda del mismo, o lo que se etiqueta en el ejemplo de la Figura 6.5 como *corner 0*, en forma de ($Rvec = Rx, Ry, Rz$) como vector de rotación y ($Tvec = Tx, Ty, Tz$) como vector de translación.

Detectar un solo marcador puede fallar por diferentes razones, como malas condiciones de iluminación, movimiento rápido de la cámara, obstrucciones, etc. Para superar ese problema, ArUco permite el uso de tablas de marcadores como la mostrada en la Figura 6.6. Cada tabla de marcadores está compuesta por varios marcadores en ubicaciones conocidas. Presentan dos ventajas principales. Primero, dado que hay más de un marcador, es menos probable perderlos todos de vista al mismo tiempo. Segundo, cuanto más marcadores se detecten, más puntos están disponibles para calcular los parámetros extrínsecos de la cámara. Como consecuencia, se obtiene una mayor precisión.

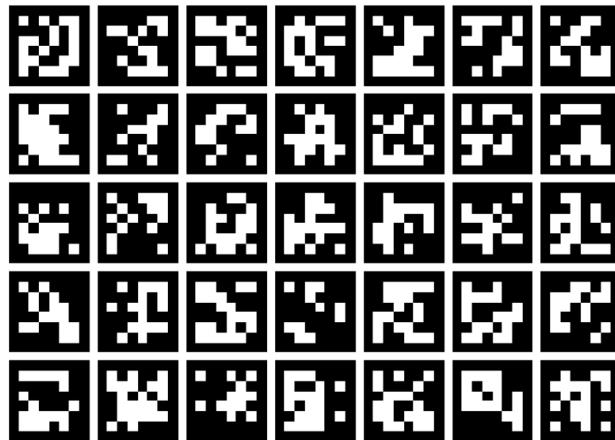


Figura 6.6: Esquema de una tabla de marcadores.

6.5.3 Generación de marcadores a partir de diccionarios predefinidos

El marcador fiduciario utilizado en este proyecto se genera a partir de un diccionario predefinido de ArUco. En la implementación actual se emplea el diccionario DICT_6X6_250, que contiene 250 marcadores únicos de 6×6 bits cada uno. Este diccionario fue seleccionado por ofrecer un buen equilibrio entre:

- **Robustez en la detección:** Los marcadores de 6×6 bits proporcionan suficiente información para una detección fiable incluso con cierta degradación de la imagen.
- **Capacidad del diccionario:** Con 250 marcadores únicos disponibles, hay suficientes IDs para implementar múltiples cubos sin conflictos.
- **Replicabilidad:** Al usar un diccionario estándar, los experimentos son fácilmente reproducibles.

La generación del cubo marcador sigue un esquema sistemático donde cada cara del cubo contiene una matriz 2×2 de marcadores del mismo diccionario. Los identificadores (IDs) de los marcadores se asignan secuencialmente siguiendo la fórmula:

```
firstId = cara_id * 4
ids = {firstId, firstId + 1, firstId + 2, firstId + 3}
```

Donde `cara_id` corresponde al número de cara del cubo (0-5). De esta forma:

- Cara 0: marcadores con IDs 0, 1, 2, 3
- Cara 1: marcadores con IDs 4, 5, 6, 7

- Cara 2: marcadores con IDs 8, 9, 10, 11
- Cara 3: marcadores con IDs 12, 13, 14, 15
- Cara 4: marcadores con IDs 16, 17, 18, 19
- Cara 5: marcadores con IDs 20, 21, 22, 23

Esta implementación permite utilizar cualquier diccionario predefinido o personalizado de ArUco, siempre y cuando el diccionario contenga al menos 24 marcadores únicos y se respete el orden secuencial de asignación de IDs para cada cara. La modularidad del sistema permite cambiar el diccionario sin necesidad de alterar la lógica de seguimiento del cubo.

6.5.4 Algoritmo de cálculo de pose del cubo

El proceso de determinación de la pose del cubo se realiza en varias etapas que transforman las detecciones individuales de cada cara en una pose unificada del centro del cubo.

Transformación de coordenadas de cara a centro

Cuando ArUco detecta una cara del cubo, devuelve la posición y orientación correspondientes a la esquina superior izquierda de esa cara (corner 0 en la Figura 6.5). Para obtener la pose del centro del cubo, es necesario aplicar una transformación que tenga en cuenta tanto la posición como la orientación actual del marcador.

6.5.5 Diseño geométrico del marcador cúbico

Debido a la versatilidad de las piezas con las que se pretende usar el marcador, se implementó priorizando eliminar las occlusiones del marcador por la pieza. Por este motivo se diseñó como un cubo, de forma que al menos una cara sería visible en todo momento. Se evaluaron otras formas geométricas (prismas rectangulares, pirámides, cilindros), pero estas demostraron menor efectividad de detección con la cámara utilizada en el desarrollo.

El diseño final consiste en un cubo donde cada cara contiene una matriz 2×2 de marcadores del mismo diccionario ArUco. Esta configuración proporciona las siguientes ventajas:

- **Redundancia:** Al tener múltiples marcadores por cara, el sistema puede mantener el tracking incluso si algunos marcadores están ocluidos.
- **Precisión mejorada:** La detección de múltiples marcadores en la misma cara permite calcular la pose con mayor precisión mediante promediado.
- **Robustez:** La forma cúbica garantiza que siempre habrá al menos una cara visible desde cualquier ángulo de observación.

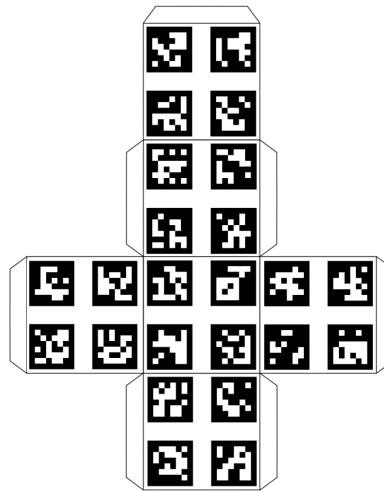


Figura 6.7: Layout de las caras del marcador fiduciario.

Se llevaron a cabo pruebas con distintos diccionarios de marcadores, modificando las tolerancias para los márgenes entre marcadores y los bordes del cubo con el fin de poder mantener unas dimensiones manejables sin comprometer el tamaño de cada marcador individual. Finalmente, fruto de los tests, se llegó al cubo de la Figura 6.8.

6.5.6 Transformación de coordenadas de cara a centro del cubo

El cubo se descompone como se puede ver en la Figura 6.7. En un primer momento la detección devuelve la posición y rotación de cada cara de forma individual. Dado que se busca la posición del centro del cubo, sobre cada valor obtenido, se aplica una transformación que tiene en cuenta tanto la posición como la orientación actual del marcador.

Para el cálculo del desplazamiento necesario para cada eje se utiliza la matriz de rotación (obtenida mediante la fórmula de Rodrigues) para asegurar que el desplazamiento se realiza en la dirección correcta respecto a la orientación actual del marcador. Este proceso se aplica para los tres ejes mediante la función `moveAxis`, cuya implementación detallada se encuentra en el Apéndice A.

