UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Escenas 3D mediante campos de radiancia neuronales: Plataforma colaborativa de entrenamiento, optimización y renderizado

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Granell Robles, Pablo

Tutor/a: Juan Lizandra, María Carmen

CURSO ACADÉMICO: 2023/2024

Resumen

En este TFG se ha desarrollado una plataforma para facilitar el entrenamiento, renderizado y visualización de escenas mediante campos de radiancia neuronal, incluyendo comprensión de la escena vía segmentación panóptica. La plataforma: 1) inicializa una escena desde un vídeo capturado manualmente; 2) entrena una escena mediante campos de radiancia neuronal a partir de este vídeo; 3) optimiza la escena con técnicas de primitivas de gráficos neuronales instantáneas. Su visualización se realiza en tiempo real desde un cliente visor. Un cliente remoto tiene control completo de la visualización del cliente visor desde cualquier red doméstica. Además, la plataforma permite la segmentación panóptica de los objetos en la escena manteniendo la consistencia temporal independientemente del ángulo. La plataforma también permite la creación, eliminación y manipulación de los objetos de la escena. Para ello altera el mapa de profundidad de la escena y a partir de ahí edita la textura. El objetivo principal es desarrollar una plataforma fácil de usar para un profesional no informático que pretenda crear escenas realistas para diferentes propósitos de una forma simple. Los subojetivos incluyen optimizar el entrenamiento de la escena 3D con técnicas de primitivas de gráficos neuronales instantáneas, crear un instalador fácil de usar que automatice toda la puesta en marcha e instalación del cliente visor/remoto, un túnel que funcione por códigos sencillos facilitando la conexión entre ambos clientes incluso en redes domésticas restrictivas, conexión con la plataforma de entrenamiento (GPU propia, Kaggle, etc.) para una mayor facilidad y un cliente visor fácil e intuitivo.

Palabras clave: Campos de resplandor, Plataforma, Tensor, Segmentación Panóptica, MLP, 3D, Inferencia, Tiempo Real, Control remoto, colaborativo, IA

Resum

En aquest TFG s'ha desenvolupat una plataforma per facilitar l'entrenament, la renderització i la visualització d'escenes mitjançant camps de radiància neuronal, incloent-hi comprensió de l'escena via segmentació panòptica. La plataforma: 1) inicialitza una escena des d'un vídeo capturat manualment; 2) entrena una escena mitjançant camps de radiància neuronal a partir d'aquest vídeo; 3) optimitza l'escena amb tècniques de primitives de gràfics neuronals instantànies. La seva visualització es fa en temps real des d'un client visor. Un client remot podrà tenir control complet de la visualització del client visor des de qualsevol xarxa domèstica. A més, la plataforma permet la segmentació panòptica dels objectes a l'escena mantenint la consistència temporal independentment de l'angle. La plataforma també permet la creació, l'eliminació i la manipulació dels objectes de l'escena. Per fer-ho altera el mapa de profunditat de l'escena i a partir d'açí edita la textura. L'objectiu principal és crear una plataforma fàcil d'usar per a un professional no informàtic que pretengui crear escenes realistes per a diferents propòsits d'una manera simple. Els subobjectius inclouen optimitzar l'entrenament de l'escena 3D amb tècniques de primitives de gràfics neuronals instantànies, crear un instal·lador fàcil d'usar que automatitze tota la posada en marxa i instal·lació del client visor/remot, un túnel que funcione per codis senzills facilitant la connexió entre tots dos clients fins i tot en xarxes domèstiques restrictives, connexió amb la plataforma d'entrenament (GPU pròpia, Kaggle, etc.) per a més facilitat i un client visor fàcil i intuïtiu.

Paraules clau: Camps de resplendor, Plataforma, Tensor, Segmentació Panòptica, MLP, 3D, Inferència, Temps Real, Control remot, col·laboratiu, IA

Abstract

This TFG develops a platform to facilitate the training, rendering and visualization of scenes using neural radiance fields, including scene understanding via panoptic segmentation. The platform: 1) initializes a scene from a manually captured video; 2) train a scene using neural radiance fields from this video; 3) optimize the scene with instant neural graphics primitives techniques. Its visualization is achieved in real time from a viewer client. A remote client has complete control of the viewer client display from any home network. Additionally, the platform enables panoptic segmentation of objects in the scene while maintaining temporal consistency regardless of angle. The platform also allows the creation, deletion and manipulation of objects in the scene. To do this, alter the depth map of the scene and from there edit the texture. The main goal is to create an easy-to-use platform for a non-IT professional who aims to create realistic scenes for different purposes in a simple way. Subobjectives include optimizing 3D scene training with instant neural graphics primitives techniques, creating an easy-to-use installer that automates the entire setup and installation of the viewer/remote client, a tunnel that works by simple codes making it easy to connect between both clients even on restrictive home networks, connection with the training platform (own GPU, Kaggle, etc.) for greater ease and an easy and intuitive viewer client.

Key words: Radiance fields, Platform, Tensor, Panoptical Segmentation, MLP, 3D, Inference, Real Time, Remote Control, Collaborative, AI

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

La última década ha sido testigo de un avance significativo en la visualización y generación de escenas tridimensionales, impulsado por los notables desarrollos en inteligencia artificial, aprendizaje profundo y gráficos computacionales [1].

Uno de los avances más notables en este campo es el uso de campos de radiancia neuronal (también conocidos como NeRF, por sus siglas en inglés), que permiten crear representaciones 3D realistas a partir de conjuntos de imágenes 2D capturadas desde diferentes ángulos. Los NeRF, introducidos por Mildenhall et al. [2], han demostrado ser una herramienta poderosa para la síntesis de nuevas vistas desde ángulos arbitrarios. Su capacidad para generar imágenes altamente detalladas y fotorrealistas ha revolucionado la forma en que se generan y representan las escenas tridimensionales.

Los campos de radiancia neuronal han encontrado una amplia variedad de aplicaciones en diferentes sectores gracias a su capacidad para generar representaciones tridimensionales detalladas y realistas. La industria del cine y los videojuegos ha adoptado NeRF para crear efectos visuales realistas y detallados, así como para el renderizado de entornos completamente generados por ordenador. Esta tecnología permite a directores y desarrolladores de videojuegos capturar entornos y objetos del mundo real y recrearlos digitalmente con gran precisión, lo que no solo mejora la calidad visual de las producciones, sino que también las hace más económicas e incluso ofrece una alternativa viable a la fotogrametría tradicional [3].

Los arquitectos y diseñadores también utilizan NeRF para visualizar, editar y validar proyectos arquitectónicos y de diseño de interiores, ya que esta tecnología aumenta las capacidades espaciales de estos entornos. Esta tecnología permite crear y editar modelos tridimensionales precisos de edificios y espacios arquitectónicos, lo que facilita la presentación de complejos proyectos a clientes y la toma de decisiones durante el proceso de diseño [4].

Este TFG se centra en el desarrollo de una plataforma que facilita la puesta en marcha, el entrenamiento, la renderización y la visualización de escenas 3D mediante campos de radiancia neuronal, entre otros métodos. La plataforma está diseñada para democratizar el acceso a la creación de escenas 3D realistas, ofreciendo una herramienta intuitiva, sencilla y accesible a profesionales sin formación en informática. La plataforma está diseñada para abstraer las complejidades técnicas subyacentes, como la conexión cliente-servidor, la transferencia y el procesamiento de datos. Su diseño intuitivo permite a los usuarios aprender a utilizarla en menos de un día, sin sacrificar la disponibilidad de funciones avanzadas para quienes las necesiten.

Los siguientes apartados detallan la motivación que impulsa este TFG, los objetivos propuestos y la estructura de la memoria.

1.1 Motivación

La motivación que hay detrás de este TFG reside en la creciente demanda de herramientas que utilicen tecnología de última generación sin las barreras que esta produce debido a la complejidad técnica propia del campo en el que se encuentra. En este contexto, surge la necesidad de una plataforma que facilite a los profesionales de diversas áreas la creación y manipulación de escenas 3D de alta calidad, sin requerir conocimientos técnicos avanzados en el campo de la visualización y generación tridimensional.

A pesar de su interés y aplicabilidad a nuestro caso de uso, las tecnologías actuales presentan ciertas barreras que limitan su accesibilidad:

- **Complejidad técnica:** la mayoría de las implementaciones de NeRF requieren conocimientos avanzados de programación.
- **Requisitos de hardware:** muchas soluciones existentes demandan recursos computacionales costosos y especializados en los que algunos profesionales pueden no estar especializados, como computación paralela y GPUs.
- **Falta de interfaces intuitivas:** las herramientas actuales carecen de interfaces amigables para usuarios no técnicos y, además, obligan a utilizar terminales en algún momento del proceso.

Esta plataforma se ha desarrollado con el objetivo de ser accesible y fácil de usar, para que los usuarios puedan centrarse en las ventajas de la tecnología y en el contenido de sus proyectos, sin verse obstaculizados por las dificultades técnicas asociadas al procesamiento y la renderización de escenas 3D. El entrenamiento puede ejecutarse en un servidor remoto, lo que garantiza que incluso aquellos con recursos de hardware limitados puedan beneficiarse de estas tecnologías de visualización avanzadas.

Este TFG se alinea con la tendencia creciente en la industria hacia soluciones colaborativas y remotas, que facilitan la cooperación entre equipos distribuidos geográficamente, como se ha evidenciado con el auge del trabajo remoto durante y tras la pandemia. La capacidad de controlar y visualizar escenas en tiempo real desde cualquier ubicación abre nuevas posibilidades para la colaboración en proyectos de diseño, arquitectura, entretenimiento y otros campos.

La facilidad de uso de la plataforma y su capacidad de colaboración remota la convierten en una prometedora herramienta de apoyo en diferentes ámbitos, como el de las terapias. En particular, puede ser de gran utilidad para la estimulación cognitiva de adultos [5] y para la reducción del estigma asociado a la terapia, al presentarse como una experiencia interactiva con elementos multijugador [6].

La tecnología NeRF se presenta como una solución prometedora para la generación de escenas 3D, y varias soluciones, como Jawset [7] y Photonerf [8], ya se están adaptando a ella (véase la figura 1.1 que muestra un fotograma de postshot subido por «studio Duckbill» publicitando Jawset Postshot en Twitter (1)). Sin embargo, la complejidad inherente a esta tecnología y su implementación subyacente aún presentan desafíos para su adopción generalizada.

