

PROYECTO FIN DE GRADO

TÍTULO: Análisis y mejora de modelos de representación de escenas 3D: un enfoque desde NeRF hasta DS-NeRF

AUTOR: Alejandro Hernández Herrera

TITULACIÓN: Grado en Ingeniería Telemática

DIRECTOR: Manuel Villa Romero

TUTOR: Jaime Sancho Aragón

DEPARTAMENTO: Departamento de Ingeniería Telemática y Electrónica

VºBº TUTOR/A

Miembros del Tribunal Calificador:

PRESIDENTA: Julia María García

TUTOR: Jaime Sancho Aragón

SECRETARIO/A: Eduardo Juárez Martínez

Fecha de lectura:

Calificación:

El Secretario/La Secretaria,

Abstract

In recent years, three-dimensional reconstruction and computer vision have seen significant growth across fields such as augmented reality, virtual reality, cinematography, medicine, and engineering. These disciplines enable the creation of highly detailed and accurate three-dimensional models from two-dimensional data, driving advancements in areas like surgical planning, autonomous navigation, and cultural heritage preservation.

In this context, Neural Radiance Fields (NeRF) models have emerged as a powerful tool for generating high-quality three-dimensional representations from a limited set of images. By leveraging deep neural networks, NeRF learns to represent the volumetric density and color of a scene, allowing for the synthesis of new views from different angles.

However, despite its successes, NeRF has notable limitations, such as its difficulty in handling complex occlusions and the requirement for a large number of reference images to generate accurate reconstructions in certain scenarios. To address these challenges, variants like Depth Supervised NeRF (DS-NeRF) have been developed, incorporating depth supervision to enhance geometric accuracy and reduce artifacts in the generated models. This project is driven by the desire to thoroughly evaluate the impact of depth supervision on three-dimensional reconstruction, as well as to propose and test modifications that optimize both performance and visual quality across different conditions, contributing to the development of more robust and efficient technologies with broad applications.

The primary objective of this Bachelor's Thesis is to explore, analyze, and compare the performance of these models to assess whether the depth supervision technique proposed by DS-NeRF represents an advancement over the original technology. The results obtained in this study reveal that, while DS-NeRF introduces an innovative approach with depth supervision, it does not consistently outperform NeRF in terms of image quality, training speed, or scalability. The introduction of depth maps in DS-NeRF appears to be particularly beneficial in scenarios with a very limited number of views (ranging from 2 to 5), but as the number of views increases to 10 or more, NeRF demonstrates superior performance, especially in scenes with complex textures. Additionally, DS-NeRF incurs a higher computational cost, resulting in a significantly slower training speed. This reinforces the notion that NeRF remains a more robust and scalable model for three-dimensional reconstruction under standard conditions, while DS-NeRF may be more suited to specific applications where depth information is critical and the number of views is constrained.

With the aim of further exploring and optimizing the concept introduced by DS-NeRF, this project also seeks to implement and evaluate two modifications, utilizing Microsoft Azure

Kinect DK cameras to obtain depth information. These cameras provide precise data via Time of Flight (ToF) sensors, which are integrated into the neural network training process to improve three-dimensional reconstruction. The tests conducted have shown that, although both variants enhance the geometric reconstruction of the scenes, issues such as shadows and lack of sharpness in color reconstruction persist, due to inconsistencies in the depth data arising from the inherent errors of the ToF sensors. The fact that improvements in depth representation do not lead to better color rendering reinforces the idea that these two objectives may not always be aligned. An optimal three-dimensional model does not necessarily guarantee enhanced color representation and may, in fact, negatively affect it. Despite these limitations, the modified versions of DS-NeRF demonstrate higher accuracy in geometric reconstruction compared to the original DS-NeRF, with one also offering greater efficiency in training time.

Resumen

En los últimos años, la reconstrucción tridimensional y la visión por ordenador han experimentado un crecimiento significativo en campos como la realidad aumentada, la realidad virtual, la cinematografía, la medicina y la ingeniería. Estas disciplinas permiten la creación de modelos tridimensionales detallados y precisos a partir de datos bidimensionales, impulsando avances en sectores como la planificación quirúrgica, la navegación autónoma y la preservación del patrimonio cultural.

Dentro de este contexto, los modelos *Neural Radiance Fields* (NeRF) han emergido como una herramienta poderosa para generar representaciones tridimensionales de alta calidad a partir de un conjunto limitado de imágenes. A través del uso de redes neuronales profundas, NeRF aprende una representación de la densidad volumétrica y el color de una escena, permitiendo la síntesis de nuevas vistas desde diferentes perspectivas.

Sin embargo, a pesar de sus logros, NeRF presenta limitaciones importantes, como su dificultad para gestionar occlusiones complejas y la necesidad de una gran cantidad de imágenes de referencia para generar reconstrucciones precisas en determinados escenarios. Para abordar estas limitaciones, se han desarrollado variantes como Depth Supervised NeRF (DS-NeRF), que incorpora supervisión de la profundidad para mejorar la precisión geométrica y reducir los artefactos presentes en las representaciones generadas. Este proyecto se motiva por el interés de evaluar a fondo el impacto de la supervisión de la profundidad en la reconstrucción tridimensional, así como de proponer y probar modificaciones que optimicen tanto el rendimiento como la calidad visual en diversas condiciones, contribuyendo al desarrollo de tecnologías más robustas y eficientes aplicables en múltiples campos.

El primer objetivo de este Proyecto de Fin de Grado es explorar, analizar y comparar el rendimiento de estos modelos para determinar si la técnica de supervisión de la profundidad propuesta por DS-NeRF representa un avance respecto a la tecnología original. Los resultados obtenidos en este apartado muestran que, aunque DS-NeRF propone un enfoque innovador mediante la supervisión por profundidad, no supera consistentemente a NeRF en cuanto a calidad de imagen, velocidad de entrenamiento o escalabilidad. La introducción de mapas de profundidad en DS-NeRF parece ser más beneficiosa en escenarios un número de vistas extremadamente limitado (de 2 a 5), pero cuando se incrementa el número de vistas a 10 o más, NeRF muestra un desempeño superior, especialmente en escenas con texturas complejas. Además, DS-NeRF presenta una mayor carga computacional que deriva en una velocidad de entrenamiento significativamente inferior. Esto refuerza la idea de que NeRF sigue siendo un modelo más robusto y escalable para la reconstrucción tridimensional en condiciones estándar, mientras que DS-NeRF

podría tener aplicaciones más específicas en escenarios donde la información de profundidad sea crítica y el número de vistas sea limitado.

A fin de seguir explorando y optimizando el concepto que propone DS-NeRF, este proyecto también tiene como objetivo la implementación y evaluación de dos modificaciones, utilizando cámaras Microsoft Azure Kinect DK para obtener información de profundidad. Estas cámaras ofrecen datos precisos mediante sensores de tipo *Time of Flight* (ToF), que se integran en el proceso de entrenamiento de las redes neuronales para mejorar la reconstrucción tridimensional. Las pruebas realizadas han demostrado que, aunque ambas variantes mejoran la reconstrucción geométrica de las escenas, persisten problemas como sombras y falta de nitidez en la reconstrucción del color, atribuibles a inconsistencias en los datos de profundidad derivados de los errores inherentes a los sensores ToF. El hecho de que la mejora en la representación de la profundidad no se traduzca en una mejora en el renderizado de color refuerza la idea de que ambos objetivos no siempre están alineados. Un modelado tridimensional óptimo no necesariamente garantiza una mejora en la representación del color y, de hecho, puede incluso afectar negativamente a esta. A pesar de estas limitaciones, las versiones modificadas de DS-NeRF muestran una mayor precisión en la reconstrucción geométrica en comparación con DS-NeRF, y una de ellas ofrece además una mayor eficiencia en el tiempo de entrenamiento.

Índice general

1. Introducción	15
1.1. Marco y motivación del proyecto	15
1.2. Objetivos	16
1.2.1. Objetivos técnicos	16
1.2.2. Objetivos teóricos	17
1.3. Estructura del resto de la memoria	17
2. Marco Tecnológico	19
2.1. Machine Learning	19
2.1.1. Redes Neuronales	20
2.1.2. <i>Multilayer Perceptron (MLP)</i>	21
2.1.3. <i>Deep Learning</i>	22
2.1.4. Tipos de aprendizaje y problemas comunes durante el entrenamiento . . .	24
2.1.5. <i>Backpropagation</i>	26
2.2. Estimación de Profundidad	28
2.2.1. Visión Estereoscópica	29
2.2.2. Métodos de estimación de profundidad	30
2.3. Métodos de reconstrucción y renderizado 3D	31
2.3.1. Fotogrametría	32
2.3.2. Estructura desde Movimiento	35
2.3.3. Trazado de Rayos	36
2.3.4. <i>Volumetric Rendering</i>	40
3. Estado del Arte	43
3.1. NeRF	43

3.1.1. Fundamentos Teóricos	44
3.1.2. Arquitectura de red	46
3.1.3. Flujo de trabajo de NeRF	47
3.2. DS-NeRF	50
3.2.1. Fundamentos Teóricos	50
3.2.2. Flujo de trabajo de DS-NeRF	53
4. Especificaciones y restricciones de diseño	57
5. Descripción de la solución desarrollada	59
5.1. Información del <i>dataset</i> con imágenes de profundidad	60
5.2. Modificación sobre DS-NeRF	61
5.3. Modificación sobre NeRF	64
6. Resultados	67
6.1. Herramientas y métodos utilizados	67
6.2. Detalles de implemetación	69
6.3. Comparación de NeRF y DS-NeRF con <i>datasets</i> ejemplo	70
6.3.1. Entrenamiento	70
6.3.2. Renderizado de vistas	75
6.4. Modificaciones propias	76
6.4.1. Entrenamiento	76
6.4.2. Renderizado de vistas	80
7. Presupuesto	83
7.1. Costes Materiales	83
7.2. Costes de Personal	84
7.3. Costes Total	85
8. Impacto del proyecto	87
9. Conclusiones	91

ÍNDICE GENERAL	9
9.1. Conclusión	91
9.2. Trabajos futuros	92
A. Resultados adicionales	99
A.1. Pruebas con NeRF	99
A.2. Pruebas con DS-NeRF	102
A.3. Pruebas con modificaciones	105

Índice de figuras

1.1.	NeRF pipeline.	16
2.1.	Estructura neurona de McCulloch y Pitts [9].	20
2.2.	Arquitectura <i>Multilayer Perceptron</i> [12].	22
2.3.	<i>Deep Neural Network</i> [14].	23
2.4.	<i>Autoencoder</i> [17].	25
2.5.	<i>Dropout</i> [18].	25
2.6.	Punto de convergencia [23].	29
2.7.	Ejemplo de visión estereo con dos cámaras [25].	30
2.8.	Ejemplo de cámara con sensor de tiempo de vuelo [27].	31
2.9.	Ejemplo de puntos y cámaras en el espacio [29].	32
2.10.	Base de datos de descriptores de características [30].	35
2.11.	(a) imagen tomada de la escena. (b) nube de puntos de baja densidad. (c) nube de puntos de alta densidad [30].	37
2.12.	(izquierda) proyección en paralelo. (derecha) proyección en perspectiva [33].	39
3.1.	<i>Input y output</i> de la arquitectura NeRF [4].	43
3.2.	Arquitectura de red <i>fully connected</i> de NeRF [4].	46
3.3.	Diagrama de flujo de NeRF.	47
3.4.	Nube de puntos de densidad baja [44].	51
3.5.	Logaritmo de la varianza promedio con diferente número de vistas [42].	52
3.6.	Diagrama de flujo de DS-NeRF.	53
5.1.	Diagrama de flujo de NeRF.	61
5.2.	Diagrama de flujo de NeRF.	65

6.1. Comparación del rendimiento entre NeRF y DS-NeRF para el <i>dataset Flowers</i> con 10 vistas para el entrenamiento.	71
6.2. Comparación del rendimiento entre NeRF y DS-NeRF para el <i>dataset Room</i> con 10 vistas para el entrenamiento.	72
6.3. Comparación visual de diferentes objetos y técnicas.	74
6.4. Comparación gráfica del rendimiento obtenido en el entrenamiento con cada tecnología para el <i>dataset</i> propio.	77
6.5. Comparación visual de los resultados obtenidos con el <i>dataset</i> propio.	79
6.6. Comparación visual de la predicción de la profundidad de la escena conseguida con cada tecnología.	81
A.1. Comparación visual de los resultados obtenidos del entrenamiento de modelos NeRF con diferentes conjuntos de imágenes - Parte 1.	100
A.2. Comparación visual de los resultados obtenidos del entrenamiento de modelos NeRF con diferentes conjuntos de imágenes - Parte 2.	101
A.3. Comparación visual de los resultados obtenidos del entrenamiento de modelos NeRF con diferentes conjuntos de imágenes - Parte 3.	102
A.4. Comparación visual de los resultados obtenidos del entrenamiento de modelos NeRF con diferentes conjuntos de imágenes - Parte 1.	103
A.5. Comparación visual de los resultados obtenidos del entrenamiento de modelos NeRF con diferentes conjuntos de imágenes - Parte 2.	104
A.6. Comparación visual de los resultados obtenidos del entrenamiento de modelos NeRF con diferentes conjuntos de imágenes - Parte 3.	105
A.7. Comparación visual de la predicción del color de la escena conseguida con cada tecnología - Parte 1.	106
A.8. Comparación visual de la predicción del color de la escena conseguida con cada tecnología - Parte 2.	107

Índice de tablas

6.1.	Características de las secuencias utilizadas para evaluar los modelos. La notación "X/Y indica que, para el mismo conjunto de datos, se entrenaan modelos tanto con X como con Y imágenes para el entrenamiento, de un total de Y disponibles.	69
6.2.	Comparación de resultados de PSNR (dB) para el <i>dataset Horns</i>	75
6.3.	Comparación de resultados de tiempo de renderizado de imágenes para varios <i>dataset</i> de ejemplo en segundos.	76
6.4.	Comparación objetiva de los datos obtenidos después de 200,000 iteraciones entre NeRF, DS-NeRF y ambas modificaciones propuestas.	76
6.5.	Comparación de resultados del tiempo de renderizado de imágenes en segundos para el <i>dataset</i> propio en cada tecnología.	82
7.1.	Costes Materiales	84
7.2.	Costes de Personal	84
7.3.	Coste Total del Proyecto	85
A.1.	Resultados del entrenamiento con NeRF para diferentes <i>datasets</i>	99
A.2.	Resultados del entrenamiento con DS-NeRF para diferentes <i>datasets</i>	102

Capítulo 1

Introducción

1.1. Marco y motivación del proyecto

En los últimos años, el campo de la visión por ordenador y la reconstrucción tridimensional han cobrado una importancia creciente en diversos campos como la realidad virtual, la realidad aumentada, la cinematografía, la medicina y la ingeniería. Este ámbito interdisciplinario combina técnicas de procesamiento de imágenes, aprendizaje profundo y geometría computacional para construir modelos tridimensionales detallados y precisos. La capacidad de generar modelos 3D de alta fidelidad a partir de imágenes 2D es crucial para avanzar en estas áreas.

Tecnologías como la fotogrametría digital, combinadas con algoritmos de aprendizaje profundo, permiten crear modelos 3D a partir de conjuntos de datos RGB en dos dimensiones, como en aplicaciones de mapeo de ciudades o reconstrucción de entornos urbanos, donde estas técnicas proporcionan detalles finos en grandes superficies [1]. En robótica y navegación autónoma, la visión por ordenador y la reconstrucción 3D permiten a los sistemas identificar obstáculos, calcular trayectorias y comprender el entorno con una mayor fiabilidad, resultando esencial para mejorar la eficiencia y seguridad de estos sistemas [2]. En patrimonio cultural, estas técnicas permiten documentar y preservar digitalmente monumentos y obras de arte mediante modelos 3D, lo que contribuye a la conservación y restauración precisa de estos bienes [1]. En medicina, el uso de estas tecnologías ha sido explorado para la producción de modelos 3D a partir de imágenes médicas, como resonancias magnéticas y tomografías computarizadas, lo que mejora la planificación quirúrgica y la visualización detallada de estructuras anatómicas complejas. Estos modelos 3D permiten una representación precisa de órganos y vasos sanguíneos, facilitando el diagnóstico y la intervención médica sin necesidad de procedimientos invasivos [3]. Con estos avances, la visión por ordenador y la reconstrucción 3D continúan siendo un pilar fundamental en diversas industrias, donde la necesidad de modelos tridimensionales precisos es cada vez más demandante.

En este contexto, los modelos *Neural Radiance Fields* (NeRF) han surgido como una solución

innovadora, utilizando redes neuronales profundas para producir representaciones tridimensionales de escenas complejas a partir de un conjunto limitado de imágenes, aprendiendo una representación de la densidad volumétrica y el color de la escena y así permitir la síntesis de nuevas vistas desde cualquier perspectiva. Este enfoque ha superado muchas de las limitaciones de los métodos tradicionales de reconstrucción 3D basados en mallas y texturas, proporcionando una mayor precisión y calidad visual. En la Figura 1.1, extraída del trabajo original, se puede observar el *pipeline* de NeRF. Dado un set de imágenes en dos dimensiones y las poses donde se toman, NeRF aprende a representar la forma en tres dimensiones, de forma que nuevas vistas pueden ser generadas. [4]

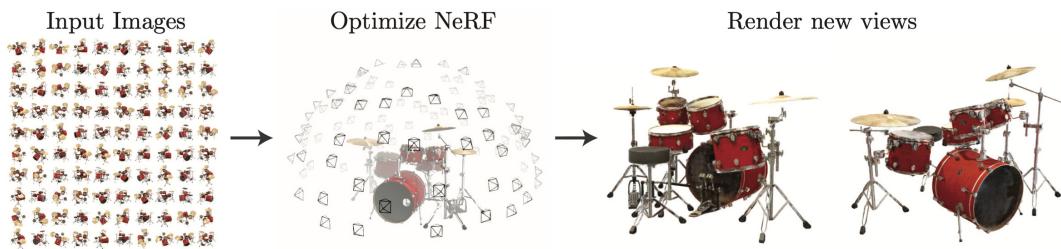


Figura 1.1. NeRF pipeline.

Sin embargo, a pesar de sus avances, NeRF enfrenta desafíos importantes, como la dificultad para manejar escenas con occlusiones complejas, muchas veces necesitando un gran volumen de imágenes para obtener un resultado correcto, y la necesidad de mejorar la eficiencia computacional. Para abordar estas limitaciones, se han propuesto diversas variantes del modelo original, entre ellas DS-NeRF (Depth Supervised-NeRF), que propone integrar información de profundidad para refinar la reconstrucción tridimensional y reducir los artefactos y ruidos presentes en las representaciones originales.

Este proyecto se centra en explorar y analizar los modelos NeRF y DS-NeRF, con el objetivo de evaluar sus capacidades y limitaciones en la reconstrucción tridimensional de escenas complejas. Además, se propone el desarrollo de un nuevo modelo basado en DS-NeRF, que utilice imágenes de profundidad tomadas por cámaras Microsoft Azure Kinect DK [5] para optimizar el concepto de supervisión del aprendizaje de la profundidad en el entrenamiento de la red y, de esa manera, conseguir más precisión y eficiencia en la representación de detalles finos en un tiempo reducido.

1.2. Objetivos

1.2.1. Objetivos técnicos

Desde el punto de vista técnico, el objetivo general del proyecto es explorar el campo de la reconstrucción tridimensional utilizando redes neuronales basadas en información de color y de

profundidad.

Los objetivos específicos asociados al objetivo general son:

- Analizar, evaluar y comparar dos arquitecturas de redes neuronales para la reconstrucción 3D: NeRF y DS-NeRF.
- Desarrollar una arquitectura de red neuronal propia que utilice como entrada imágenes capturadas por una cámara de tiempo de vuelo (*ToF*).
- Evaluar y validar la eficacia de la arquitectura propuesta utilizando una escena propia de imágenes a color y de profundidad.

1.2.2. Objetivos teóricos

En lo que se refiere a los objetivos académicos, los objetivos a tener en cuenta son:

- Adquirir conocimiento extenso sobre el funcionamiento y la arquitectura de modelos de aprendizaje profundo.
- Desarrollar una sólida base en geometría proyectiva y técnicas de reconstrucción tridimensional, con enfoque en la teoría y práctica de la proyección de puntos en un espacio tridimensional a un plano bidimensional.
- Acrecentar la capacidad de analizar y evaluar el rendimiento de diversos modelos de aprendizaje profundo mediante el uso de métricas cuantitativas y cualitativas, diseñando experimentos, interpretando resultados y mejorando los modelos basándose en un riguroso análisis crítico.
- Identificar oportunidades para la innovación en el campo del aprendizaje profundo y la visión por ordenador, aplicando conceptos avanzados para desarrollar nuevas soluciones que superen las limitaciones de las técnicas existentes.

1.3. Estructura del resto de la memoria

Este proyecto de Fin de Grado se estructura en un total de nueve capítulos, incluyendo la introducción como el primero de ellos.

En el Capítulo 2, se presenta un análisis de los conocimientos teóricos y técnicos con los que es necesario familiarizarse para abordar la lectura del resto de la memoria. Se exponen, en primer lugar, la arquitectura, diseño y funcionamiento de los algoritmos de *machine learning*. Además, se contempla especialmente el modelo de red neuronal utilizado en los ejemplos que se analizan posteriormente. Después, dado que la solución propuesta se basa en la estimación

y supervisión de la profundidad, se analiza en qué consiste y algunas de las técnicas utilizadas para la estimación de ésta. Finalmente, se introducen los métodos tradicionales de renderizado y reconstrucción 3D que aprovecha y en los que se basa el modelo NeRF.

Ya que este proyecto ha tenido una etapa extensa de análisis teórico y práctico de dos soluciones ya existentes para comprender exactamente qué hacen y en qué se basan, en el Capítulo 3, se ofrece un análisis en detalle de los fundamentos teóricos específicos y la arquitectura de los modelos NeRF y DS-NeRF.

En el Capítulo 4 se exponen las distintas especificaciones del proyecto. Contiene, entre otras, qué debe hacer la solución desarrollada, cómo, en cuánto tiempo o qué calidad deben tener los resultados obtenidos.

La descripción completa de la propuesta de arquitectura desarrollada se encuentra en el Capítulo 5. Se explica exactamente en qué consiste, en qué se basa y cómo se ha conseguido el funcionamiento correcto de esta nueva solución basada en la supervisión de la profundidad de una escena durante el aprendizaje de la red neuronal, utilizando para ello imágenes de profundidad tomadas por cámaras Microsoft Azure Kinect DK [5]. Se muestran primero las herramientas que han sido necesarias para su desarrollo e implementación, y después se procede a hablar de las diferentes versiones creadas para este proyecto de Fin de Grado.

Una vez descrita la solución desarrollada, en el Capítulo 6, se procede a mostrar todos los resultados obtenidos en las diferentes pruebas realizadas. En primer lugar, se realiza una comparación cualitativa y cuantitativa de los resultados obtenidos en todos los experimentos con NeRF y con DS-NeRF, para comprobar hasta qué punto la modificación representa una mejora respecto al original y de qué forma. En segundo lugar, se incluye una presentación de los resultados obtenidos con las diferentes versiones del nuevo modelo implementado, con el objetivo de determinar cuál es la solución más eficaz.

En el Capítulo 7 se analizan los presupuestos, tanto materiales como de personal, necesarios para reproducir este trabajo.

En el Capítulo 8 se proporciona una visión del impacto que el proyecto puede tener en las diferentes áreas en las que se enfoca.

Finalmente, se incluyen las conclusiones y los posibles trabajos futuros basados en este proyecto en el Capítulo 9.

Capítulo 2

Marco Tecnológico

2.1. Machine Learning

El aprendizaje automático tiene sus orígenes a mediados del siglo XX, cuando investigadores comenzaron a explorar cómo enseñar a las máquinas a aprender de conjuntos de datos. Su evolución fue marcada por varios períodos de rápido avance y crecimiento. En 1957, Frank Rosenblatt introdujo el perceptrón, uno de los primeros modelos de redes neuronales, lo que dio un gran impulso inicial al campo. Posteriormente, el desarrollo de técnicas como los árboles de decisión, las máquinas de vectores de soporte (SVM) y los algoritmos de agrupamiento (*clustering*) sentaron las bases de nuevas metodologías. En las últimas décadas, el auge del aprendizaje profundo, junto con el aumento en la disponibilidad de grandes volúmenes de información y recursos computacionales, ha renovado el interés en el aprendizaje automático, promoviendo su aplicación práctica para resolver problemas complejos del mundo real [6].

De manera concisa, como escribe Zhi-Hua Zhou en su libro *Machine learning* [7], mientras que los humanos podemos aprender de la experiencia, los ordenadores también pueden aprender a través del uso de técnicas de *Machine Learning* (ML), que se define como la técnica que mejora el rendimiento de los sistemas utilizando métodos computacionales. En estos sistemas, la experiencia existe en forma de datos, y la tarea principal del Machine learning es la de desarrollar algoritmos de aprendizaje que construyen modelos a partir de éstos. Alimentando estos algoritmos con muestras de información se obtiene un modelo que puede hacer predicciones o nuevas observaciones.

El objetivo principal de esta disciplina es que el algoritmo consiga generalizar a partir de los datos (*training set*), es decir, que para una entrada desconocida (*testing set*), sea capaz de obtener una salida coherente. Este proceso debe ocurrir sin que el algoritmo haya sido específicamente programado para esa entrada. En este punto, el enfoque difiere de la programación tradicional.

Este proyecto se basa en el uso de redes neuronales artificiales, por lo que el resto de la

sección se enfoca en éstas.

2.1.1. Redes Neuronales

Una red neuronal es un modelo de inteligencia artificial que facilita el procesamiento de datos en sistemas computacionales mediante una arquitectura inspirada en el funcionamiento del cerebro humano. Este tipo de modelo se organiza en capas de neuronas artificiales conectadas entre sí, formando un sistema adaptable que aprende mediante la retropropagación de errores y la optimización de sus parámetros. A través de este proceso, la red ajusta sus parámetros para mejorar continuamente su rendimiento en tareas específicas. Este enfoque ha posibilitado la resolución de problemas complejos, como el reconocimiento de rostros y la generación automática de resúmenes de documentos, alcanzando altos niveles de precisión [8].

El primer modelo de neurona artificial denominado *threshold logic unit* (TLU) o perceptrón, fue propuesto por McCulloch y Pitts en 1943 [9]. En la Figura 2.1 se observa la estructura de una de éstas neuronas.

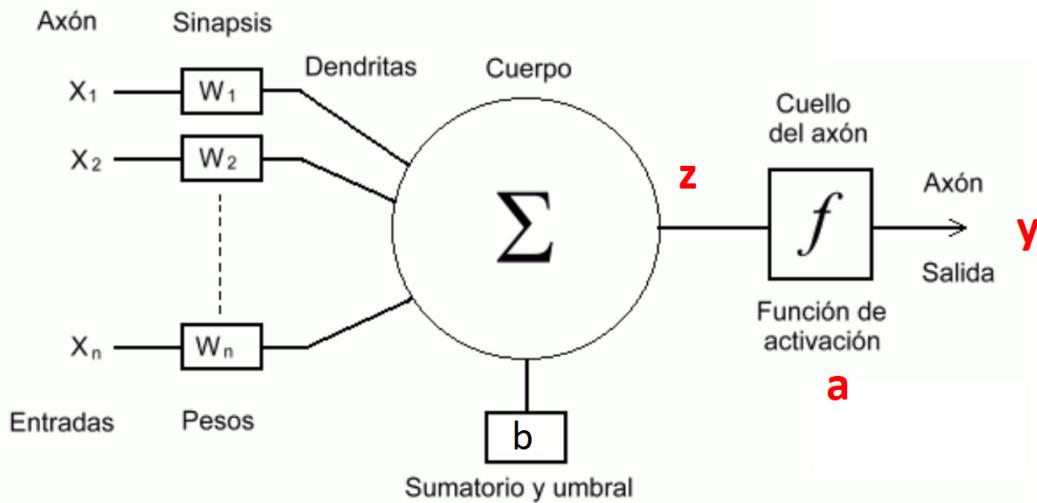


Figura 2.1. Estructura neurona de McCulloch y Pitts [9].

El modelo parte de n entradas ($x_1, x_2, \dots, x_j, \dots, x_n$). Cada entrada tiene un peso asociado ($w_1, w_2, \dots, w_j, \dots, w_n$), que determina la importancia de cada entrada en el cálculo de la salida. A cada entrada con su peso se le aplica una ponderación lineal más un umbral o bias, b , que es un número real que determina si la neurona se activa o no:

$$z = b + \sum_{i=1}^n w_i x_i$$

Seguido de este cálculo, se aplica una función de activación no lineal que, en el modelo de

McCulloch y Pitts es una función escalonada del tipo:

$$y = f(z) \begin{cases} 1 & \text{si } z \geq 0 \\ 0 & \text{si } z < 0 \end{cases}$$

Este cálculo resulta en una salida digital o binaria de forma que, si la suma ponderada $\sum_{i=1}^n w_i x_i$ es mayor o igual al umbral b , la neurona se activa y la salida es 1 y si la suma ponderada es menor que el umbral, la neurona no se activa y la salida es 0. En el caso en el que la suma ponderada sea igual al umbral, se define una recta que representa la región de decisión.