Una vez ajustada la posición del centro, es necesario corregir la orientación de cada cara para mantener un sistema de coordenadas consistente en todo el cubo. Para ello, la función `cubeCoordinates` aplica las rotaciones necesarias según la cara detectada, usando la cara 0 como referencia. Los detalles completos del código fuente de estas transformaciones de coordenadas están disponibles en el mismo apéndice.

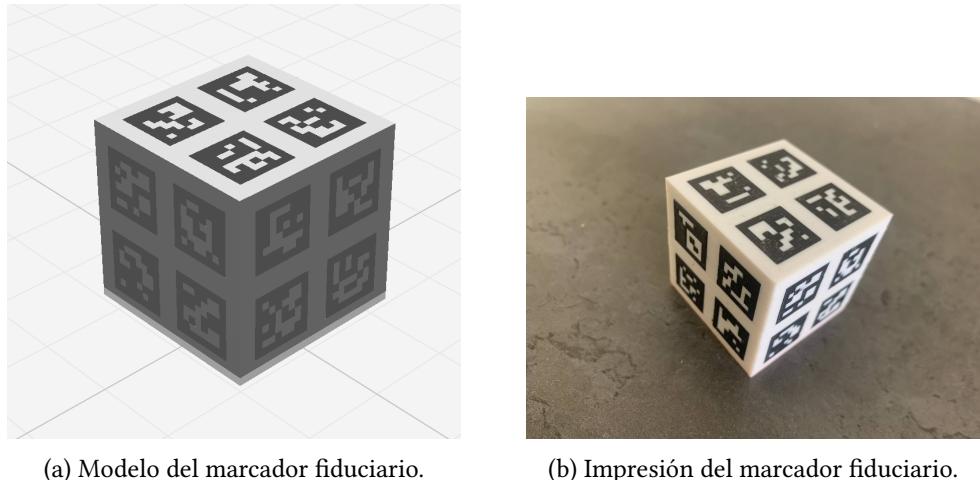


Figura 6.8: Diseño final del marcador fiduciario.

Fusión de múltiples detecciones

Cuando se detectan múltiples caras del cubo simultáneamente (hasta 3 caras pueden ser visibles en una sola imagen), el sistema debe combinar las estimaciones individuales para obtener una pose más precisa y estable. El algoritmo implementado utiliza un promediado simple de los vectores de traslación y rotación.

El promediado de vectores de rotación requiere especial atención debido a la naturaleza circular de las rotaciones y posibles discontinuidades en la representación de Rodrigues. El sistema implementa verificaciones para evitar promediar rotaciones que difieran significativamente, lo que podría indicar detecciones erróneas o ambigüedades en la orientación.

En caso de que únicamente se detectase una cara del cubo, los vectores de rotación y translación finales serían los obtenidos llegados a este punto. Como se observa en la Figura 6.9, es posible detectar hasta 3 caras simultáneamente en una única imagen, lo que permite aplicar el algoritmo de fusión descrito anteriormente para obtener una estimación más robusta de la pose.

Procesamiento de secuencias de vídeo

Hasta ahora, se ha descrito el comportamiento de la aplicación para una única imagen. El procesamiento de secuencias de vídeo consiste en aplicar iterativamente este algoritmo a cada frame de la secuencia. Una vez obtenidos los valores de pose para un frame, el sistema espera al siguiente frame y ejecuta nuevamente el pipeline de detección.

El diagrama muestra claramente las etapas principales del proceso implementado:

- 1. Inicialización del sistema:** Apertura de la cámara, carga de parámetros de calibración, creación de las tablas de marcadores para cada cara del cubo e inicialización del detector

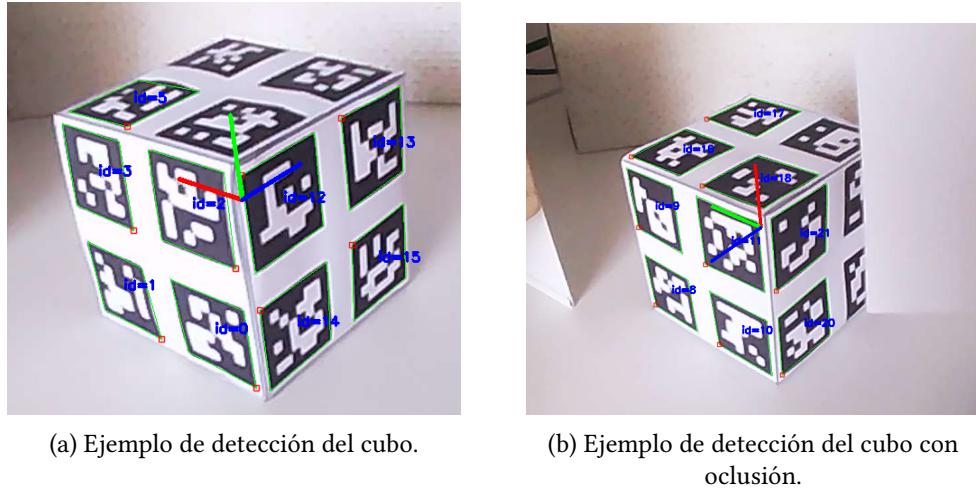


Figura 6.9: Imágenes sobre las que se estima la pose del cubo.

ArUco.

2. **Captura de frame:** Obtención del frame de vídeo desde la cámara.
3. **Detección de marcadores:** Identificación inicial de marcadores ArUco en la imagen.
4. **Refinamiento de detección:** Mejora de la precisión de detección aplicando el refinamiento específico para cada tabla de marcadores del cubo.
5. **Procesamiento por cara:** Para cada una de las seis caras del cubo, se calculan las correspondencias entre puntos de imagen y puntos del objeto 3D.
6. **Cálculo de pose individual:** Estimación de pose mediante solvePnP para cada cara que tenga correspondencias válidas.
7. **Transformación al centro:** Conversión de las coordenadas de cada cara al centro del cubo utilizando las transformaciones geométricas correspondientes.
8. **Promediado de poses:** Combinación de todas las poses válidas obtenidas para generar una estimación final más robusta.
9. **Comunicación de resultados:** Retorno de la pose final calculada o pose vacía si no se detectaron marcadores.

El diagrama de flujo completo del algoritmo, desde la captura de imagen hasta la obtención de la pose final del cubo, se puede consultar en el Apéndice B.

6.6 Implementación del Passthrough en Exposure Render

6.6.1 Captura de imagen

En un primer momento para la obtención de imágenes sobre las que poder realizar el seguimiento del marcador fiduciario se pretendían usar las propias cámaras frontales del HTC Vive Pro 2. Se trata de un par de cámaras colocadas longitudinalmente a lo largo del frontal del casco, que permiten su uso en aplicaciones de realidad aumentada y realidad mixta. Se trató de acceder a las mismas mediante el uso de SRworks C++ SDK y con librerías de terceros a pesar de no tener éxito. Posteriormente tras conversaciones con el equipo de soporte se concluyó que no era posible el acceso a las cámaras de forma nativa. Solo a través de Unreal Engine o Unity en su defecto, lo que nos hizo descartar esta posibilidad. Cabe destacar que durante el desarrollo del proyecto HTC retiró el soporte de la extensión de passthrough en OpenXR, por lo que versiones posteriores del runtime dejaron de anunciar esta funcionalidad a través de la API de extensiones de OpenXR. La alternativa era acceder a las imágenes a través de una cámara de terceros. Esta opción presentaba la desventaja de imposibilitar el tratamiento de la imagen directamente en GPU, pero era la única viable dentro del contexto del TFG.