(1) Fuente: Vídeo https://x.com/DuckbillStudio/status/1761186168828490152

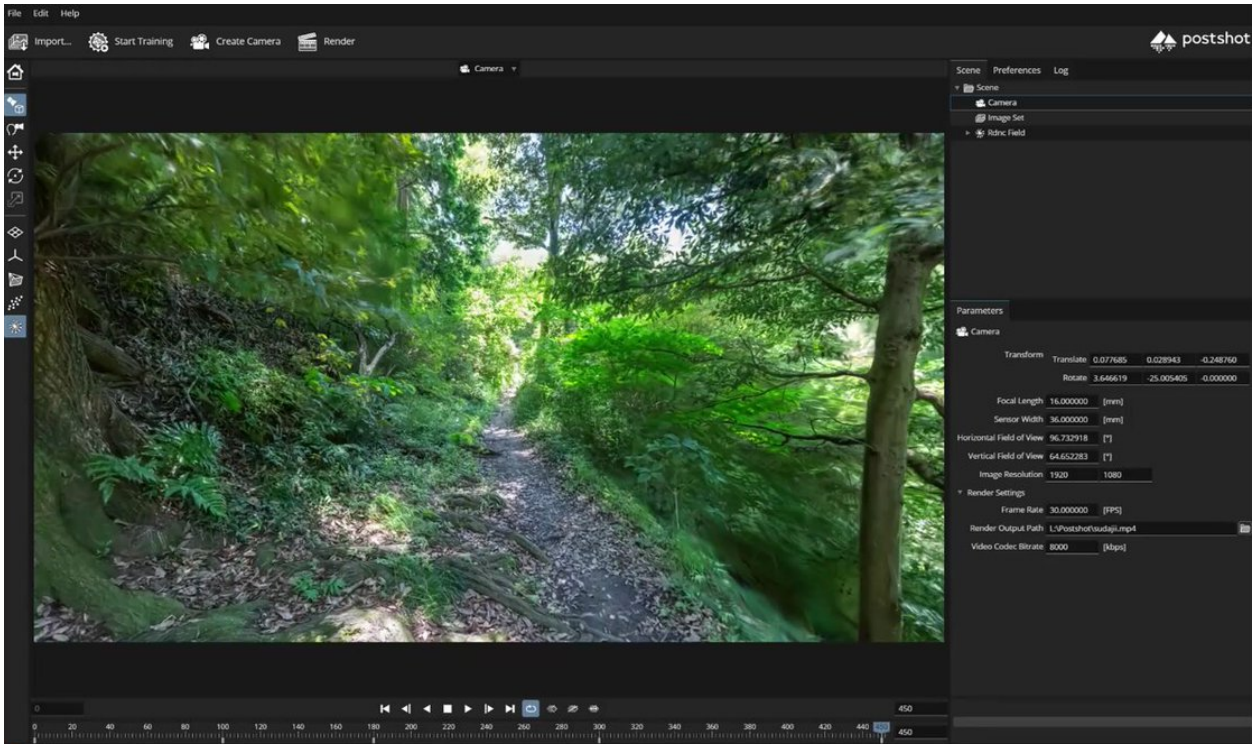

Figura 1.1: Fotograma Postshot renderizando una escena de un bosque con campos de radiancia neuronal

En última instancia, esta plataforma tiene el potencial de democratizar el acceso a las tecnologías de visualización y generación de escenas 3D, allanando el camino para su adopción en una amplia gama de industrias.

1.2 Objetivos

El objetivo principal es el desarrollo de una plataforma (llamada *Cloudstudio*) que facilite el entrenamiento, la renderización y la visualización de escenas 3D utilizando campos de radiancia neuronal. Esta plataforma busca abstraer la complejidad inherente a NeRF, democratizando el acceso a esta tecnología y haciéndola accesible a usuarios sin conocimientos técnicos avanzados.

Para lograrlo, la plataforma tiene como subobjetivos:

1. Implementar un instalador intuitivo que automatice la resolución de dependencias y la inicialización del servidor.
2. Configurar automáticamente el servidor responsable de la renderización de las escenas, ocultando los requisitos de computación e instalación al usuario.
3. Procesar automáticamente los datos de la escena a partir de un vídeo capturado manualmente, incluyendo la carga de archivos al servidor y la extracción de la información necesaria para el entrenamiento.
4. Entrenar la escena utilizando NeRF, optimizándolo gradualmente para mantener la responsividad y garantizar un rendimiento fluido en tiempo real.
5. Implementar un visor intuitivo y fácil de usar que permita la visualización en tiempo real de la escena desde cualquier red, con capacidad de control remoto y sincronización de cámara entre clientes.

La plataforma se basa en NeRFstudio [9], un backend de código abierto popular en la comunidad de investigación de NeRF. Esto permite una configuración exhaustiva del sistema, facilita la implementación de mejoras y ofrece una amplia gama de métodos NeRF. Por defecto, la plataforma utiliza el método más rápido con ajustes optimizados para el tipo de servidor y caso de uso propuestos, pero esto se puede modificar fácilmente a través de una interfaz intuitiva.

Este TFG también explora la optimización del proceso de entrenamiento utilizando técnicas avanzadas como Gaussian Splatting [10], que ha demostrado una mayor eficiencia en el entrenamiento, reduciendo los tiempos de procesamiento y minimizando los artefactos visuales.

En resumen, esta plataforma busca democratizar el acceso a la creación de escenas 3D realistas, simplificando el proceso y abstrayendo las complejidades técnicas del usuario. Su diseño intuitivo y su arquitectura robusta, basada en tecnologías de vanguardia en inteligencia artificial, aprendizaje profundo y gráficos computacionales, permiten una experiencia de creación fluida y transparente. La plataforma tiene el potencial de impactar en múltiples áreas, desde la creación de contenido multimedia hasta aplicaciones en realidad virtual y aumentada, ofreciendo una solución robusta y accesible para la generación y manipulación de entornos 3D de alta calidad.

1.3 Metodología de Desarrollo

Para desarrollar este TFG, hemos seguido una metodología iterativa que comprende las siguientes fases de desarrollo:

1. **Investigación y análisis:** se llevó a cabo una exhaustiva investigación sobre la utilización de campos de radiancia neuronal para la síntesis de escenas 3D y de las diversas aplicaciones que pueden surgir. Además, se analizaron las diferentes implementaciones y enfoques existentes, identificando sus ventajas y desventajas, para poder elegir uno una vez especificados los requisitos del TFG.

2. **Definición de requisitos:** basándose en la investigación realizada y en las necesidades del TFG, se definieron los requisitos de la plataforma en función del caso de uso.

3. **Estudio de posibles soluciones:** se exploraron tres posibles soluciones para abordar los desafíos planteados en el desarrollo de la plataforma. Se analizaron las ventajas y desventajas de cada solución teniendo en cuenta los requisitos y especificaciones definidos en la fase anterior.

4. **Elección de servicios y protocolos:** una vez elegida la arquitectura principal, se seleccionan los servicios y protocolos necesarios para integrar toda la infraestructura en una sola plataforma.

5. **Implementación:** la plataforma se desarrolló de forma iterativa, comenzando con un prototipo básico y añadiendo funcionalidades progresivamente, hasta optimizar los resultados. Se siguió un enfoque de desarrollo ágil, con ciclos cortos de implementación y pruebas.

6. **Resultados:** se realizaron pruebas exhaustivas de cada componente y de la plataforma en su conjunto con el objetivo de comprobar su rendimiento y usabilidad en cada iteración. Además, se evaluó la plataforma con usuarios potenciales, se recopiló su feedback y se realizaron ajustes y mejoras en base a sus comentarios.

7. **Documentación:** se elaboró esta memoria como una documentación detallada sobre el funcionamiento de la plataforma, que incluye un marco teórico, una descripción más detallada de la implementación y un historial de la línea de pensamiento seguida a través de las diferentes implementaciones, que refleja la metodología iterativa utilizada.

Como se puede ver en la lista anterior, para desarrollar la plataforma *Cloudstudio* se ha seguido una metodología de desarrollo iterativo similar a Extreme Programming (XP). Aunque no se adhiere estrictamente a todos los principios de esta, sí incorpora varios de sus elementos clave.

Extreme Programming es una metodología de desarrollo ágil que se centra en la adaptabilidad, la retroalimentación y la entrega frecuente de software funcional. A continuación, se analiza la relación entre la metodología utilizada en este TFG y los principios de XP.

1.3.1. Relación con Extreme Programming

Como se puede observar en la tabla 1.1, la metodología seguida en el desarrollo de *Cloudstudio* incorpora varios elementos de XP, como el desarrollo iterativo, el diseño simple, las pruebas y la propiedad colectiva del código.

Principio de XP	**Implementado**	**Descripción**
Planificación iterativa	Parcialmente	Se definieron objetivos y requisitos generales, pero también se adaptaron durante el desarrollo como se ve en las primeras versiones.
Pequeñas entregas	Sí	Desde las primeras implementaciones se añadieron funcionalidades de forma gradual hasta la versión final.
Diseño simple	Sí	Se priorizó la simplicidad en la arquitectura y la interfaz de usuario.
Pruebas	Sí	Se realizaron pruebas unitarias y de integración para verificar la funcionalidad del sistema.
Programación en parejas	Sí	El TFG se desarrolló de forma colectiva junto con la ayuda del tutor.
Propiedad colectiva	Sí	El TFG es de código abierto en su totalidad, disponible en repositorio público.
Integración continua	Sí	Se realizaron integraciones frecuentes y se automatizó el proceso.
Ritmo sostenible	Sí	Se mantuvo un ritmo de trabajo constante y se evitaron las jornadas excesivas.
Cliente in situ	Parcialmente	No había un cliente pero se buscó un feedback constante sobre la funcionalidad esperada.
Estándares de codificación	Sí	Se siguieron estándares de codificación para garantizar la legibilidad del código.

Tabla 1.1: Relación entre la metodología seguida en *Cloudstudio* y los principios de XP

1.3.2. Conclusión

La metodología de desarrollo iterativo utilizada en este TFG, inspirada en los principios de XP, ha demostrado ser efectiva para el desarrollo de la plataforma *Cloudstudio*. La iteratividad, la simplicidad del diseño, las pruebas constantes y la refactorización del código han contribuido a la calidad del software y a la satisfacción del cliente.

Si bien la metodología seguida en este TFG no es una implementación estricta de Extreme Programming, sí adopta varios de sus principios clave para optimizar el proceso de desarrollo y la calidad del producto final.

1.4 Estructura de la memoria

Esta memoria contiene seis capítulos, además de la bibliografía y los anexos. Su estructura es la siguiente:

- El primer capítulo, la introducción, presenta la motivación detrás de este TFG, el objetivo general y los objetivos específicos, la metodología seguida para llevar a cabo el TFG y la estructura de la memoria.
- El segundo capítulo presenta el marco teórico de los campos de radiancia neuronal, incluyendo detalles de su funcionamiento, la evolución del campo de la síntesis de escenas 3D, su evolución desde su introducción en 2020, sus aplicaciones actuales y su estado actual en términos de eficiencia y funcionalidad.

  También se incluye una sección que describe NeRFstudio, una plataforma de código abierto que se utilizará como base para el desarrollo de la plataforma propuesta en este TFG.
- En el tercer capítulo se analiza el problema existente y se detallan las especificaciones y requisitos necesarios para el desarrollo de la plataforma. Se presenta un estudio comparativo de tres posibles soluciones, analizando sus ventajas y desventajas como criterios para su selección.

  Al final, se elige estrictamente qué solución se va a llevar a cabo y se dejan los detalles y las decisiones sobre su implementación para el apartado de detalles de la solución.
- El cuarto capítulo presenta la solución propuesta y detalla su implementación práctica, incluyendo el servidor, el visor, el instalador y la arquitectura cliente-servidor.
- El quinto capítulo presenta los resultados obtenidos, incluyendo el feedback recibido, junto con los cambios realizados y un análisis de las fortalezas y debilidades de la implementación.
- El sexto capítulo, las conclusiones reflexionan sobre los resultados obtenidos, la relación del TFG con los conceptos estudiados en la carrera y las posibles líneas de trabajo futuro que surgen a raíz del estado de implementación final.
- Luego, se encuentra la bibliografía siguiendo el estilo APA.
- En los anexos se encuentran los Objetivos de Desarrollo Sostenible (ODS) y un glosario con los términos relevantes para esta memoria.

CAPÍTULO 2
Campos de radiancia neuronal

La representación de escenas 3D a partir de imágenes 2D ha sido un objetivo fundamental en la visión artificial y los gráficos por ordenador. Por ejemplo, puede verse en el número de publicaciones anuales en SIGGRAPH. Antes de la llegada de los NeRF, las técnicas dominantes se basaban principalmente en dos enfoques:

- **Fotogrametría:** esta técnica consiste en capturar múltiples imágenes de un objeto o escena desde diferentes ángulos, como se ilustra en la figura 2.1 con Photo Tourism. En ella se muestra una vista existente de una colección de fotos no organizada y, mediante el uso de algoritmos tradicionales, cómo se reconstruye la geometría 3D y la textura para crear una escena navegable.