Este modelo puede utilizarse como clasificador entrenable, pero una sola neurona de McCulloch y Pitts solo permite realizar una clasificación binaria. Esto se debe a que la salida binaria (0 o 1) restringe el modelo a distinguir únicamente entre dos clases o estados distintos.

En 1949, mediante su libro *The organization of behavior: A neuropsychological theory* [10], el psicólogo Donald Hebb formula la regla de Hebb, que describe cómo las conexiones entre las neuronas (sinapsis) se fortalecen o se debilitan en función de la actividad conjunta de las neuronas pre y postsinápticas [11], de tal forma que el peso del enlace entre dos neuronas es mayor si las dos neuronas se activan simultáneamente, reduciéndose en caso de activarse por separado. Para un mayor análisis del trabajo de Hebb y cómo se implementa en redes neuronales artificiales, consultar [10] o [11].

A partir del momento en el que es posible conectar neuronas, se comienzan a desarrollar redes neuronales en diferentes capas. En las Subsección 2.1.2 se presenta una de las arquitecturas más sencillas, que es la utilizada en las tecnologías abordadas en este proyecto.

2.1.2. *Multilayer Perceptron (MLP)*

MLP es una de las primeras, más comunes y más simples arquitecturas de redes neuronales utilizadas en el campo del aprendizaje automático. En la Figura 2.2 se puede observar gráficamente cómo se estructuran las neuronas en tres capas.

- **Capa de entrada:** está formada por neuronas que representan las características de los datos de entrada; cada neurona corresponde a una característica y su valor refleja el valor de dicha característica. Estas neuronas no realizan ningún cálculo, simplemente transmiten los valores a las neuronas de la primera capa oculta.
- **Capas ocultas:** que pueden ser una o más, reciben los *inputs* de la capa anterior, aplican una suma ponderada sobre los valores recibidos, añaden un sesgo o umbral, y el resultado pasa a través de una función de activación.
- **Capa de salida:** produce la predicción final, cuyo número de neuronas depende de la

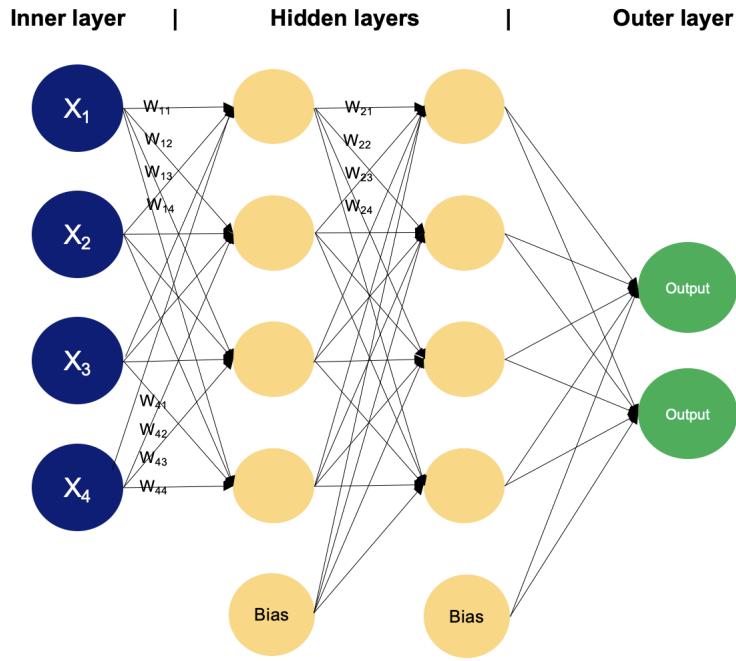


Figura 2.2. Arquitectura *Multilayer Perceptron* [12].

naturaleza del problema; en clasificación binaria, por ejemplo, puede haber una neurona para cada clase, retornando probabilidades.

La simplicidad de este modelo radica en su estructura, donde cada neurona de una capa está conectada a todas las neuronas de la siguiente capa mediante conexiones ponderadas, cuyos valores se ajustan durante el proceso de entrenamiento [13].

Como se ha introducido anteriormente con la regla de Hebb, los pesos en las conexiones en las neuronas representan su fuerza. Cada conexión entre dos neuronas tiene un peso asociado. Estos pesos se modifican durante el entrenamiento y determinan el impacto del *output* de una neurona en la siguiente.

La predicción generada en la capa de salida será comparada con el valor real y , y según difiera del resultado predicho, se irán modificando los pesos en la red para mejorar estas predicciones. La técnica más común de optimizar los resultados es el *backpropagation*, en la que se profundizará en la Subsección 2.1.5.

2.1.3. Deep Learning

El *deep learning* o aprendizaje profundo es una subárea del aprendizaje automático, que se basa en el uso de redes neuronales con múltiples capas para modelar y comprender patrones complejos en datos a gran escala. Los modelos de *deep learning* son conocidos como redes neuronales profundas o DNN (*Deep Neural Networks*).

En la Figura 2.3, se presenta un ejemplo de la estructura de una DNN. Cuando una red tiene más de tres capas, incluyendo las capas de entrada y salida, se considera una red neuronal profunda. Estas redes pueden incluir una variedad de capas, como capas convolucionales, capas de *max-pooling*, y capas densas, entre otras. El uso de estas arquitecturas permite mejorar la capacidad de modelado y la precisión en la resolución de problemas complejos [6].

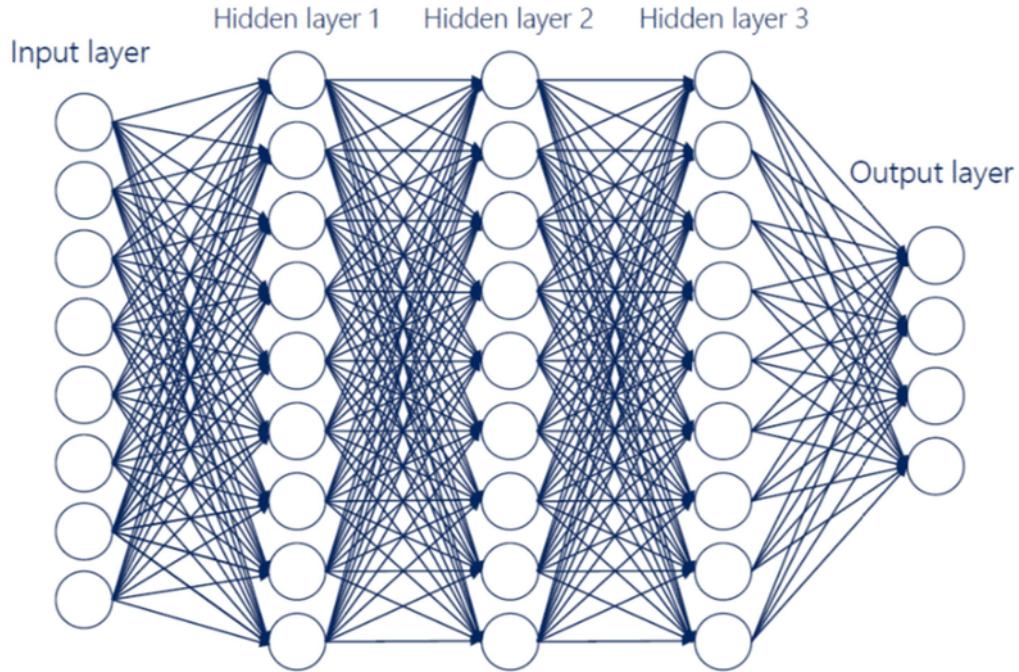


Figura 2.3. Deep Neural Network [14].

A pesar de los avances en este campo, las redes neuronales profundas presentan varias limitaciones y problemas inherentes a su uso que deben ser considerados, los cuales se discuten en la Subsección 2.1.4.

El *deep learning* se ha aplicado ampliamente en tareas de visión por ordenador, como el procesamiento de imágenes y el reconocimiento de patrones, así como en visión estereoscópica, donde se utilizan imágenes de dos cámaras para calcular la profundidad espacial [8]. En el ámbito de la reconstrucción 3D, estas técnicas se emplean para recuperar la estructura y geometría tridimensional de objetos o escenas a partir de un conjunto de imágenes 2D de entrada. Un ejemplo representativo es el desarrollo de sistemas como *Neural Implicit Surfaces* (NeuS), que permiten la reconstrucción precisa de superficies tridimensionales complejas a partir de un número limitado de vistas [15]. Además, el aprendizaje profundo se utiliza en la estimación de profundidad, donde se analiza la geometría de la escena para calcular la distancia de los objetos a la cámara. Algunos algoritmos han logrado generar mapas de profundidad a partir de imágenes bidimensionales, representando con precisión la distancia de los objetos en la imagen, como en la tecnología desarrollada por, entre otros autores, *Clément Godard* [16].

2.1.4. Tipos de aprendizaje y problemas comunes durante el entrenamiento

En el campo del *machine learning*, existen varias formas de entrenar sistemas, destacando dos de ellas por su uso más extendido: el aprendizaje supervisado y el no supervisado. Aunque en este contexto se trata dentro del ámbito del *deep learning*, es importante señalar que estos enfoques no se limitan exclusivamente a redes neuronales profundas, sino que se aplican también en otros enfoques del aprendizaje automático.

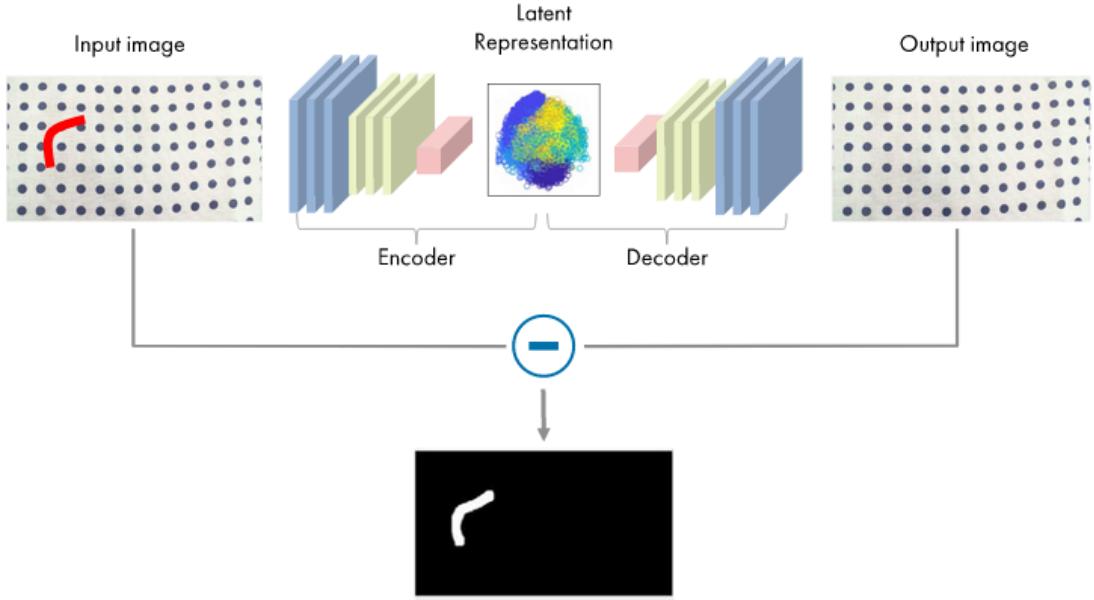
-Aprendizaje supervisado: en este primer enfoque, el sistema se entrena con un conjunto de datos etiquetados, es decir, datos con un valor de salida asociado. El objetivo es que el modelo aprenda a relacionar las entradas con las salidas correctas, basándose en la observación de múltiples ejemplos. En una DNN, los datos etiquetados permiten seleccionar una función de pérdida adecuada según el problema, junto con un optimizador. Durante cada iteración, el modelo realiza predicciones sobre el conjunto de entrenamiento. La función de pérdida mide la discrepancia entre las predicciones del modelo y las etiquetas reales, y el optimizador ajusta los pesos de la red a través del algoritmo de retropropagación (*backpropagation*), minimizando dicha pérdida [13].

-Aprendizaje no supervisado: aquí, el modelo aprende a partir de datos no etiquetados, con el objetivo de descubrir patrones, estructuras o relaciones intrínsecas en los datos. En el contexto del aprendizaje profundo, uno de los enfoques más comunes es el uso de autocodificadores (*autoencoders*), un tipo de red neuronal que se entrena para replicar los datos de entrada en su salida a través de una capa oculta que representa una codificación latente de los datos. Durante el entrenamiento, un codificador aprende a comprimir los datos de entrada en una representación compacta, mientras que un decodificador los reconstruye a partir de esta representación latente. Esta técnica es aplicable a diversos tipos de datos, como imágenes o series temporales, siendo generalizable a muchos contextos [8]. En la Figura 2.4, podemos observar cómo se utiliza un autocodificador para detectar y eliminar anomalías en una imagen.

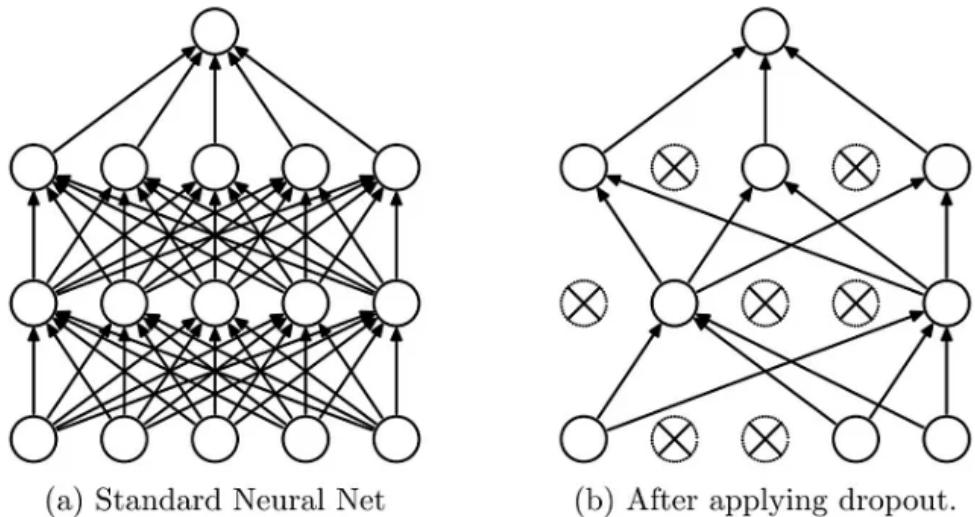
El entrenamiento de DNNs es un proceso complejo que puede enfrentar una serie de desafíos que afectan tanto el rendimiento como la estabilidad del modelo resultante. A continuación, se mencionan varios problemas frecuentes y diferentes estrategias para mitigarlos:

- Al aumentar el número de capas ocultas en una red, el número de parámetros a entrenar crece significativamente, lo que requiere mayor potencia de cómputo y más tiempo para su entrenamiento.
- Otro problema recurrente en el entrenamiento de DNNs es el sobreajuste u *overfitting*. Este fenómeno ocurre cuando el modelo memoriza los detalles y el ruido del conjunto de entrenamiento, lo que perjudica su capacidad para generalizar a nuevos datos no vistos. El sobreajuste es particularmente frecuente cuando se utiliza un conjunto de datos excesivamente pequeño o cuando el modelo es demasiado complejo.

Para conseguir mitigar el *overfitting*, existe una técnica llamada *dropout*. El principio de

Figura 2.4. *Autoencoder* [17].

esta técnica consiste en desactivar aleatoriamente neuronas (junto con sus conexiones) de la red neuronal durante el entrenamiento, impidiendo que aporten a la red en la iteración correspondiente. Esto evita que las unidades se adapten demasiado entre sí [18]. En la Figura 2.5, observamos gráficamente como se deselecciónan ciertas neuronas de la DNN.

Figura 2.5. *Dropout* [18].

- Otro de los desafíos que puede surgir es el problema de la desaparición y explosión de los gradientes. Durante el entrenamiento de una red neuronal, se utiliza el algoritmo de

retropropagación, explicado en la Subsección 2.1.5, para ajustar los pesos de la red. Este algoritmo se basa en el cálculo de los gradientes de la función de pérdida con respecto a los pesos de la red. Estos gradientes se utilizan para actualizar los pesos y reducir el error de predicción.

El problema de la desaparición de gradientes o *vanishing gradient* implica que, al realizar *backpropagation* para actualizar los pesos de la red, la magnitud de la señal se reduce progresivamente a medida que se propaga hacia las capas más profundas, lo que resulta en modificaciones insignificantes en los pesos, impidiendo que la red aprenda correctamente. En contraste, la explosión de gradientes o *exploding gradient* ocurre cuando los gradientes crecen demasiado, lo que provoca actualizaciones inestables y grandes variaciones en los pesos, desestabilizando la red y degradando su rendimiento.

La solución más extendida a estos problemas es una inicialización adecuada de pesos que mantengan los valores iniciales en un rango adecuado. Uno ejemplo de método utilizado es la inicialización de Xavier [19]. Además, la tasa de aprendizaje (η), que se explica en la Subsección 2.1.5, juega un papel crucial: una tasa de aprendizaje demasiado pequeña puede agravar el problema de la desaparición de gradientes, mientras que una tasa excesivamente alta puede intensificar la explosión de gradientes.

2.1.5. *Backpropagation*

Como ya se ha introducido anteriormente, una de las formas más comunes para el entrenamiento de las redes neuronales es el algoritmo de retropropagación o *backpropagation*. Introducido por *D. Rumelhart, G. E. Hinton y R. J. Williams* en 1986 en el artículo *Learning representations by back-propagating errors* [20], en el que se basa la siguiente explicación, se define como un procedimiento reiterativo de ajuste de los pesos dentro de las conexiones de la red.

Antes de comenzar con el proceso de retropropagación, una vez escogido el número de capas y neuronas por capa y la función de activación, se debe determinar varios parámetros clave que influirán en la eficacia del entrenamiento de la red neuronal:

- La tasa de aprendizaje (η) determina el tamaño de los pasos con los que se actualizan los pesos durante el entrenamiento. Un valor muy alto puede hacer que el modelo no converja, mientras que un valor muy bajo puede hacer que el entrenamiento sea extremadamente lento.
- Como ya se indica en la Subsección 2.1.4, es necesario definir la función de pérdidas que se encargue de comparar el resultado obtenido de la red con el resultado que tendría que haber salido. En el caso de la solución desarrollada, se utiliza el error cuadrático medio [21]:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2$$

En esta ecuación:

\hat{y}_i representa la predicción del modelo para el ejemplo i ,

y_i es la etiqueta verdadera del ejemplo i ,

n es el número total de ejemplos en el conjunto de datos.

- El algoritmo de optimización define cómo se ajustan los pesos basándose en los gradientes calculados.
- Como se introdujo en la Subsección 2.1.4, la forma en que se inicializan los pesos puede afectar significativamente el entrenamiento.
- Además, el tamaño del lote, es decir, cuántos ejemplos de entrenamiento se utilizan en una sola actualización de los pesos, y el número de épocas, o el número de veces que el algoritmo de entrenamiento pasa por todo el conjunto de datos, toman gran parte en el rendimiento final de la red neuronal.

Para conseguir que se minimice la función de pérdida y, por lo tanto, se mejore el rendimiento del modelo, se aplica la operación de gradiente sobre la función de pérdidas, lo que se conoce como descenso del gradiente o *gradient descent*. En el contexto de *backpropagation*, se usa para ajustar los pesos de la red neuronal en la dirección del gradiente negativo de la función de pérdida con respecto a los pesos:

$$w = w - \eta \frac{\partial L}{\partial w}$$

donde η es la tasa de aprendizaje y $\frac{\partial L}{\partial w}$ es el gradiente de la función de pérdidas con respecto a los pesos.

A continuación se presentan las diferentes fases en el proceso de *backpropagation*:

- **Propagación hacia adelante:** los datos de entrada se pasan a través de la red para obtener la salida predicha. Cada neurona realiza una operación de activación:

$$a_j = \sigma \left(\sum_i w_{ij} x_i + b_j \right)$$

donde a_j es la activación de la neurona j , w_{ij} es el peso entre la neurona i y j , x_i es la entrada, b_j es el umbral o sesgo, y σ es la función de activación. Este paso ya fue explicado en la Subsección 2.1.1.

- **Cálculo del error:** se calcula el error en la capa de salida usando la función de pérdida mencionada anteriormente
- **Propagación hacia atrás:** se calculan los gradientes de la función de pérdida con respecto a cada peso en la red. Se utiliza la regla de la cadena para calcular estos gradientes desde la capa de salida hasta la capa de entrada.

Para cada capa l , el gradiente de la pérdida con respecto a los pesos se calcula como:

$$\delta^l = \frac{\partial L}{\partial a^l} \cdot \sigma'(z^l)$$

$$\frac{\partial L}{\partial w^l} = \delta^l (a^{l-1})^T$$

donde δ^l es el error de la capa l , σ' es la derivada de la función de activación, y z^l es el valor de la suma ponderada antes de aplicar la activación.

- **Actualización de pesos:** finalmente, se actualizan los pesos utilizando los gradientes calculados y la tasa de aprendizaje:

$$w^l = w^l - \eta \frac{\partial L}{\partial w^l}$$

Este proceso de *forward* y *backward propagation* se repite para cada lote de datos o *batch* hasta que la función de pérdidas se minimice adecuadamente o se cumpla algún criterio de parada, como el número máximo de iteraciones o la convergencia del error.

Para más detalles sobre el algoritmo y su implementación, consultar el artículo original [20].

2.2. Estimación de Profundidad

La estimación de profundidad es un concepto clave en la visión artificial y la percepción humana. Se refiere a la capacidad de determinar la distancia a la que se encuentran los objetos en una imagen o en nuestro entorno.

Para percibir la profundidad tridimensional de los objetos, el ser humano utiliza la visión estereoscópica mediante los ojos, que están separados por una distancia pequeña. Este fenómeno se basa en el principio de la disparidad binocular, donde cada ojo ve una imagen ligeramente diferente de la misma escena debido a su posición única. En la Subsección 2.2.1 se profundizará en este concepto, que supone un gran reto en el ámbito de la computación.

En el contexto de la visión artificial, la estimación de profundidad implica el uso de algoritmos para reconocer la tridimensionalidad de imágenes bidimensionales. Estos algoritmos pueden elaborar un mapa representando la profundidad a la que se encuentran cada uno de los objetos

de una imagen [22]. En el área de la reconstrucción 3D, utilizando múltiples cámaras o puntos de vista para estimar la distancia de los objetos en una escena permite reconstruir modelos tridimensionales precisos del entorno.

2.2.1. Visión Estereoscópica

Como se ha mencionado anteriormente, este es el mecanismo que los seres humanos, al igual que muchos animales, utilizan para comprender el mundo en tres dimensiones. Se produce cuando el cerebro fusiona las imágenes 2D provenientes de ambos ojos en una sola. Esta representación integrada nos permite percibir la profundidad de la escena.

La estereoscopía se basa en las diferencias en las imágenes que llegan a nuestros ojos, que al estar separados unos centímetros, proporcionan dos puntos de vista ligeramente distintos. Si pudiéramos capturar las imágenes que se proyectan en nuestras retinas, y las pudiéramos superponer de alguna manera, apreciaríamos que algunos objetos presentan disparidad.

Esta disparidad retiniana es la distancia entre la proyección de un punto de la escena en la retina derecha y la proyección del mismo punto en la retina izquierda y se suele medir en grados. Cada punto en el espacio tiene una disparidad que depende de la profundidad y de la distancia a la que convergen nuestros ojos. Este punto de convergencia se refiere al punto en el espacio donde las líneas de visión de ambos ojos se cruzan cuando miran un objeto [23]. En la Figura 2.6 podemos ver un ejemplo.

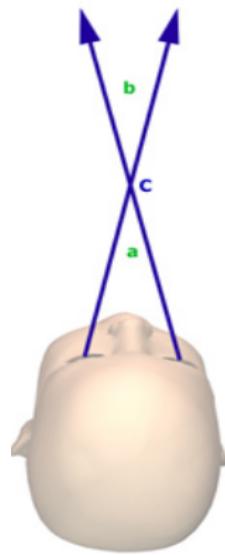


Figura 2.6. Punto de convergencia [23].

La visión estereoscópica es posible gracias a que el sistema visual es capaz de combinar dos imágenes con disparidad en una única imagen en relieve. Esta habilidad se llama fusión y el sentido resultante estereopsis.

2.2.2. Métodos de estimación de profundidad

En la Subsección 2.2.1 se ha explicado cómo el cerebro humano interpreta el mundo 3D gracias a la disparidad binocular. Ahora, se introducen algunos de los métodos que se han desarrollado para estimar la profundidad en el campo de la visión por ordenador. Estos métodos buscan emular la capacidad humana de entender el entorno tridimensional utilizando diferentes técnicas y tecnologías.

- **Visión estéreo:** técnica bien establecida en la visión por ordenador que permite obtener mediciones tridimensionales utilizando dos o más cámaras dispuestas en diferentes posiciones. Este método se basa en la triangulación de los puntos proyectados desde cada cámara. Cada píxel de una cámara digital captura la luz que llega al sensor, y si se identifica una característica común en las imágenes capturadas por ambas cámaras, se puede calcular la disparidad entre ellas. Esta disparidad se utiliza para obtener información sobre la profundidad. Al cruzar los rayos desde diferentes puntos de vista, es posible reconstruir la ubicación tridimensional del objeto y sus características [24]. En la Figura 2.7 se presenta un ejemplo gráfico de este concepto.

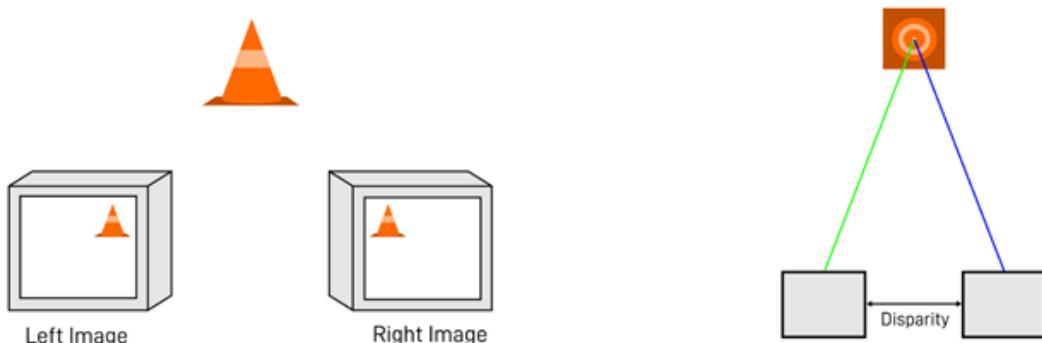


Figura 2.7. Ejemplo de visión estereo con dos cámaras [25].

- **Tiempo de Vuelo (*Time-of-Flight, ToF*):** permite capturar imágenes 3D sin necesidad de escanear el objeto. Esta técnica se basa en iluminar el objeto con una fuente de luz modulada. La posición del objeto se determina midiendo el tiempo de viaje de un pulso emitido y su retorno al sensor tras ser reflejado por el objeto. Las cámaras empleadas para capturar la escena en el desarrollo del modelo propuesto utilizan esta técnica para estimar la profundidad de los objetos presentes.

La variante de esta técnica más utilizada actualmente en cámaras con sensor ToF es la del cambio de fase. Primero, el sensor ToF emite una luz modulada, generalmente en forma de una onda sinusoidal. La luz modulada viaja hacia el objeto, se refleja en su superficie y

regresa al sensor. El sensor recibe esta luz reflejada y mide el cambio de fase entre la onda emitida y la onda recibida [26]:

- **Fase de la onda emitida:** ϕ_{emitida}
- **Fase de la onda recibida:** ϕ_{recibida}
- **Cambio de fase:** $\Delta\phi = \phi_{\text{recibida}} - \phi_{\text{emitida}}$

La distancia d al objeto se puede calcular utilizando el cambio de fase $\Delta\phi$, la velocidad de la luz c y la frecuencia de modulación f :

$$d = \frac{c \cdot \Delta\phi}{4\pi f}$$

En la Figura 2.8 podemos ver un ejemplo gráfico de este concepto:

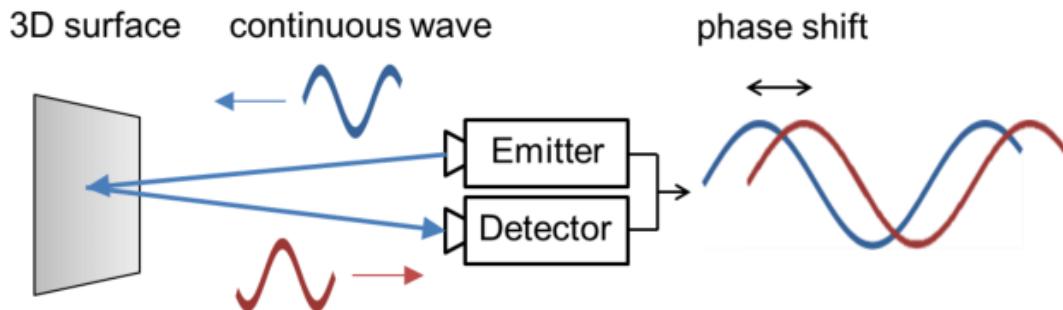


Figura 2.8. Ejemplo de cámara con sensor de tiempo de vuelo [27].