6.6.2 Integración del seguimiento

Con el fin de no aumentar la cantidad de dependencias de un proyecto de la envergadura de Exposure Render, se trató de implementar la solución de tracking como una librería independiente que se pudiese integrar en el proyecto.

La integración del sistema de tracking ArUco en Exposure Render se realizó como una librería independiente que proporciona la funcionalidad de seguimiento de marcadores fiduciarios. El punto central de esta integración es la función `getCubePoseMatrix`, que actúa como interfaz principal entre el sistema de tracking y la aplicación de realidad aumentada.

Función `getCubePoseMatrix`

La función `getCubePoseMatrix` es la interfaz principal que proporciona la pose del cubo marcador en el espacio tridimensional. Los parámetros de esta función incluyen configuraciones para visualización, dimensiones físicas del marcador, rutas de calibración y directorios de configuración.

Los detalles completos de la signatura de esta función y sus parámetros se pueden consultar en el Apéndice A.

Implementación como hilo independiente

Para minimizar la latencia y mantener la fluidez del renderizado principal, el sistema de tracking se implementó como un hilo independiente mediante la clase `Tracking` que encapsula toda la funcionalidad de seguimiento.

El flujo de ejecución del tracking opera en un hilo separado (`poseThread`) que ejecuta continuamente el siguiente proceso:

1. **Captura de pose del headset:** Obtiene la matriz de transformación actual del casco VR desde el sistema OpenXR.
2. **Captura de imagen:** Adquiere una nueva imagen desde la cámara del sistema.
3. **Procesamiento de marcadores:** Ejecuta la detección y seguimiento de marcadores ArUco llamando a `getCubePoseMatrix`.
4. **Sincronización de datos:** Actualiza de forma thread-safe la pose más reciente del cubo y la imagen procesada.
5. **Actualización del overlay:** Envía la imagen procesada al sistema de overlay para visualización en realidad aumentada.

La gestión de concurrencia utiliza mecanismos de sincronización para garantizar la integridad de los datos compartidos:

- `std::mutex`: Protege el acceso a las variables `_latestPose` y `_latestCameraImage`.
- `std::atomic<bool>`: Controla el estado de ejecución del hilo mediante `_poseThreadRunning`.
- **Lock guards**: Garantizan el acceso exclusivo durante las operaciones de lectura y escritura de datos compartidos.

Esta arquitectura asegura que el hilo principal de renderizado no se vea bloqueado por las operaciones de procesamiento de imagen, manteniendo una alta tasa de refresco en la experiencia de realidad aumentada mientras proporciona datos de tracking actualizados de forma continua.

Estructura de datos `PoseMatrix4x4` y transformaciones de coordenadas

Para facilitar la integración con Exposure Render, la librería de tracking utiliza la estructura `PoseMatrix4x4`, que representa una matriz de 4×4 . Esta estructura encapsula tanto la rotación como la traslación del marcador en el espacio 3D.

La función `getCubePoseMatrix` implementa una serie de transformaciones para que la pose del cubo detectada por ArUco sea directamente utilizable por el motor de renderizado de Exposure Render. El proceso incluye:

1. **Construcción de la matriz de transformación:** Conversión de vectores de rotación y traslación de ArUco a matriz de transformación 4×4 que representa la pose del marcador en el sistema de coordenadas de la cámara.
2. **Inversión para obtener la transformación del mundo al marcador:** La matriz de ArUco representa la transformación del marcador a la cámara, por lo que se invierte para obtener la transformación que posiciona objetos virtuales respecto al marcador detectado.
3. **Conversión de sistemas de coordenadas:** Compatibilización entre el orden row-major de OpenCV y column-major de OSG.

Los detalles matemáticos completos de estas transformaciones, incluyendo el código de implementación y las ecuaciones específicas, se pueden consultar en el Apéndice A.

Transformación al World space Finalmente, la pose del cubo se transforma desde el espacio de coordenadas de la cámara al **World space** del entorno VR. Esta transformación se realiza mediante la multiplicación de la matriz del cubo por la matriz de transformación del headset VR, que se obtiene del sistema OpenXR:

```
1 cv::Mat headsetMat = poseMatrixToCvMat(headsetPose);
2 finalTransform = headsetMat * cubeModelTransposed;
```

La transformación completa se puede expresar matemáticamente como una cadena de operaciones matriciales:

$$\mathbf{T}_{world} = \mathbf{H} \cdot (\mathbf{T}_{ArUco}^{-1})^T$$

Donde:

- \mathbf{T}_{world} : Matriz de transformación final del cubo en el **World space**
- \mathbf{H} : Matriz de transformación del headset VR en el **World space** (obtenida de OpenXR)
- \mathbf{T}_{ArUco} : Matriz de transformación vista del cubo detectada por ArUco (de cubo a cámara)
- $(\mathbf{T}_{ArUco}^{-1})^T$: Matriz transpuesta de la inversa de \mathbf{T}_{ArUco} (modelo del cubo en espacio de cámara)

Esta cadena de transformaciones convierte la pose detectada por ArUco desde el espacio de coordenadas de la cámara al [World space](#) compatible con Exposure Render.

Este enfoque permite que el objeto virtual renderizado por Exposure Render se mantenga perfectamente alineado con el marcador físico en el mundo real, independientemente de los movimientos del usuario y la cámara. La secuencia completa de transformaciones asegura que las convenciones de coordenadas de ArUco, OpenCV, [OSG](#) y OpenXR sean compatibles entre sí, proporcionando una experiencia de realidad aumentada coherente y precisa.

6.6.3 Consideraciones de implementación alternativa

Durante el desarrollo del proyecto, surgieron limitaciones técnicas en la integración completa con Exposure Render que condujeron a la exploración de enfoques alternativos de visualización. Estas limitaciones incluyeron incompatibilidades organizativas en cuanto al acceso al [HMD](#) y complejidades en la transformación entre los sistemas de [Tracking](#) y renderizado, entre otros.

Como resultado de estas consideraciones técnicas, se desarrolló una implementación de renderizado independiente que permitió validar la funcionalidad del sistema de seguimiento de marcadores y demostrar la viabilidad del enfoque propuesto. Esta solución alternativa mantuvo los principios fundamentales de seguimiento en tiempo real, proporcionando una base sólida para futuras integraciones con sistemas de renderizado más complejos.

6.7 Aplicación Independiente de Realidad Aumentada

Con el fin de demostrar la viabilidad del sistema de [Tracking](#) desarrollado y proporcionar una herramienta práctica para la visualización de modelos anatómicos sobre marcadores fiduciarios, se implementó una aplicación independiente de realidad aumentada. Esta aplicación combina las capacidades de seguimiento ArUco desarrolladas anteriormente con un sistema de renderizado 3D que permite cargar y visualizar modelos anatómicos de diversas fuentes.