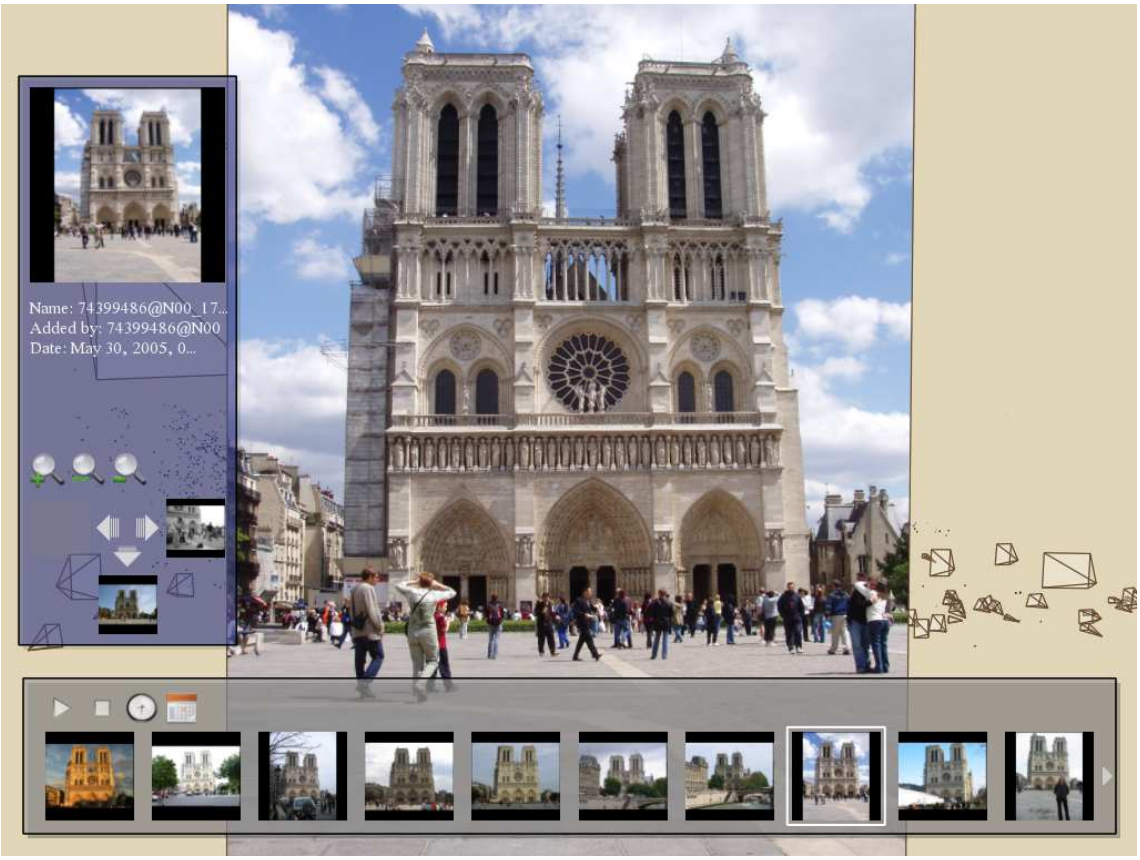

Figura 2.1: Vista existente de una colección de fotos no organizada (2)

(2) Fuente: Figura 1(c) de Snavely et al. [11]

- **Reconstrucción basada en la profundidad:** estos métodos utilizan sensores de profundidad, como Kinect con sus cámaras RGB-D, como se muestra en la figura 2.2. Esta nube de puntos muestra «Kinect» como ejemplo de cámara RGB-D, representada en color cian, junto con la nube de puntos obtenida por el escáner láser «FARO LS880», representada en color blanco, para capturar información de profundidad directamente. Esta información se utiliza posteriormente para generar mallas 3D.

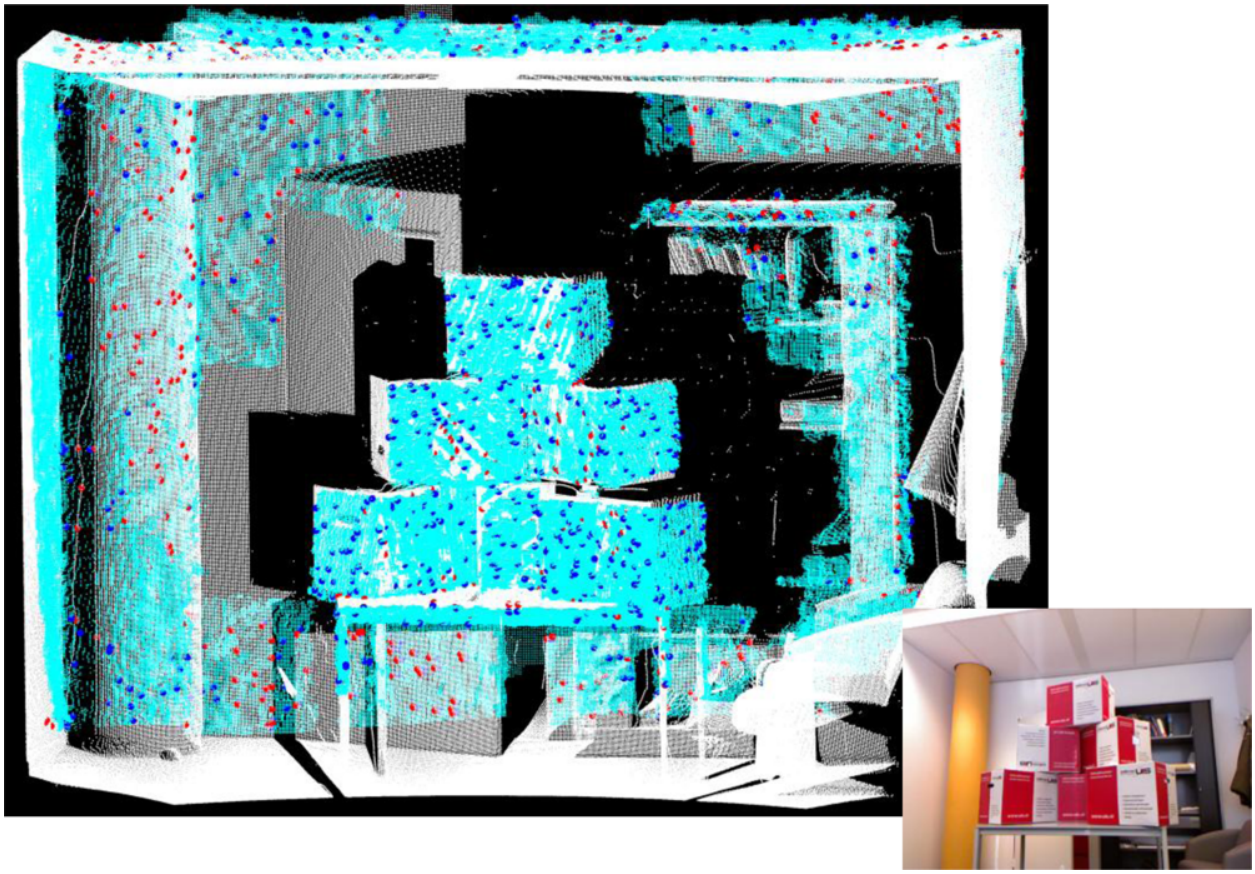

Figura 2.2: Comparación de la nube de puntos de «Kinect» con «FARO LS880» (3)

A pesar de ser tecnologías de vanguardia en su momento, estos métodos tradicionales a menudo presentaban limitaciones en cuanto a:

- **Calidad visual:** la fotogrametría puede generar artefactos visibles en áreas con superficies reflectantes u oclusiones complejas. La reconstrucción basada en la profundidad, por otro lado, a menudo produce modelos 3D con geometría simplificada o ruidosa, dependiendo de la sensibilidad de las cámaras de profundidad utilizadas. Ambos métodos pueden carecer de la capacidad para capturar detalles finos de la escena.

- **Capacidad para manejar oclusiones complejas:** tanto la fotogrametría como la reconstrucción basada en la profundidad pueden tener dificultades para reconstruir áreas ocluidas [13] en las imágenes de entrada, ya que no extrapolan información más allá de las muestras capturadas. Esto implica la necesidad de una gran cantidad de muestras para cubrir la escena completa y minimizar la presencia de huecos en la geometría reconstruida.

(3) Fuente: Figura 6 de Khoshelham et al. [12]

- **Eficiencia computacional:** el procesamiento de grandes conjuntos de datos de imágenes o nubes de puntos puede ser computacionalmente intensivo, requiriendo una gran capacidad de almacenamiento y potencia de procesamiento. Esto limita la escalabilidad de estos métodos, especialmente en comparación con los métodos basados en campos de radiancia neuronal, que pueden aprovechar la paralelización y la aceleración por hardware (GPU) para mejorar su eficiencia.

La introducción de los campos de radiancia neuronal en marzo de 2020 por Mildenhall et al. [2] marcó un punto de inflexión en el campo de la representación de escenas 3D. Su novedoso enfoque, basado en redes neuronales entrenadas con aprendizaje profundo, permitió sintetizar nuevas vistas de la escena desde ángulos arbitrarios con una alta fidelidad visual, utilizando solo un conjunto disperso de imágenes de entrada. Esta técnica superó muchas de las limitaciones de los métodos tradicionales, abriendo nuevas posibilidades para la generación y manipulación de escenas 3D.

A diferencia de los métodos tradicionales que construyen explícitamente una malla 3D, los NeRF representan la escena como un campo continuo de densidad y color. Este campo se modela mediante una red neuronal, que se entrena utilizando un conjunto de imágenes de entrada para aprender la correspondencia entre posiciones 3D, direcciones de vista y valores de color y densidad. La representación paramétrica y compacta del campo de radiancia permite sintetizar cualquier vista arbitraria de la escena renderizando los rayos que la atraviesan y acumulando los colores y las densidades a lo largo de su trayectoria. Este proceso genera representaciones pseudo-3D de alta fidelidad visual, capturando detalles finos de la escena y manejando oclusiones complejas de forma natural.

2.1 Funcionamiento de los NeRF

En esencia, un NeRF es una función parametrizada, entrenada mediante aprendizaje profundo, que mapea una posición 3D y una dirección de vista a un color y una densidad. Esta función se implementa típicamente como una red neuronal multicapa (MLP), que toma como entrada la posición 3D y la dirección de vista y produce como salida el color RGB y la densidad correspondientes a esa posición y dirección.

El proceso de renderización con un NeRF comprende los siguientes pasos:

1. **Muestreo de rayos:** se trazan rayos desde la cámara virtual a través de cada píxel de la imagen que se desea renderizar.

2. **Consulta del campo de radiancia:** para cada rayo, se toman muestras a lo largo de su trayectoria en el espacio 3D. Estas muestras se utilizan como entrada para el NeRF, que devuelve el color y la densidad correspondientes a cada punto muestreado.

3. **Renderización volumétrica:** los colores y densidades muestreados se combinan utilizando una función de renderización volumétrica, como la integración de la ecuación de renderización, para producir el color final del píxel. La integración volumétrica realiza una combinación ponderada de los colores y densidades a lo largo del rayo, teniendo en cuenta la opacidad acumulada. Esto permite una representación natural de oclusiones y una transición suave entre diferentes regiones de la escena, sin requerir un paso de homogeneización de píxeles explícito como en algunos métodos fotogramétricos.

La capacidad de los NeRF para representar escenas complejas con alta fidelidad visual reside en su capacidad para aprender representaciones continuas de la geometría y la apariencia de la escena. Al codificar la información 3D en la función de campo de radiancia, los NeRF pueden manejar oclusiones complejas y representar detalles finos que a menudo se pierden con los métodos tradicionales basados en mallas explícitas.