2.3. Métodos de reconstrucción y renderizado 3D

El área de reconstrucción y renderizado 3D se refiere a los campos de estudio y práctica que se ocupan de la generación de imágenes tridimensionales y la recreación de objetos o escenas en 3D a partir de datos diversos. En concreto, la reconstrucción 3D consiste en crear modelos tridimensionales de objetos o escenas a partir de datos obtenidos a través de diferentes métodos, ya sean fotografías, escaneos láser u otras mediciones, y el renderizado es el proceso de generar una imagen 2D a partir de una reconstrucción 3D mediante el uso de algoritmos y técnicas computacionales.

En los siguientes apartados, se profundiza en diversas técnicas de reconstrucción y renderizado 3D que las arquitecturas estudiadas y desarrolladas en este proyecto aprovechan para sintetizar vistas nuevas y realistas a partir de imágenes RGB y de profundidad como entrada.

2.3.1. Fotogrametría

Como se indica en el libro *Photogrammetric 3D reconstruction of small objects for a real-time fruition*, la fotogrametría aprovecha un set de imágenes tomadas desde distintos puntos de vista para construir un modelo digital en tres dimensiones de la escena capturada, detectando múltiples puntos en cada imagen basándose en la forma y en las diferencias de color y asociando puntos homólogos entre diferentes imágenes para determinar la posición del objeto, su orientación y su forma en el espacio 3D [28].

Previo al proceso de adquisición de imágenes, algunos parámetros de cámara deben ser ajustados para tomar fotografías adecuadas para lograr una reconstrucción óptima en las fases posteriores:

- Una distancia focal media es la mejor opción para reducir cualquier distorsión de perspectiva (unos 50 milímetros para una cámara de formato completo)
- Una profundidad de campo reducida puede traer limitaciones en la toma de imágenes. Profundidad de campo o *depth of focus* se refiere a la zona de distancia a lo largo del eje óptico de una cámara fotográfica en la que los objetos están nítidos en la imagen, por lo tanto, si es demasiado reducido, solo una pequeña porción de la imagen aparecerá lo suficientemente nítida. Como posible solución, se puede incrementar la profundidad de campo aumentando el valor f , el ratio de la distancia focal al diámetro de la pupila de entrada.

La captura de imágenes debe darse en condiciones lumínicas homogéneas para evitar discontinuidades en el color en las texturas que serán extraídas en las fases siguientes. Además, el contraste debe ser establecido para evitar grandes sombras o zonas sobreexpuestas que puedan obstaculizar la identificación de puntos en las imágenes. En la Figura 2.9 podemos observar una representación gráfica de cómo diferentes cámaras detectan el mismo punto en el espacio desde diferentes perspectivas, para una posterior reconstrucción en tres dimensiones.

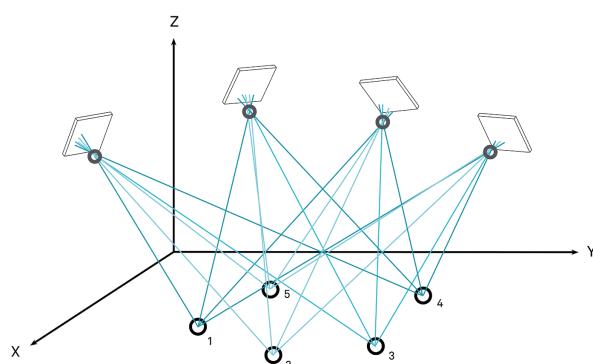


Figura 2.9. Ejemplo de puntos y cámaras en el espacio [29].

Una reconstrucción fiable requiere una correcta calibración de cámara dirigida a estimar parámetros intrínsecos, para reducir efectos de distorsión de lente, y parámetros extrínsecos, para describir la orientación de los objetos en el espacio y las posiciones cámara-objeto. La calibración es el proceso de determinar los valores intrínsecos de una cámara, también conocidos como orientación interior. Estos valores incluyen la distancia focal, el tamaño del formato, el punto principal y la distorsión de la lente.

Una correcta calibración de la cámara requiere comprender la matriz intrínseca K y las matrices extrínsecas. El libro *Multiple View Geometry in Computer Vision* de Richard Hartley y Andrew Zisserman [24], en el que se basa la siguiente explicación, profundiza en la proyección de cámaras y las matrices mencionadas.

La proyección de puntos 3D en un espacio del mundo hacia un plano de imagen 2D es un proceso fundamental basado en un modelo matemático que utiliza matrices intrínsecas y extrínsecas.

- La **matriz intrínseca K** encapsula las propiedades internas de la cámara, como la distancia focal y el punto principal. Permite transformar coordenadas 3D en el espacio de la cámara a coordenadas de píxeles en la imagen, ajustando la proyección de acuerdo con las características ópticas de la cámara. Se define generalmente como:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Donde:

- f_x y f_y son las distancias focales en las direcciones x e y , respectivamente.
- c_x y c_y representan las coordenadas del punto principal (el centro de la imagen) en el sistema de coordenadas de la cámara.
- La última fila de la matriz K es para asegurar que las coordenadas homogeneizadas se manejen adecuadamente.
- Las matrices extrínsecas, que incluyen la matriz de rotación R y el vector de traslación T , establecen la relación entre el sistema de coordenadas del mundo y el sistema de coordenadas de la cámara. Se utilizan para transformar las coordenadas de los puntos 3D del mundo al sistema de coordenadas de la cámara, permitiendo así que los puntos sean vistos desde la perspectiva de la cámara. Se pueden representar como:

$$\begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix}$$

Donde:

- R es una matriz de rotación que orienta la cámara en el espacio 3D.
- T es un vector de traslación que posiciona la cámara en el espacio.

El proceso de proyección de un punto 3D \mathbf{P} en el espacio del mundo a un punto en la imagen 2D \mathbf{p} se lleva a cabo en varias etapas:

1. **Transformación del espacio del mundo al espacio de la cámara:**

$$\mathbf{P}_c = R\mathbf{P} + T$$

Aquí, \mathbf{P}_c son las coordenadas del punto en el sistema de la cámara.

2. **Proyección a coordenadas homogéneas:** las coordenadas del punto en el espacio de la cámara se representan en forma homogénea:

$$\mathbf{P}_c = \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix}$$

3. **Aplicación de la matriz intrínseca:**

$$\mathbf{p} = K\mathbf{P}_c$$

Esto transforma las coordenadas del espacio de la cámara a coordenadas de píxeles en la imagen. Al aplicar la matriz intrínseca, se obtienen las coordenadas 2D (x, y) en el plano de la imagen, a partir de las coordenadas homogéneas.

4. **Normalización:** finalmente, se normalizan las coordenadas de la imagen dividiendo por la tercera coordenada (la coordenada homogénea) para obtener las coordenadas en el plano de imagen:

$$\mathbf{p} = \begin{bmatrix} \frac{x_c}{z_c} \\ \frac{y_c}{z_c} \end{bmatrix}$$

El modelo de cámara es un modelo matemático que aproxima la transformación proyectiva original a la realidad física de las lentes. Al ser cámaras no métricas, los parámetros de orientación interna de la cámara (focal, punto principal, distorsión radial y tangencial, etc.) no se conocen con exactitud; por lo tanto, será necesario realizar una calibración para determinar estos parámetros en cada trabajo. Los algoritmos Estructura desde Movimiento o *Structure from Motion* (SfM), que se tratan en la Subsección 2.3.2 son capaces de estimar parámetros internos automáticamente. Además, SfM es el algoritmo más común para el proceso de reconstrucción.

2.3.2. Estructura desde Movimiento

Como se ha mencionado en la anterior Subsección, SfM es el algoritmo de reconstrucción 3D más común.

El primer paso del *workflow* es el proceso de extracción de puntos clave, en el que SfM extrae una serie de características de cada imagen, que podrán ser utilizados posteriormente para la correspondencia de imágenes. Los puntos de interés o *keypoints* son identificados a lo largo de toda la imagen, seguido de la creación de un descriptor de características, el cual se calcula transformando los gradientes de la imagen local en una representación que es insensible a las variaciones de iluminación y orientación. Un descriptor de características es esencialmente una representación compacta y única de una característica en una imagen que puede ser utilizada para relacionar diferentes imágenes entre sí en el proceso de reconstrucción. Estos descriptores son lo suficientemente únicos como para permitir que las características se correspondan en grandes conjuntos de datos [30]. En la Figura 2.10 se observa cómo se descompone una imagen en una base de datos de descriptores de características.

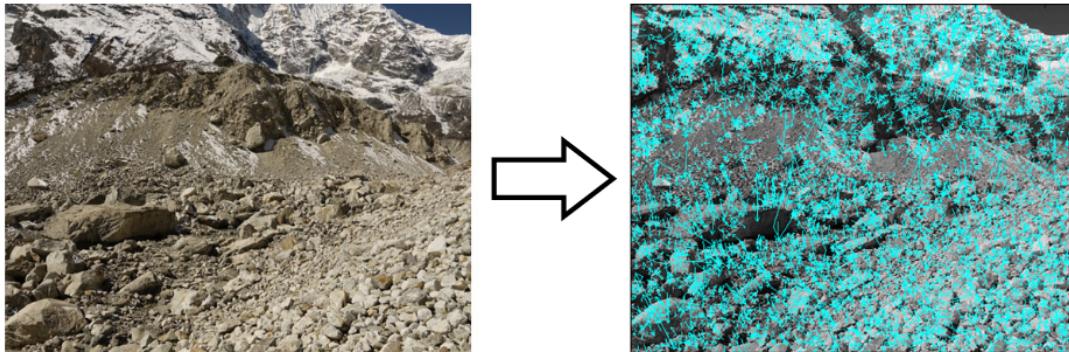


Figura 2.10. Base de datos de descriptores de características [30].

El siguiente paso es la reconstrucción de la escena en tres dimensiones. Se estiman las poses de cámara y se extrae una nube de puntos de baja densidad, también conocida como *sparse*, utilizando el sistema *Bundler* [31]. Los puntos clave de múltiples imágenes se emparejan utilizando algoritmos de aproximación como el de *nearest neighbours* y se establecen *tracks* relacionando puntos clave específicos en un conjunto de imágenes. Los *tracks* que involucran dos o más puntos clave y tres imágenes se utilizan para la reconstrucción en la nube de puntos, y los demás serán descartados. Esta práctica retira de la base de datos elementos transitorios como personas moviéndose, u otros elementos no estáticos capturados de manera no intencionada, ya que no son apropiados para una reconstrucción, dado que su posición es relativa en todas las imágenes.

Los *tracks* imponen restricciones en la orientación de las poses de la cámara, que se reconstruyen utilizando una transformación de similitud o *similarity transformation*, mientras que la minimización de errores se logra utilizando una solución de mínimos cuadrados no lineales. Finalmente, la triangulación se utiliza para estimar las posiciones de los puntos 3D y reconstruir

incrementalmente la geometría de la escena, fijada en un sistema de coordenadas relativo. Se produce la nube de puntos *sparse* utilizando el paquete de ajuste de *Bundler*.

El paso final es el post-procesamiento, en el que, entre otros procesos, se transforma de un sistema de coordenadas relativo a uno absoluto, aunque el método más efectivo consiste en realizar una transformación de coordenadas durante la importación de datos , especificando el sistema de coordenadas actual y el de destino antes de la indexación.

Una nube de puntos de densidad mejorada puede generarse a partir de una *sparse* mediante los algoritmos de *Clustering View for Multi-view Stereo* (CMVS) y de *Patch-based Multi-view Stereo* (PMVS2) [32], utilizando las posiciones de cámara derivadas de *Bundler* como entradas. SfM y *Multi-view Stereo* (MVS) son técnicas complementarias en visión por ordenador, utilizadas en la reconstrucción 3D . SfM se emplea para estimar los parámetros de la cámara y proporcionar una reconstrucción de baja densidad, mientras que MVS se utiliza para refinar esta reconstrucción y proporcionar una salida 3D más densa. La Figura 2.11 muestra las diferencias entre una nube de puntos *sparse* y una densa.

Otra solución que genera una nube de puntos densa es el uso de las arquitecturas NeRF, en las que se profundizará en la Sección 3.

2.3.3. Trazado de Rayos

El trazado de rayos o *Raytracing* es un algoritmo de renderizado que consiste en simular el comportamiento de los rayos de luz que inciden sobre un objeto y que se visualizan desde una perspectiva. El algoritmo traza los rayos desde el observador procesando las posibles intersecciones de ese rayo con los objetos y trazándose otro rato si fuera necesario por reflexión o refracción. Un rayo consiste en un vector situado en un punto del espacio que se utiliza para simular un rayo de luz que parte del observador y atraviesa la escena. Cuando hablamos de observador, realmente hablamos de la cámara que captura la imagen. Un plano de proyección es el plano en el que la escena se está renderizando [33].

A menudo es necesario aplicar cambios a la posición, tamaño u orientación de un objeto para que se alineen correctamente con otros objetos y con la cámara, y se requiere hacerlo indicando los parámetros exactos que se quieren cambiar. A esto se le llama transformaciones. Al aplicar trasformaciones a un punto se utiliza una matriz de una dimensión superior. En este caso veremos cómo son para un punto en tres dimensiones. A continuación, se detallan las transformaciones:

- **Traslación:** mover un punto a una distancia determinada. El vector T indica la dirección de la traslación y su magnitud indica la distancia deseada.

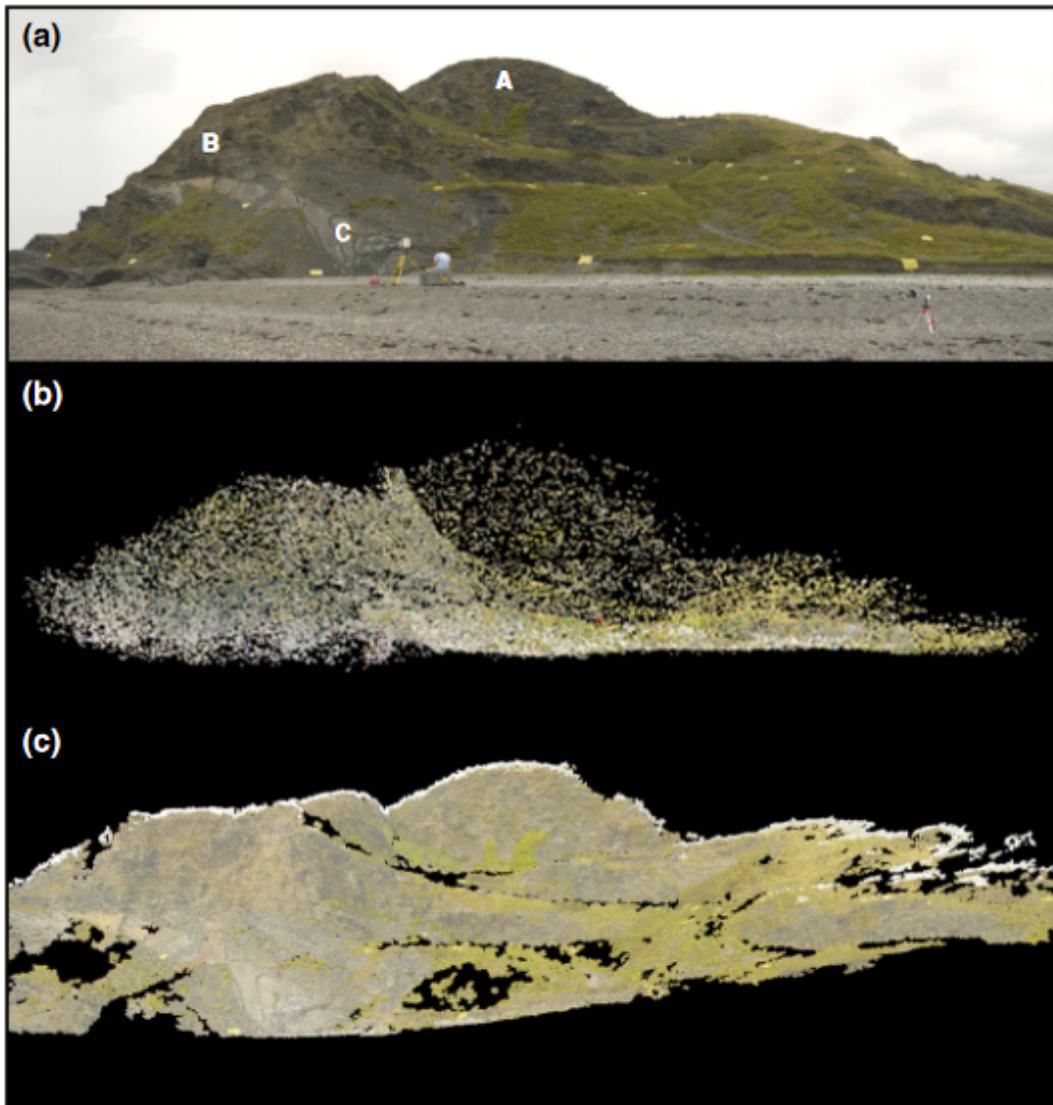


Figura 2.11. (a) imagen tomada de la escena. (b) nube de puntos de baja densidad. (c) nube de puntos de alta densidad [30].

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (2.1)$$

Con esta operación los componentes de T en cada eje se suman a las coordenadas del punto obteniendo así la translación.

- **Escalado:** Agrandar o encojer un objeto en un factor S determinado aplicándola a cada punto.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (2.2)$$

Al aplicar el escalador a cada punto sus coordenadas se multiplican por el factor de escalado deseado.

- **Rotación:** La matriz de rotación aplica un desplazamiento a un punto con respecto al ángulo β determinado. Para el caso de tres dimensiones, es necesario especificar el eje sobre el que se rota, ya que existen tres ejes de rotación.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \beta & -\sin \beta & 0 \\ 0 & \sin \beta & \cos \beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.3)$$

Rotación respecto al eje X

$$\begin{bmatrix} \cos \beta & 0 & \sin \beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \beta & 0 & \cos \beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.4)$$

Rotación respecto al eje Y

$$\begin{bmatrix} \cos \beta & -\sin \beta & 0 & 0 \\ \sin \beta & \cos \beta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.5)$$

Rotación respecto al eje Z

Si se desea rotar una figura sin cambiar su posición, se necesita trasladarla al origen (0,0,0), aplicar la rotación, y trasladarla a la posición anterior. Si no, la figura simplemente se desplazará alrededor del eje especificado sin rotar sobre sí misma.

El componente más importante del trazado de rayos son las propiedades del observador, ya que de éste depende el resultado de la imagen final, al influir en varios aspectos que determinan la visualización de la imagen.

- **Tipo de proyección:** De qué forma la cámara visualiza la escena. O bien el observador se mantiene fijo en un punto y desde ahí se trazan rayos, todos diferentes, hacia la escena, lo

que se conoce como perspectiva, o bien se trazan rayos paralelos desde un observador en el infinito hacia el plano de proyección, conocido como paralela. En la Figura 2.12 observamos un ejemplo de proyección en paralelo y en perspectiva.

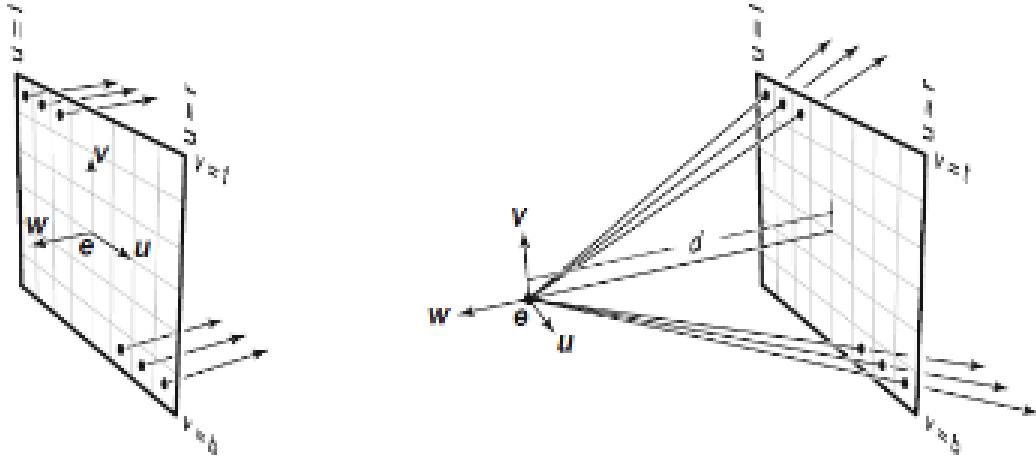


Figura 2.12. (izquierda) proyección en paralelo. (derecha) proyección en perspectiva [33].

- **Vector ViewUP (VUP):** este vector determina la dirección hacia la que el observador considera arriba.
- **Near, Far:** Distancia mínima y máxima de renderizado. Es decir, los objetos de la escena que se encuentren fuera del rango, no serán renderizados en la imagen resultante.

Una vez determinados los diferentes parámetros y definidos los componentes, se implementa un algoritmo en el cual, para cada píxel de la imagen, se calcula un rayo que parte del observador y para por un punto correspondiente a ese píxel. Si el rayo se interseca con algún objeto, se calcula el vector normal n a la superficie del objeto y se evalúa el modelo de iluminación para determinar el color del píxel. En caso de que el rayo no se intersecte con ningún objeto, se asigna el color de fondo al píxel actual.

El proceso para determinar el color de cada píxel en el *Raytracing* consiste en determinar primero cuál es la superficie visible y luego aplicar un modelo de iluminación basado en las propiedades del material. No se entrará en detalles acerca del cálculo de intersecciones o los diferentes modelos de iluminación. Para más detalle en estas cuestiones, consultar [33].

Este modelo presenta ventajas, como su sencillez y su capacidad para combinarse con otros algoritmos, pero también tiene desventajas como su costo computacional o que, por sí solo no consigue efectos demasiado realistas. Existen otras técnicas utilizadas para el renderizado de imágenes diferentes a la de *Raytracing*, y arquitecturas como NeRF, las cuales emplean técnicas similares para evaluar la densidad y el color a lo largo de los rayos que atraviesan la escena.

2.3.4. *Volumetric Rendering*

El renderizado de volumen en gráficos por ordenador es un área fundamental para lograr efectos visuales realistas en la representación de escenas en tres dimensiones. A diferencia del renderizado tradicional de superficies, en el que los objetos se representan principalmente por sus fronteras geométricas, el renderizado de volumen se centra en simular cómo la luz interactúa con materiales que ocupan un espacio tridimensional, como el humo, las nubes o el agua.

En este contexto, el artículo *Ray Tracing Volume Densities* de James T. Kajiya y Brian P. Von Herzen representó un avance significativo [34], ya que introdujo técnicas avanzadas para modelar y simular materiales volumétricos mediante el trazado de rayos, sentando las bases para desarrollos posteriores en el campo del renderizado volumétrico.

Kajiya y Von Herzen proponen un modelo basado en la densidad volumétrica $\sigma(\mathbf{x})$, donde cada punto \mathbf{x} dentro del volumen se caracteriza por una densidad que determina la cantidad de luz que puede absorber, dispersar o emitir. La interacción de la luz con los materiales volumétricos se describe mediante la ecuación de renderizado volumétrico:

$$L(\mathbf{o}, \mathbf{d}) = L_{\text{emisión}}(\mathbf{o}, \mathbf{d}) + \int_0^{t_{\max}} \sigma(\mathbf{x} + t\mathbf{d}) L(\mathbf{x} + t\mathbf{d}, -\mathbf{d}) dt$$

donde:

- $L(\mathbf{o}, \mathbf{d})$ es el valor de luz que llega al punto de vista \mathbf{o} a lo largo de la dirección \mathbf{d} .
- $L_{\text{emisión}}(\mathbf{o}, \mathbf{d})$ representa la luz emitida directamente desde el punto \mathbf{o} hacia la dirección \mathbf{d} .
- $\sigma(\mathbf{x})$ es la densidad volumétrica en el punto \mathbf{x} , que determina la cantidad de luz absorbida o dispersada por unidad de distancia.
- t_{\max} es la distancia máxima a la que se extiende el volumen en la dirección \mathbf{d} .

Cómo la luz interactúa con los materiales volumétricos se describe utilizando la ecuación recién descrita, que tiene en cuenta tanto la luz emitida por el volumen como la luz que atraviesa y se dispersa en el mismo. La absorción y dispersión de la luz dentro del volumen están directamente relacionadas con la densidad volumétrica $\sigma(\mathbf{x})$. A mayor densidad, mayor es la cantidad de luz que se absorbe o dispersa a medida que los rayos de luz atraviesan el volumen.

El algoritmo de trazado de rayos, explicado en la anterior Subsección, es crucial para simular la propagación de la luz a través de volúmenes. Se implementa trazando rayos desde la cámara a través del volumen y calculando cómo interactúan estos rayos con la densidad y propiedades ópticas del medio en cada punto. Este proceso implica resolver la integral volumétrica para cada rayo, considerando tanto la absorción como la dispersión a lo largo de su trayectoria.

Para una explicación completa, consultar la publicación original de *Kajiya y Von Herzen* [34].

Capítulo 3

Estado del Arte

3.1. NeRF

NeRF es una técnica innovadora en el campo de la visión por ordenador y los gráficos en tres dimensiones que permite generar imágenes fotorrealistas y vistas sintéticas de escenas complejas a partir de un conjunto limitado de imágenes en dos dimensiones. Fue introducida en 2020 por un equipo de investigadores de *UC Berkeley* y *Google Research*, incluyendo a *Ben Mildenhall*, *Pratul P. Srinivasan*, *Matthew Tancik*, *Jonathan T. Barron*, *Ravi Ramamoorthi* y *Ren Ng* [4].

Como se aprecia en la Figura 3.1, la arquitectura NeRF consiste en un algoritmo que representa una escena estática utilizando una red neuronal profunda no convolucional *fully connected*, también conocida como MLP, que toma un *input* en forma de una coordenada continua 5D: La posición (x, y, z) de un punto y la dirección desde la que se observa (θ, ϕ). El *output* es la densidad volumétrica y la radiancia emitida, que mide cuánta energía es emitida o recibida en una dirección específica desde una superficie específica, dependiendo de la vista desde esa posición en el espacio.

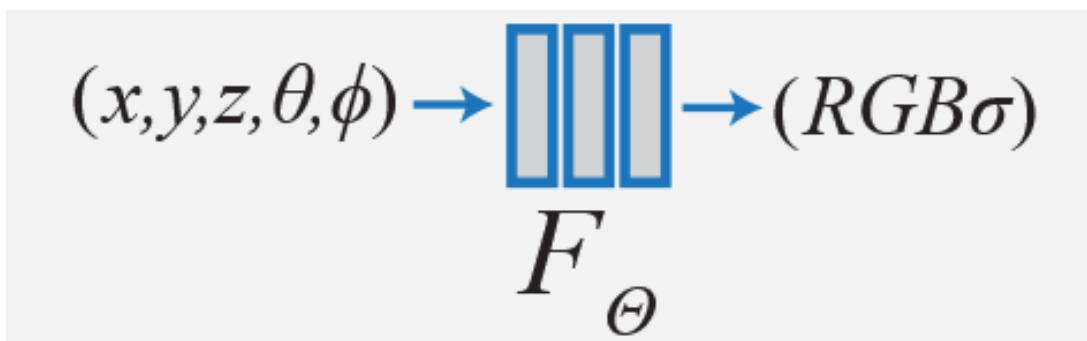


Figura 3.1. *Input y output* de la arquitectura NeRF [4].

Como se explica en detalle posteriormente, NeRF sintetiza nuevas vistas introduciendo en la red coordenadas 5D posicionadas a lo largo de rayos de cámara y utilizando técnicas clásicas

de renderizado volumétrico para proyectar los colores y densidades de salida en una imagen. Dado que el renderizado volumétrico es naturalmente diferenciable, es decir, que las funciones matemáticas utilizadas permiten calcular derivadas de manera directa, el único *input* requerido para optimizar la representación es un conjunto de imágenes con poses de cámara conocidas.

3.1.1. Fundamentos Teóricos

Como se ha introducido antes, NeRF representa una escena continua como una función de vectores 5D cuyo *input* es una posición 3D $\mathbf{x} = (x, y, z)$ y una dirección de vista o *viewing direction* en 2D (θ, ϕ) (en la práctica se expresa como un vector 3D en coordenadas Cartesianas d), y cuyo *output* es el color emitido $c = (r, g, b)$ y la densidad volumétrica σ . Se approxima la representación de la escena 5D continua con una red MLP $F_\Theta : (\mathbf{x}, d) \rightarrow (c, \sigma)$ y se optimizan sus pesos Θ para mapear desde cada coordenada 5D a su correspondiente densidad volumétrica y color emitido, dependiente de la vista.