6.7.1 Arquitectura de la aplicación

La aplicación se estructura en varios módulos funcionales que operan de manera coordinada:

- **Módulo de captura:** Gestiona la adquisición de imágenes desde la cámara web y la inicialización de los parámetros de calibración.
- **Módulo de tracking:** Utiliza la librería de seguimiento ArUco desarrollada para detectar y seguir el marcador cúbico en tiempo real.

- **Módulo de renderizado:** Implementa el sistema de visualización 3D utilizando [GLFW](#), [GLEW](#) y [GLM](#) para el renderizado con OpenGL.
- **Módulo de carga de modelos:** Integra [ASSIMP](#) para soportar múltiples formatos de archivo 3D (FBX, OBJ, STL, etc.).

El punto de entrada principal de la aplicación coordina la ejecución de todos los módulos mediante hilos independientes para garantizar un rendimiento óptimo.

6.7.2 Sistema de renderizado basado en OpenGL

El sistema de renderizado implementado utiliza OpenGL con shaders programables para proporcionar capacidades de visualización 3D avanzadas.

El renderizador OpenGL proporciona las siguientes características:

- **Modelo de iluminación:** Implementa un [Modelo de iluminación Phong](#) con componentes ambiente, difusa y especular.
- **Soporte para texturas:** Permite aplicar texturas a los modelos cuando están disponibles en el archivo del modelo.
- **Renderizado fuera de pantalla:** Utiliza framebuffers para renderizar la escena en una textura que posteriormente se compone con la imagen de la cámara.
- **Compatibilidad con múltiples formatos:** Mediante [ASSIMP](#), soporta formatos como FBX, OBJ, 3DS, COLLADA, etc.

El pipeline de renderizado se basa en dos tipos de shaders principales que operan en diferentes etapas del proceso de renderizado:

Vertex Shader

El vertex shader se encarga de procesar cada vértice individual del modelo 3D. Su función principal es transformar las coordenadas de los vértices desde el espacio del modelo al espacio de pantalla mediante la aplicación de las matrices de transformación (modelo, vista y proyección). Además, prepara la información necesaria para el fragment shader, incluyendo las posiciones transformadas en el espacio mundial, las normales corregidas para la iluminación y las coordenadas de textura.

Fragment Shader

El fragment shader opera sobre cada píxel de las superficies renderizadas e implementa el [Modelo de iluminación Phong](#) para calcular la iluminación final. Este shader combina tres componentes de iluminación: la luz ambiente (que proporciona una iluminación base uniforme), la luz difusa (que simula la [Reflexión lambertiana](#) dependiente del ángulo de incidencia) y la luz especular (que genera los brillos característicos de las superficies). El resultado final se combina con las texturas del modelo cuando están disponibles, proporcionando una apariencia adecuada.

6.7.3 Sistema de carga de modelos

La aplicación implementa un sistema de carga de modelos basado en la librería [ASSIMP](#), que permite trabajar con múltiples formatos de archivo 3D de manera unificada. El sistema aprovecha las capacidades de [ASSIMP](#) para procesar automáticamente la geometría del modelo y convertirla en estructuras de datos optimizadas para el renderizado en tiempo real.

El proceso de carga se estructura en tres etapas principales. En primer lugar, se inicializa el importador de [ASSIMP](#) con flags de post-procesamiento específicos que garantizan la compatibilidad con el pipeline de renderizado:

```

1 Assimp::Importer importer;
2 const aiScene* scene = importer.ReadFile(filename,
3     aiProcess_Triangulate | aiProcess_GenNormals |
4     aiProcess_FlipUVs);

```

Estos flags aseguran que todos los polígonos del modelo estén triangulados, que las normales se generen automáticamente si no están presentes en el archivo original, y que las coordenadas de textura estén orientadas correctamente para el sistema de coordenadas de OpenGL.

En la segunda etapa, se valida la estructura del archivo cargado y se extrae la primera malla disponible en la escena. [ASSIMP](#) organiza los modelos en una jerarquía de nodos que pueden contener múltiples mallas, pero para simplificar la implementación se procesa únicamente la primera malla encontrada.

Finalmente, se iteran los vértices de la malla para extraer las posiciones, normales y coordenadas de textura, almacenándolas en vectores que posteriormente serán transferidos a la GPU:

```

1 for (unsigned int i = 0; i < ai_mesh->mNumVertices; ++i) {
2     mesh.vertices.push_back(ai_mesh->mVertices[i]);
3     mesh.normals.push_back(ai_mesh->mNormals[i]);
4     mesh.texCoords.push_back(ai_mesh->mTextureCoords[0][i]);
5 }

```

Este enfoque permite cargar modelos anatómicos complejos procedentes de software de segmentación médica como 3D Slicer, así como modelos creados en aplicaciones de modelado 3D estándar, proporcionando flexibilidad en las fuentes de datos sin comprometer el rendimiento del sistema de renderizado.

6.7.4 Flujo de ejecución de la aplicación

El flujo principal de la aplicación coordina todos los componentes del sistema:

1. **Inicialización de rutas:** Se establecen las rutas a los archivos de calibración, marcadores y modelos 3D.
2. **Creación del hilo de captura:** Se inicia un hilo independiente para el [Tracking](#) de marcadores mediante `captureThreadFunc`.
3. **Inicialización del renderizador:** Se crea e inicializa el sistema de renderizado (OpenGL o OpenCV según la configuración).
4. **Bucle principal de renderizado:** Se ejecuta el bucle principal que combina las imágenes de la cámara con los modelos 3D renderizados.

```

1 int main()
2 {
3     // Configuración de parámetros del proyecto
4     std::string calibrationPath = getCalibrationPath();
5     std::string markersPath = getMarkersPath();
6     std::string modelsPath = getModelsPath();
7
8     // Inicialización del hilo de tracking
9     std::thread captureThread(captureThreadFunc,
10         true, // showVisualization
11         markerSideLength,
12         markerGapLength,
13         calibrationPath,
14         markersPath);
15
16     // Inicialización del renderizador
17     if (!initializeRenderer(cameraResolution, modelsPath)) {
18         std::cerr << "Error: Failed to initialize renderer" <<
19         std::endl;
20         return -1;
21     }
22
23     // Bucle principal de renderizado

```

```
23     while (!shouldClose()) {
24         Mat renderedFrame = renderFrame(pose, cameraImage);
25
26         if (!renderedFrame.empty()) {
27             displayFrame(renderedFrame);
28         }
29
30         processEvents();
31     }
32
33     return 0;
34 }
```

Esta arquitectura modular permite que la aplicación funcione como una herramienta independiente y versátil para la visualización de modelos 3D en realidad aumentada. El diseño implementado demuestra la efectividad del sistema de [Tracking](#) desarrollado y proporciona una base sólida para futuras extensiones y mejoras. La separación en módulos independientes facilita la integración con otros sistemas de renderizado y permite adaptar la solución a diferentes casos de uso manteniendo la robustez del seguimiento de marcadores.