2.1.1. Red Neuronal Multicapa

La red neuronal multicapa es el componente central de un NeRF, responsable de aprender la representación del campo de radiancia de la escena. Esta red mapea la información de posición 3D y dirección de vista a los valores de color y densidad correspondientes, permitiendo la reconstrucción de la escena a partir de un conjunto de imágenes de entrada.

La arquitectura típica de la MLP en NeRFs consta de varias capas completamente conectadas, con una función de activación no lineal en cada capa, como la unidad lineal rectificada (ReLU). La estructura general de la MLP es la siguiente:

1. **Entrada:** La entrada a la MLP es un vector de 5 dimensiones que codifica la posición 3D (x, y, z) y la dirección de vista 2D (θ, ϕ).
2. **Capas ocultas:** Cada capa oculta contiene un número determinado de neuronas, que aplican una función de activación no lineal a su entrada. Las funciones de activación más comunes en NeRFs son ReLU y, en algunos casos, funciones suaves como Softplus.

   Estas capas ocultas permiten a la MLP aprender una función no lineal compleja que mapea la posición 3D y la dirección de vista a los valores de color y densidad. Esta función, específica para cada escena, se obtiene durante el entrenamiento del NeRF. Por lo tanto, es necesario entrenar un NeRF para cada escena que se desea generar, ya que el entrenamiento no se puede generalizar a escenas no vistas. Sin embargo, existen investigaciones en curso que exploran la posibilidad de transferir el conocimiento aprendido entre escenas [14].
3. **Capa de salida:** la capa de salida produce un vector de 4 dimensiones que representa el color RGB (r, g, b) y la densidad volumétrica (σ) en la posición 3D y dirección de vista dadas. A diferencia de las capas ocultas, la capa de salida no utiliza una función de activación no lineal.

Los pesos y sesgos de las conexiones entre las neuronas en cada capa, conocidos como hiperparámetros de la red, se ajustan durante el proceso de entrenamiento mediante algoritmos de optimización basados en retropropagación y descenso de gradiente. Para guiar el entrenamiento y obtener una representación precisa de la escena, se utilizan funciones de pérdida que cuantifican la diferencia entre las imágenes renderizadas por el NeRF y las imágenes de referencia. Estas funciones de pérdida proporcionan una señal de error que se utiliza para ajustar los hiperparámetros de la red y minimizar la diferencia entre las imágenes renderizadas y las imágenes de referencia.

2.1.2. Función de Renderizado Volumétrico

La función de renderizado volumétrico integra la información de color y densidad a lo largo de la trayectoria de un rayo para generar el color final del píxel correspondiente. Este proceso simula la interacción de la luz con el volumen de la escena, teniendo en cuenta

la absorción y la dispersión de la luz a medida que atraviesa el medio. La función de renderizado volumétrico permite crear representaciones visualmente realistas que capturan efectos como oclusiones, sombras y translucidez.

La ecuación de renderizado volumétrico describe matemáticamente cómo la luz interactúa con un medio participativo, teniendo en cuenta la absorción y la dispersión de la luz a medida que atraviesa el volumen. En NeRFs, esta ecuación se integra numéricamente a lo largo de cada rayo trazado desde la cámara virtual a la escena, acumulando las contribuciones de color y densidad de los puntos muestreados a lo largo del rayo. Este proceso permite obtener una representación precisa de la apariencia de la escena desde la perspectiva de la cámara.

Las oclusiones, que se producen cuando un objeto bloquea la vista de otro, se manejan de forma natural durante el proceso de renderizado volumétrico. La opacidad acumulada a lo largo del rayo determina la contribución de cada punto muestreado al color final del píxel. A medida que un rayo atraviesa la escena, la opacidad acumulada aumenta, lo que reduce la contribución de los puntos posteriores a lo largo del rayo. Esto produce un efecto de oclusión, donde los objetos más cercanos a la cámara ocultan los objetos más lejanos, creando una sensación de profundidad realista.

La ecuación de renderizado volumétrico describe la interacción de la luz con un volumen. En el contexto de los NeRFs, esta ecuación se integra numéricamente [15] a lo largo de la trayectoria de cada rayo, como se muestra a continuación:

$$C(r) = \int_{t_n}^{t_f} T(t)\sigma(r(t))c(r(t), d)dt \tag{2.1}$$

Donde:

* $C(r)$ es el color del rayo r, que representa la acumulación de color a lo largo del rayo desde el punto de entrada t_n hasta el punto de salida t_f.

* $T(t)$ representa la transmitancia acumulada a lo largo del rayo hasta el punto t. La transmitancia indica la fracción de luz que no ha sido absorbida ni dispersada a lo largo del rayo hasta el punto t.

* $\sigma(r(t))$ es la densidad volumétrica en el punto $r(t)$ del rayo. La densidad volumétrica determina la probabilidad de que la luz interactúe con el medio en ese punto, contribuyendo a la absorción y la dispersión de la luz.

* $c(r(t), d)$ es el color emitido por el medio en el punto $r(t)$ en la dirección d, siendo d la dirección del rayo.

Para optimizar la eficiencia del renderizado, se utilizan diversas técnicas de muestreo que permiten seleccionar puntos a lo largo del rayo de manera inteligente, concentrando las muestras en las regiones más relevantes de la escena. Algunas de las técnicas de muestreo más comunes son:

- **Muestreo Estratificado:** esta técnica divide el rayo en intervalos y toma muestras en puntos aleatorios dentro de cada intervalo. Esto garantiza una distribución más uniforme de las muestras a lo largo del rayo, reduciendo el ruido en la imagen renderizada.

- **Muestreo por Importancia:** este método concentra las muestras en las regiones con mayor densidad volumétrica, donde la luz interactúa más con el medio. Al dedicar más recursos de muestreo a estas áreas, se mejora la eficiencia del renderizado y se reduce el ruido en las regiones más visibles de la imagen.

En resumen, la MLP actúa como una función de mapeo, aprendiendo una representación continua del campo de radiancia de la escena. Para ello, toma como entrada un vector de 5 dimensiones que representa la posición 3D y la dirección de vista. A través de una serie de transformaciones lineales (multiplicaciones de matrices) y no lineales (funciones de activación), la MLP transforma este vector de entrada en un vector de salida de 4 dimensiones que representa el color RGB y la densidad volumétrica en la posición 3D y dirección de vista dadas. Esta representación compacta y continua del campo de radiancia permite sintetizar nuevas vistas de la escena desde cualquier posición y dirección.

Posteriormente, se utiliza la ecuación de renderizado volumétrico para renderizar la escena, integrando la información de color y densidad aprendida por la MLP a lo largo de cada rayo trazado desde la cámara virtual. Este proceso produce una imagen final que representa la apariencia de la escena desde la perspectiva de la cámara.

2.2 Evolución de los NeRF

Desde su introducción, los NeRF han experimentado un rápido desarrollo, con numerosas mejoras en rendimiento, consistencia, funcionalidad y extensiones que abordan sus limitaciones y amplían sus capacidades. En esta sección, se revisan algunos de los modelos NeRF más relevantes que han surgido desde la publicación del trabajo original de Mildenhall et al. [2].

Los principales objetivos de la investigación en NeRF desde 2020 han sido mejorar la eficiencia del entrenamiento y la renderización, la capacidad para manejar escenas dinámicas y la calidad visual de las representaciones. Algunos de los modelos más relevantes que han contribuido a estos avances son (4):

- **Instant Neural Graphics Primitives (Instant NGP) [16]:** Instant NGP utiliza una representación alternativa del campo de radiancia basada en una codificación hash multirresolución. Esta representación permite un acceso eficiente a la información del campo de radiancia, acelerando significativamente el entrenamiento y la renderización de los NeRF. Instant NGP permite la interacción en tiempo real con escenas complejas, incluso en hardware de consumo.

  En el contexto de la investigación realizada para este trabajo, se ha evaluado Instant NGP, obteniendo resultados prometedores en términos de calidad visual y eficiencia. La representación del campo de radiancia obtenida con Instant NGP presenta un bajo nivel de ruido, y el entrenamiento es notablemente rápido. En un ordenador modesto (GTX 1650 Ti, 4 GB de VRAM), se ha logrado un rendimiento de aproximadamente 25 FPS, que se puede aumentar a 35 FPS con técnicas de reescalado. Instant NGP se presenta con una interfaz de usuario completa (como se muestra en la figura 2.3 que muestra un recorte de la interfaz de Instant NGP entrenando y renderizando a la vez la común escena del tractor de Lego.). Es importante destacar que Instant NGP requiere la extracción previa de las poses de la cámara y las matrices de proyección intrínseca a partir del conjunto de imágenes de entrada, lo que se puede realizar utilizando scripts auxiliares.

(4) Fuente: Lista de modelos elaborada en base a la información del modelo «nerfacto» [9]

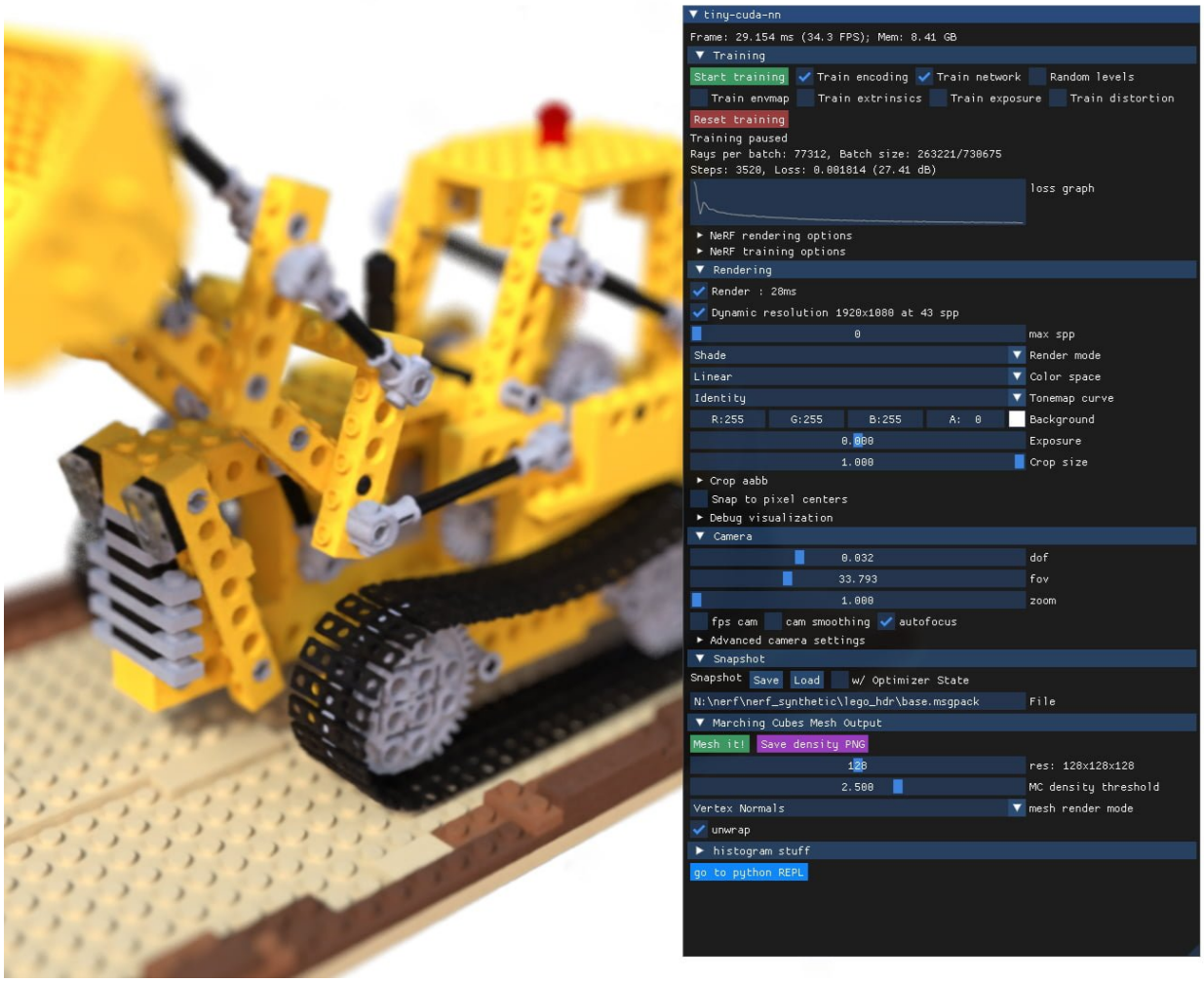

Figura 2.3: Interfaz de Instant NGP entrenando el tractor de Lego (5)

- **MipNeRF-360 [17]:** MipNeRF-360 es una extensión de MipNeRF diseñada específicamente para manejar escenas a gran escala, incluyendo escenas panorámicas de 360 grados. Este método aborda uno de los principales desafíos de los NeRF tradicionales: la dificultad para escalar a entornos grandes y complejos.