NeRF renderiza el color de cada rayo que pasa por la escena utilizando principios de renderizado de volumen clásicos, como los introducidos en la Subsección 2.3.4 y detallados en el trabajo *Ray Tracing Volume Densities* de James T. Kajiya y Brian P. Von Herzen [34].

La densidad volumétrica $\sigma(x)$ se puede interpretar como la probabilidad diferencial de un rayo terminando en una partícula infinitesimal en la posición \mathbf{x} [4]. El color esperado $C(r)$ de un rayo de cámara $r(t) = o + td$ con límites *Near*, *Far* t_n y t_f es:

$$C(r) = \int_{t_n}^{t_f} T(t) \sigma(r(t)) c(r(t), d) dt,$$

donde

$$T(t) = \exp \left(- \int_{t_n}^t \sigma(r(s)) ds \right)$$

La función $T(t)$ denota la transmitancia acumulada a través del rayo desde t_n hasta t , es decir, la probabilidad de que el rayo viaje desde t_n hasta t sin chocar con ninguna otra partícula. Renderizar una vista requiere estimar esta integral para cada rayo de cámara a través de cada píxel de la cámara virtual deseada.

Se estima numéricamente esta integral continua utilizando cuadraturas, una técnica para aproximar integrales definidas que no pueden resolverse de forma analítica. Se utiliza un muestreo estratificado donde se divide $[t_n, t_f]$ en N intervalos de manera uniforme y luego se extrae una muestra uniformemente al azar dentro de cada intervalo:

$$t_i \sim U \left[\frac{t_n + (i-1)(t_f - t_n)}{N}, \frac{t_n + i(t_f - t_n)}{N} \right]$$

Se usan estas muestras para estimar $C(r)$:

$$\hat{C}(r) = \sum_{i=1}^N T_i (1 - \exp(-\sigma_i \delta_i)) c_i, \quad \text{donde} \quad T_i = \exp \left(- \sum_{j=1}^{i-1} \sigma_j \delta_j \right)$$

Donde $\delta_i = t_{i+1} - t_i$ es la distancia entre muestras adyacentes. Esta función para calcular $\hat{C}(r)$ a partir del conjunto de valores (c_i, σ_i) es diferenciable y se reduce al modelo tradicional de combinación de imágenes mediante el uso de un canal alfa con valores $\alpha_i = 1 - \exp(-\sigma_i \delta_i)$. En composición alfa tradicional, cada píxel de una imagen tiene un valor de alfa asociado que indica su nivel de transparencia. Puede variar de 0 (transparente) a 1 (opaco).

Ya descritos los conceptos clave necesarios para modelar una escena y renderizar nuevas vistas a partir de esta representación, con el objetivo de mejorar las representaciones, se introducen dos mejoras a la arquitectura:

- **Positional encoding:** alineado con el trabajo de *Rahaman* [35], que muestra que las redes neuronales profundas tienden a sobreajustarse y a tener dificultades para generalizar al trabajar con funciones de demasiada baja frecuencia, se encuentra que operar directamente con *inputs* de coordenadas $xyz\theta\phi$ resulta en renderizados de calidad pobre al representar complejas variaciones de color y geometría. *Rahaman* también muestra que, mapeando los *inputs* a espacios dimensionales más altos utilizando funciones de alta frecuencia antes de pasarlos a la red, se permite una mejor adaptación a datos que contienen variaciones complejas.

La arquitectura NeRF reformula F_Θ como una composición de dos funciones $F_\Theta = F'_\Theta \circ \gamma$, donde γ es un mapeo de las dimensiones reales \mathbb{R} a un espacio dimensional \mathbb{R}^{2L} y F'_Θ es la red MLP. La función de codificación que se utiliza es:

$$\gamma(p) = (\sin(2^0 \pi p), \cos(2^0 \pi p), \dots, \sin((2^{L-1}) \pi p), \cos((2^{L-1}) \pi p))$$

Esta función se aplica a cada una de las tres coordenadas \mathbf{x} , normalizadas entre $[-1, 1]$, y a las tres componentes de las direcciones de vista en coordenadas cartesianas \mathbf{d} , las cuales también están normalizadas por defecto en el intervalo $[-1, 1]$. Se asigna $L = 10$ por defecto para $\gamma(x)$ y $L = 4$ para $\gamma(d)$.

- **Hierarchical Volume Sampling:** la razón detrás de la introducción de esta técnica en la arquitectura es lo ineficiente que resulta evaluar de manera densa la escena en N puntos en cada rayo de cámara, siendo N el número de puntos que se introduzcan en la red de cada rayo. Esto se debe a que la mayoría de los puntos de una escena suelen capturar espacio libre y regiones ocultas que no contribuyen a la imagen renderizada. Se utiliza el trabajo previo de *Marc Levoy* [36] para diseñar una representación jerárquica que coloca las muestras de manera proporcional a la importancia esperada que tendrá en el renderizado final.

Para ello, en lugar de optimizar solo una red, se optimizan dos de manera simultánea: una gruesa y una fina. Primero se selecciona un conjunto de N_c posiciones utilizando el procedimiento de muestreo de rayos explicado anteriormente, y se evalúa la red gruesa con ellas. Con el *output* de esta red, se produce una nueva selección de puntos a lo largo de cada rayo para representar las partes más relevantes del volumen. Para ello, se reescribe el color compuesto de la red gruesa $\hat{C}_c(r)$ como la suma ponderada de todos los colores muestreados c_i , a lo largo del rayo:

$$\hat{C}_c(r) = \sum_{i=1}^{N_c} w_i c_i, \quad \text{donde } w_i = T_i(1 - \exp(-\sigma_i \delta_i)).$$

3.1.2. Arquitectura de red

En la Figura 3.2 está representada la arquitectura de la red neuronal profunda que utiliza NeRF para optimizar sus renderizados.

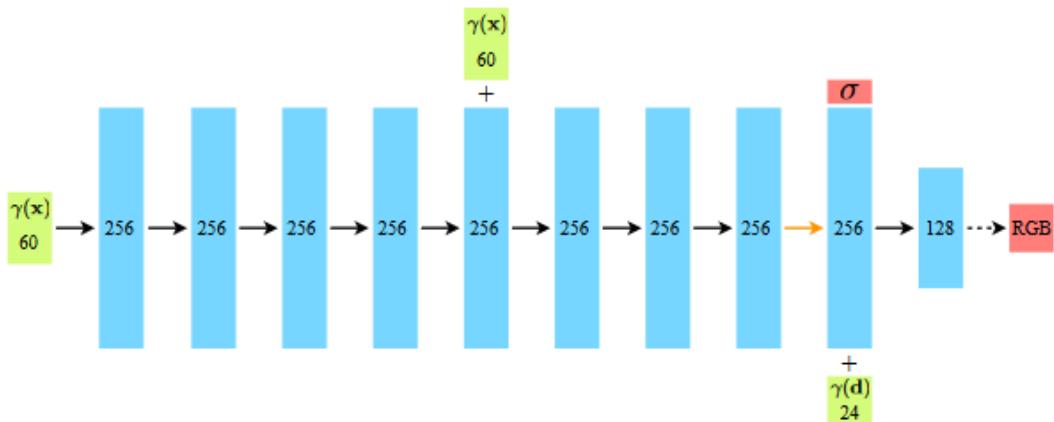


Figura 3.2. Arquitectura de red *fully connected* de NeRF [4].

Los vectores *input* se muestran en verde, las capas ocultas en azul y los vectores de salida en rojo. Los números dentro de cada bloque representan las dimensiones del vector. Las flechas negras indican capas con activaciones ReLU, las flechas naranjas indican capas sin activación, las flechas negras punteadas indican capas con activación sigmoide, y el símbolo + denota concatenación de vectores.

La codificación posicional de la ubicación de entrada $\gamma(\mathbf{x})$ atraviesa un conjunto de 8 capas ReLU completamente conectadas, cada una con 256 canales. Se incluye una conexión de salto que concatena esta entrada a la activación de la quinta capa. Una capa adicional produce la densidad del volumen σ , que se rectifica usando ReLU para asegurar que la densidad del volumen de salida no sea negativa, y un vector de características de 256 dimensiones. Este vector de características se concatena con la codificación posicional de la dirección de visualización de entrada $\Theta(\mathbf{d})$, y se procesa mediante una capa ReLU *fully connected* adicional con 128 canales. Una capa final,

con activación sigmoide, produce la radiancia RGB emitida en la posición \mathbf{x} , vista por un rayo con dirección \mathbf{d} [4].

3.1.3. Flujo de trabajo de NeRF

NeRF puede trabajar con varios tipos de *dataset*, que presentan las imágenes y poses de cámara, necesarios para el funcionamiento de la red, de diferente forma. En esta memoria, por ser el tipo de *dataset* más utilizado en las pruebas realizadas, nos centraremos en el funcionamiento de la red para *Local Light Field Fusion* (LLFF) [37], una técnica basada en la fusión de campos de luz locales para generar nuevas vistas de una escena a partir de múltiples imágenes capturadas desde diferentes ángulos. En el contexto de NeRF, el *dataset* LLFF se utiliza como un conjunto de datos de referencia para entrenar y evaluar modelos. Esto es posible porque ambos métodos comparten la necesidad de datos multivista para generar nuevas vistas sintéticas de una escena.

La Figura 3.3 representa un diagrama del flujo de trabajo de NeRF. A continuación, se comenta en detalle qué procesos se llevan a cabo en su entrenamiento y su uso:

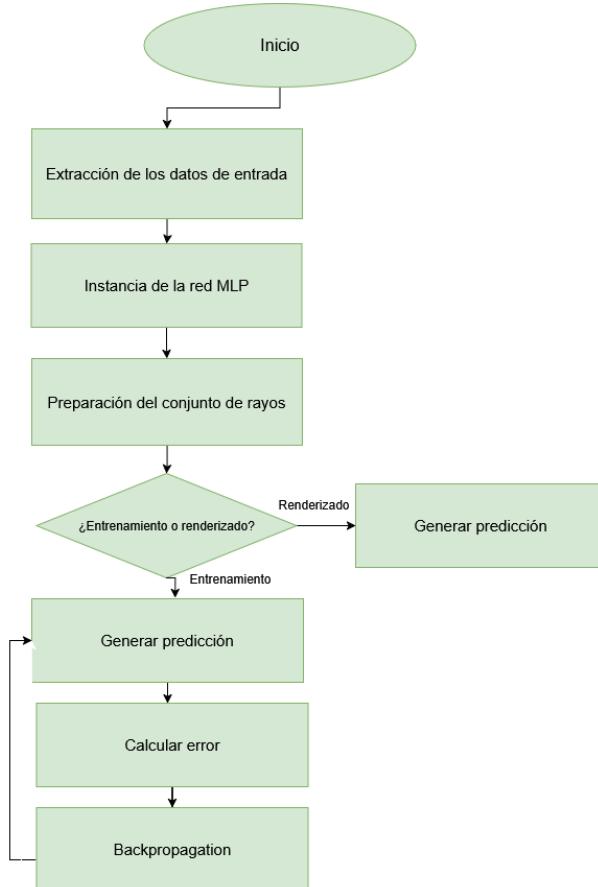


Figura 3.3. Diagrama de flujo de NeRF.

- **Extracción de los datos de entrada:** en primer lugar, se extraen las poses de cámara,

que se organizan en una matriz que contiene, para cada pose, su matriz de rotación, que describe cómo la cámara está rotada respecto al marco de referencia global, el vector de traslación, que va a definir la posición de la cámara en el espacio en relación con la referencia, e información adicional como la escala de la escena. Después se extraen los límites (cerca y lejos) de la escena del archivo cargado, que van a definir las barreras en las que se genere el modelo, es decir, acotan el espacio tridimensional que NeRF va a evaluar. Después, utilizando *Imageio* [38], una biblioteca de Python que permite leer y escribir una amplia variedad de formatos de imagen, se extraen las imágenes del *dataset* y se organizan en matrices.

Después de diferentes reajustes en las matrices, se definen poses para el patrón en espiral que se utilizará para generar vistas nuevas en la fase de *testing*. Además, se extraen las imágenes del *dataset* que se utilizarán también en la fase de *testing*, para que los resultados puedan ser comparadas con los datos originales y extraer la pérdida.

- **Generación de la matriz K :** en el modelo de cámara pinhole, la matriz K o matriz de parámetros intrínsecos, descrita en la Subsección 2.3.1 se utiliza para transformar las coordenadas de un punto en el espacio tridimensional a coordenadas en el plano de la imagen.
- **Creación del modelo NeRF:** donde se instancia el modelo MLP de NeRF.

Primero, se crea una función que aplicará la codificación posicional a un tensor de entrada, generando características adicionales para cada coordenada x, y, z.

Después, crea un modelo de red NeRF para la red gruesa y otro para la red fina, explicado anteriormente. Utilizando *Torch* [39], una biblioteca de computación científica con soporte para aprendizaje automático, *PyTorch* [40], una biblioteca de código abierto basada en *Torch*, y *PyTorch.nn*, que proporciona capas de red predefinidas, funciones de activación y diferentes herramientas de modelado, se genera la arquitectura de red descrita en la Subsección 3.1.2.

Finalmente, se crea el optimizador de tipo Adam [41], y se crean diccionarios que se utilizarán en el entrenamiento y fase de pruebas, que contienen los modelos creados y diferentes parámetros relevantes, como el número de partes en las que se dividirán los rayos de la escena y cuyos puntos definidos posteriormente servirán como entrada para la red.

- **Preparación del conjunto de rayos:** utilizando las dimensiones de las imágenes, la matriz K y las poses de cámara, se obtiene la dirección de los rayos que pasan a través de los píxeles de cada imagen, calculando, para cada uno, la coordenada x e y de la dirección del rayo con K . Después, con la componente de rotación de las poses, se describe cómo la rotación afecta las direcciones del rayo obteniendo, finalmente, un vector de dirección de los rayos de luz de tres dimensiones para cada píxel de la cuadrícula. Además, a partir de

las poses de cámara, se define la matriz de transformación, que representa la posición de la cámara en el espacio tridimensional, necesaria para procedimientos posteriores.

Obtenidos los rayos, se crea una matriz que contiene todos los píxeles de las imágenes que se van a utilizar en el entrenamiento, con su matriz de transformación, el vector de dirección del rayo que lo atraviesa y su RGB. De esta matriz se irán extrayendo fragmentos que serán imprescindibles en el entrenamiento de la red.

- **Entrenamiento de la red:** para cada iteración del entrenamiento se escoge un número predefinido de píxeles del conjunto antes mencionado, separando los rayos del color del píxel, que se utilizará para comparar el resultado obtenido con el original.

En este punto, para cada rayo del conjunto, las tres componentes del vector de dirección de cada rayo se multiplica por, generalmente , 64 espacios generados a partir de un tensor de 64 números equiespaciados desde el punto límite inicial y final de la escena, extraídos anteriormente, al que se le introduce ruido y se expande a las dimensiones del conjunto de rayos de entrada, consiguiendo los desplazamientos a lo largo de los rayos. Después, se suman las posiciones de cámara a los desplazamientos calculados, resultando en las coordenadas tridimensionales de todos los puntos a lo largo de los rayos de dirección, habiendo tenido en cuenta la posición de la cámara y los desplazamientos a lo largo de los rayos. Estos puntos sirven de *input* a la red neuronal creada anteriormente.

A estos puntos, junto con sus direcciones de vista, se les aplica la codificación posicional, resultando en 63 características de posición y 27 de dirección de vista para cada punto, con los parámetros estándar. Estos puntos entran a la red gruesa y pasan por las capas, produciendo una salida que contiene la predicción del RGB y de la densidad volumétrica para cada píxel del conjunto.

Con estos resultados brutos se van a generar diferentes conjuntos de valores con sentido semántico, como un mapa RGB, realizando la integración volumétrica de los colores a lo largo de cada rayo, considerando la densidad y transparencia acumulada de cada punto muestreado. También uno de profundidad calculando, para cada rayo, una media ponderada de las distancias de los puntos muestreados, considerando sus densidades. Además, los llamados *weights* o pesos, donde cada valor representa el producto acumulativo de los elementos anteriores a lo largo de cada rayo, siendo elementos los diferentes puntos en los que se ha dividido anteriormente, es decir, cuánto contribuye al color final cada punto que se ha pasado por la red. Estos valores son relevantes para determinar qué puntos se van a evaluar en la red fina.

Se calcula la función de de densidad de probabilidad (PDF) a partir de los pesos, lo que proporciona una representación probabilística de la contribución de cada punto a la imagen renderizada. Con esto se calcula la función de distribución acumulativa (CDF). Después de un proceso de sampleo jerárquico, utilizando estas distribuciones, se consiguen las muestras más representativas del conjunto que se introducen en la red fina, que tiene igual forma que la gruesa, produciendo el resultado que representa el *output* final de esa iteración.

Con los resultados del RGB predicho y los valores originales se calcula el error utilizando la función de pérdida, se realiza el proceso de *backpropagation* y se procede con la siguiente iteración, hasta que se alcance el número de iteraciones definido en los parámetros.

- **Renderizado de vistas:** Si la red ya ha sido entrenada y solo se busca renderizar imágenes sintéticas a partir de un modelo previamente entrenado, el flujo de trabajo sigue un proceso similar al que seguiría durante el entrenamiento. Para cada imagen completa que se requiera, se realiza el proceso de muestreo, evaluación y renderizado en esa vista: se generan los rayos de cámara a partir de la posición y orientación de la cámara en el espacio 3D, se define el conjunto de puntos a lo largo del rayo del cual se evaluará la función de radiancia, los puntos muestreados en el rayo se pasan a través de la red neuronal NeRF preentrenada, que genera las dos salidas y, después de la integración volumétrica, se genera una imagen completa para esa perspectiva.

3.2. DS-NeRF

Depth Supervised-NeRF [42] es una extensión de la técnica NeRF que incorpora información de profundidad para mejorar la calidad de la reconstrucción 3D. Incluye mapas de profundidad que proporcionan información adicional sobre las distancias desde la cámara hasta los objetos en la escena. Estos mapas se utilizan como datos de supervisión adicionales durante el entrenamiento de la red neuronal.

La información de profundidad viene de COLMAP [43]. NeRF ya utiliza esta solución de SfM para estimar las poses de cámara y demás parámetros necesarios para la reconstrucción. Sin embargo, COLMAP, por defecto, también devuelve una nube de puntos de baja densidad con sus errores de reproyección correspondientes. NeRF resuelve implícitamente un problema de correspondencia entre múltiples vistas. No obstante, una versión de baja densidad de este problema ya ha sido resuelta por SfM, cuya solución es devuelta mediante los puntos clave en 3D de COLMAP. Por lo tanto, la supervisión de la profundidad aporta una ampliación de la búsqueda de la correspondencia utilizando estos *keypoints* explícitos. En la Figura 3.4 se observa una representación de una nube de puntos *sparse* que devuelve COLMAP.

3.2.1. Fundamentos Teóricos

Partiendo de la explicación de la Subsección 3.1.1, en la que se explica que, dados un punto en 3D $\mathbf{x} \in \mathbb{R}^3$ y una dirección de vista específica $\mathbf{d} \in \mathbb{R}^3$, NeRF aprende una función implícita F que estima la densidad volumétrica $\sigma(x)$ y el color RGB c de forma $F_\Theta : (\mathbf{x}, d) \rightarrow (c, \sigma)$. Para renderizar una pose dada \mathbf{P} , se lanzan rayos r desde del centro de proyección de \mathbf{P} , o , en la dirección \mathbf{d} derivada de sus parámetros intrínsecos. Se integra el campo de radiancia implícita a lo largo del rayo para calcular la radiancia de cada objeto que se encuentre en \mathbf{d} de la forma



Figura 3.4. Nube de puntos de densidad baja [44].

$$\hat{C} = \int_0^\infty T(t)\sigma(t)c(t) dt.$$

La expresión $h(t) = T(t)\sigma(t)$ representa una distribución de probabilidad continua sobre la distancia de rayo t que describe la probabilidad de que el rayo termine en t . Por cuestiones prácticas, NeRF asume que la escena se encuentra entre una barrera cercana y una lejana t_n, t_f . Para asegurarse de que $h(t)$ suma 1, la implementación de NeRF a menudo trata t_f como un muro opaco. Con esta definición, el color renderizado puede expresarse como una expectativa:

$$\hat{C} = \int_0^\infty h(t)c(t) dt = \mathbb{E}_{h(t)}[c(t)]$$

La distribución $h(t)$ describe la contribución de muestras de radiancia a lo largo del rayo para lograr el valor final. La mayoría de escenas consisten en espacio vacío y superficies opacas que restringen la contribución a cortarse a partir de la superficie más cercana. Esto implica que la distribución de rayo ideal de un punto de una imagen con el objeto más cercano con profundidad \mathbf{D} debe ser $\delta(t - \mathbf{D})$. Como muestra la Figura 3.5, la varianza empírica que cuantifica la variabilidad de un conjunto de datos observados disminuye cuantas más vistas de entrenamiento en las distribuciones de terminación de rayos. Por lo tanto, se entiende que los modelos entrenados con muchas vistas tienden a generar distribuciones de rayos que se acercan a la función δ . Esto apoya la idea de supervisión de la profundidad de terminación de los rayos [42].

Como ya se ha mencionado, NeRF necesita imágenes con las matrices de cámara asociadas (P_1, P_2, \dots), estimados normalmente con paquetes de SfM como COLMAP [43], que también devuelven puntos clave 3D $\{\mathbf{X} : x_1, x_2, \dots \in \mathbb{R}^3\}$ con referencias para qué puntos se ven desde cada cámara $j : X_j \subset X$. Dada la imagen I_j y su cámara P_j , se estima la profundidad de los puntos clave visibles $x_i \in X_j$ proyectando x_i con P_j , tomando el valor z reproyectado como la profundidad estimada del punto clave D_{ij} .

Por cuestiones de ruido en las imágenes, o en errores de renderizado de COLMAP, D_{ij} es un parámetro ruidoso. Se puede medir la fiabilidad de un punto clave x_i utilizando el error de reproyección $\hat{\sigma}_i$ en las vistas donde aparece el punto. En DS-NeRF, se modela la localización de la primera superficie interceptada por un rayo como una variable aleatoria \mathbb{D}_{ij} que está distribuida

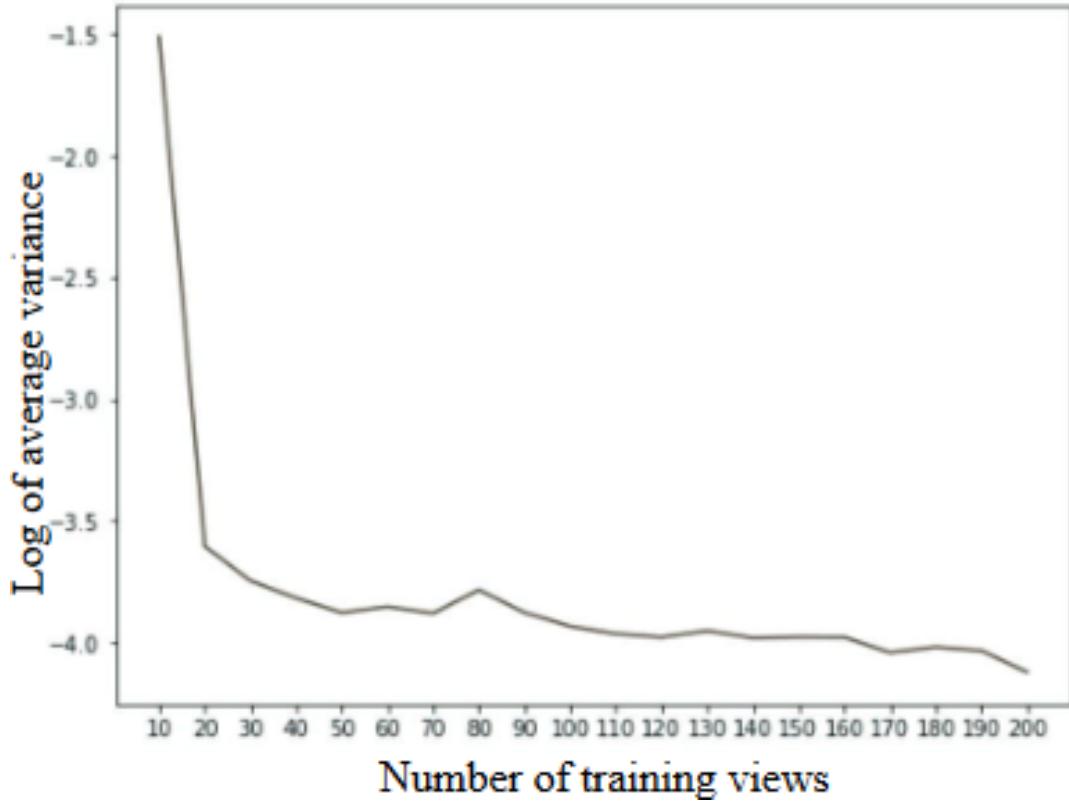


Figura 3.5. Logaritmo de la varianza promedio con diferente número de vistas [42].

alrededor de la profundidad estimada por COLMAP D_{ij} con varianza $\hat{\sigma}_i : \mathbb{D}_{ij} \sim \mathcal{N}(D_{ij}, \hat{\sigma}_i)$. El objetivo es minimizar la divergencia KL entre la distribución del rayo renderizado $h_{ij}(t)$ correspondiente a las coordenadas de imagen de x_i y la estimación de la profundidad ruidosa. La siguiente expresión permite que la distribución de profundidad o de terminación de rayos $h(t)$ se entrene con la supervisión de la profundidad de COLMAP:

$$\mathbb{E}_{\mathbb{D}_{ij}} \text{KL}[\delta(t - \mathbb{D}_{ij}) \| h_{ij}(t)] = \text{KL}[\mathcal{N}(D_{ij}, \hat{\sigma}_i) \| h_{ij}(t)] + \text{const}$$

En cuanto al error de distribución de rayos, o error de profundidad, se calcula mediante la siguiente expresión:

$$L_{\text{Depth}} = \mathbb{E}_{x_i \in X_j} \int \log h(t) \exp \left(-\frac{(t - D_{ij})^2}{2\hat{\sigma}_i^2} \right) dt \approx \mathbb{E}_{x_i \in X_j} \sum_k \log h_k \exp \left(-\frac{(t_k - D_{ij})^2}{2\hat{\sigma}_i^2} \right) \Delta t_k$$

El error total se expresa como $L = L_{\text{Color}} + \lambda_D L_{\text{Depth}}$ donde λ_D es un hiperparámetro que controla el balance entre la supervisión del color y de la profundidad.

En el artículo original de DS-NeRF [42] se encuentra la explicación completa y los experi-

mentos realizados por los autores.

3.2.2. Flujo de trabajo de DS-NeRF

Dado que DS-NeRF parte de la implementación *Pytorch* de NeRF, de la que se habla en la Subsección 3.1.3, se comenta en este apartado los cambios realizados que permiten la supervisión de la profundidad en el entrenamiento. En la Figura 3.6 se muestra el diagrama de flujo. En azul están representados los procesos que han sido modificados para introducir este nuevo concepto en el entrenamiento de las redes. Se comenta, a continuación, en qué consisten exactamente los cambios realizados:

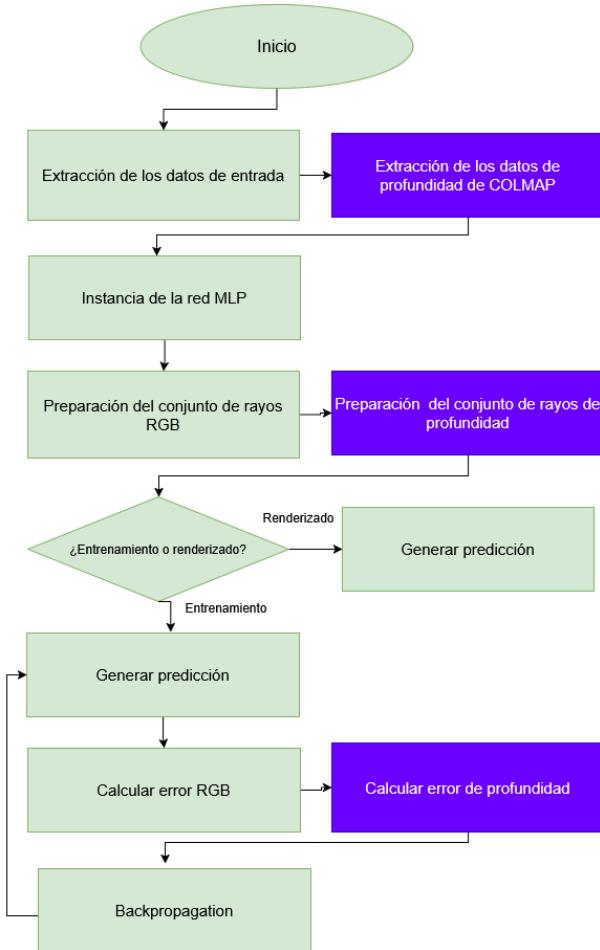


Figura 3.6. Diagrama de flujo de DS-NeRF.