Capítulo 7

Conclusiones y trabajo futuro

Como cierre del trabajo, cabe reflexionar sobre el desarrollo del mismo así como sobre su estado actual, y su futuro. Dado por finalizado el trabajo, se han alcanzado buena parte de los objetivos fijados en su concepción:

- Se identificaron métodos para la extracción de secciones de [TC](#) así como para el refinado de las mismas. También se analizaron las posibilidades de impresión optimizando el uso de los materiales y de las capacidades de las impresoras. Esto ha permitido trabajar sobre piezas en un nivel más visual e interactivo.
- Se ha llevado a cabo un estudio de las soluciones existentes para el seguimiento 3D realizando pruebas de las mismas sobre los datos propios para extraer las ventajas y desventajas de cada solución en un nivel práctico.
- Se han diseñado e impreso una gran cantidad de marcadores fiduciarios sobre los que se han realizado pruebas de forma iterativa con el fin de refinar y optimizar el diseño hasta obtener un resultado que satisface las características del proyecto. Todo esto mediante el uso de software libre creando así un proceso adaptable a otros casos de uso.
- Se desarrolló un sistema robusto de seguimiento de marcadores cúbicos basado en ArUco que permite detectar hasta 3 caras simultáneamente, proporcionando mayor precisión y resistencia a occlusiones parciales.
- Se implementó un algoritmo de transformación de coordenadas que permite convertir las poses detectadas por ArUco en matrices de transformación compatibles con sistemas de renderizado 3D, resolviendo las incompatibilidades entre diferentes convenciones de coordenadas.
- Se creó una aplicación independiente de realidad aumentada que demuestra la viabilidad del sistema de tracking desarrollado, integrando capacidades de carga de modelos 3D mediante [ASSIMP](#) y renderizado con iluminación [Modelo de iluminación Phong](#).

- Se estableció una arquitectura modular que permite la integración del sistema de tracking como librería independiente, facilitando su incorporación en proyectos existentes como Exposure Render.
- Se ha integrado parcialmente la solución en el proyecto de Iglesias-Guitian et al., sentando las bases para una integración completa que posibilite casos de uso médicos reales.

7.0.1 Enriquecimiento Formativo

Es imprescindible destacar la vertiente formativa que presenta este trabajo dada su naturaleza como trabajo de fin de grado. El autor ha tenido la posibilidad de trabajar sobre una serie de campos de lo más variados que comprenden la imagen médica, la impresión 3D, la visión artificial, la realidad virtual y el renderizado en un motor de trazado de rayos, entre otros. Por otra parte, mencionar la exposición a un ambiente investigador en el CITIC del que poder empaparse de la forma de trabajar y la cooperación entre iguales. También se trató de una primera puesta en práctica de los conceptos aprendidos sobre la gestión de proyectos que resultó enriquecedora.

7.0.2 Trabajo futuro

En la actualidad el proyecto tiene distintas vertientes que pueden ser desarrolladas en el futuro:

Integración completa con Exposure Render

Una de las líneas de trabajo más prometedoras consiste en completar de forma exitosa la integración del sistema de tracking con Exposure Render. Aunque se han establecido las bases arquitectónicas y se ha desarrollado la librería de tracking como módulo independiente, quedan aspectos por resolver relacionados con el acceso nativo a las cámaras del HTC Vive Pro 2 y la optimización del pipeline de transformación de coordenadas. La consecución de esta integración permitiría aprovechar las capacidades avanzadas de renderizado volumétrico de Exposure Render para visualizar modelos anatómicos extraídos de TC con calidad fotorrealista en tiempo real.

Desarrollo de benchmarks para evaluación de marcadores

Sería valioso desarrollar un sistema de benchmarking que permita evaluar objetivamente la efectividad de distintos tipos de marcadores fiduciarios en diferentes condiciones. Este benchmark incluiría métricas como precisión de detección, robustez frente a occlusiones, estabilidad del tracking en movimiento y rendimiento bajo diferentes condiciones de iluminación.

Los resultados permitirían optimizar el diseño de marcadores para casos de uso específicos y proporcionarían una base científica para futuras investigaciones en el campo.

Implementación de filtrado de Kalman

Si bien el sistema actual de seguimiento es completamente funcional, en sistemas donde la tasa de refresco de la cámara no es suficientemente alta, el seguimiento puede dar una sensación de escalonado. Una posible solución sería implementar un filtro de Kalman que, bien configurado, ayudaría a estimar pasos intermedios entre imágenes y a reducir el ruido en las mediciones de pose. Esto proporcionaría una experiencia más fluida y permitiría mantener el seguimiento incluso en frames donde el marcador no sea detectado temporalmente [20].

Mejoras en realidad aumentada mediante fusión de cámaras

Una segunda vertiente de mejora consistiría en reconstruir sobre las imágenes extraídas del casco un ambiente de realidad aumentada más robusto. Esto incluiría utilizar las imágenes de ambas cámaras del HTC Vive Pro 2 para crear una imagen compuesta estereoscópica que permita una mejor detección del marcador fiduciario y una experiencia de realidad aumentada más inmersiva. Aprovechando los desarrollos de la interfaz de realidad virtual del proyecto de Iglesias-Guitian et al., se podría trabajar con la pieza virtual modificando sus parámetros visuales en tiempo real.

Extensión a múltiples marcadores

Finalmente, otra línea de desarrollo interesante sería la extensión del sistema para soportar el seguimiento simultáneo de múltiples marcadores cúbicos independientes. Esto permitiría el seguimiento de varios objetos anatómicos simultáneamente o la implementación de sistemas de referencia más complejos que mejoren la precisión general del seguimiento mediante triangulación entre marcadores.

Apéndices

Apéndice A

Detalles de Implementación

EN este apéndice se incluyen los detalles técnicos de implementación del sistema de tracking, incluyendo el código fuente completo de las funciones principales y los algoritmos de transformación de coordenadas.

A.1 Implementación del algoritmo de tracking

A.1.1 Función moveAxis

La función `moveAxis` permite desplazar la pose del marcador a lo largo de un eje específico del sistema de coordenadas del cubo:

```
1 cv::Vec3d moveAxis(cv::Vec3d tvec, cv::Vec3d rvec, double distance,
2   int axis)
3 {
4   cv::Mat rotationMatrix;
5   cv::Mat rotationMatrixTransposed;
6   Rodrigues(rvec, rotationMatrix);
7   rotationMatrixTransposed = rotationMatrix.t();
8   double* rz = rotationMatrixTransposed.ptr<double>(axis); //  
x=0, y=1, z=2
9   tvec[0] -= rz[0] * distance;
10  tvec[1] -= rz[1] * distance;
11  tvec[2] -= rz[2] * distance;
12  return tvec;
```

A.1.2 Aplicación de las transformaciones al centro del cubo

El siguiente código muestra cómo se aplican las transformaciones para mover la pose desde la esquina del marcador hasta el centro del cubo:

```

1 tvecs = moveAxis(tvecs, rvecs, -SIDELENGTH, 0);
2 tvecs = moveAxis(tvecs, rvecs, -SIDELENGTH, 1);
3 tvecs = moveAxis(tvecs, rvecs, -SIDELENGTH, 2);