  La clave de la eficiencia de MipNeRF-360 reside en su estrategia de muestreo jerárquico. En lugar de muestrear puntos uniformemente a lo largo de los rayos, MipNeRF-360 utiliza una estructura de pirámide de imágenes (mipmap) para guiar el muestreo hacia las regiones más relevantes de la escena. Esta estrategia de muestreo adaptativo reduce significativamente el número de evaluaciones de la función de renderizado necesarias, lo que permite representar escenas mucho más grandes y complejas que con los métodos NeRF tradicionales, manteniendo una alta calidad visual.

- **NeRF– [18]:** NeRF– se desarrolló para abordar algunas de las limitaciones del NeRF original, enfocándose en mejorar la eficiencia computacional y la velocidad de entrenamiento sin comprometer significativamente la calidad de la reconstrucción 3D.

  NeRF– utiliza un esquema de muestreo adaptativo que prioriza las regiones más relevantes de la escena durante el proceso de renderizado, optimizando así el uso

(5) Fuente: Tercera imagen del testbed tomada de NVlabs (https://github.com/NVlabs/instant-ngp/blob/master/docs/assets_readme/testbed.png)

de recursos computacionales. Este enfoque permite obtener una representación más compacta de las escenas tridimensionales, lo que resulta en modelos más ligeros y fáciles de manejar, sin sacrificar significativamente la calidad visual.

- **NeRF-W [15]:** NeRF-W amplía el modelo NeRF original para manejar escenas dinámicas, es decir, escenas que cambian con el tiempo. Para ello, introduce un componente temporal en la función de campo de radiancia, lo que permite modelar la evolución de la escena a lo largo del tiempo.

  NeRF-W descompone la escena en un componente estático y un componente dinámico. El componente estático se representa mediante un único NeRF, mientras que el componente dinámico se modela utilizando una serie de NeRFs, cada uno correspondiente a un instante de tiempo específico. Esta descomposición permite a NeRF-W representar escenas con objetos en movimiento y cambios en la iluminación de manera realista. Al modelar explícitamente el componente dinámico, NeRF-W se vuelve más robusto y aplicable a escenarios del mundo real, superando las limitaciones del NeRF original en escenas no controladas.

- **Ref-NeRF [19]:** Ref-NeRF introduce un novedoso método para aprovechar la información de múltiples imágenes de entrada al renderizar una nueva vista. En lugar de procesar cada imagen de forma independiente, Ref-NeRF utiliza una red neuronal para encontrar correspondencias entre píxeles en diferentes imágenes y fusionar la información de color de múltiples vistas.

  Este enfoque multivista permite mejorar significativamente la calidad de las reconstrucciones, especialmente en áreas con oclusiones o texturas complejas. Al combinar información de múltiples vistas, Ref-NeRF puede "rellenar" los huecos de información que pueden estar presentes en una sola vista, lo que resulta en imágenes más completas y realistas.

Como veremos posteriormente, estos modelos y sus estudios son la base del modelo por defecto que utilizaremos.

Además de las mejoras en la eficiencia y la calidad visual, los NeRF también se están integrando con otras tecnologías, como la segmentación semántica [20] y la edición 3D [21]. Algunos ejemplos de estas integraciones son:

- **Panoptic Neural Radiance Fields (PNeRF) [20]:** Este método combina NeRF con la segmentación panóptica para permitir la segmentación de objetos y la edición 3D en las escenas reconstruidas.

- **EditNeRF [21]:** Este método permite editar objetos en escenas NeRF utilizando una representación basada en capas que separa la geometría, la apariencia y la iluminación.

2.3 Aplicaciones de los NeRF

Los campos de radiancia neuronal han encontrado una amplia variedad de aplicaciones en diferentes sectores gracias a su capacidad para generar representaciones tridimensionales detalladas y realistas. A continuación, se enumeran algunas de las aplicaciones más destacadas de los NeRF:

- **Realidad virtual y aumentada:** Los NeRF permiten crear experiencias inmersivas y fotorrealistas en realidad virtual (RV) y realidad aumentada (RA) [22], abriendo

nuevas posibilidades para el entretenimiento, la educación, el diseño y otras industrias.

- **Robótica y conducción autónoma:** Los NeRF se pueden utilizar para crear modelos 3D precisos del entorno, lo que resulta crucial para la navegación, la planificación de trayectorias y la toma de decisiones en robots [23] y vehículos autónomos.
- **Reconstrucción del patrimonio cultural:** Los NeRF permiten documentar digitalmente y preservar sitios históricos, artefactos y obras de arte con gran detalle [24], facilitando su estudio, conservación y acceso para las generaciones futuras. Además, se pueden utilizar para mapear grandes áreas y realizar análisis espaciales.
- **Diseño y arquitectura:** Los arquitectos y diseñadores utilizan NeRF para visualizar, editar y validar proyectos arquitectónicos y de diseño de interiores [25]. Esta tecnología permite crear y editar modelos tridimensionales precisos de edificios y espacios, facilitando la presentación de proyectos a clientes y la toma de decisiones durante el proceso de diseño.

2.4 Estado actual de los NeRF

El ámbito de los campos de radiancia neuronal se encuentra en constante evolución, con nuevas investigaciones y desarrollos que surgen continuamente. A pesar de los significativos avances logrados en los últimos años, aún existen desafíos que la comunidad de investigación está abordando activamente. Algunos de estos desafíos son:

- **Complejidad computacional:** El renderizado de escenas con NeRF puede ser computacionalmente intensivo, lo que dificulta su uso en dispositivos con recursos limitados o para aplicaciones en tiempo real que requieren altas tasas de fotogramas. La optimización del rendimiento de los NeRF es un área de investigación activa [26].
- **Escalabilidad a escenas a gran escala:** Los NeRF aún enfrentan desafíos para representar escenas muy grandes y complejas, como ciudades enteras o paisajes extensos [27]. La investigación en esta área se centra en desarrollar métodos para descomponer escenas grandes en subescenas más pequeñas que se puedan procesar de forma independiente y luego combinar para formar una representación completa.
- **Generalización a nuevos dominios:** Si bien los NeRF han demostrado un buen rendimiento en dominios específicos, como escenas de interiores o exteriores, su generalización a nuevos dominios, como escenas submarinas o microscópicas, aún requiere mayor investigación [28]. La adaptación de los NeRF a nuevos tipos de datos y a las características específicas de cada dominio es un desafío que la comunidad está abordando.

A pesar de estos desafíos, los NeRF han demostrado ser una tecnología disruptiva con un enorme potencial para transformar la forma en que creamos, interactuamos y experimentamos el mundo tridimensional. Los métodos de síntesis de escenas basados en esta arquitectura, como el Gaussian Splatting, son aún más prometedores, ya que ofrecen una mayor eficiencia y calidad visual.

A medida que la investigación y el desarrollo continúen, podemos esperar ver aplicaciones aún más innovadoras y transformadoras de los NeRF en los próximos años, expandiendo su impacto en una amplia gama de campos, desde la realidad virtual y aumentada hasta la robótica y la medicina.

2.5 NeRFstudio

NeRFstudio [9] es una librería de Python de código abierto diseñada para facilitar la investigación y el desarrollo de campos de radiancia neuronal. Su arquitectura modular y configurable proporciona una colección de implementaciones de última generación de diferentes métodos NeRF, junto con un conjunto de herramientas y utilidades para facilitar el entrenamiento, la evaluación y la visualización de modelos NeRF.

NeRFstudio se ha convertido en una plataforma popular en la comunidad de investigación de NeRF gracias a su flexibilidad, facilidad de uso y a la activa comunidad de desarrolladores. Algunas de las características clave de NeRFstudio que han contribuido a su popularidad son:

- **Modularidad:** NeRFstudio está diseñado de forma modular, lo que permite a los usuarios intercambiar fácilmente diferentes componentes, como la arquitectura del modelo, la función de pérdida y el optimizador. Estos componentes se configuran fácilmente a través de archivos de configuración, lo que permite personalizar el método de acuerdo con las necesidades específicas de cada proyecto. La figura 2.4 muestra un fragmento del archivo de configuración de los métodos implementados por el equipo de NeRFstudio, en este caso mostrando el principio de la configuración del entrenamiento de «nerfacto».

Cloudstudio / nerfstudio / configs / **method_configs.py**

Code Blame 744 lines (714 loc) · 27.2 KB

```
69   descriptions = {
86   method_configs["nerfacto"] = TrainerConfig(
87       method_name="nerfacto",
88       steps_per_eval_batch=500,
89       steps_per_save=2000,
90       max_num_iterations=30000,
91       mixed_precision=True,
92       pipeline=VanillaPipelineConfig(
93           datamanager=ParallelDataManagerConfig(
94               dataparser=NerfstudioDataParserConfig(),
95               train_num_rays_per_batch=4096,
96               eval_num_rays_per_batch=4096,
97           ),
98           model=NerfactoModelConfig(
99               eval_num_rays_per_chunk=1 << 15,
100              average_init_density=0.01,
101              camera_optimizer=CameraOptimizerConfig(mode="SO3xR3"),
```

Figura 2.4: Archivo de configuración nerfacto en NeRFstudio (6)

- **Flexibilidad:** NeRFstudio admite el entrenamiento y la evaluación de una amplia gama de modelos NeRF en diversos conjuntos de datos, incluyendo escenas de interiores, exteriores, objetos individuales y entornos a gran escala. La plataforma

(6) Fuente: method_configs.py disponible en GitHub (https://github.com/pablogranell/Cloudstudio/blob/main/nerfstudio/configs/method_configs.py)

ofrece soporte para diferentes formatos de datos, incluyendo imágenes y vídeos capturados con dispositivos móviles. Además, permite la integración de analizadores de datos personalizados para procesar información de diferentes tipos de sensores, como cámaras RGB, cámaras de profundidad y escáneres LiDAR. La portabilidad de NeRFstudio, basada en Python y librerías multiplataforma, facilita su uso en diferentes sistemas operativos.

- **Facilidad de uso:** NeRFstudio proporciona una interfaz de línea de comandos sencilla e intuitiva para entrenar y evaluar modelos NeRF. Además, ofrece una API completa que facilita la integración de NeRFstudio en otros sistemas y la creación de interfaces de usuario personalizadas.
- **Código abierto:** NeRFstudio se distribuye bajo la licencia Apache 2.0 (7), lo que permite a los usuarios modificar el código fuente, implementar nuevas funciones y contribuir a la plataforma. Su naturaleza de código abierto fomenta la colaboración y la innovación en la comunidad de NeRF. NeRFstudio proporciona "pipelines" que facilitan la integración de nuevos modelos NeRF y ponen a disposición de los desarrolladores las herramientas necesarias para la implementación de nuevos métodos.