- **Extracción de los datos de COLMAP:** en esta implementación, lo primero es extraer los datos que devuelve COLMAP en forma de ficheros binarios.

Primero se lee el archivo *images.bin* que contiene, para cada imagen del *dataset* un identificador, valores de orientación de la cámara en el espacio, valores que representan la posición de la cámara en el espacio 3D y, lo más importante, dónde en la imagen se encuentran los

puntos destacados en la imagen y sus correspondencias con los puntos 3D reconstruidos. Después, se lee el *points3D.bin*, que contiene un identificador único para cada punto 3D, tres valores que representan la posición del punto en el espacio, tres valores para el color del punto, su error de reproyección y una lista que contiene qué imágenes y desde qué píxel de cada imagen es visible.

Una vez obtenidos los datos de COLMAP, se va a obtener la profundidad de cada punto con respecto a las cámaras que lo visualizan, de tal forma que, el resultado final sea conocer, para cada imagen, qué tan lejos está cada punto visible desde la cámara a lo largo de su eje de visión (el eje z de la cámara), es decir, la distancia desde la cámara hacia el punto en la dirección en que la cámara está mirando. Para ello, se aplica el siguiente método:

- **Posición relativa del punto 3D:** se resta la posición de la cámara de las coordenadas del punto 3D para obtener un vector que representa la ubicación del punto con respecto a la cámara.
- **Proyección sobre el eje z:** luego, se proyecta el vector resultante sobre la dirección de visión de la cámara (representada por el eje z de la cámara), utilizando el producto punto.
- **Escalado:** finalmente, se ajusta el resultado multiplicando por un factor de escala. Este factor se utiliza en varias ocasiones en NeRF y DS-NeRF para convertir coordenadas reconstruidas por COLMAP a unidades físicas reales (metros), ya que COLMAP no reconstruye escenas directamente en unidades físicas absolutas, como metros, sino que la escala de la escena puede estar en unidades arbitrarias. Por medio de un factor de escala determinado mediante, se supone, una elección empírica, se ajustan las coordenadas arbitrarias de COLMAP a un rango adecuado para unidades reales.

Del proceso anterior se obtiene la siguiente fórmula:

$$\text{depth} = (\mathbf{R}_z^T \cdot (\mathbf{p} - \mathbf{C})) \cdot sc$$

Donde:

- $\mathbf{p} - \mathbf{C}$ es el vector que representa el desplazamiento del punto 3D respecto a la cámara.
- \mathbf{R}_z^T es la traspuesta del vector que representa la dirección del eje z de la cámara, y su producto escalar con $\mathbf{p} - \mathbf{C}$ nos da la proyección de este desplazamiento en la dirección z.
- El factor sc ajusta la escala de las coordenadas a unidades reales.

Una vez mapeados estos puntos, se van incluyendo en una lista de diccionarios de *Python* en el que se relaciona, para cada imagen del *dataset*, el valor de la profundidad de cada uno, con sus coordenadas y sus errores de reproyección.

- **Uso de la distancia focal:** para esta arquitectura, no se utiliza la matriz K para calcular los rayos de la cámara, sino que se utiliza directamente la distancia focal. Este método es más simple y eficiente, pero menos flexible y preciso.
- **Preparación del conjunto de rayos:** en el proceso de generar los rayos de cámara y preparar el conjunto del que se irán extrayendo fragmentos, para los puntos RGB que se extraen de las imágenes, se realiza de la misma forma. Para los puntos extraídos de los archivos de COLMAP, se lleva a cabo un proceso similar, en el que, para cada punto se obtienen el vector de dirección de los rayos de luz de tres dimensiones y la posición de la cámara en el espacio tridimensional. Estos datos se juntan con, esta vez, el valor de la profundidad y se incluyen en una lista.
- **Entrenamiento de la red:** para el entrenamiento de la red, se vuelven a tomar diferentes *batches* de puntos en cada iteración. Sin embargo, esta vez, se van a juntar en el mismo *batch* puntos extraídos de las imágenes, de los que se conoce su color, con puntos extraídos de COLMAP, de los que se conoce su valor de profundidad. A la salida de la red, una vez obtenidos los valores predichos por la red, se obtiene un valor de pérdida comparando el RGB predicho con el real, y otro valor de pérdida comparando la profundidad predicha con la que se obtiene de COLMAP. Para el cálculo de la pérdida total, se introduce un parámetro que indica qué porcentaje de la pérdida de profundidad se le quiere agregar a la pérdida de color. Por defecto, se pone a 0.1.

Capítulo 4

Especificaciones y restricciones de diseño

A continuación se exponen las distintas especificaciones del proyecto:

- La solución debe estar desarrollada mediante las técnicas de *Machine Learning*.
- La solución desarrollada debe obtener representaciones de escenas en tres dimensiones y cualquier vista de la escena en dos dimensiones a partir de imágenes de color e imágenes de profundidad.
- La calidad del resultado debe ser, al menos, lo más parecido posible a lo obtenido mediante las técnicas NeRF y DS-NeRF en cuanto a tiempo de entrenamiento necesario y calidad del modelo final.
- El formato de los resultados debe proporcionar suficiente información para realizar la comparación con los actuales.

Capítulo 5

Descripción de la solución desarrollada

La solución propuesta se basa en las tecnologías recién descritas NeRF, que es la primera que surgió, y DS-NeRF, que ya es una modificación de la original. El objetivo es el mismo que el segundo, mejorar la reconstrucción de escenas 3D mediante NeRFs utilizando supervisión de profundidad.

DS-NeRF pretende optimizar la calidad de las reconstrucciones tridimensionales, reduciendo las inconsistencias geométricas y acelerando la convergencia del entrenamiento, al incorporar información de mapas de profundidad como una fuente adicional de supervisión. La solución que se describe en este apartado no va a utilizar la profundidad de COLMAP en forma de mapas de baja densidad, sino mediante imágenes de profundidad tomadas con cámaras Microsoft Azure Kinect DK [5], que utilizan tecnología de tiempo de vuelo (ToF) para medir la distancia entre el sensor y los objetos de la escena, donde cada píxel en la imagen de profundidad representa la distancia entre la cámara y el objeto o superficie capturada por éste en milímetros.

Como se ha explicado en apartados anteriores, DS-NeRF se basa principalmente en el uso de información de profundidad de COLMAP, que genera una representación tridimensional de la escena mediante un conjunto disperso de coordenadas con datos de distancias y errores de reproyección , es decir, una nube de puntos de baja densidad, formada tomando el conjunto de imágenes RGB de la secuencia en cuestión y, a través del análisis de coincidencias entre las imágenes, se reconstruye un modelo 3D de la escena. Este conjunto sirve como ampliación de la búsqueda de la correspondencia. El problema que se ha observado es la escasez de información que proporcionan estos mapas. En el caso del *dataset* preparado para este proyecto, 4163 puntos en tres dimensiones, que si bien pueden ser útiles para conjuntos de datos limitados, representan una pequeña porción del total de la profundidad. Con esta nueva información, y considerando el margen de error asociado, se podrá tener una visión más amplia de las distancias en la escena. . Millones de puntos cuya supervisión en el entrenamiento de redes NeRF permitirán comprobar

si la idea de supervisar la profundidad realmente puede representar una mejora sobre el modelo original.

En este trabajo de Fin de Grado, se presentan dos modificaciones a DS-NeRF: la primera, expuesta en la Subsección 5.2, parte directamente de DS-NeRF y la segunda, expuesta en la Subsección 5.3, parte del código original de NeRF. El objetivo de ambas es el recién descrito, pero llevado a cabo de diferente manera, y con unas características finales diferentes. En el Capítulo 6 se comparan los resultados obtenidos con ambas modificaciones y se determina cuál es el más eficaz.

5.1. Información del *dataset* con imágenes de profundidad

Para entrenar modelos con las modificaciones propuestas, se ha utilizado un conjunto de datos tomados en el CITSEM capturado con cámaras Microsoft Azure Kinect DK [5], que consiste en 8 imágenes tomadas de una escena en RGB con formato PNG, de dimensiones 1920x1080, y sus correspondientes imágenes de profundidad, donde cada píxel contiene información sobre la distancia de un objeto al dispositivo, tomadas desde los mismos ángulos y en formato YUV, que, aunque habitualmente se emplea para imágenes de vídeo en color, puede almacenar los valores de profundidad en escala de grises utilizando solamente el canal de luminancia **Y** y almacenando los valores con una precisión de 16 bits, lo que proporciona un rango de hasta 65,536 niveles en cada píxel, permitiendo una representación detallada de la distancia en la escena [45].

Las imágenes a color han sido procesadas por COLMAP y han generado los archivos necesarios para poder ser utilizadas en NeRF: los archivos *cameras.txt*, *images.txt*, y *points3D.txt*, cuyas características y uso en el código está explicado en la Subsección 3.2.2.

Una vez procesadas, para conseguir que el *dataset* pueda entrar en el tipo LLFF y sus datos puedan ser extraídos para el entrenamiento, es necesario generar el archivo *poses_bounds.npy*, que va a contener las posiciones y orientaciones de las cámaras desde las cuales se capturan las imágenes de la escena, es decir, las matrices de rotación y traslación de cada cámara, y los límites de la escena, que definen el rango de profundidad o distancia en la escena capturada. Para ello, los autores de DS-NeRF [46] incluyen un código en el que, incluyendo el directorio donde se guarda el *dataset*, genera el archivo utilizando los datos proporcionados previamente por COLMAP.

Después de este proceso, se tiene elconjunto de 8 imágenes RGB, sus pares de profundidad, las matrices de poses y límites de escena necesarios para que el *dataset* pueda ser utilizado para entrenar modelos NeRF, DS-NeRF y las diferentes modificaciones propuestas para este proyecto.

5.2. Modificación sobre DS-NeRF

Esta primera modificación parte directamente de DS-NeRF. Busca introducir la información de profundidad de las imágenes en formato YUV de la misma forma que se introduce la extraída de COLMAP y que, finalmente, sea compatible utilizar ambas profundidades en el entrenamiento del modelo. En la Figura 5.1 se muestra el diagrama de flujo con las nuevas implementaciones necesarias para el funcionamiento de esta nueva variación. En azul están representados los procesos añadidos para DS-NeRF, como se explica en la Subsección 3.2.2, mientras que en rojo se indican los implementados para este proyecto. A continuación, se explica en detalle en qué consisten éstos:

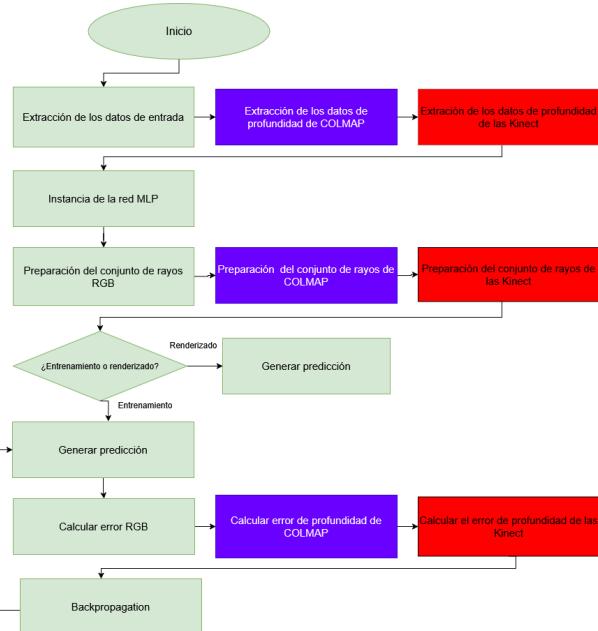


Figura 5.1. Diagrama de flujo de NeRF.

- **Extracción de los datos de profundidad de las Kinect:** Después de extraer los datos de la misma forma que se explica en la Subsección 3.2.2, se introduce la extracción, calibración y procesamiento de los datos de las imágenes de profundidad arrojados por las cámaras Kinect para que puedan ser introducidos en la red. Para ello se han codificado dos funciones:

- ***read_yuv_file*:** lee un archivo YUV binario que contiene información de profundidad codificada en 16 bits por píxel. Su tarea principal es convertir los datos binarios del archivo en una estructura *NumPy* 2D ($height \times width$) que representa la imagen de profundidad. Para ello, calcula el tamaño total de cada fotograma en función de las dimensiones de la imagen (ancho y alto), lee los datos binarios del archivo YUV y convierte estos datos en un array *NumPy* de tipo uint16, que refleja los valores

de profundidad capturados por la cámara Kinect en formato YUV. Finalmente, da forma a los datos en una matriz 2D que representa los valores de profundidad para cada píxel de la imagen.

- ***load_yuv_depth:*** tiene como objetivo cargar múltiples imágenes de profundidad desde archivos YUV en un directorio, convertirlas en arrays *NumPy* y procesarlas para obtener las coordenadas 2D de los píxeles y sus respectivas profundidades. La salida final es una estructura lista para ser utilizada en la estructura específica de DS-NeRF para la reconstrucción 3D. Para ello, en primer lugar y utilizando *read_yuv_file*, lee todas las imágenes y las almacena en una lista.

Seguido de leer las imágenes de profundidad, viene la normalización de los datos, imprescindible para que el resultado final sea coherente. Tiene como objetivo hacer coincidir los valores de profundidad obtenidos de dos fuentes diferentes: las imágenes de profundidad capturadas por la cámara Kinect y las distancias estimadas por COLMAP. Ambas tecnologías entregan la información en escalas distintas. Es crucial unificarlas para garantizar que la profundidad obtenida de ambas fuentes sea concordante, permitiendo su uso de manera efectiva en la reconstrucción 3D, que requiere datos lo más precisos y consistentes posible. El procedimiento que se ha seguido involucra la comparación directa de distancias en píxeles específicos que han sido estimados por ambas fuentes:

- En primer lugar, se seleccionan píxeles comunes en las imágenes donde tanto COLMAP como la Kinect han proporcionado una estimación de profundidad. Esto es posible, como se explica en la Subsección 3.2.2, ya que los datos de COLMAP, inicialmente en tres dimensiones, se modifican para obtener la distancia de cada punto con respecto a las cámaras que lo visualizan, cuyas posiciones coinciden con las de las imágenes tomadas por las Kinect.
- Para cada píxel seleccionado, se registran las profundidades proporcionadas por la cámara Kinect, que proporciona datos en milímetros, y por COLMAP, cuyos valores están en su propia escala, influenciada por varios factores de la reconstrucción de la escena.
- Se observa que la profundidad generada por la Kinect y COLMAP puede diferir únicamente por un factor de escala S . Esto significa que, en promedio, la relación entre las profundidades estimadas por COLMAP y Kinect sigue la fórmula:

$$D_{\text{COLMAP}} \approx S \cdot D_{\text{Kinect}}$$

Donde D_{COLMAP} es la profundidad estimada por *COLMAP*, y D_{Kinect} es la profundidad obtenida por la cámara Kinect.

- Para obtener el factor de escala S se calcula la razón promedio entre los valores de profundidad correspondientes entre COLMAP y la Kinect para los píxeles seleccionados:

$$S = \frac{\sum_i D_{\text{COLMAP},i}}{\sum_i D_{\text{Kinect},i}}$$

Donde S representa el ajuste necesario para convertir las profundidades capturadas por la Kinect a la escala de COLMAP.

- Finalmente, la normalización de las imágenes de profundidad de la Kinect se realiza dividiendo los valores de profundidad por el factor de escala obtenido previamente de la forma:

$$D'_{\text{Kinect}} = \frac{D_{\text{Kinect}}}{S}$$

Donde D_{Kinect} son los valores de profundidad originales capturados por la Kinect, ya convertidos previamente a metros dividiéndolos por 1000, y D'_{Kinect} son los valores de profundidad ajustados a la escala de *COLMAP*.

Una vez están normalizados los datos de profundidad, se generan dos matrices i y j con las coordenadas 2D de los píxeles (índices x e y), utilizando *np.meshgrid*. Estas coordenadas representan la ubicación de cada píxel en la imagen. Estas matrices se expanden y repiten para que coincidan con la cantidad de imágenes cargadas, generando arrays que contienen las coordenadas 2D de cada píxel para cada imagen. Despues, se combinan con los valores de profundidad en un array donde cada elemento contiene las coordenadas 2D de un píxel y su profundidad correspondiente. Esto crea una estructura de datos (i , j , profundidad) para cada píxel en cada imagen.

Posteriormente, se crea una máscara para cada imagen, que filtra los píxeles donde la profundidad no es cero. Esto es importante, ya que los píxeles con profundidad cero representan zonas sin información válida. Se seleccionan solo los píxeles válidos (aquellos con una profundidad distinta de cero), y se almacenan tanto las coordenadas 2D como las profundidades en:

- Una lista que contiene los puntos válidos de cada imagen en forma de (i , j , profundidad), posteriormente convertida a un array de objetos.
- Otra lista de diccionarios que, imitando la estructura de datos de profundidad de DS-NeRF, separa las coordenadas y profundidades para cada imagen.

La función retorna estos dos objetos recién mencionados. Posteriormente se hará uso de ambos en el entrenamiento de la red.

Con estas funciones se consigue el objetivo de adaptar los datos que proporcionan las Kinect para que tengan la misma forma y, por lo tanto, puedan ser utilizados de la misma manera, que los datos de profundidad que ya nos daba COLMAP en DS-NeRF.

- **Preparación del conjunto de rayos de las Kinect:** Este paso se implementa después de obtener los conjuntos de rayos RGB y de profundidad de COLMAP, tal y como se

explica en la Subsección 3.1.3 para los primeros y en la Subsección 3.2.2 para los rayos de profundidad.

El procedimiento será idéntico al llevado a cabo con la profundidad de COLMAP. Se calculan los rayos que pasan por los píxeles con información de profundidad, utilizando las coordenadas y la pose de la cámara, se combinan estos rayos con los valores de profundidad correspondientes y se almacenan los rayos generados en una lista para, al finalizar, concatenar todos los rayos en un solo array *yuv_rays_depth*. Finalmente, se barajan aleatoriamente los rayos del array con *np.random.shuffle*, preparando los datos para su uso posterior en entrenamiento o procesamiento de imágenes sintéticas.

El resultado final, después de añadir este procedimiento, son tres arrays *rays_rgb*, *rays_depth* y *yuv_rays_depth*. Cada array relaciona rayos de cámara con píxeles de, o bien imágenes RGB, como en el caso de *rays_rgb*, de tal forma que el color del píxel correspondiente representa el color del objeto en la escena que el rayo intersecta, o bien de imágenes de profundidad, como en los otros dos arrays, de tal forma que la profundidad mide cuán lejos está el píxel del objeto o superficie a lo largo del rayo.

- **Calcular error de profundidad de las Kinect:** Una vez definidos los rayos de cámara y agrupados en los respectivos arrays, los siguientes procedimientos, dentro ya del entrenamiento de la red neuronal, incluirán reunir conjuntos de muestras o *batches*, tomadas de los tres arrays recién descritos y agrupándolos en el mismo conjunto, separar los rayos que, posteriormente, servirán para definir puntos a lo largo de éstos cuyas coordenadas tridimensionales servirán como *inputs* para la red neuronal, de su respectiva relación de color o profundidad, que se utilizará como *target* con el que comparar el resultado arrojado por la red.

Una vez la red ha producido una salida, de nuevo, se separan los resultados según su procedencia y se calcula, para cada caso, el error cuadrático medio entre el valor predicho y el real. Cada error se agrega a la pérdida total, y se lleva a cabo el proceso de *backpropagation*.

Sintetizando lo anterior, el proceso de entrenamiento de la red tiene la misma estructura que el de DS-NeRF, que ya era muy similar al de NeRF, pero añadiendo la nueva información en forma de millones de valores de profundidad de la escena conocidos, procesados y adaptados a la estructura y la escala de NeRF, con los que se espera que la red produzca una reconstrucción más precisa y detallada de la escena.

5.3. Modificación sobre NeRF

Esta segunda modificación se ha implementado sobre NeRF, por lo que no va a utilizar la profundidad de COLMAP ni la estructura de datos que genera . En este caso relacionan los píxeles de cada imagen con su RGB y, como novedad, con su profundidad, extraída de

las imágenes tomadas por la Kinect. En la Figura 5.2 se muestra el diagrama de flujo con la implementación para esta versión en color rojo:

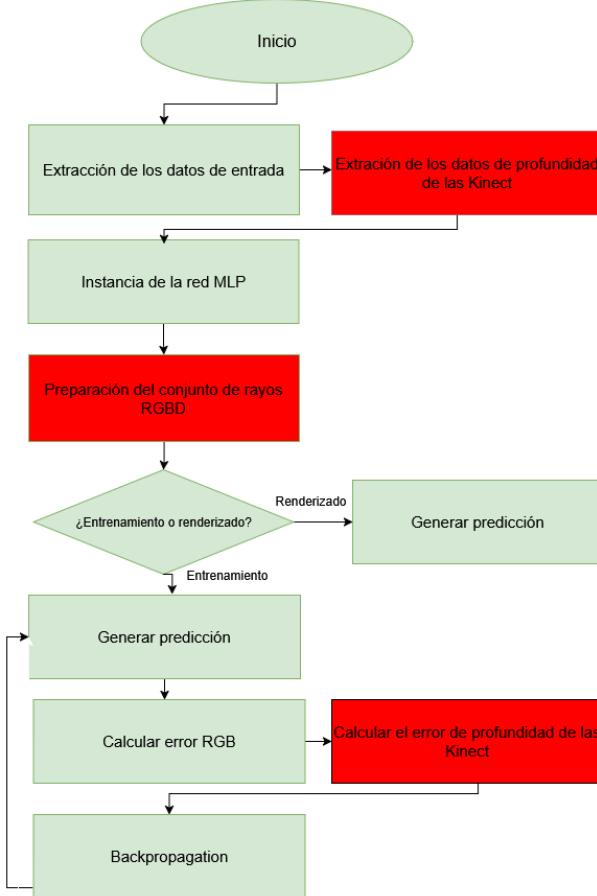


Figura 5.2. Diagrama de flujo de NeRF.

- **Extracción de los datos de profundidad de las Kinect:** una vez extraídos los datos de entrada como se describe en la Subsección 3.1.3, se procede a extraer los datos de profundidad de la Kinect de manera similar a la detallada en la anterior versión, con el mismo proceso de normalización, pero con variaciones en la salida de la función `load_yuv_depth`: en este caso no se va a utilizar la estructura de datos de profundidad de DS-NeRF, en el que se relacionaban las distancias con las coordenadas de los píxeles correspondientes en forma de una lista de diccionarios. Para esta versión, la salida consiste en una lista `yuv_images` con un array de la forma (profundidad, altura, anchura) para cada píxel de las vistas de la escena, de forma que cada uno tienen asociada la profundidad que ha tomado la Kinect.
- **Preparación del conjunto de rayos RGBD:** así como en la anterior modificación se extendía la idea de crear diferentes conjuntos de rayos de cámara según la procedencia de los datos con los que se relacionan, en este caso, ya que las coordenadas de las vistas para RGB y profundidad de la Kinect son las mismas, se opta por relacionar los mismos

rayos de cámara con ambas medidas al mismo tiempo. Para ello, una vez creada la lista de arrays *rays_rgb* con los rayos de cámara y su color correspondiente, tal y como se explica en la Subsección 3.1.3, se amplían sus dimensiones para que puedan albergar una variable además del RGB, como es el espacio. Utilizando los datos de las Kinect previamente extraídos de forma adaptada para este formato, se consigue relacionar cada rayo de cámara con el color del píxel que atraviesa, así como con su profundidad.

- **Calcular error de profundidad de las Kinect:** para el cálculo de la pérdida, al igual que en las versiones anteriores, se toman muestras de datos del conjunto recién descrito y se separan los rayos del RGB y de la profundidad. Una vez la red ha producido una salida de color y espacio para cada muestra se procede, como en el resto de versiones, a calcular el error cuadrático medio entre la predicción y la realidad. Después de calcular el error de color, se calcula el de profundidad. Teniendo en cuenta que las cámaras Kinect no miden la profundidad en la totalidad de la imagen, dejando un gran porcentaje sin cubrir y que, a diferencia del caso anterior, en este punto no se han manejado estas zonas sin información, .es necesario filtrar los resultados basados en datos irrelevantes para que la red solo aprenda de información válida. Para ello, se genera una máscara booleana que identifica los píxeles o puntos donde la profundidad no es cero mirando el conjunto de valores reales de profundidad del *batch*, y se seleccionan tanto los valores reales como los predichos cuya información es relevante. Con éstos valores se calcula el valor de la pérdida para la profundidad y se añade al error total que se utilizará para el proceso de *backpropagation*.

Este nuevo procedimiento, más simplificado que el anterior, busca mantener la calidad de la reconstrucción mientras se optimiza la velocidad de iteración en el entrenamiento, aprovechando la simplicidad inherente a NeRF. Además, incorpora el aprendizaje de la profundidad de la escena, lo cual es una novedad frente a NeRF, que solo aprende del color. Además, esta implementación mejora los datos de entrada en comparación con DS-NeRF, al no depender de una información espacial limitada.

Capítulo 6

Resultados

6.1. Herramientas y métodos utilizados

Para el entrenamiento y evaluación de modelos NeRF y sus modificaciones, es necesario emplear diversas herramientas y equipos. Se utiliza un ordenador de alta gama con sistema operativo Ubuntu y una tarjeta gráfica NVIDIA GeForce RTX 3090 [47]. La reconstrucción 3D y la renderización basada en NeRF son tareas que requieren una potencia computacional significativa debido a su naturaleza intensiva en términos de cálculo. Por lo tanto, un ordenador de alta gama con una potente CPU y una GPU avanzada resulta esencial para manejar estas cargas de trabajo. Ubuntu, siendo un sistema operativo robusto y ampliamente utilizado en la investigación y el desarrollo de aprendizaje profundo, proporciona un entorno Linux que ofrece flexibilidad y compatibilidad con una amplia gama de bibliotecas y herramientas de software necesarias para el desarrollo de la solución. La tarjeta gráfica NVIDIA GeForce RTX 3090, con sus numerosos núcleos CUDA y una capacidad de memoria VRAM de 24 GB, permite el entrenamiento eficiente de modelos de aprendizaje profundo que requieren grandes cantidades de datos y operaciones de cálculo intensivo. Además, la arquitectura Ampere de la RTX 3090 permite un buen rendimiento en tareas de renderizado y aprendizaje profundo, acelerando significativamente el tiempo de entrenamiento y la inferencia.

Para la captura del *dataset* preparado para este proyecto, se utilizan cámaras de profundidad Microsoft Azure Kinect DK [5], que hacen uso de sensores Time-of-Flight que miden el tiempo que tarda la luz en rebotar desde los objetos hacia la cámara, permitiendo crear mapas de profundidad precisos en 3D con suficiente precisión. También incluyen cámaras RGB para capturar imágenes en color. La compatibilidad de las cámaras Azure Kinect DK con diversos entornos de desarrollo y su soporte para lenguajes de programación como Python y C++ facilita la integración de los datos de profundidad en el *pipeline* de entrenamiento de NeRF. Al pertenecer al centro de investigación CITSEM [48], estas cámaras se encuentran bien mantenidas y calibradas, asegurando la fiabilidad y consistencia de los datos de profundidad capturados, lo

cual es crucial para el entrenamiento de modelos robustos y precisos.

El rendimiento de las tecnologías puestas a prueba y la calidad de los modelos entrenados se van a medir en forma de tiempo de entrenamiento requerido para conseguir las representaciones que se muestran, cuya calidad se medirá utilizando, como medida subjetiva, ejemplos de imágenes de referencia junto con las generadas por NeRF, DS-NeRF, o las modificaciones propias y, como medida objetiva, el *Peak Signal-to-Noise Ratio* (PSNR) [49], métrica utilizada comúnmente para evaluar la calidad de la reconstrucción de una señal comparando la calidad de una imagen comprimida o generada con su versión original, basándose en el error cuadrático medio. Se expresa en decibelios, de manera que una puntuación PSNR más alta generalmente indica que la imagen generada tiene mayor calidad y está más cerca de la imagen de referencia. Se calcula como:

$$PSNR = 10 \cdot \log_{10} \left(\frac{MAX_I^2}{MSE} \right) \quad (6.1)$$

Donde:

- **MAX_I**: Es el valor máximo posible de un píxel en la imagen. Para imágenes de 8 bits por canal, este valor es 255.
- **MSE (Mean Squared Error)**: Es el error cuadrático medio entre los valores de los píxeles de la imagen original y la imagen generada.