```

A.1.3 Función cubeCoordinates

La función `cubeCoordinates` aplica las rotaciones necesarias según la cara detectada, usando la cara 0 como referencia:

```

1 void cubeCoordinates(int id, cv::Vec3d& rvecs, cv::Vec3d& tvecs,
                      float sideLength)
2 {
3     tvecs = moveAxis(tvecs, rvecs, -SIDELENGTH, 0);
4     tvecs = moveAxis(tvecs, rvecs, -SIDELENGTH, 1);
5     tvecs = moveAxis(tvecs, rvecs, -SIDELENGTH, 2);
6     switch (id)
7     {
8         case 1://cara 1
9             rvecs = rotateXAxis(rvecs, -M_PI / 2);
10            break;
11        case 2://cara 2
12            rvecs = rotateXAxis(rvecs, M_PI);
13            break;
14        case 3://cara 3
15            rvecs = rotateYAxis(rvecs, M_PI / 2);
16            rvecs = rotateZAxis(rvecs, M_PI);
17            break;
18        case 4://cara 4
19            rvecs = rotateXAxis(rvecs, M_PI);
20            rvecs = rotateYAxis(rvecs, -M_PI / 2);
21            break;
22        case 5://cara 5
23            rvecs = rotateXAxis(rvecs, M_PI / 2);
24            break;
25        default://La cara 0 no precisa rotar
26            break;
27    }
28}

```

A.2 Transformaciones de coordenadas entre sistemas

A.2.1 Conversión del espacio de cámara al espacio del cubo

ARuco proporciona la pose del marcador como vectores de rotación y traslación que se convierten en una matriz de transformación 4×4 :

```
1 cv::Mat cubeTransform = buildTransformation(rvecFinal, tvecFinal);
```

A.2.2 Inversión para obtener la matriz modelo

Para obtener la pose del cubo en el espacio de coordenadas de la cámara:

```
1 cv::Mat cubeModel1;
2 cv::invert(cubeTransform, cubeModel1);
```

A.2.3 Conversión entre sistemas row-major y column-major

Para compatibilizar OpenCV (row-major) con OSG (column-major):

```
1 cv::Mat cubeModel1Transposed;
2 cv::transpose(cubeModel1, cubeModel1Transposed);
```

A.3 Interfaz principal de tracking

A.3.1 Función getCubePoseMatrix

La función principal que proporciona la pose del cubo marcador:

```
1 PoseMatrix4x4 getCubePoseMatrix(
2     bool showVisualization,
3     double markerSideLength,
4     double markerGapLength,
5     const std::string& calibrationFilePath,
6     const std::string& boardDirPath,
7     ImageData& undistortedImage);
```

Esta función encapsula toda la funcionalidad de tracking y actúa como interfaz principal entre el sistema de seguimiento y la aplicación de realidad aumentada.

Apéndice B

Diagramas y Gráficos Detallados

En este apéndice se incluyen los diagramas detallados y gráficos extensos que complementan el desarrollo del proyecto.

B.1 Diagrama de flujo completo del sistema de tracking

B.2 Planificación temporal detallada

B.3 Tabla detallada de tareas del proyecto

APÉNDICE B. DIAGRAMAS Y GRÁFICOS DETALLADOS *B.3. Tabla detallada de tareas del proyecto*

Nº	Funcionalidad	Est. (H)	Sprint
1	Importar datos en formato DICOM en 3D Slicer	2	1
2	Generar secciones para poder exportar de 3DSlicer	40	1
3	Arreglar el modelo para que sea viable para impresión	12	1
4	Imprimir modelo	15	1
5	Instalar librerías necesarias para el desarrollo	4	2
6	Familiarizarse con la librería	12	2
7	Prototipar pruebas para tests	40	2
8	Implementar herramienta para tracking del marcador	40	2
9	Testear prototipos	25	2
10	Generar modelo a partir del prototipo final	33	2
11	Imprimir prototipo final	16	3
12	Instalar y compilar Exposure Render	80	3
13	Familiarizarse con Exposure Render	20	3
14	Implementar captura de imagen	30	4
15	Integrar sistema de tracking	40	4
16	Validar sistema integrado	15	4
17	Diseñar motor de render alternativo	25	5
18	Implementar solución OpenGL	60	5
19	Validar solución independiente	20	5
20	Redacción de memoria	80	6
21	Revisión final del proyecto	15	6
Total		584	

Tabla B.1: Tabla completa de tareas del proyecto con estimaciones detalladas y asignación a sprints.