NeRFstudio se ha convertido rápidamente en una herramienta popular entre los investigadores en el campo de los NeRF, ya que proporciona un marco flexible y potente para experimentar con diferentes métodos y desarrollar nuevas aplicaciones. Su código abierto y su activa comunidad de desarrolladores lo convierten en una plataforma ideal para la investigación y el desarrollo de soluciones basadas en NeRF.

En la actualidad, debido a su flexibilidad, facilidad de uso, código abierto y su activa comunidad de desarrolladores, NeRFstudio se considera una de las mejores plataforma para investigar, desarrollar y evaluar nuevas implementaciones de NeRF, según el consenso de la comunidad de investigadores en los campos de la visión artificial y los gráficos por ordenador.

(7) La licencia completa está disponible en https://github.com/nerfstudio-project/nerfstudio/blob/main/LICENSE

CAPÍTULO 3
Estudio del problema

Para desarrollar una plataforma robusta y efectiva, es fundamental establecer un conjunto claro de requisitos y especificaciones que guíen el proceso de diseño e implementación. En este capítulo, se definen los requisitos de la plataforma, se detallan las especificaciones para su implementación y se analizan diferentes soluciones.

Tras analizar el funcionamiento, las ventajas y las desventajas de diferentes soluciones potenciales, se seleccionará la opción más adecuada para la implementación de la plataforma, considerando los requisitos y especificaciones previamente establecidos.

3.1 Requisitos

Los requisitos de la plataforma se dividen en requisitos funcionales, que describen lo que la plataforma debe hacer, y requisitos no funcionales, que especifican "cómo" la plataforma debe funcionar. Estos requisitos se han definido teniendo en cuenta las necesidades de los usuarios potenciales y las capacidades actuales de la tecnología NeRF.

A continuación, se enumeran los requisitos considerados para el desarrollo de la plataforma:

1. **Facilidad de uso:** La plataforma debe ser accesible para usuarios sin conocimientos de programación o de redes neuronales. La interfaz de usuario debe ser intuitiva y fácil de usar, permitiendo a los usuarios realizar todas las tareas necesarias sin necesidad de escribir código o configurar parámetros complejos.

2. **Rendimiento:** El backend debe ser capaz de entrenar modelos NeRF en tiempos razonables, aprovechando la aceleración por hardware para optimizar el proceso de entrenamiento. La renderización de la escena debe ser lo suficientemente rápida para permitir la interacción en tiempo real, incluso con múltiples clientes conectados.

3. **Claridad visual:** Las escenas renderizadas deben tener una alta calidad visual, con suficiente detalle para permitir la identificación de los elementos relevantes de la escena, incluso durante el proceso de entrenamiento. La calidad visual debe mejorar a medida que avanza el entrenamiento, convergiendo hacia una representación fotorrealista de la escena.

4. **Modularidad:** La plataforma debe ser modular, permitiendo a los usuarios seleccionar entre diferentes métodos NeRF, configurar los parámetros del modelo y añadir nuevos métodos a medida que se desarrollen. La plataforma debe ser flexible para

manejar diferentes tipos de datos de entrada, como imágenes, vídeos y nubes de puntos.

5. **Flexibilidad:** La plataforma debe ser compatible con diferentes sistemas operativos y navegadores web. La arquitectura del sistema debe permitir la migración a diferentes servidores o entornos de ejecución sin necesidad de realizar cambios significativos en el código.

6. **Colaboración y control remoto:** La plataforma debe soportar la colaboración multiusuario, permitiendo a varios usuarios visualizar y manipular la misma escena simultáneamente. La plataforma debe ofrecer funcionalidades de control remoto, permitiendo a los usuarios controlar la visualización y la manipulación de la escena desde diferentes ubicaciones.

7. **Automatización:** La plataforma debe ser autónoma en su instalación y ejecución. El proceso de entrenamiento y renderización debe automatizarse al máximo, minimizando la intervención del usuario. La plataforma debe ser resistente a errores, implementando mecanismos de recuperación automática para garantizar su funcionamiento continuo.

3.2 Especificaciones

Una vez definidos los requisitos generales de la plataforma, se detallan las especificaciones que guiarán su implementación. Estas especificaciones concretan las características, funcionalidades y diseño de la plataforma, asegurando que cumple con los requisitos establecidos y las necesidades de los usuarios.

A continuación, se presentan las especificaciones para cada requisito, siguiendo la misma estructura que la lista de requisitos:

1. **Facilidad de uso:**
   - El visor debe tener un diseño sencillo e intuitivo, con iconos claros y etiquetas descriptivas para todas las funciones.
   - El visor debe guiar a los usuarios a través de un flujo de trabajo claro y conciso, facilitando la carga de datos y la configuración inicial.

2. **Rendimiento:**
   - El backend debe ser capaz de entrenar un modelo NeRF utilizando una GPU de gama media, manteniendo un rendimiento fluido para la renderización y transmisión de la escena a los clientes visores.
   - El visor debe mantener una tasa de al menos 60 FPS durante la interacción en tiempo real con la escena, independientemente de la capacidad del backend para renderizar la escena a esa velocidad.
   - El backend debe soportar al menos dos clientes conectados simultáneamente a la misma escena, sin degradación significativa del rendimiento en el entrenamiento o la renderización.

3. **Claridad visual:**
   - La calidad visual debe ser suficiente para permitir el reconocimiento de formas y contornos después de solo 2000 iteraciones de entrenamiento.

- La calidad visual debe mejorar gradualmente a medida que avanza el entrenamiento, mostrando mejoras notables en la nitidez y los detalles hasta alcanzar una representación fotorrealista de la escena.

4. **Modularidad:**
   - La plataforma debe soportar la integración de módulos de análisis de datos para procesar diferentes formatos de entrada, incluyendo un número ilimitado de imágenes y vídeos de cualquier duración
   - La plataforma debe ser modular para permitir la selección e implementación de al menos dos métodos NeRF diferentes de forma sencilla, a través de la interfaz de usuario o archivos de configuración.
5. **Flexibilidad:**
   - La plataforma debe ser compatible con Windows y Linux, incluyendo todas sus funciones de entrenamiento, análisis de datos y renderizado. Se recomienda utilizar un lenguaje de programación multiplataforma para facilitar la portabilidad del sistema.
   - El visor debe funcionar en los navegadores web más comunes, como Chrome, Firefox y Safari, en sus versiones más recientes, asegurando su accesibilidad desde diferentes dispositivos.
6. **Colaboración y control remoto:**
   - El visor debe implementar un sistema de sincronización de cámara que replique la posición y la orientación de la cámara del cliente "controlador" en las vistas de los demás clientes conectados.
   - El sistema de sincronización de cámara debe ser configurable, permitiendo a los usuarios activar o desactivar la sincronización, seleccionar el cliente "controlador" y permitir el control independiente de la cámara para cada cliente.
7. **Automatización:**
   - El instalador debe automatizar la configuración del entorno de ejecución, incluyendo la instalación de las dependencias necesarias (Python, librerías, etc.), la configuración de la conexión con Kaggle y la creación de accesos directos para facilitar el uso de la plataforma.
   - El sistema debe registrar todos los eventos importantes y errores en un archivo de registro detallado, accesible a través de la interfaz de usuario o la línea de comandos, para facilitar la depuración y el análisis del funcionamiento de la plataforma.
   - La plataforma debe implementar mecanismos de recuperación automática para reconectarse al backend en caso de fallo del servidor o pérdida de conexión, garantizando un funcionamiento continuo y minimizando la interrupción del flujo de trabajo del usuario.

3.3 Relación entre Requisitos y Especificaciones según ISO/IEC 25010

3.3.1. Introducción

La norma ISO/IEC 25010:2011 (8) define un modelo de calidad para productos de software, estableciendo un conjunto de características que se deben considerar al evaluar la calidad de un sistema. Este modelo es fundamental para garantizar que el software cumple con los estándares de calidad esperados y satisface las necesidades de los usuarios.

A continuación, se analiza cómo los requisitos y especificaciones de la plataforma se alinean con las características de calidad definidas por la norma ISO/IEC 25010:2011. Esta alineación permite asegurar que la plataforma cumple con los estándares internacionales de calidad del software, además de satisfacer las necesidades específicas de los usuarios.

3.3.2. Análisis de Requisitos según ISO/IEC 25010:2011

Requisito	**Característica ISO/IEC 25010:2011**	**Subcaracterística**
Facilidad de uso	Capacidad de Interacción	Aprendizaje
		Operabilidad
		Reconocibilidad de la adecuación

Tabla 3.1: Relación del requisito de Facilidad de uso con ISO/IEC 25010:2011

El requisito de facilidad de uso se alinea con la característica de *capacidad de interacción* de la norma ISO/IEC 25010:2011 (ver tabla 3.1). La interfaz gráfica intuitiva de la plataforma y la posibilidad de realizar todas las operaciones sin necesidad de programación contribuyen a la *operabilidad* y el *aprendizaje* del sistema. El diseño sencillo, con iconos claros y etiquetas descriptivas, facilita la *reconocibilidad de la adecuación*, permitiendo a los usuarios comprender rápidamente el propósito y la funcionalidad del software.

Requisito	**Característica ISO/IEC 25010:2011**	**Subcaracterística**
Rendimiento	Eficiencia de Desempeño	Comportamiento temporal
		Utilización de recursos
		Capacidad

Tabla 3.2: Relación del requisito de Rendimiento con ISO/IEC 25010:2011

El requisito de rendimiento se relaciona con la característica de *eficiencia de desempeño* de la norma ISO/IEC 25010:2011 (ver tabla 3.2). La capacidad de la plataforma para entrenar modelos NeRF en tiempos razonables y mantener una tasa de al menos 60 FPS durante la interacción en tiempo real refleja un buen *comportamiento temporal*. El uso de la aceleración por hardware para optimizar el proceso de entrenamiento demuestra una *utilización de recursos* eficiente. La capacidad del backend para gestionar al menos dos

(8) ISO/IEC 25010:2011 Sistemas y software de ingeniería — Requisitos de calidad y evaluación (SQuaRE) — Modelo de calidad. https://www.iso.org/standard/50868.html

clientes conectados simultáneamente, sin degradación significativa del rendimiento, evidencia la *capacidad* del sistema.

Requisito	**Característica ISO/IEC 25010:2011**	**Subcaracterística**
Claridad visual	Adecuación Funcional	Corrección funcional

Tabla 3.3: Relación del requisito de Claridad visual con ISO/IEC 25010:2011

El requisito de claridad visual se alinea con la característica de *adecuación funcional* de la norma ISO/IEC 25010:2011, específicamente con la subcaracterística de *corrección funcional* (ver tabla 3.3). La capacidad de la plataforma para manejar efectos visuales complejos y mejorar la calidad visual durante el entrenamiento demuestra que el sistema produce resultados correctos y precisos, cumpliendo con las necesidades de los usuarios.

Requisito	**Característica ISO/IEC 25010:2011**	**Subcaracterística**
Modularidad	Mantenibilidad	Modularidad
		Capacidad para ser modificado

Tabla 3.4: Relación del requisito de Modularidad con ISO/IEC 25010:2011

El requisito de modularidad se alinea con la característica de *mantenibilidad* de la norma ISO/IEC 25010:2011 (ver tabla 3.4). La capacidad de la plataforma para soportar diferentes tipos de entrada, permitir la configuración de parámetros y la selección de métodos NeRF demuestra una buena *modularidad*. La facilidad para incorporar nuevos métodos y técnicas de NeRF refleja una alta *capacidad para ser modificado*, lo que facilita la evolución y el mantenimiento del software a largo plazo.