Para medir el tiempo de entrenamiento junto con el indicador PSNR a lo largo del proceso, la pérdida y el número de iteraciones por segundo, se ha elaborado un código en Python. Al comenzar el proceso de entrenamiento de la red, el código comienza a medir. Cuando el modelo se entrena un número de iteraciones específico, se registra en un documento de texto el número actual de iteraciones, el valor de PSNR, el valor de la pérdida y las iteraciones por segundo, dividiendo el número de iteraciones desde la última medida entre el tiempo que ha transcurrido. Con estas medidas, se puede determinar con el ritmo de iteración del programa, así como la velocidad a la que disminuye la pérdida y aumenta el PSNR, lo que va a permitir una comparación objetiva del rendimiento de las diferentes tecnologías que se ponen a prueba. El mismo código se utiliza para medir el tiempo que las tecnologías tardan en renderizar una imagen sintética, una vez entrenado el modelo.

Los *dataset* que se han utilizado forman parte de un paquete de ejemplo disponible en la web de NeRF [4], creado por los desarrolladores, que consiste en varios conjuntos de datos de tipo LLFF preparados para ser introducidos en la red. Además, como se indica en la Subsección 5.1, se ha preparado un *dataset* propio con imágenes de profundidad, procesado por COLMAP y configurado para entrar en el tipo LLFF. Con estos datos se han generado modelos utilizando todas las tecnologías mencionadas en el proyecto y bajo las mismas condiciones. En la Tabla 6.1 se

reflejan las características de las secuencias. Su comparación permite conocer las posibilidades de la tecnología original en cuanto a calidad de resultados y tiempo requerido con datos de ejemplo y propios, y compararlas con las modificaciones estudiadas y propuestas basadas en la supervisión de la profundidad, y así concluir si este método representa una mejora y, si es el caso, en qué aspectos y de qué forma.

Nombre	Resolución	Número de imágenes para <i>training</i>	Número de imágenes para <i>testing</i>
<i>Flower</i>	1080x756	10/30	2/5
<i>Room</i>	1080x756	10/35	2/6
<i>Orchids</i>	1080x756	10/22	2/4
<i>Horns</i>	1080x756	10/55	2/8
<i>Trex</i>	1080x756	10/48	2/7
UPM	1920x1080	7	1

Tabla 6.1. Características de las secuencias utilizadas para evaluar los modelos. La notación "X/Y" indica que, para el mismo conjunto de datos, se entrena modelo tanto con X como con Y imágenes para el entrenamiento, de un total de Y disponibles.

6.2. Detalles de implementación

Al igual que en el caso de los experimentos realizados por los autores de la arquitectura NeRF [4], se utiliza el paquete SfM COLMAP [43] para estimar las poses de cámara y los parámetros intrínsecos de los *datasets* con los que se experimenta. COLMAP es un software de propósito general para la Estructura a partir del Movimiento (SfM) y Estereotipo de Múltiples Vistas (MVS). Ofrece una amplia gama de funciones para la reconstrucción de colecciones de imágenes. NeRF estima estos parámetros como parte del proceso de reconstrucción.

En cada iteración, se seleccionan aleatoriamente un conjunto de rayos de cámara del conjunto de todos los píxeles del *dataset*, y se utiliza el muestreo jerárquico descrito en la Subsección 3.1.1. Este proceso tiene como objetivo optimizar el uso de puntos en los rayos al evaluar primero una red "gruesa" con un conjunto de N_c posiciones iniciales, las cuales son seleccionadas para abarcar toda la escena de manera uniforme. Posteriormente, la salida de esta red gruesa se utiliza para seleccionar de manera más eficiente nuevos puntos de interés, representando las partes más relevantes del volumen, que serán evaluados por la red "fina", encargada de refinar la predicción con mayor precisión, utilizando N_f muestras adicionales en estas áreas de interés.

Ambas redes trabajan en conjunto: la red gruesa se encarga de muestrear el espacio de forma más general, mientras que la red fina refina estas muestras basándose en la importancia de las zonas seleccionadas. El color predicho por la red gruesa para un rayo r , $\hat{C}_c(r)$, se calcula como la suma ponderada de los colores muestreados c_i en las posiciones seleccionadas, mientras que el color final predicho, $\hat{C}_f(r)$, se obtiene tras refinar el muestreo utilizando la red fina. La pérdida total es el error cuadrático entre los colores reales y los predichos por ambas redes:

$$L = \sum_{r \in R} \left(\left\| \hat{C}_c(r) - C(r) \right\|_2^2 + \left\| \hat{C}_f(r) - C(r) \right\|_2^2 \right)$$

donde R es el conjunto de rayos en cada iteración, $C(r)$ es el color real, $\hat{C}_c(r)$ es el color predicho por la red gruesa y $\hat{C}_f(r)$ es el color predicho por la red fina. Aunque el renderizado final proviene de $\hat{C}_f(r)$, la optimización de la red gruesa es esencial para guiar de forma eficiente el muestreo jerárquico y mejorar la precisión de la red fina.

En los siguientes experimentos realizados, se utilizan conjuntos de 4096 rayos por iteración, cada uno sampleados a $N_c = 64$ coordenadas en el volumen grueso y $N_f = 128$ coordenadas adicionales en el volumen fino. Se utiliza el optimizador Adam [41] con una *learning rate* η que comienza en 5×10^{-4} y decrece exponencialmente a 5×10^{-5} durante el curso de la optimización. Los demás hiperparámetros de Adam se mantienen en sus valores predeterminados de $\beta_1 = 0,9$, $\beta_2 = 0,999$ y $\epsilon = 10^{-7}$.

6.3. Comparación de NeRF y DS-NeRF con *datasets* ejemplo

En primer lugar, se compara el rendimiento de NeRF y DS-NeRF cuando se entrena modelos utilizando varios de los *dataset* de ejemplo, en concreto, *Flower* y *Room*. En el Anexo A se encuentran los resultados del resto de los test realizados. Las pruebas presentadas en el artículo original de DS-NeRF [42] muestran un pico de PSNR alcanzado en un tiempo de dos a tres veces más rápido que NeRF. Esta mejora se incrementa cuantas menos vistas se utilizan en el entrenamiento. El artículo concluye que DS-NeRF consigue mejor calidad de reconstrucción que NeRF en menos tiempo cuando hay menos vistas, especialmente, de dos a cinco. A medida que aumenta el número de imágenes, se observa que el rendimiento de ambos modelos tiende a igualarse. Si bien DS-NeRF demuestra un mejor rendimiento con pocas vistas en el artículo original, se busca determinar qué sucede cuando aumentan las vistas y se pretende conseguir una mayor calidad de reconstrucción. Por lo tanto, se opta por una comparación con diez vistas para el entrenamiento. Aun así, esta cantidad sigue siendo un conjunto limitado de datos para técnicas como NeRF, que generalmente requiere muchas más vistas para generar buenas reconstrucciones.

6.3.1. Entrenamiento

Se muestran comparativas de tiempo, PSNR y pérdida de los resultados obtenidos con 10 imágenes para el entrenamiento en las siguientes gráficas y, en la Figura 6.3, una comparativa visual en diferentes etapas del entrenamiento. En primer lugar, se muestran los parámetros objetivos de rendimiento obtenidos para cada conjunto de datos, que se muestran en la Figura 6.1 y Figura 6.2.

A partir de estos resultados, se pueden destacar los siguientes puntos:

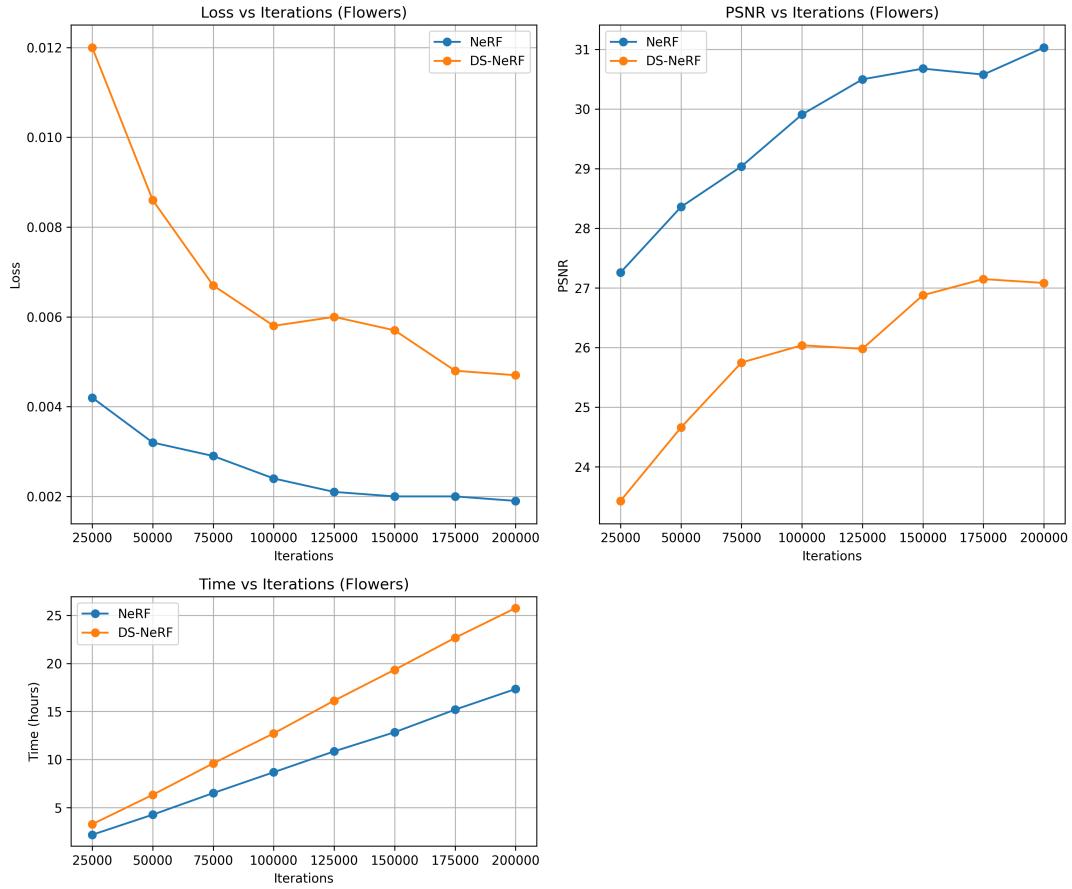


Figura 6.1. Comparación del rendimiento entre NeRF y DS-NeRF para el *dataset Flowers* con 10 vistas para el entrenamiento.

■ Calidad de imagen (PSNR)

- En ambos casos, NeRF supera a DS-NeRF en términos de PSNR, lo que indica que produce imágenes de mayor calidad con 10 vistas de entrenamiento.
- DS-NeRF, a pesar de utilizar la supervisión por mapas de profundidad, no logra alcanzar la calidad de imagen de NeRF. Esto podría deberse a que la complejidad añadida del modelo, mediante la supervisión de la profundidad, podría estar interfiriendo en su capacidad de generalización.

■ Velocidad de entrenamiento

- DS-NeRF es significativamente más lento que NeRF en ambos *dataset*. En *Flower*, DS-NeRF toma alrededor de un 50 % más de tiempo para completar 200,000 iteraciones en comparación con NeRF.
- El tiempo adicional se debe a los cálculos involucrados en la supervisión de mapas de profundidad. La integración de esta información añade una capa extra de procesamiento que afecta la eficiencia computacional.

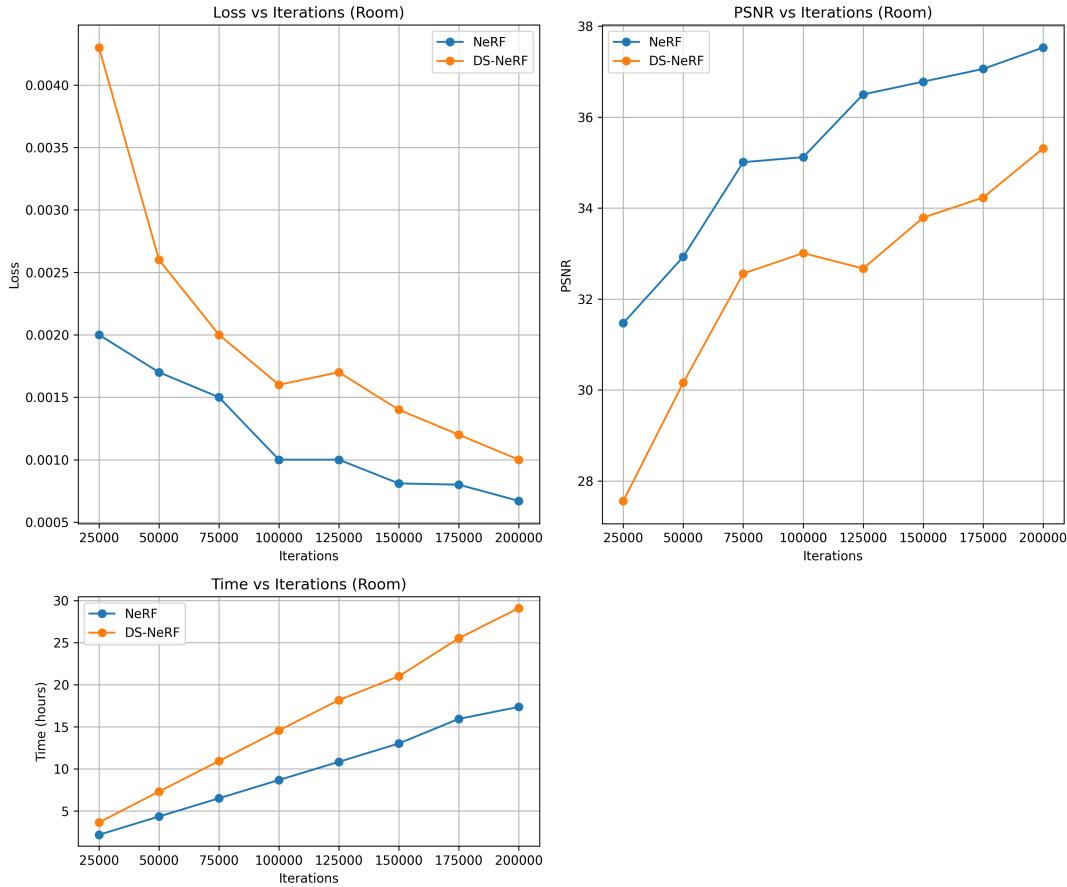


Figura 6.2. Comparación del rendimiento entre NeRF y DS-NeRF para el *dataset Room* con 10 vistas para el entrenamiento.

- NeRF, al no depender de dicha información adicional, es más eficiente en términos de tiempo, completando las mismas iteraciones en menos horas.

■ Convergencia de la pérdida (*Loss*)

- DS-NeRF muestra una convergencia inicial más rápida, mostrando una reducción significativa de la pérdida en las primeras iteraciones, seguramente gracias a la supervisión de la profundidad.
- Sin embargo, después de las 100,000 iteraciones, DS-NeRF muestra una desaceleración en su mejora, mientras que NeRF continúa disminuyendo su pérdida de forma más constante.
- NeRF termina con una pérdida mucho menor que DS-NeRF, lo que indica que, a largo plazo, es más eficaz en refinar la reconstrucción.

■ Impacto del *dataset*

- En *Flower*, NeRF tiene un mejor desempeño, logrando una diferencia notable en

PSNR respecto a DS-NeRF. Esto sugiere que NeRF maneja mejor las texturas y detalles finos con 10 vistas.

- En *Room*, DS-NeRF mejora su rendimiento, pero sigue estando por debajo de NeRF. Esto indica que, aunque la supervisión de profundidad puede ayudar en escenas más estructuradas, sigue sin representar una mejora con respecto al modelo original.

■ Escalabilidad y recursos

- NeRF demuestra ser más escalable, ya que con menos tiempo de entrenamiento logra una mayor calidad de imagen.
- DS-NeRF, aunque promete mejoras mediante la supervisión de profundidad, es menos eficiente en términos computacionales y no muestra un rendimiento superior cuando se superan las 5 vistas de entrenamiento. El *overhead* computacional se incrementa sin que haya una mejora significativa en PSNR.

En general, teniendo en cuenta los resultados obtenidos por los autores del artículo original de DS-NeRF [42], y los recién expuestos, se puede sugerir que el beneficio de la supervisión por profundidad de DS-NeRF es más evidente en escenarios con aún menos vistas (como en el rango de 2 a 5 vistas que se menciona en la literatura), donde la información adicional de la profundidad podría ser más crítica para la reconstrucción. Sin embargo, cuando las vistas aumentan a 10, la diferencia entre ambos modelos se reduce, y NeRF parece tener un desempeño superior en general, lo que indica que, con una cantidad moderada de vistas, NeRF es suficientemente robusto para reconstruir escenas de alta calidad, mientras que la ventaja de la supervisión de profundidad en DS-NeRF no tiene el mismo impacto y se convierte, más bien, en una desventaja.

Cabe destacar la diferencia que presentan los resultados obtenidos en el artículo original frente a los obtenidos en esta serie de experimentos en cuanto a PSNR. Los autores presentan una gráfica comparativa del PSNR obtenido NeRF y DS-NeRF para 10 vistas, sin especificar de qué *dataset* se ha obtenido, con DS-NeRF obteniendo mejores resultados que NeRF.

En estos casos, es importante asegurarse de que los parámetros utilizados en el entrenamiento son los mismos. Estos parámetros se definen en los archivos de configuración y, posteriormente, quedan escritos en los documentos de texto *args.txt*. Son cruciales, ya que afectarán directamente al resultado final. Se habla de parámetros como la tasa de aprendizaje, el rango de profundidad (*near* y *far*) o la resolución de las imágenes de entrada y salida . Una serie de archivos de configuración estándar para los *dataset* de ejemplo se incluyen en la descarga de DS-NeRF, pero en ningún lugar se asegura que éstos fueron los utilizados para las pruebas expuestas en el artículo. No es posible conseguir los archivos *args.txt* donde se detallan los parámetros que utilizaron los autores, a pesar de haber sido solicitado en el apartado de *issues* del *Github* de DS-NeRF [46]. Por lo tanto, se asume que los parámetros utilizados en las pruebas realizadas para este proyecto, basados en la configuración predeterminada, son correctos. A pesar de los esfuerzos por replicar las condiciones de los experimentos originales, no se ha identificado una

razón concreta que explique la discrepancia entre los resultados obtenidos en ambas series de pruebas.

A continuación, para comentar de manera los resultados con los parámetros subjetivos , en la Figura 6.3, se puede ver una comparativa visual:

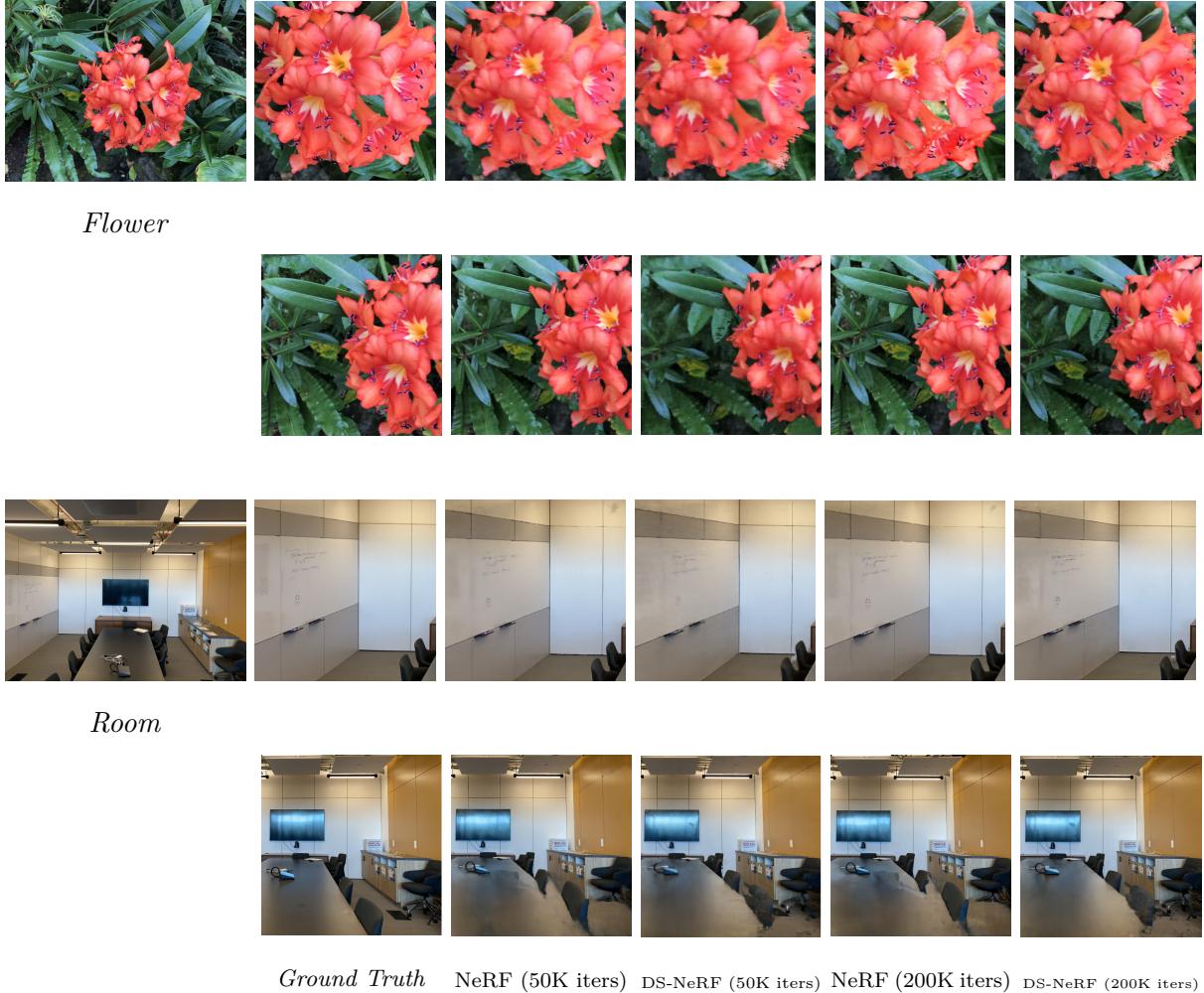


Figura 6.3. Comparación visual de diferentes objetos y técnicas.

Como se puede observar a simple vista, si bien ambas tecnologías son capaces de producir un resultado aceptable, podemos observar que, en comparación con NeRF, DS-NeRF presenta imperfecciones y un nivel de detalle en zonas de texturas complejas menor. Para el resultado obtenido en *Room*, no es capaz de hacer legible el texto escrito en la pizarra y, además, no soluciona las imperfecciones que presenta NeRF para representar correctamente los límites de las escenas. En el caso de *Flower*, si bien el nivel de detalle obtenido en ambos casos es bueno, se observan ciertas imperfecciones que en NeRF no se presentan, como en los bordes de los pétalos de las hojas, además de un menor nivel de detalle en el fondo de la escena.

En uno de los problemas comentados por los usuarios de DS-NeRF en el *GitHub* [46], se presentan una serie de resultados subóptimos obtenidos utilizando, en el caso específico 54 vistas

de entrenamiento para un *dataset* de ejemplo, llamado *Horns*. Al igual que en el obtenido en este proyecto, NeRF consigue un rendimiento notablemente mejor en cuanto a PSNR que DS-NeRF al final de 200K iteraciones. Sumado a esto, uno de los autores originales, *Kangle Deng*, responde comentando que también ha observado un PSNR subóptimo cuando se entrena DS-NeRF con más de 20 imágenes. Además, apunta que quizás se debe a que los objetivos de minimizar la pérdida de RGB y profundidad no siempre coinciden. Dicho de otra forma, sugiere que los objetivos de reconstruir correctamente las distancias de la escena y, al mismo tiempo, obtener una reconstrucción precisa de colores y texturas pueden entrar en conflicto o no estar alineados. sobre todo si se pretenden reconstruir superficies detalladas. Añadir supervisión de profundidad cuando ya se tienen suficientes vistas de entrenamiento podría no ofrecer una reconstrucción RGB tan sólida, pero produciría profundidades más razonables. Finalmente, reconoce que siguen buscando una buena manera de conseguir que la supervisión de profundidad funcione mejor en configuraciones con vistas completas.

En la Tabla 6.2 se compara el resultado obtenido por el usuario que escribió el problema en *GitHub* con algunos de los resultados obtenidos para este proyecto, con el mismo *dataset Horns* y con las mismas iteraciones finales. Ambos se asemejan, a diferencia de los obtenidos en el artículo original. Resalta el hecho de que DS-NeRF consiga un PSNR muy superior con un menor número de imágenes de entrenamiento. Esto podría deberse a una sobrecarga de complejidad al añadir más vistas, ya que el modelo tiene más información que procesar y más detalles que aprender, además de correr el riesgo de crear más ambigüedades en la correspondencia entre las distintas imágenes. También podría deberse a que el modelo tenga dificultades para mantener la coherencia geométrica entre todas las vistas al intentar ajustar tanto la profundidad como la radiancia simultáneamente. En cualquier caso, de nuevo, NeRF demuestra ser más flexible, aprendiendo implícitamente la estructura tridimensional, y no ajustándose a la profundidad proporcionada que, además de ser muy limitada, es imperfecta.

NeRF (<i>issue</i>)	DS-NeRF (<i>issue</i>)	NeRF 10 <i>view</i>	DS-NeRF 10 <i>view</i>	DS-NeRF 54 <i>view</i>
28.85	23.87	29.16	25.51	23.97

Tabla 6.2. Comparación de resultados de PSNR (dB) para el *dataset Horns*.

6.3.2. Renderizado de vistas

Para comparar el renderizado de vistas, se han probado varios modelos entrenados con los mismos conjuntos de imágenes en ambas tecnologías, y se ha medido el tiempo que tardan en generar una serie de imágenes. Se ha hecho una media del tiempo que ha tardado NeRF y DS-NeRF en generar cada imagen en segundos para obtener la siguiente Tabla 6.3. Cabe resaltar que las imágenes generadas tienen una resolución de 1008x756 píxeles. Cuanto de mayor tamaño sean las imágenes a generar, es lógico que mayor sea el tiempo que tarde el modelo en generarlas.

Como podemos observar, los modelos entrenados con ambas tecnologías son capaces de ge-

	NeRF (s)	DS-NeRF (s)
<i>Flower</i>	22.66	23.14
<i>Room</i>	22.5	23.07
<i>Horns</i>	22.59	23.17

Tabla 6.3. Comparación de resultados de tiempo de renderizado de imágenes para varios *dataset* de ejemplo en segundos.

nerar vistas sintéticas a velocidades similares. DS-NeRF tampoco es capaz de superar al original en este área, aunque es cierto que no era uno de los objetivos de ésta. Se considera innecesario presentar muestras visuales del resultado puesto que ya se presentó una comparativa de vistas generadas en el apartado anterior.

6.4. Modificaciones propias

6.4.1. Entrenamiento

A continuación, se presentan y comentan los resultados obtenidos después de entrenar NeRF y DS-NeRF con el *dataset* preparado para este proyecto, al igual que las dos modificaciones propias preparadas para utilizar imágenes de profundidad en el entrenamiento de las redes. Con esta comparación se podrá determinar hasta qué punto se experimenta una mejora cuando se supervisa la profundidad que incorporan los datos de las cámaras Kinect [5], y cuál de las modificaciones es más robusta y genera mejores representaciones. En la Imagen 6.4 se observa de manera gráfica las métricas objetivas obtenidas en diferentes etapas del entrenamiento y en la Tabla 6.4 se presentan los resultados numéricos finales.

Modelo	Tiempo (horas)	PSNR (dB)	Loss
NeRF	17.32	34.42	0.0010
DS-NeRF	28.32	32.33	0.0015
DS-NeRF MOD	29.66	30.02	0.0015
NeRF MOD	17.41	30.74	0.0024

Tabla 6.4. Comparación objetiva de los datos obtenidos después de 200,000 iteraciones entre NeRF, DS-NeRF y ambas modificaciones propuestas.