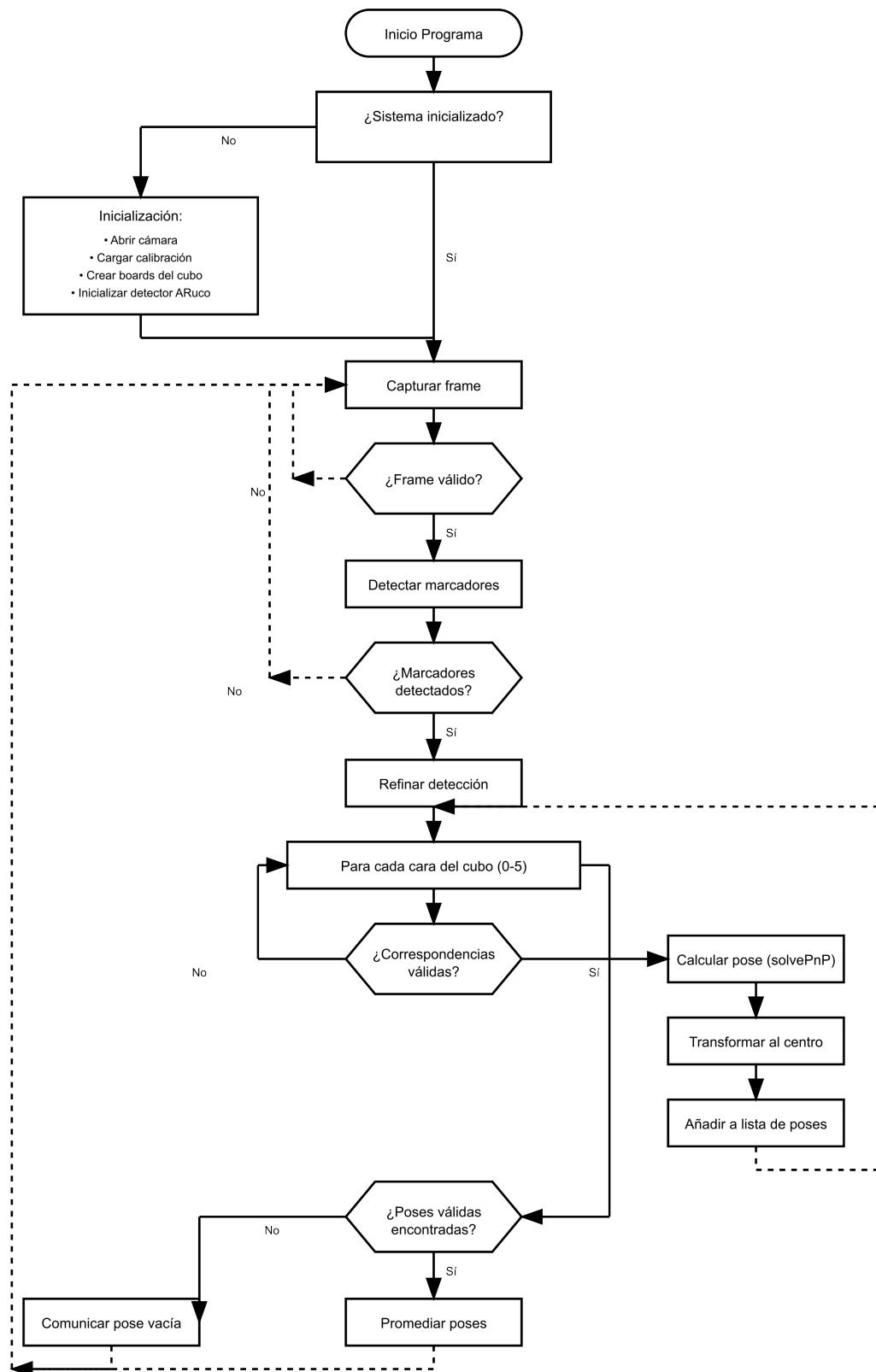


Figura B.1: Diagrama de flujo detallado del sistema de tracking implementado.

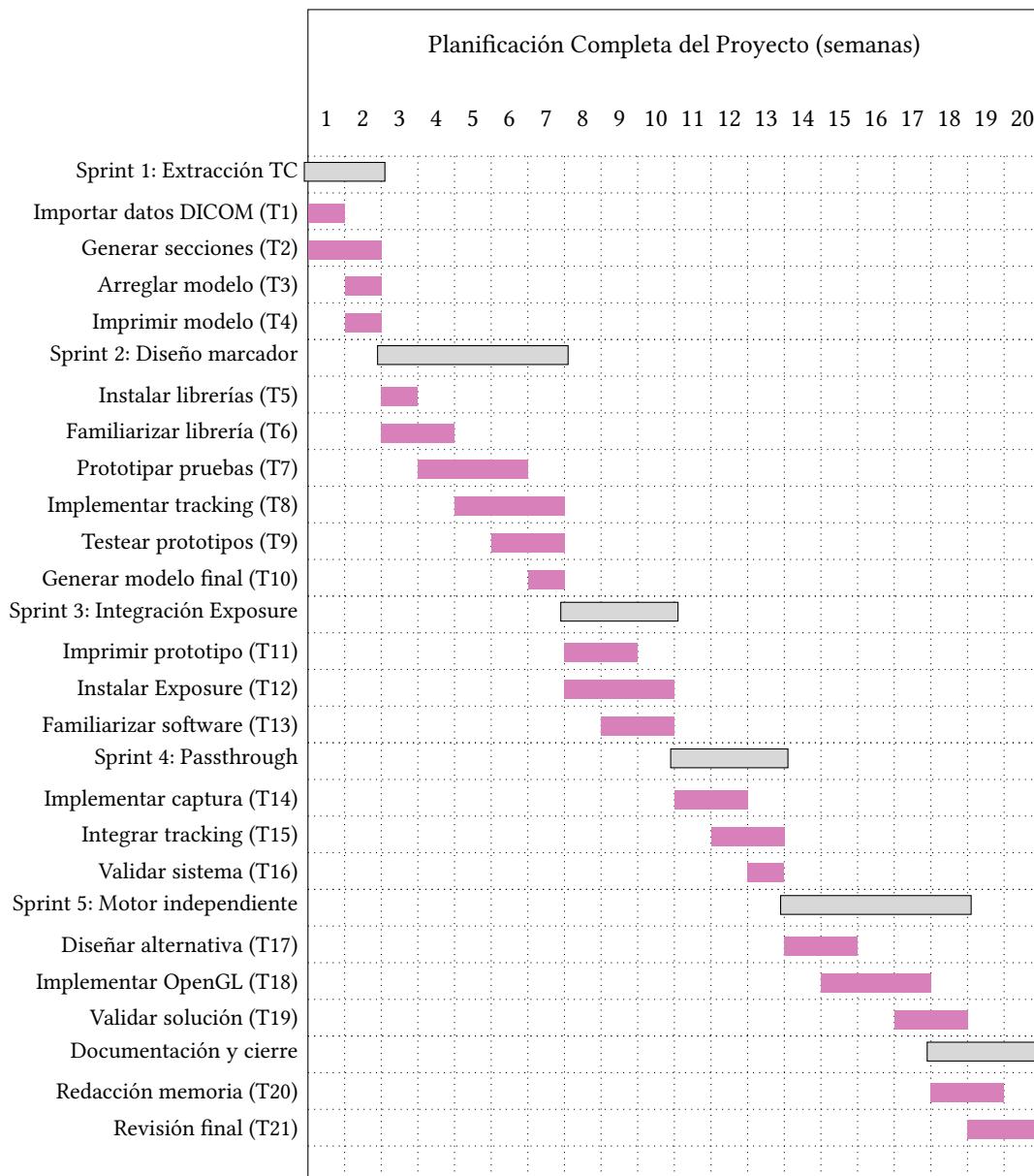


Figura B.2: Diagrama de Gantt completo del proyecto mostrando todos los sprints, tareas y dependencias temporales.

Lista de acrónimos

API Interfaz de programación de aplicaciones. 14–16

AR Realidad Aumentada. 1, 2, 7

ASSIMP Asset Import Library. 16, 42, 43, 46

CAD Computer-Aided Design. 6, 11

CGLS Mínimos Cuadrados Conjugados. 5

CITIC Centro de Investigación en Tecnologías de la información y Comunicaciones. 26, 47

DICOM Digital Imaging and Communication In Medicine. 13

DVR Direct Volume Rendering. 9

FBP Retroproyección Filtrada. 5

GLEW OpenGL Extension Wrangler. 17, 42

GLFW Graphics Library Framework. 16, 42

GLM OpenGL Mathematics. 17, 42

GPU Graphical Processing Unit. 9

HMD Head Mounted Display. 2, 10, 22, 24, 27, 41

MCRT Monte Carlo Ray Tracing. 9

MLEM Máxima Verosimilitud de la Expectativa-Maximización. 5

MR Realidad Mixta. 7

OpenCV Open Source Computer Vision Library. 16

OSG OpenSceneGraph. 16, 41

RLS Recursive Least Squares. 9

SDK Software Development Kit. 24

TC Tomografía Computerizada. ii, iv, 2, 4–6, 9, 10, 16, 24, 27–29, 46, 47

VR Realidad Virtual. 7

XR Realidad Extendida. 1, 7, 14

Glosario

Modelo de iluminación Phong Modelo empírico desarrollado por Bui Tuong Phong que calcula la iluminación de una superficie combinando tres componentes: luz ambiente, luz difusa y luz especular. Es ampliamente utilizado en gráficos por computador por su simplicidad y resultados visualmente aceptables. [42](#), [43](#), [46](#)