Requisito	**Característica ISO/IEC 25010:2011**	**Subcaracterística**
Flexibilidad	Flexibilidad	Adaptabilidad
		Instalabilidad

Tabla 3.5: Relación del requisito de Flexibilidad con ISO/IEC 25010:2011

El requisito de flexibilidad se relaciona con la característica de *portabilidad* de la norma ISO/IEC 25010:2011 (ver tabla 3.5). La capacidad de la plataforma para funcionar en diferentes sistemas operativos y navegadores web demuestra una buena *adaptabilidad*. La facilidad para cambiar de servidor o entorno de ejecución refleja una buena *instalabilidad*, lo que facilita el despliegue y la utilización de la plataforma en diferentes entornos.

Requisito	**Característica ISO/IEC 25010:2011**	**Subcaracterística**
Colaboración y control remoto	Compatibilidad	Interoperabilidad

Tabla 3.6: Relación del requisito de Colaboración y control remoto con ISO/IEC 25010:2011

El requisito de colaboración y control remoto se alinea con la característica de *compatibilidad* de la norma ISO/IEC 25010:2011, específicamente con la subcaracterística de

interoperabilidad (ver tabla 3.6). La capacidad de la plataforma para permitir a múltiples usuarios visualizar y manipular la misma escena de forma simultánea, con sincronización de cámara, demuestra una buena *interoperabilidad* entre diferentes instancias del sistema.

Requisito	**Característica ISO/IEC 25010:2011**	**Subcaracterística**
Automatización	Fiabilidad	Recuperabilidad
	Mantenibilidad	Capacidad para ser probado
	Flexibilidad	Instalabilidad

Tabla 3.7: Relación del requisito de Automatización con ISO/IEC 25010:2011

El requisito de automatización se relaciona con tres características de la norma ISO/IEC 25010:2011 (ver tabla 3.7): *fiabilidad*, *mantenibilidad* y *flexibilidad*. La capacidad de la plataforma para recuperarse automáticamente de fallos demuestra una buena *recuperabilidad*. El sistema de registro detallado facilita las pruebas y el diagnóstico, mejorando la *capacidad para ser probado*. La capacidad del instalador para configurar completamente el entorno, incluyendo la instalación de las dependencias necesarias, refleja una buena *instalabilidad*.

3.3.3. Conclusión

El análisis de la alineación de los requisitos de la plataforma con las características de calidad definidas por la norma ISO/IEC 25010:2011 proporciona una serie de ventajas, entre las que se destacan:

1. **Garantía de calidad**: Al alinear los requisitos de la plataforma con estándares internacionales como ISO/IEC 25010:2011, se asegura que la plataforma cumple con criterios de calidad reconocidos globalmente, lo que aumenta la confianza de los usuarios en el sistema.
2. **Facilidad de evaluación**: La alineación con ISO/IEC 25010:2011 proporciona un marco de referencia claro para evaluar el cumplimiento de los objetivos de calidad de la plataforma, tanto durante el desarrollo como después de su despliegue.
3. **Mejora continua**: Al identificar claramente las características de ISO/IEC 25010:2011 que se relacionan con cada requisito, se pueden enfocar los esfuerzos de mejora de forma más efectiva, priorizando las áreas que requieren mayor atención.
4. **Comunicación clara**: La alineación con la norma ISO/IEC 25010:2011 permite comunicar de forma clara y concisa las características de calidad de la plataforma a los usuarios potenciales, utilizando un lenguaje estandarizado y reconocido internacionalmente.

En resumen, la alineación de los requisitos de la plataforma con la norma ISO/IEC 25010:2011 proporciona un marco sólido para el desarrollo, la evaluación y la mejora continua de la plataforma. Esta alineación no solo valida la calidad del sistema, sino que también facilita la comunicación de las características de calidad a los usuarios potenciales. La plataforma, al cumplir con los estándares internacionales de calidad del software, se posiciona de manera favorable para satisfacer las necesidades de los usuarios y ofrecer una experiencia de usuario óptima.

3.4 Posibles soluciones

En base a los requisitos y especificaciones establecidos, se han identificado tres posibles soluciones para la implementación de la plataforma. Cada solución presenta ventajas y desventajas, que se analizarán a continuación para determinar la opción más adecuada.

Es importante destacar que las soluciones que se analizan a continuación se centran en la base de la plataforma, especialmente en la implementación del backend de entrenamiento y visualización de las escenas. Los protocolos de comunicación, las librerías, las herramientas y la forma de crear una implementación sencilla de estas se detallarán en el capítulo siguiente, que describe la solución propuesta.

El análisis de las soluciones comenzará con las opciones más complejas y completas, y luego se explorarán opciones más simples. El objetivo es encontrar una solución que cumpla con todos los requisitos y especificaciones, pero que también sea fácil de mantener y actualizar.

3.4.1. Sistema propio de renderizado

Una posible solución consiste en desarrollar un motor de renderizado completamente nuevo para la plataforma, específicamente diseñado para NeRF. Esta solución requeriría una inversión significativa en tiempo y recursos, pero ofrecería un control total sobre todas las características y la optimización del rendimiento. Un ejemplo de un motor de renderizado propio basado en CUDA es Instant-NGP [16], desarrollado por NVIDIA.

Ventajas:

- **Control total:** Esta solución proporciona un control total sobre todas las características, funcionalidades y protocolos de la plataforma, lo que permite una personalización y optimización a medida.
- **Optimización del rendimiento:** El motor de renderizado se puede optimizar para las necesidades específicas de NeRF y el hardware disponible, lo que permite maximizar el rendimiento y la calidad visual.
- **Flexibilidad en las dependencias:** Se tiene la libertad de elegir las librerías externas y los frameworks que mejor se adapten a las necesidades del TFG.

Desventajas:

- **Alta inversión de recursos:** El desarrollo de un motor de renderizado propio requiere una inversión significativa en tiempo, recursos humanos y experiencia técnica.
- **Complejidad técnica y mantenimiento:** La alta complejidad técnica del desarrollo de un motor de renderizado puede dar lugar a errores, un rendimiento inferior al esperado y la necesidad de un mantenimiento constante para garantizar su correcto funcionamiento y la incorporación de nuevas características.
- **Conocimiento especializado:** Se requiere un profundo conocimiento de CUDA y otras tecnologías de programación paralelas de GPU para desarrollar un motor de renderizado eficiente y optimizado.

Debido a la alta inversión de recursos, la complejidad técnica y la necesidad de un conocimiento especializado, el desarrollo de un motor de renderizado propio no se considera una opción viable para este TFG. Como se ha mencionado anteriormente, el desarrollo de Instant-NGP [16], un motor de renderizado propio para NeRF, requirió un esfuerzo conjunto de varios autores durante dos años, además de la investigación previa en tecnologías complementarias, como se describe en [29].

3.4.2. Reimplementación de Instant NGP

Otra opción es basar la plataforma en una reimplementación de Instant NGP, adaptándolo a las necesidades específicas del TFG. Immersive Neural Graphics Primitives [22] es un ejemplo de una reimplementación de Instant NGP para su uso en realidad virtual.

Sin embargo, Instant NGP no está diseñado para su uso en un entorno cliente-servidor, ya que no proporciona una interfaz de comunicación para conectarse a un servidor remoto. Además, la falta de una API pública limita la posibilidad de controlar y extender las funcionalidades del visor de Instant NGP.

Ventajas:

- **Alto rendimiento y eficiencia:** Instant NGP ofrece un alto rendimiento y eficiencia en el entrenamiento y la renderización de escenas NeRF, gracias a su representación basada en codificación hash.
- **Control sobre la implementación:** Al basarse en una reimplementación de Instant NGP, se tiene control sobre el código fuente, lo que permite adaptarlo a las necesidades específicas del TFG.
- **Flexibilidad para mejoras:** El control sobre el código fuente permite implementar mejoras y nuevas funciones en el motor de renderizado.

Desventajas:

- **Complejidad de la comunicación cliente-servidor:** La reimplementación de Instant NGP para un entorno cliente-servidor requeriría el desarrollo de un protocolo de comunicación desde cero, lo que implica una complejidad técnica significativa.
- **Problemas de propiedad intelectual:** Instant NGP es una tecnología desarrollada por NVIDIA, lo que podría plantear problemas de propiedad intelectual al utilizarla como base de la plataforma.
- **Mantenimiento a largo plazo:** La dependencia de CUDA y la reimplementación de Instant NGP podrían dificultar el mantenimiento de la plataforma a largo plazo, ya que el toolkit de CUDA se actualiza con frecuencia, lo que podría requerir la adaptación del código.

Debido a la falta de soporte para comunicación cliente-servidor, las limitaciones de la API y las posibles restricciones de licencia de NVIDIA, la reimplementación de Instant NGP no se considera una opción viable para este TFG.

3.4.3. Método basado en NeRFstudio

NeRFstudio [9] es una plataforma de código abierto que ofrece una arquitectura modular y flexible, con un backend y un visor separados. Esta separación facilita la modifi-

cación y extensión de la funcionalidad de la plataforma para adaptarla a las necesidades específicas del TFG.

Ventajas:

- **Arquitectura modular:** NeRFstudio está diseñado con una arquitectura modular que facilita la adición de nuevos módulos, la modificación de los módulos existentes y la personalización de la plataforma para diferentes casos de uso.
- **Optimizaciones de rendimiento:** NeRFstudio implementa optimizaciones para el entrenamiento y el renderizado eficientes de modelos NeRF, incluyendo el método «nerfacto», que ofrece un buen equilibrio entre calidad visual y velocidad de procesamiento.
- **Comunidad activa y código abierto:** NeRFstudio es un proyecto de código abierto con una comunidad activa de desarrolladores que contribuyen constantemente a su mejora, expansión y soporte. Esto garantiza un desarrollo continuo de la plataforma, la corrección de errores y la incorporación de nuevas funcionalidades.
- **Funcionalidad completa:** NeRFstudio proporciona una funcionalidad completa para el entrenamiento, la renderización y la visualización de modelos NeRF, incluyendo un visor web interactivo y scripts de entrenamiento personalizables.

Desventajas:

- **Desarrollo continuo:** Al estar en constante desarrollo, algunas funciones de NeRFstudio pueden estar en fase experimental o presentar cambios en su interfaz o rendimiento. Es importante tener en cuenta que el desarrollo continuo también implica la incorporación de nuevas características y mejoras.
- **Complejidad de la instalación y configuración inicial:** NeRFstudio, al estar dirigido principalmente a investigadores y desarrolladores, puede presentar una cierta complejidad en su instalación y configuración inicial para usuarios sin experiencia en Python o en entornos de desarrollo de software.

En base al análisis comparativo, NeRFstudio se selecciona como la solución más adecuada para este TFG. Su arquitectura modular, la disponibilidad de su código fuente y la activa comunidad de desarrolladores ofrecen una base sólida para la implementación de la plataforma. Las desventajas de NeRFstudio, como la complejidad de la instalación y la configuración inicial, se abordarán mediante el desarrollo de un instalador intuitivo y una interfaz de usuario amigable, como se describe en los siguientes capítulos.

3.5 Solución propuesta

La elección de NeRFstudio como base para el desarrollo de la plataforma se ha realizado tras un análisis de las alternativas disponibles, en el que se han considerado principalmente criterios como la modularidad, la complejidad, la flexibilidad y la disponibilidad de código abierto, que indirectamente afectan a la comunidad presente y al soporte que esta puede ofrecer.