▪ Calidad de imagen (PSNR)

- El modelo original NeRF muestra una mejora constante en el PSNR a medida que aumentan las iteraciones. Comienza con un PSNR de 29.66 dB en 50,000 iteraciones y llega hasta 34.42 dB en 200,000. Este es el mejor rendimiento en cuanto a calidad visual entre todos los modelos.
- DS-NeRF tiene un PSNR más bajo al comienzo, con 26.65 dB en 50,000 iteraciones, pero también mejora de manera consistente hasta alcanzar 32.33 dB en 200,000. Esto

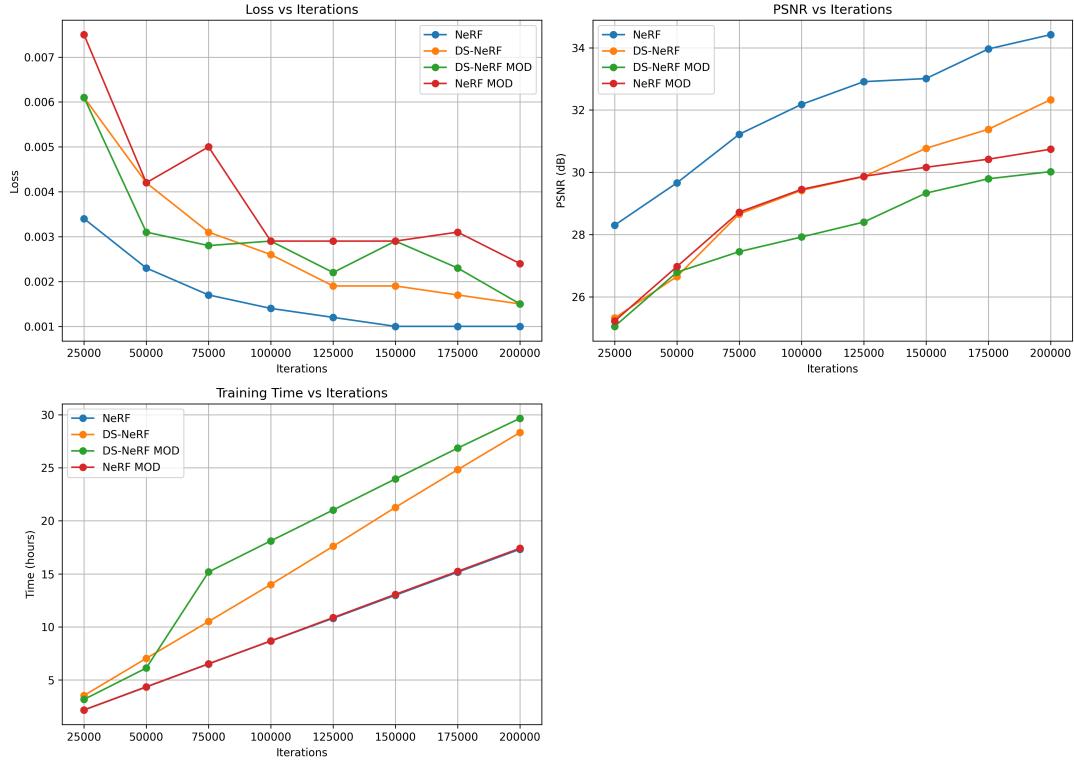


Figura 6.4. Comparación gráfica del rendimiento obtenido en el entrenamiento con cada tecnología para el *dataset* propio.

sugiere que DS-NeRF es menos eficiente optimizando el PSNR en comparación con el modelo original NeRF .

- Las modificaciones propias están por debajo en términos de PSNR, lo que indica que, sobre el papel, no mejoran la calidad visual de las reconstrucciones.

■ Tiempo de entrenamiento

- NeRF y NeRF *MOD* son los modelos más eficientes en términos de tiempo de entrenamiento, con tiempos similares. Esto era esperado, dado que la modificación se aplica principalmente antes del proceso de entrenamiento. Por lo tanto, el tiempo de entrenamiento debía ser casi idéntico
- DS-NeRF es, de nuevo, significativamente más lento debido a su mayor complejidad computacional. DS-NeRF *MOD* normalmente debería tener tiempos similares a DS-NeRF, pero un problema técnico causó un tiempo de entrenamiento más alto de lo esperado.

■ Convergencia de la pérdida (*Loss*)

- El valor de *loss* de NeRF disminuye consistentemente con el número de iteraciones, comenzando en 0.0023 y alcanzando 0.0010. Este modelo tiene la mejor reducción de pérdida, con los valores más bajos al final del entrenamiento.

- DS-NeRF y la modificación implementada con imágenes de profundidad a partir de éste, presentan el mismo valor de pérdida final, 0.0015. Sin embargo, DS-NeRF demuestra una reducción de la pérdida constante, mientras que DS-NeRF *MOD* mantuvo un estancamiento en 0.0029 durante, aproximadamente, 50,000 iteraciones. Esto indica que, en términos de pérdida, es menos consistente.
- NeRF *MOD* es la que peores resultados en cuanto a pérdida presenta. En 50,000 iteraciones se tiene un valor de 0.0042. Puede ser lógico pensar que, en las primeras fases del entrenamiento, esta modificación tenga valores más altos de pérdida que NeRF, ya que se le introduce también la pérdida que genera la supervisión de la profundidad. Sin embargo, al final del entrenamiento, es el modelo que más pérdida tiene con 0.0024, y es el que más altibajos sufre durante el proceso.

■ Impacto del *dataset*

- NeRF *MOD* y DS-NeRF *MOD* integran supervisión de profundidad con estos mapas generados por cámaras Kinect, lo que teóricamente debería mejorar la reconstrucción 3D en áreas donde NeRF o DS-NeRF podrían tener problemas con occlusiones o áreas no vistas.
- Ambos modelos muestran un PSNR más bajo y tienen ciertas dificultades para reducir la pérdida en comparación con NeRF y DS-NeRF. Esto sugiere que la calidad de las imágenes de profundidad obtenidas con las cámaras Kinect no es lo suficientemente precisa como para mejorar la reconstrucción, y el ruido en los datos de profundidad podría estar generando errores en la estimación de la geometría de la escena. Es importante recordar que los datos objetivos no tienen por qué estar completamente alineados con los resultados visuales.

En la Figura 6.5 se aprecian diferentes ejemplos visuales de los resultados obtenidos por cada tecnología. En el Anexo A se presentan más imágenes representativas.

En este caso, al tener menos vistas para el entrenamiento, se observan más occlusiones y zonas borrosas en el resultado de NeRF, como se observa en la primera imagen de esta tecnología, observando la pizarra. Sin embargo, el nivel de nitidez obtenido en el mapa, en los rostros, o en el logotipo de la UPM está mejor conseguido en esta primera tecnología. Por otra parte, DS-NeRF, si bien no consigue la definición de NeRF en zonas clave, es capaz de solventar algunas zonas borrosas que éste sí presenta. Si bien sería necesario observar múltiples vistas generadas por ambas tecnologías para apreciar correctamente la diferencia, si se analiza las capturas de la Figura 6.5, DS-NeRF solventa la zona borrosa en la pizarra con cierto nivel de nitidez. La falta de vistas provoca que resalte más la influencia de la información de profundidad de COLMAP en el entrenamiento, a pesar de que el resultado de PSNR de NeRF sea significativamente superior.

En cuanto a los resultados obtenidos por las modificaciones propuestas en este proyecto, se aprecia a simple vista que el resultado es bastante similar entre sí. De nuevo, sin observar un

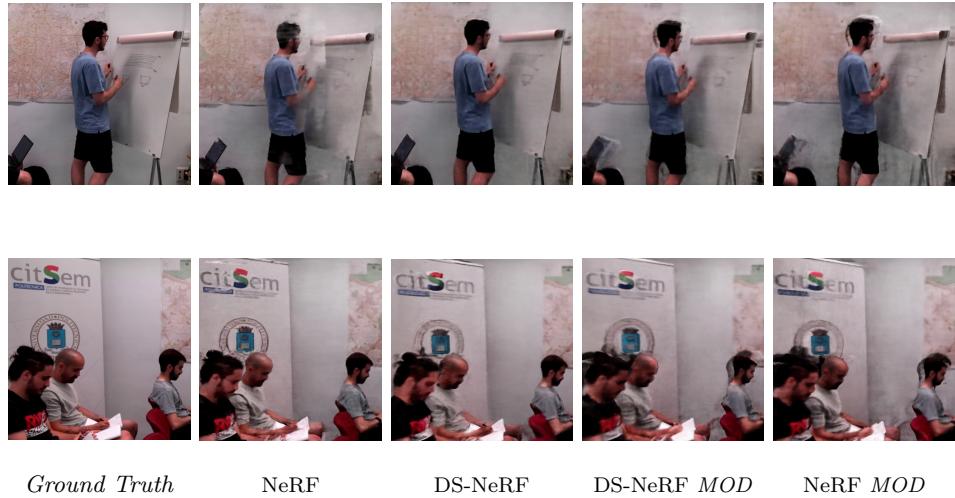


Figura 6.5. Comparación visual de los resultados obtenidos con el *dataset* propio.

número significativo de imágenes no se reconocen del todo las diferencias entre modelos pero, en estos últimos casos, las occlusiones y las zonas borrosas se minimizan aún más que con DS-NeRF. El nivel de nitidez conseguido con estas modificaciones en general es similar al de éste, pero con una diferencia que se aprecia a simple vista en varias zonas de las capturas de la Figura 6.5, como en las cabezas o el ordenador: estos errores visuales, tales como sombras y artefactos, ocurren debido a los siguientes motivos:

- Los mapas de profundidad generados por las cámaras Kinect, al utilizar tecnología infrarroja para la estimación, sufren de limitaciones inherentes en cuanto a la precisión, especialmente cuando se enfrentan a superficies complejas como cabellos o zonas de baja reflectancia, iluminación inadecuada o distancias mas allá de 4 o 5 metros.
- Las diferencias espaciales entre las vistas de entrenamiento causadas, de nuevo, por el ruido, pueden causar errores en la interpolación de la geometría, es decir, el proceso de estimar puntos no capturados directamente por los datos de entrada, utilizando la información de las vistas y puntos cercanos. La inconsistencia entre vistas tiene como resultado estos artefactos visuales.
- A pesar de que las imágenes RGB y los mapas de profundidad comparten poses de cámara al haber sido tomadas desde las mismas posiciones, la fusión de información de color y de profundidad puede no ser perfecta si la calidad de los datos de profundidad es inconsistente entre diferentes vistas. Correspondencias incorrectas entre los puntos en las imágenes RGB y los puntos en los mapas de profundidad pueden causar sombras, más comunes en las áreas que están parcialmente ocluidas o donde hay superficies complejas.

Si se renderiza la vista reservada para *testing* y se obtienen sus valores de profundidad con cada tecnología, podemos comparar su rendimiento haciendo predicciones de las dimensiones de

la escena. La Figura 6.6 contiene imágenes de las vistas que han generado las tecnologías después de 200,000 iteraciones, además de la imagen captada por la Kinect desde el mismo ángulo. El resultado de NeRF no se incluye ya que éste no utiliza la profundidad en su entrenamiento y, por lo tanto, no la aprende. Inmediatamente se aprecian las diferencias entre los resultados. La información de profundidad extra que incorporan las cámaras Kinect permiten un resultado significativamente más preciso y nítido si se utiliza en el entrenamiento de redes NeRF, comparado con el que el que produce la tecnología que sólo utiliza la profundidad de COLMAP. En este sentido, las modificaciones propuestas sí representan una gran mejora con respecto a lo que devulve la Kinect, que muestra una versión reducida de la imagen con errores en los contornos de las figuras, y con lo que es capaz de generar DS-NeRF, una predicción claramente más difusa. Al igual que con las vistas RGB generadas, ambas modificaciones consiguen una representación de la profundidad similar entre sí.

Como primera conclusión que se puede extraer de esta comparación puede destacar que, si bien NeRF vuelve a estar por encima si se presta atención únicamente a los datos objetivos, si se observan las imágenes generadas, la realidad es distinta. En este caso en el que se cuenta con una escena propia, más compleja y en el que se cuentan con menos vistas de entrenamiento, la información extra de la profundidad de COLMAP con la que se supervisa el entrenamiento de DS-NeRF supone una cierta mejora con respecto a NeRF.

En cuanto a las modificaciones, los resultados obtenidos son similares. La gran diferencia entre estos modelos, por lo que quizá la modificación sobre NeRF sea la mejor implementada, es el tiempo de entrenamiento. Precisamente por partir de NeRF, la tecnología itera a la misma velocidad que éste, al igual que DS-NeRF itera igual que su modificación. A falta de realizar pruebas con otros conjuntos de datos para una mejor conclusión, se puede establecer que una tecnología es más eficiente que la otra.

Finalmente, comparando los resultados de los modelos originales con sus modificaciones, aunque los errores de los sensores ToF de las cámaras y la falta de correspondencia que provocan al tratar de encajar esta información con los puntos RGB desencadenen sombras, falta de nitidez y, en cuanto a la reconstrucción de color, un peor resultado, es cierto que la predicción que hace la red de la profundidad de la escena es considerablemente mejor en las tecnologías propuestas para este proyecto de Fin de Grado, como se ha observado claramente en la Figura 6.6.

6.4.2. Renderizado de vistas

En cuanto al tiempo que tarda cada tecnología en generar una vista nueva, en la Tabla 6.5 está la comparación de lo que ha tardado cada tecnología para una imagen 1920x1080. Como era de esperar, puesto que las modificaciones no interfieren en el renderizado, los tiempos son similares, y las variaciones percibidas se considera se deben al servidor *mizar* en el que se ejecutan las pruebas.



Kinect



DS-NeRF

DS-NeRF *MOD*NeRF *MOD*

NeRF (s)	DS-NeRF (s)	DS-NeRF MOD (s)	NeRF MOD (s)
61.5	62.01	66.1	60.7

Tabla 6.5. Comparación de resultados del tiempo de renderizado de imágenes en segundos para el *dataset* propio en cada tecnología.

Capítulo 7

Presupuesto

Para desglosar los presupuestos de un Proyecto de Fin de Grado en el que se han estudiado y modificado NeRF y DS-NeRF, realizado pruebas de entrenamiento, desarrollado modificaciones y utilizado hardware y software específicos, el desglose debe considerar tanto costes materiales como costes de personal.

7.1. Costes Materiales

Los costes materiales corresponden a los recursos físicos y tecnológicos que se han utilizado para llevar a cabo el proyecto. Estos incluyen tanto el hardware necesario para la captura de datos y entrenamiento de los modelos, como el software utilizado en las distintas etapas de desarrollo y pruebas. En la Tabla 7.1 se encuentra el desglose.

- ***Hardware:***

- **Cámaras Microsoft Azure Kinect [5]** (8 unidades): utilizadas para crear el *dataset* propio utilizado para comparar las diferentes tecnologías. Estas cámaras pertenecen al CITSEM [48].
- **Tarjeta gráfica NVIDIA RTX 3090 [47]**: también cedida por el CTISEM, que cuenta con ocho de éstas, aunque solo se incluye el precio de una de ellas en estos presupuestos. Se han utilizado para el entrenamiento de modelos para su evaluación y comparación. Considerando el uso que se ha hecho de este recurso durante el entrenamiento de modelos, es necesario estimar un coste de uso. En esta clase de proyectos, este coste puede estar relacionado con el tiempo de uso del servidor, pero como es proporcionado por la institución sin coste adicional, no se refleja este coste.
- **Ordenador cedido y ordenador personal**: como los anteriores medios, el primero también cedido por el CITSEM. Ambos han sido utilizados en cada fase necesaria para completar el proyecto.

- **Software:** teniendo en cuenta que todas las herramientas *software* utilizadas, tanto COLMAP [43], como los diferentes entornos de desarrollo (Pytorch, TensorFlow, bibliotecas de visión por ordenador, etc.), son de gratuitas de código abierto y que no se han utilizado licencias de coste adicional, ya que el entorno de edición LaTeX ha sido Overleaf, gratuito en la versión utilizada, los costes del *software* utilizados son cero euros.

Producto	Coste (euros)
Cámaras Microsoft Azure Kinect (8 unidades)	3200 euros
Tarjeta gráfica NVIDIA RTX 3090	1500 euros
Ordenador cedido	1000 euros
Ordenador personal	500 euros
Software COLMAP	0 euros
Entorno de desarrollo (PyTorch, TensorFlow, etc.)	0 euros
Licencias <i>software</i>	0 euros
Total Materiales	6200 euros

Tabla 7.1. Costes Materiales

7.2. Costes de Personal

En la Tabla 7.2 se encuentra el desglose de los costes de personal. Éstos engloban el tiempo y el esfuerzo dedicados por las personas implicadas en el proyecto, tanto del tutor como del estudiante. Estos costes reflejan el valor del trabajo intelectual y las horas invertidas en tareas como la supervisión del proyecto, el desarrollo de las modificaciones de los modelos y la creación del dataset.

- **Tutor académico:** se estima coste del tutor encargado de la supervisión y orientación, basado en su dedicación al proyecto, suponiendo unas 30 horas empleadas.
- **Desarrollo por parte del estudiante:** aunque el trabajo del estudiante puede no tener un coste directo, se puede valorar el esfuerzo invertido para dar una idea del coste real del desarrollo del proyecto, estimando un trabajo de 320 horas en total.

Personal	Horas	Salario (euros/h)	Coste (euros)
Estudiante (desarrollo y pruebas)	320	15	4800
Tutor (orientación y supervisión)	30	30	900
Total Personal	—	—	5700

Tabla 7.2. Costes de Personal

7.3. Costes Total

Finalmente, en la Tabla 7.3 se muestra la suma de los costes de material y los costes de personal, obteniendo así el coste total del presente Proyecto de Fin de Grado.

Descripción	Coste (euros)
Coste Material	6200
Coste Personal	5700
Coste Total	11900

Tabla 7.3. Coste Total del Proyecto

Capítulo 8

Impacto del proyecto

Este proyecto aborda el estudio y desarrollo de técnicas avanzadas de reconstrucción tridimensional basadas en los modelos NeRF y DS-NeRF, incorporando mejoras en la supervisión de la profundidad mediante el uso de imágenes capturadas por cámaras Microsoft Azure Kinect. Los resultados obtenidos no solo han demostrado una mejora significativa en la precisión de la reconstrucción de la profundidad, sino que también han revelado aplicaciones y ventajas potenciales en diversos campos tecnológicos. A continuación, se analizan las principales implicaciones tecnológicas, económicas, sociales y medioambientales derivadas del trabajo realizado, así como su posible contribución a los Objetivos de Desarrollo Sostenible (ODS) [50].

Si se habla de impacto tecnológico, las modificaciones propuestas han demostrado una mejora significativa en la precisión de la reconstrucción tridimensional. Este avance abre nuevas oportunidades para aplicaciones que requieran un análisis detallado de la geometría, como la realidad virtual y aumentada, la robótica y la navegación autónoma, donde la exactitud en la representación de los objetos es crucial, a menudo por encima de la fidelidad en el color. Por ejemplo, las tecnologías de AR y VR dependen en gran medida de la precisión en la reconstrucción de entornos 3D para generar experiencias inmersivas. Estas mejoras permiten una integración más exacta de objetos virtuales en escenarios reales, aumentando la fidelidad visual. En áreas como los juegos y simulaciones en VR, las versiones modificadas de NeRF y DS-NeRF facilitan la creación de entornos más detallados y realistas [1]. Además, en el ámbito de la arquitectura y la construcción, los sistemas desarrollados en este proyecto, junto con otras soluciones para la reconstrucción de escenas complejas [51], pueden aplicarse en la generación de modelos 3D detallados, útiles en diseño arquitectónico, restauración de edificios o inspección de infraestructuras. Finalmente, en el sector de la salud y el análisis biomédico, las imágenes 3D son esenciales para la planificación quirúrgica y el diagnóstico asistido por ordenador. Las mejoras en la reconstrucción tridimensional permiten crear modelos más exactos de órganos o tejidos, incrementando la precisión en intervenciones médicas [3].

En el caso del impacto económico, el uso de cámaras Microsoft Azure Kinect, en lugar de

tecnologías más costosas como los sistemas *Light Detection and Ranging* (LiDAR) [52], posiciona este proyecto como una solución más asequible para obtener reconstrucciones tridimensionales precisas. Estos sistemas de teledetección utilizan pulsos de luz láser para medir distancias y obtener representaciones tridimensionales precisas de superficies, objetos o terrenos, lo cual resulta ideal para ámbitos como la cartografía y topografía, la arqueología o arquitectura de precisión. Su precisión y resolución espacial son significativamente mayores que las de las cámaras Kinect, elevando el coste notoriamente. Las cámaras Kinect son más económicas al estar dirigidas a un mercado de consumo general y usar tecnologías menos precisas pero suficientemente adecuadas para muchas aplicaciones. En conclusión, la propuesta de este proyecto frente a LiDAR proporciona una solución más asequible para aplicaciones que no requieren una precisión, manteniendo una buena relación entre costo y precisión en la reconstrucción tridimensional.

El desarrollo de tecnologías más accesibles para la creación de entornos tridimensionales detallados puede tener importantes beneficios en el ámbito social, pudiendo contribuir en la creación de entornos simulados con alta fidelidad geométrica, donde la precisión en la profundidad puede ser clave, como experiencias más inmersivas para la educación y la cultura. Se habla de, por ejemplo, el desarrollo de proyectos educativos interactivos como experiencias inmersivas en museos y exposiciones [1]. Además, como se comenta anteriormente, en el campo de la medicina, que afecta directamente al bienestar social, la mejora en la precisión de la reconstrucción de la profundidad podría aplicarse en el desarrollo de herramientas avanzadas de diagnóstico por imagen, donde la información tridimensional precisa puede mejorar los resultados médicos y reducir riesgos.

Aunque este proyecto no tiene un impacto medioambiental directo, el uso de infraestructuras de procesamiento eficientes contribuye a la reducción de la huella de carbono. Los superordenadores y centros de datos que sustentan la computación de alto rendimiento requieren grandes cantidades de energía para operar. Las operaciones de procesamiento intensivo de datos, como el entrenamiento de modelos de *deep learning*, pueden consumir una gran cantidad de energía, lo que incrementa la huella de carbono si la fuente de energía proviene de fuentes no renovables. En este proyecto, se han desarrollado dos modificaciones que logran reconstruir la profundidad de las escenas de forma más precisa que DS-NeRF, además de necesitar menos tiempo para obtener mejores resultados. Una de estas versiones se beneficia también de una mayor velocidad en el entrenamiento de la red neuronal. Al optimizar el tiempo de entrenamiento y mejorar la eficiencia del sistema, se reduce el consumo energético en comparación con DS-NeRF y otros enfoques más tradicionales de reconstrucción 3D, lo que contribuye a una menor huella de carbono.

Por último, se determina el potencial de contribuir a los ODS en varias áreas:

- **ODS 9: Industria, Innovación e Infraestructura:** El avance en tecnologías accesibles y eficientes para la reconstrucción 3D, como el caso de este proyecto, promueve la innovación tecnológica y puede ser aplicado en diversos sectores industriales, desde la robótica hasta el entretenimiento, fomentando un desarrollo más sostenible en cuanto a recursos.

- **ODS 11: Ciudades y Comunidades Sostenibles:** Las mejoras en las tecnologías de modelado tridimensional pueden ser aplicadas en la preservación del patrimonio cultural, ayudando a la conservación y restauración digital de sitios históricos y contribuyendo a la construcción de ciudades más inteligentes y sostenibles.
- **ODS 12: Producción y Consumo Responsables:** Al reducir los costos de hardware y optimizar el uso de recursos computacionales por motivos descritos anteriormente, este proyecto puede ayudar a reducir la cantidad de recursos materiales y energéticos necesarios para realizar tareas avanzadas de reconstrucción 3D, promoviendo una producción tecnológica más responsable.

Capítulo 9

Conclusiones

En este último apartado se presentan las principales conclusiones derivadas del trabajo realizado en el proyecto, así como las implicaciones más relevantes que se han identificado a lo largo del desarrollo. Se abordará un análisis general del problema propuesto, las soluciones implementadas y los resultados obtenidos, permitiendo evaluar el éxito y las limitaciones del enfoque planteado. Asimismo, se exponen los nuevos conocimientos derivados de la investigación, señalando aspectos del problema que podrían abordarse en futuras investigaciones. Finalmente, se plantearán posibles líneas de trabajo a desarrollar para continuar mejorando los resultados y resolver los desafíos pendientes.

9.1. Conclusión

Este Proyecto de Fin de Grado ha abordado el estudio y análisis de los modelos NeRF y DS-NeRF, con el objetivo de evaluar sus capacidades y limitaciones en la reconstrucción tridimensional de escenas complejas. Como parte del desarrollo, se propusieron una serie de modificaciones a partir de las anteriores tecnologías, que introdujo la supervisión de la profundidad utilizando imágenes capturadas por cámaras Microsoft Azure Kinect. Estas nuevas implementaciones han demostrado una mejora significativa en la reconstrucción de la profundidad de las escenas, logrando una precisión superior que DS-NeRF comparando los modelos entrenados con el *dataset* preparado para este propósito. Aunque la reconstrucción de imágenes RGB no alcanzó la misma calidad que las tecnologías base, los avances en la predicción de la profundidad sitúan a la modificación propuesta como una solución más eficiente y precisa en cuanto a detalles geométricos.

De la comparación entre las tecnologías base, derivada del estudio en profundidad y la evaluación de los resultados objetivos y subjetivos obtenidos tras el entrenamiento de modelos con conjuntos de datos idénticos para ambas tecnologías, se extraen diversas conclusiones. Se observó una discrepancia significativa, sin una justificación clara, entre los resultados reportados

por los autores del trabajo original de DS-NeRF [42] y los obtenidos en este proyecto, así como por otros usuarios de la tecnología. En este contexto, la técnica de supervisión de la profundidad mediante COLMAP demostró, en general, un rendimiento inferior en cuanto a PSNR y calidad visual subjetiva para conjuntos de datos con 10 o más imágenes, tras 200.000 iteraciones. Aunque NeRF no genera información de profundidad y DS-NeRF sí es capaz de reconstruir la geometría de la escena, la precisión de esta reconstrucción es limitada debido al escaso número de puntos de profundidad 3D disponibles. Estos resultados sugieren que la reconstrucción precisa de colores y texturas y la reconstrucción precisa de la geometría de la escena podrían no estar alineados como objetivos. Un modelado tridimensional óptimo no necesariamente implica una mejora en la representación de color y, de hecho, podría perjudicarla en escenarios con más de 10 vistas. Por último, es importante destacar que NeRF presenta un mejor rendimiento en cuanto a tiempos de entrenamiento, lo que indica que la estructura utilizada para la supervisión de profundidad en DS-NeRF conlleva un entrenamiento más lento en términos de iteraciones por segundo.

En cuanto a las modificaciones propuestas e implementadas para este proyecto, las pruebas realizadas mostraron que, al comparar los modelos originales con las modificaciones desarrolladas, se alcanzó una mayor precisión en la reconstrucción de la geometría de la escena, especialmente en lo que respecta a la predicción de la profundidad. No obstante, problemas como la aparición de sombras y la falta de nitidez en la reconstrucción de los colores, derivados de los errores en los sensores ToF de las cámaras Kinect y las dificultades para alinear correctamente la información de profundidad con los puntos RGB, siguen siendo un reto para lograr una calidad visual óptima. A pesar de estas limitaciones, los resultados indican que las modificaciones mejoran notablemente la reconstrucción de la geometría de la escena en comparación con las versiones originales. Sin embargo, en términos de velocidad de entrenamiento, la variante planteada sobre NeRF mostró una ventaja considerable, ya que mantiene la eficiencia de NeRF, iterando a la misma velocidad, mientras que la modificación a partir de DS-NeRF hereda su mayor tiempo de procesamiento debido a la supervisión adicional de profundidad. Aunque aún es necesario realizar más pruebas con otros conjuntos de datos para obtener conclusiones definitivas, estos avances sugieren que, para ciertas aplicaciones donde la precisión en la profundidad es prioritaria sobre la calidad del color, las modificaciones propuestas ofrecen una solución más eficaz.

9.2. Trabajos futuros

A continuación, se detallan las posibles líneas de trabajo futuro que podrían surgir a partir de los resultados obtenidos en este proyecto. A pesar de los avances expuestos, existen áreas que requieren de mayor investigación y refinamiento. Algunas propuestas son las siguientes:

- **Aplicación en nuevos conjuntos de datos:** Sería conveniente realizar pruebas con más conjuntos de datos, idealmente con escenas más diversas y complejas. Esto permitiría al modelo generalizar mejor y podría proporcionar una evaluación más completa de su

desempeño en diferentes escenarios.

- **Optimización de la precisión en la reconstrucción de color:** Abordar los problemas de sombras y falta de nitidez en la reconstrucción de color causados por los errores de los sensores ToF de las cámaras Kinect, utilizando técnicas de corrección o mejorando la fusión de datos entre los puntos RGB y la profundidad.
- **Mejora de la precisión de los sensores ToF:** Investigar soluciones para reducir los errores de los sensores ToF de las cámaras Kinect, como la integración de otros sensores o la mejora en los métodos de calibración de los dispositivos para una mayor exactitud en la representación de la profundidad.