Método de Otsu Algoritmo de umbralización automática desarrollado por Nobuyuki Otsu en 1979 que determina automáticamente el valor de umbral óptimo para binarizar una imagen basándose en la minimización de la varianza intraclasa de los píxeles. [21](#)

Pinhole camera model Descripción de la relación matemática entre las coordenadas de un punto en el espacio de tres dimensiones y su proyección en el plano de imagen de una supuesta cámara estenopeica ideal. [21](#)

Reflexión lambertiana Modelo de reflexión difusa ideal desarrollado por Johann Heinrich Lambert [21] en el que la superficie refleja la luz uniformemente en todas las direcciones, con la intensidad de la luz reflejada proporcional al coseno del ángulo entre la dirección de la luz incidente y la normal de la superficie (ley del coseno de Lambert). [43](#)

Tracking Proceso de seguimiento en tiempo real de la posición y orientación de objetos o marcadores en aplicaciones de visión por computador. [41](#), [44](#), [45](#)

World space Espacio de coordenadas mundial o global que define el sistema de coordenadas absoluto del entorno 3D, en el cual se posicionan todos los objetos de la escena. [40](#), [41](#)

Bibliografía

- [1] M. Venkatesan, H. Mohan, J. R. Ryan, C. M. Schürch, G. P. Nolan, D. H. Frakes, and A. F. Coskun, “Virtual and augmented reality for biomedical applications,” *Cell Reports Medicine*, vol. 2, no. 7, p. 100348, Jul. 2021. [En línea]. Disponible en: <https://doi.org/10.1016/j.xcrm.2021.100348>
- [2] S. Garrido-Jurado, R. Muñoz-Salinas, F. Madrid-Cuevas, and M. Marín-Jiménez, “Automatic generation and detection of highly reliable fiducial markers under occlusion,” *Pattern Recognition*, vol. 47, no. 6, pp. 2280–2292, Jun. 2014. [En línea]. Disponible en: <https://doi.org/10.1016/j.patcog.2014.01.005>
- [3] C. P. Botha, B. Preim, A. E. Kaufman, S. Takahashi, and A. Ynnerman, “From individual to population: Challenges in medical visualization,” in *Mathematics and Visualization*. Springer London, 2014, pp. 265–282. [En línea]. Disponible en: https://doi.org/10.1007/978-1-4471-6497-5_23
- [4] T. Sielhorst, M. Feuerstein, and N. Navab, “Advanced medical displays: A literature review of augmented reality,” *Journal of Display Technology*, vol. 4, no. 4, pp. 451–467, Dec. 2008. [En línea]. Disponible en: <https://doi.org/10.1109/jdt.2008.2001575>
- [5] S. H. Muñiz and M. M. Casanovas, “Introducción a la tomografía computarizada,” *Revista Española de Medicina Nuclear*, vol. 25, no. 3, pp. 206–214, 2006.
- [6] G. Kontaxakis, J. J. Vaquero López, and A. Santos, “Reconstrucción de imagen en tomografía por emisión de positrones,” *Revista de la Real Academia de las Ciencias Exactas, Físicas y Naturales*, 2002.
- [7] M. J. Willemink, P. A. de Jong, T. Leiner, L. M. de Heer, R. A. J. Nievelstein, R. P. J. Budde, and A. M. R. Schilham, “Iterative reconstruction techniques for computed tomography part 1: Technical principles,” *European Radiology*, vol. 23, no. 6, pp. 1623–1631, Jan. 2013. [En línea]. Disponible en: <https://doi.org/10.1007/s00330-012-2765-y>

- [8] M. Zahera, “La fabricación aditiva, tecnología avanzada para el diseño y el desarrollo de productos,” *Economía Industrial*, no. 386, pp. 71–76, 2012.
- [9] P. Milgram and F. Kishino, “A taxonomy of mixed reality visual displays,” *IEICE Transactions on Information and Systems*, vol. 77, pp. 1321–1329, 1994.
- [10] R. T. Azuma, “A survey of augmented reality,” *Presence: Teleoperators and Virtual Environments*, vol. 6, no. 4, pp. 355–385, Aug. 1997. [En línea]. Disponible en: <https://doi.org/10.1162/pres.1997.6.4.355>
- [11] T. Kroes, F. H. Post, and C. P. Botha, “Exposure render: An interactive photo-realistic volume rendering framework,” *PLoS ONE*, vol. 7, no. 7, p. e38586, Jul. 2012. [En línea]. Disponible en: <https://doi.org/10.1371/journal.pone.0038586>
- [12] N. Max, “Optical models for direct volume rendering,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 1, no. 2, pp. 99–108, Jun. 1995. [En línea]. Disponible en: <https://doi.org/10.1109/2945.468400>
- [13] J. Díaz, T. Ropinski, I. Navazo, E. Gobbetti, and P.-P. Vázquez, “An experimental study on the effects of shading in 3d perception of volumetric models,” *The Visual Computer*, vol. 33, no. 1, pp. 47–61, Sep. 2015. [En línea]. Disponible en: <https://doi.org/10.1007/s00371-015-1151-6>
- [14] R. Englund and T. Ropinski, “Evaluating the perception of semi-transparent structures in direct volume rendering techniques,” in *SIGGRAPH ASIA 2016 Symposium on Visualization*. ACM, Nov. 2016. [En línea]. Disponible en: <https://doi.org/10.1145/3002151.3002164>
- [15] F. Lindemann and T. Ropinski, “About the influence of illumination models on image comprehension in direct volume rendering,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 1922–1931, Dec. 2011. [En línea]. Disponible en: <https://doi.org/10.1109/tvcg.2011.161>
- [16] J. A. Iglesias-Guitian, P. Mane, and B. Moon, “Real-time denoising of volumetric path tracing for direct volume rendering,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 28, no. 7, pp. 2734–2747, Jul. 2022. [En línea]. Disponible en: <https://doi.org/10.1109/tvcg.2020.3037680>
- [17] R. Moreta-Martinez, D. García-Mato, M. García-Sevilla, R. Pérez-Mañanes, J. A. Calvo-Haro, and J. Pascau, “Combining augmented reality and 3d printing to display patient models on a smartphone,” *Journal of Visualized Experiments*, no. 155, Jan. 2020. [En línea]. Disponible en: <https://doi.org/10.3791/60618>

- [18] A. G. Alvarez, P. L. Evans, L. Dovgalski, and I. Goldsmith, “Design, additive manufacture and clinical application of a patient-specific titanium implant to anatomically reconstruct a large chest wall defect,” *Rapid Prototyping Journal*, vol. 27, no. 2, pp. 304–310, jan 2021. [En línea]. Disponible en: <https://doi.org/10.1108%2Frpj-08-2019-0208>
- [19] J. E. Mebius, “Derivation of the euler-rodrigues formula for three-dimensional rotations from the general formula for four-dimensional rotations,” 2007.
- [20] G. F. Welch, “Kalman filter,” *Computer Vision: A Reference Guide*, pp. 1–3, 2020.
- [21] J. H. Lambert, *Photometria sive de mensura et gradibus luminis, colorum et umbrae*. Augustae Vindelicorum: Sumptibus viduæ Eberhard Klett, Typis Christophori Petri Detleffsen, 1760.