Bibliografía

- [1] R. Martin-Brualla, N. Radwan, M. S. Sajjadi, J. T. Barron, A. Dosovitskiy y D. Duckworth, “NeRF-W: Neural Radiance Fields in the Wild”, en *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, págs. 7210-7219.
- [2] L. Gao, R. Clark, A. Mirchev y E. Johns, “iMAP: Implicit Mapping and Positioning in Real-Time”, en *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, págs. 6229-6238.
- [3] V. autores, “NeRF for Medical Imaging”, *Preprint*, 2022, <https://arxiv.org/abs/2201.10717>.
- [4] K. Gao, Y. Gao, H. He, D. Lu, L. Xu y J. Li, “Nerf: Neural radiance field in 3d vision, a comprehensive review”, *arXiv preprint arXiv:2210.00379*, 2022.
- [5] Microsoft. “Documentación de Azure Kinect DK”. (2024), dirección: <https://learn.microsoft.com/es-es/azure/kinect-dk/>.
- [6] Y. LeCun, Y. Bengio y G. Hinton, “Deep learning”, *Nature*, vol. 521, n.º 7553, págs. 436-444, 2015.
- [7] Z.-H. Zhou, *Machine learning*. Springer nature, 2021.
- [8] J. Schmidhuber, “Deep learning in neural networks: An overview”, *Neural networks*, vol. 61, págs. 85-117, 2015.
- [9] D. de Matemática Aplicada. “Redes Neuronales Introducción”. (2021), dirección: https://dcain.etsin.upm.es/~carlos/bookAA/05.1_RedesNeuronalesIntroduccion.html.
- [10] D. O. Hebb, *The organization of behavior: A neuropsychological theory*. Psychology press, 2005.
- [11] A. Santos-del Riego y M. V. Casto Villar, “Redes de neuronas artificiales: reglas de aprendizaje”, 1996.
- [12] S. Jaiswal. “Multilayer Perceptrons in Machine Learning: A Comprehensive Guide”. (2024), dirección: <https://www.datacamp.com/tutorial/multilayer-perceptrons-in-machine-learning>.
- [13] I. Goodfellow, Y. Bengio y A. Courville, *Deep learning*. MIT press, 2016.

- [14] “Deep Neural Network (DNN) example.” (), dirección: https://www.researchgate.net/figure/Deep-Neural-Network-DNN-example_fig2_341037496.
- [15] P. Wang, L. Liu, Y. Liu, C. Theobalt, T. Komura y W. Wang, “NeuS: Learning Neural Implicit Surfaces by Volume Rendering for Multi-view Reconstruction”, *Advances in Neural Information Processing Systems*, vol. 34, 2021.
- [16] C. Godard, O. Mac Aodha y G. J. Brostow, “Unsupervised Monocular Depth Estimation with Left-Right Consistency”, en *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, págs. 270-279.
- [17] “Introducción a los autoencoders”. (), dirección: <https://es.mathworks.com/discovery/autoencoder.html>.
- [18] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever y R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting”, *The journal of machine learning research*, vol. 15, n.º 1, págs. 1929-1958, 2014.
- [19] X. Glorot e Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks”, en *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, JMLR Workshop y Conference Proceedings, 2010, págs. 249-256.
- [20] D. E. Rumelhart, G. E. Hinton y R. J. Williams, “Learning representations by back-propagating errors”, *nature*, vol. 323, n.º 6088, págs. 533-536, 1986.
- [21] “Aprendizaje automático: Una introducción al error cuadrático medio y las líneas de regresión.” (), dirección: <https://www.freecodecamp.org/espanol/news/aprendizaje-automatico-una-introduccion-al-error-cuadratico-medio-y-las-lineas-de-regresion/>.
- [22] R. R. Abril. “Estimación de profundidad”. (), dirección: <https://lamaguinaoraculo.com/deep-learning/estimacion-de-profundidad/>.
- [23] C. A. (UPC). “Fundamentos de visión estereoscópica”. (2002), dirección: <https://www.cs.upc.edu/~virtual/SGI/guions/FonamentsEstereo.pdf>.
- [24] R. Hartley y A. Zisserman, *Multiple view geometry in computer vision*. Cambridge university press, 2003.
- [25] N. Scarlet. “Visión Estéreo 3D para Aplicaciones de Visión Artificial”. (2023), dirección: <https://www.clearview-imaging.com/es/blog/visi%C3%B3n-est%C3%A9reo-3d-para-aplicaciones-de-visi%C3%B3n-artificial>.
- [26] W. Förstner, “Time-of-flight cameras and their applications”, *Time-of-Flight Imaging Sensors*, vol. 1, págs. 1-20, 2009.
- [27] “Time-of-Flight camera principle.” (), dirección: https://www.researchgate.net/figure/Time-of-Flight-camera-principle_fig4_260432866.

- [28] L. T. De Paolis, V. De Luca, C. Gatto, G. D'Errico y G. I. Paladini, "Photogrammetric 3D reconstruction of small objects for a real-time fruition", en *Augmented Reality, Virtual Reality, and Computer Graphics: 7th International Conference, AVR 2020, Lecce, Italy, September 7–10, 2020, Proceedings, Part I* 7, Springer, 2020, págs. 375-394.
- [29] "Fotogrametría". (), dirección: <https://hexagon.com/es/products/product-groups/measurement-inspection-hardware/photogrammetry>.
- [30] M. J. Westoby, J. Brasington, N. F. Glasser, M. J. Hambrey y J. M. Reynolds, "Structure-from-Motionphotogrammetry: A low-cost, effective tool for geoscience applications", *Geomorphology*, vol. 179, págs. 300-314, 2012.
- [31] S. Agarwal, N. Snavely, S. M. Seitz y R. Szeliski, "Bundle adjustment in the large", en *Computer Vision–ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5–11, 2010, Proceedings, Part II* 11, Springer, 2010, págs. 29-42.
- [32] Y. Furukawa y J. Ponce, "Accurate, dense, and robust multiview stereopsis", *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, n.º 8, págs. 1362-1376, 2009.
- [33] L. F. M. Idárraga, R. A. B. Valencia y J. M. Franco, "Estructura y funcionamiento de un algoritmo de renderizado: Raytracing",
- [34] J. T. Kajiya y B. P. Von Herzen, "Ray tracing volume densities", *ACM SIGGRAPH computer graphics*, vol. 18, n.º 3, págs. 165-174, 1984.
- [35] N. Rahaman, A. Baratin, D. Arpit y col., "On the spectral bias of neural networks", en *International conference on machine learning*, PMLR, 2019, págs. 5301-5310.
- [36] M. Levoy, "Efficient ray tracing of volume data", *ACM Transactions on Graphics (ToG)*, vol. 9, n.º 3, págs. 245-261, 1990.
- [37] B. Mildenhall, P. P. Srinivasan, R. Ortiz-Cayon y col., "Local Light Field Fusion: Practical View Synthesis with Prescriptive Sampling Guidelines", *ACM Transactions on Graphics (TOG)*, 2019.
- [38] I. Contributors, *Imageio: A Python library for reading and writing image data*, Version 2.22.4, 2024. dirección: <https://imageio.readthedocs.io/>.
- [39] R. Collobert, K. Kavukcuoglu y C. Farabet, *Torch7: A Matlab-like Environment for Machine Learning*, 2011. dirección: <http://torch.ch/>.
- [40] P. Contributors, *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, Version 1.9.0, 2024. dirección: <https://pytorch.org/>.
- [41] D. P. Kingma y J. Ba, "Adam: A method for stochastic optimization", *arXiv preprint arXiv:1412.6980*, 2014.
- [42] K. Deng, A. Liu, J.-Y. Zhu y D. Ramanan, "Depth-supervised nerf: Fewer views and faster training for free", en *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, págs. 12 882-12 891.

- [43] J. L. Schonberger y J.-M. Frahm, “Structure-from-motion revisited”, en *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, págs. 4104-4113.
- [44] “COLMAP”. (), dirección: <https://colmap.github.io/>.
- [45] R. C. Gonzalez y R. E. Woods, *Digital Image Processing*, 4th. London, UK: Pearson, 2017, ISBN: 978-0-13-335672-4.
- [46] Dunbar12138, *DS-NeRF: Depth Supervised Neural Radiance Fields*, Último acceso: 25 de noviembre de 2024, 2021. dirección: <https://github.com/dunbar12138/DSNeRF>.
- [47] NVIDIA, *GeForce RTX 3090 Specifications*, <https://www.nvidia.com/en-us/geforce/graphics-cards/30-series/rtx-3090/>, Accessed: 22-Oct-2024, 2020.
- [48] Centro de Investigación en Tecnologías Software y Sistemas Multimedia (CITSEM), *Centro de Investigación en Tecnologías Software y Sistemas Multimedia (CITSEM)*, <https://citsem.ucm.es/>, Accessed: 22-Oct-2024.
- [49] N. Instruments, *Peak Signal-to-Noise Ratio as an Image Quality Metric*, Último acceso: 29 de octubre de 2024, 2024. dirección: <https://www.ni.com/en/shop/data-acquisition-and-control/add-ons-for-data-acquisition-and-control/what-is-vision-development-module/peak-signal-to-noise-ratio-as-an-image-quality-metric.html>.
- [50] Naciones Unidas, *Objetivos de Desarrollo Sostenible*, <https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/>, Accedido: 16 de enero de 2025, 2025.
- [51] O. Mendez, S. Izadi, R. Newcombe, J. Shotton, A. Fitzgibbon y D. Kim, “Photorealistic 3D Reconstruction of Indoor Scenes with RGB-D Sensors”, *ACM Transactions on Graphics (TOG)*, vol. 39, n.º 4, págs. 1-15, 2020.
- [52] A. Wehr y U. Lohr, *LiDAR: Mapping the world in 3D*. London, UK: Taylor & Francis, 2001, Overview of LiDAR technology and its applications in various industries, ISBN: 978-0415299727.

Apéndice A

Resultados adicionales

A.1. Pruebas con NeRF

En la tabla A.1 se presentan los parámetros objetivos obtenidos del entrenamiento de modelos NeRF con diferentes conjuntos de datos. Además, la Figura A.1, Figura A.2 y Figura A.3 muestran detalles representativos de las imágenes generadas utilizando los modelos entrenados. Se probaron diferentes configuraciones de número de vistas, número de iteraciones y cantidad de rayos de luz que se muestrean en cada paso de la iteración en busca de un resultado más optimizado.

<i>Dataset</i>	Número de Vistas	Número de Iteraciones	Número de rayos tomados por iteración	Tiempo de Entrenamiento (h)	PSNR (dB)	Pérdida
<i>Flowers</i>	10	200,000	4,096	17.34	31.03	0.0019
<i>Flowers</i>	30	200,000	4,096	17.38	29.72	0.0024
<i>Room</i>	10	200,000	4,096	17.36	37.53	0.00067
<i>Room</i>	35	200,000	2,048	8.9	35.1	0.0012
<i>Horns</i>	10	200,000	4,096	17.34	29.16	0.0038
<i>Horns</i>	55	300,000	1,024	7.37	27.52	0.0049
<i>Trex</i>	10	200,000	4,096	17.38	29.26	0.0041
<i>Trex</i>	48	100,000	8,192	17.32	27.5	0.005
<i>Orchids</i>	10	200,000	4,096	17.43	24.93	0.0079
<i>Orchids</i>	22	200,000	4,096	17.44	23.22	0.011

Tabla A.1. Resultados del entrenamiento con NeRF para diferentes *datasets*

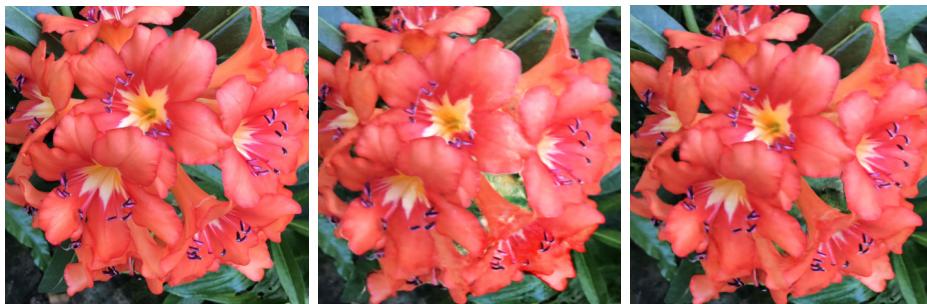
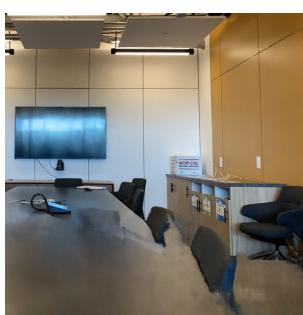
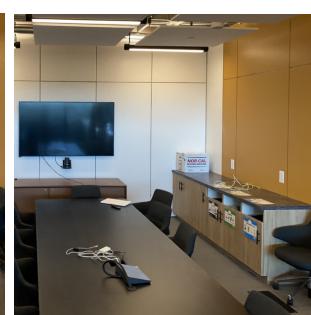
Flowers*Room**Ground Truth**NeRF 10 view**NeRF all views*

Figura A.1. Comparación visual de los resultados obtenidos del entrenamiento de modelos NeRF con diferentes conjuntos de imágenes - Parte 1.

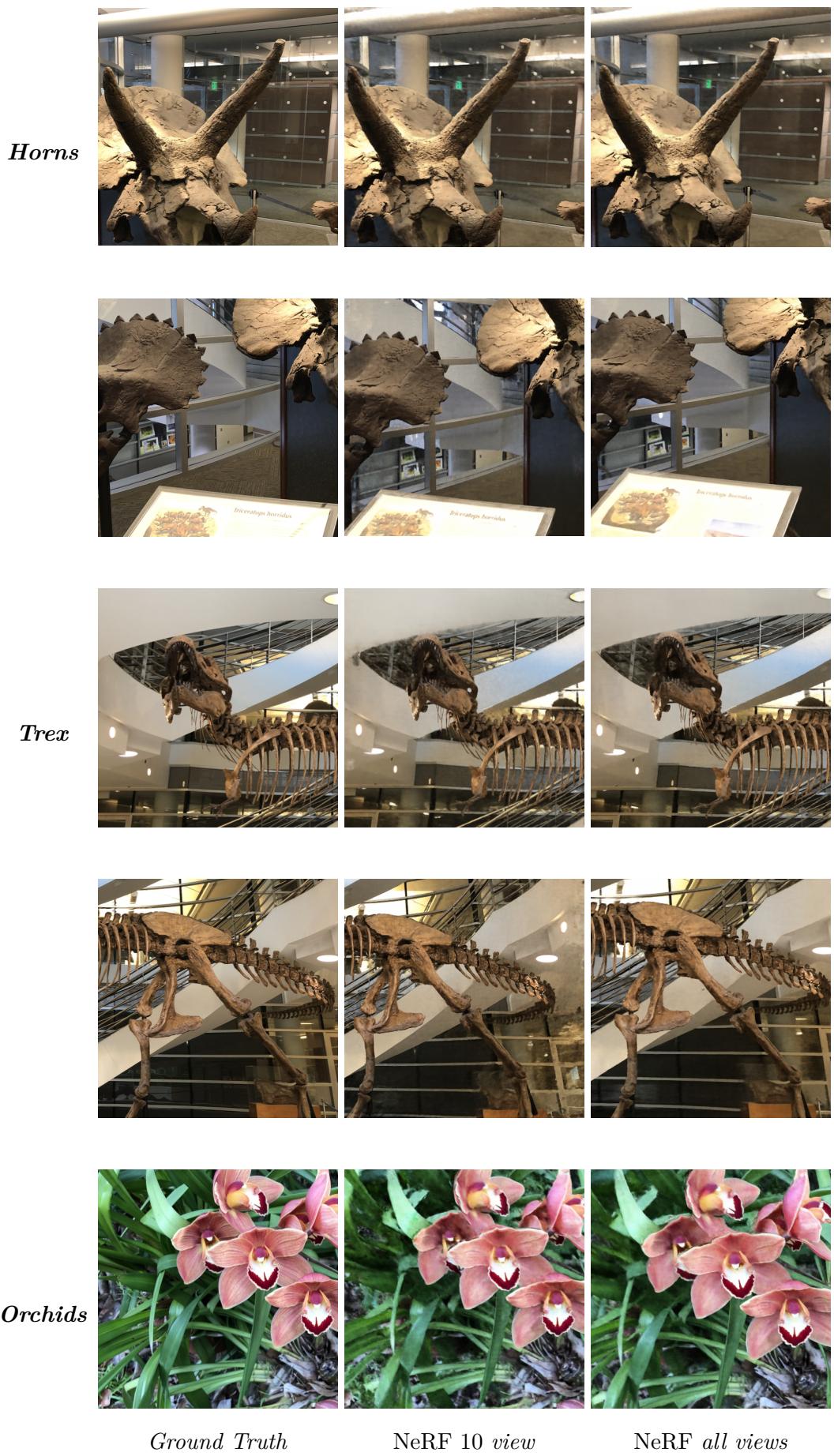


Figura A.2. Comparación visual de los resultados obtenidos del entrenamiento de modelos NeRF con diferentes conjuntos de imágenes - Parte 2.

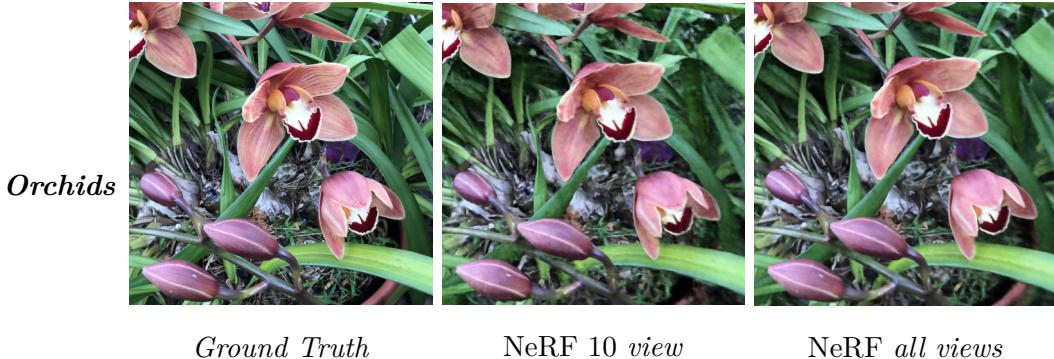


Figura A.3. Comparación visual de los resultados obtenidos del entrenamiento de modelos NeRF con diferentes conjuntos de imágenes - Parte 3.

A.2. Pruebas con DS-NeRF

En este caso, se presentan los resultados obtenidos del entrenamiento y prueba de modelos con DS-NeRF, utilizando los mismos conjuntos de datos que para los modelos NeRF. En la tabla A.2 se presentan los parámetros objetivos obtenidos y en la Figura A.4, Figura A.5 y Figura A.6, se muestran detalles significativos de las imágenes generadas.

Dataset	Número de Vistas	Número de Iteraciones	Número de rayos tomados por iteración	Tiempo de Entrenamiento (h)	PSNR (dB)	Pérdida
<i>Flowers</i>	10	200,000	4,096	25.76	27.09	0.0047
<i>Flowers</i>	30	200,000	4,096	34	27.44	0.0049
<i>Room</i>	10	200,000	4,096	29.1	35.31	0.001
<i>Room</i>	35	200,000	4,096	30.02	33.86	0.0018
<i>Horns</i>	10	200,000	4,096	28.59	25.51	0.0078
<i>Horns</i>	55	200,000	4,096	29.67	23.96	0.012
<i>Trex</i>	10	200,000	4,096	29.23	26.45	0.0056
<i>Trex</i>	48	300,000	1,024	12.25	22.97	0.023
<i>Orchids</i>	10	200,000	4,096	28,01	21.53	0.018
<i>Orchids</i>	10	400,000	512	13.87	20.05	0.025

Tabla A.2. Resultados del entrenamiento con DS-NeRF para diferentes *datasets*

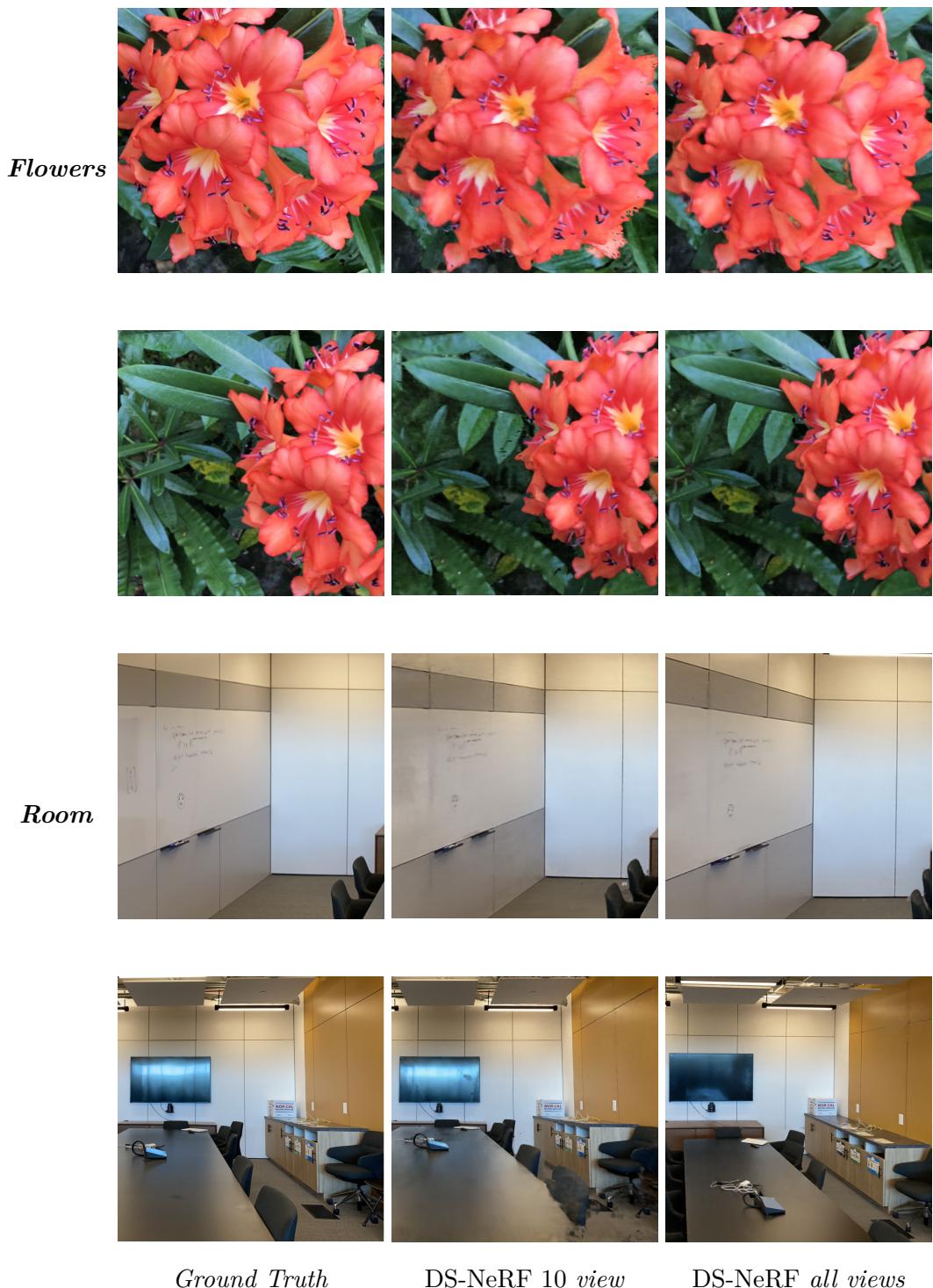


Figura A.4. Comparación visual de los resultados obtenidos del entrenamiento de modelos NeRF con diferentes conjuntos de imágenes - Parte 1.

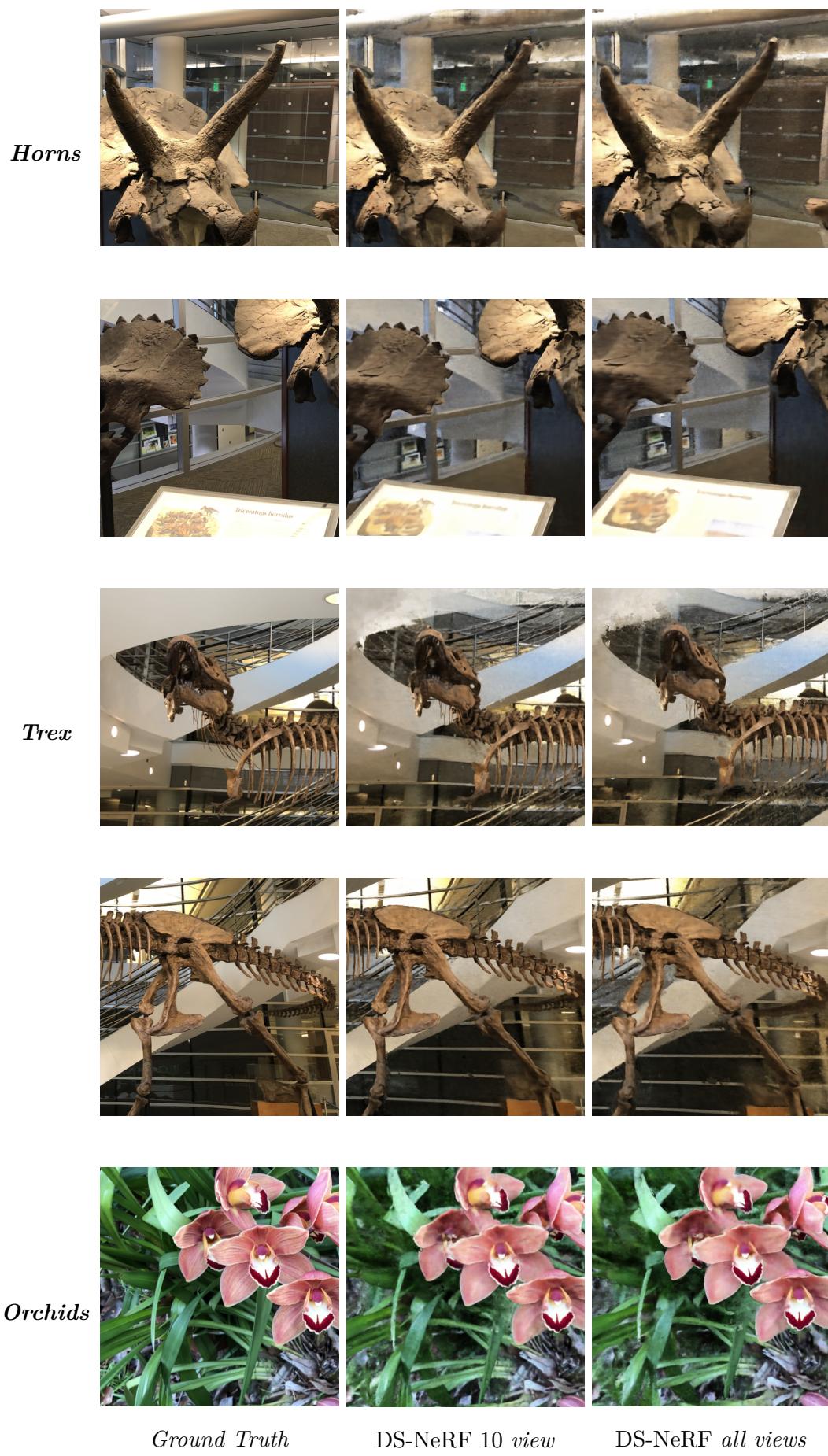


Figura A.5. Comparación visual de los resultados obtenidos del entrenamiento de modelos NeRF con diferentes conjuntos de imágenes - Parte 2.

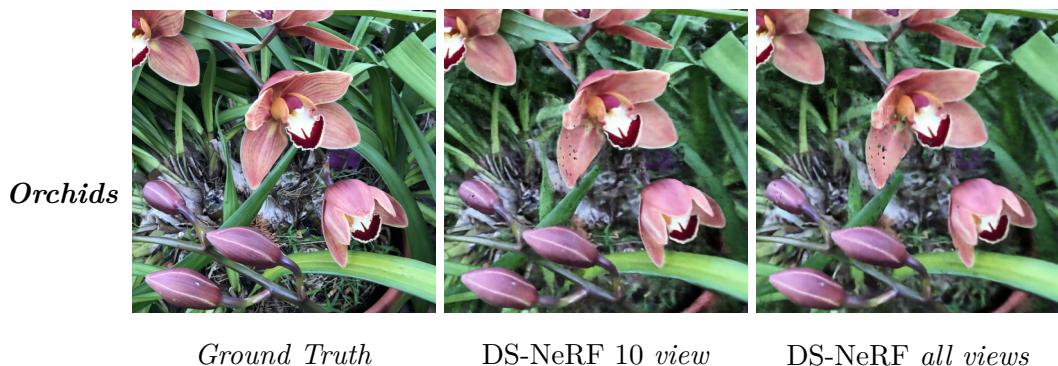


Figura A.6. Comparación visual de los resultados obtenidos del entrenamiento de modelos NeRF con diferentes conjuntos de imágenes - Parte 3.

A.3. Pruebas con modificaciones

Con el fin de proporcionar una comparación más detallada de la calidad visual de las imágenes generadas por NeRF, DS-NeRF y las modificaciones propuestas, se incluye en este anexo una presentación a mayor escala de los resultados. Aunque en el apartado de resultados ya se muestran algunos fragmentos, se ha decidido ofrecer estas imágenes a mayor escala para facilitar un análisis más exhaustivo de las características visuales, como la nitidez, la precisión de los bordes y detalles de las texturas.



Ground Truth



NeRF

Figura A.7. Comparación visual de la predicción del color de la escena conseguida con cada tecnología - Parte 1.



DS-NeRF



DS-NeRF MOD



NeRF MOD

Figura A.8. Comparación visual de la predicción del color de la escena conseguida con cada tecnología - Parte 2.