La tabla 3.8 presenta una comparación de las tres soluciones propuestas, considerando las características más relevantes para la selección de la solución óptima. Las características más importantes que han influido en la elección de NeRFstudio se resaltan en negrita.

Característica	Sistema propio	Reimplementación de Instant NGP	NeRFstudio
Facilidad de uso	Depende	Baja	Media
Rendimiento	**Potencialmente alto**	**Alto**	**Alto**
Modularidad	Depende	Limitada	**Alta**
Flexibilidad	**Alta**	Media	**Alta**
Complejidad técnica	Alta	Media	Media
Inversión de recursos	Alta	Media	Media
Comunidad y soporte	No	Limitado	**Extenso**
Código abierto	Depende	Sí	**Sí**

Tabla 3.8: Comparación de las posibles soluciones para la implementación del backend de NeRFstudio. Las fortalezas de cada solución se resaltan en negrita

Como se observa en la tabla 3.8, NeRFstudio ofrece una serie de ventajas significativas para este TFG, en comparación con las otras soluciones analizadas. Su alto rendimiento, modularidad, flexibilidad, código abierto y la extensa comunidad que lo respalda lo convierten en una base sólida para el desarrollo de la plataforma. Las limitaciones de NeRFstudio en cuanto a la facilidad de uso se abordarán mediante el desarrollo de una interfaz de usuario intuitiva y un instalador que simplifique la configuración inicial.

Considerando la modularidad, la flexibilidad y la complejidad como métricas principales, NeRFstudio se presenta como la opción más adecuada para este TFG. Su arquitectura modular permite adaptar la plataforma a las necesidades específicas del caso de uso, mientras que su naturaleza de código abierto y la amplia comunidad de desarrolladores garantizan un desarrollo continuo y un soporte técnico robusto.

3.5.1. Ventajas Adicionales de NeRFstudio

Además de las ventajas mencionadas anteriormente, NeRFstudio ofrece otras características que lo convierten en una opción atractiva para este TFG:

- **Integración con Herramientas Existentes:** NeRFstudio se integra con librerías y herramientas populares del ecosistema de Python para el aprendizaje automático y la visión artificial, como PyTorch, lo que facilita su uso, la incorporación de nuevas funciones y la interoperabilidad con otros sistemas.
- **Documentación Completa:** NeRFstudio cuenta con una documentación completa y actualizada, que incluye tutoriales, ejemplos y referencias de la API. Esta documentación facilita la curva de aprendizaje, especialmente para usuarios con conocimientos de informática y experiencia en Python.
- **Actualizaciones Constantes:** NeRFstudio se encuentra en constante desarrollo, lo que garantiza la incorporación de las últimas innovaciones en el campo de los NeRF, la actualización de su documentación y la corrección de errores. A fecha de 28 de junio de 2024, NeRFstudio ha superado la fase de desarrollo experimental y se encuentra en la versión 1.1.3.

En resumen, NeRFstudio se ha seleccionado como la base para el desarrollo de la plataforma, debido a sus numerosas ventajas y su adecuación a los requisitos del TFG. El siguiente capítulo describe la solución propuesta, detallando su implementación y las herramientas y librerías utilizadas.

CAPÍTULO 4
Detalles de la solución propuesta

Como se ha justificado en el capítulo anterior, la plataforma se basa en NeRFstudio [9] para la implementación de un backend de entrenamiento de escenas NeRF. La arquitectura modular de NeRFstudio, su código abierto y su flexibilidad permiten adaptarlo a las necesidades específicas del TFG, ofreciendo una base sólida para el desarrollo de un sistema robusto y eficiente.

En este capítulo, se describe la solución propuesta y se detalla la implementación de la plataforma, incluyendo la elección de servidor, el backend de NeRFstudio, el visor y el backend-frontend del instalador. También se explicarán las decisiones tomadas durante el proceso de desarrollo, la arquitectura cliente-servidor y, las herramientas y librerías utilizadas.

4.1 Backend de NeRFstudio

La elección del backend es crucial para el éxito de la plataforma, ya que este componente se encarga de las tareas más complejas, como el entrenamiento, la renderización y la gestión de la comunicación con los clientes. El objetivo principal del backend es abstraer la complejidad técnica del usuario, proporcionando una experiencia de uso sencilla e intuitiva.

4.1.1. Abstracción de la complejidad de la instalación

La instalación de NeRFstudio y sus dependencias puede ser un proceso complejo para usuarios sin experiencia en Python o en entornos de desarrollo de software. Para simplificar la instalación, se exploran tres soluciones potenciales:

- **Instalador ejecutable:** Esta solución consiste en crear un instalador ejecutable utilizando herramientas como PyInstaller (9), que empaqueta el código Python de la plataforma, incluyendo NeRFstudio y sus dependencias, en un único archivo ejecutable. Este instalador se encarga de instalar Python, las librerías necesarias y la plataforma en el sistema del usuario de forma automatizada. Sin embargo, esta solución presenta ciertas limitaciones:

    1. **Permisos y entorno de instalación:** La instalación puede requerir permisos de administrador en el sistema del usuario, lo que puede ser un obstáculo para algunos usuarios. Además, el instalador ejecutable no tiene conocimiento del

(9) Fuente: PyInstaller (https://pyinstaller.org/)

entorno de software existente en el sistema del usuario, lo que puede dar lugar a conflictos con otras aplicaciones o librerías instaladas.

2. **Requisitos de hardware:** El entrenamiento de modelos NeRF requiere una gran cantidad de recursos computacionales, especialmente una GPU potente. La instalación local limita la plataforma a usuarios con hardware adecuado, lo que reduce su accesibilidad.

- **Entorno Docker:** Esta solución implica la creación de una imagen Docker que contiene un entorno preconfigurado con Python, NeRFstudio y todas las dependencias necesarias. El usuario descargaría la imagen Docker y la ejecutaría en su sistema utilizando Docker. Esta solución ofrece un entorno aislado y controlado, pero también presenta ciertas desventajas:

  1. **Dependencia de Docker:** El usuario debe tener Docker instalado en su sistema, lo que puede ser un obstáculo para algunos usuarios. Además, la ejecución de Docker puede requerir permisos de administrador.
  2. **Mantenimiento de las imágenes Docker:** Cada nueva versión de NeRFstudio requeriría la creación de una nueva imagen Docker, lo que requeriría más esfuerzos de mantenimiento y descartaría en gran parte nuestros apartados de flexibilidad presentes en los apartados 3.1 y 3.2.

- **Backend en la nube:** Esta solución propone ejecutar el backend de NeRFstudio en un entorno en la nube, utilizando servicios como Google Colab o Kaggle, que ofrecen recursos computacionales gratuitos, incluyendo GPUs. El usuario se conectaría al backend en la nube a través de una interfaz web o una aplicación de escritorio. Esta solución ofrece varias ventajas:

  1. **Control del entorno:** Al ejecutar el backend en la nube, se tiene un control total sobre el entorno de software y hardware, lo que permite optimizar el rendimiento y la fiabilidad del sistema.
  2. **Acceso a recursos computacionales potentes:** Los servicios en la nube ofrecen acceso a GPUs potentes, lo que permite acelerar el entrenamiento de los modelos NeRF y mejorar la eficiencia de la plataforma.
  3. **Conexión a Internet:** El usuario necesita una conexión a Internet estable para interactuar con el backend de NeRFstudio en la nube.
  4. **Costes:** Si bien existen servicios en la nube gratuitos, como Google Colab y Kaggle, con limitaciones en los recursos o el tiempo de ejecución, el uso de servicios en la nube más potentes puede implicar costes adicionales.

4.1.2. Abstracción de la complejidad de la ejecución

El entrenamiento y la renderización de modelos NeRF con NeRFstudio pueden requerir una gran cantidad de recursos computacionales, especialmente memoria de vídeo (VRAM) y una GPU potente como se puede ver en la tabla 4.1. El modelo Nerfacto-huge, por ejemplo, puede consumir hasta 24 GB de VRAM.

Método	Descripción	Memoria RAM	Velocidad
nerfacto	Modelo por defecto	$\sim 6GB$	Rápido
nerfacto-big	Mayor calidad	$\sim 12GB$	Lento
nerfacto-huge	Aún más grande y de mayor calidad	$\sim 24GB$	El mas lento
depth-nerfacto	Supervisar en profundidad	$\sim 6GB$	Rápido

Tabla 4.1: Especificaciones de velocidad y memoria de nerfacto (10)

Para abstraer estos requisitos de hardware del usuario y hacer que la plataforma sea accesible para una mayor variedad de dispositivos, se consideran dos opciones:

- **Ejecución local:** Esta opción implica ejecutar el backend de NeRFstudio en el mismo sistema donde se ha instalado la plataforma. Si bien esta solución es sencilla de implementar, limita la plataforma a usuarios con hardware potente, ya que el entrenamiento y la renderización de modelos NeRF pueden ser computacionalmente intensivos.
- **Ejecución en la nube:** Esta opción consiste en ejecutar el backend de NeRFstudio en un servidor en la nube, liberando al usuario de los requisitos de hardware y permitiéndole acceder a la plataforma desde cualquier dispositivo con conexión a Internet. El resultado del entrenamiento y la renderización se transmite al usuario a través de una interfaz web o una aplicación de escritorio. Esta solución se puede implementar de tres maneras:
  1. **Servidor dedicado en la nube:** Se puede alquilar un servidor dedicado en la nube con los recursos necesarios para ejecutar el backend de NeRFstudio. Algunos proveedores de servicios en la nube que ofrecen este tipo de servidores son Amazon Web Services (AWS) (11) y Hugging Face (12).
  2. **Servicios de computación en la nube con GPUs prestadas:** Se pueden utilizar servicios como Google Colab (13) y Kaggle (14) que ofrecen acceso gratuito o de bajo coste a GPUs para la ejecución de tareas de aprendizaje automático.
  3. **Virtualización de GPUs en la nube:** Existen servicios que permiten virtualizar GPUs en la nube y acceder a ellas desde un sistema local, como Juice (15). Estos servicios utilizan la virtualización para compartir recursos de GPU entre diferentes usuarios, aprovechando la capacidad de procesamiento inactiva. Según Juice-Labs [30], solo se utiliza el 15 % de la potencia de procesamiento de GPU disponible en promedio, lo que indica un gran potencial para la virtualización y la optimización del uso de recursos.

4.1.3. Primeras implementaciones

Tras analizar las diferentes opciones para la implementación del backend de NeRFstudio, se procede a evaluar cada una de ellas en la práctica, con el objetivo de determinar la solución más adecuada para este TFG. Esta evaluación se basará en criterios como la facilidad de uso, el rendimiento, la fiabilidad y la escalabilidad.

Durante la fase de evaluación, se implementaron prototipos de cada una de las soluciones potenciales. Algunas de estas implementaciones se descartaron debido a limitaciones en el rendimiento, la escalabilidad o la complejidad técnica.

A continuación, se describen las soluciones que se prototiparon, incluyendo los resultados obtenidos y los motivos que llevaron a su selección o descarte.

(10) Fuente: Tabla extraída y traducida (https://docs.nerf.studio/nerfology/methods/nerfacto.html)
(11) Fuente: AWS EC2 (https://aws.amazon.com/ec2/?nc2=h_ql_prod_cp_ec2)
(12) Fuente: Hugging Face Spaces (https://huggingface.co/spaces)
(13) Fuente: Google Colab (https://colab.research.google.com/)
(14) Fuente: Kaggle (https://www.kaggle.com/)
(15) Fuente: Juice Community Version (https://github.com/Juice-Labs/Juice-Labs)

Instalación local

Se comenzó por evaluar la opción de ejecutar el backend de NeRFstudio localmente. El objetivo principal de esta evaluación era comprender las dificultades de la instalación, los requisitos de hardware y la complejidad que se podía abstraer del usuario final.

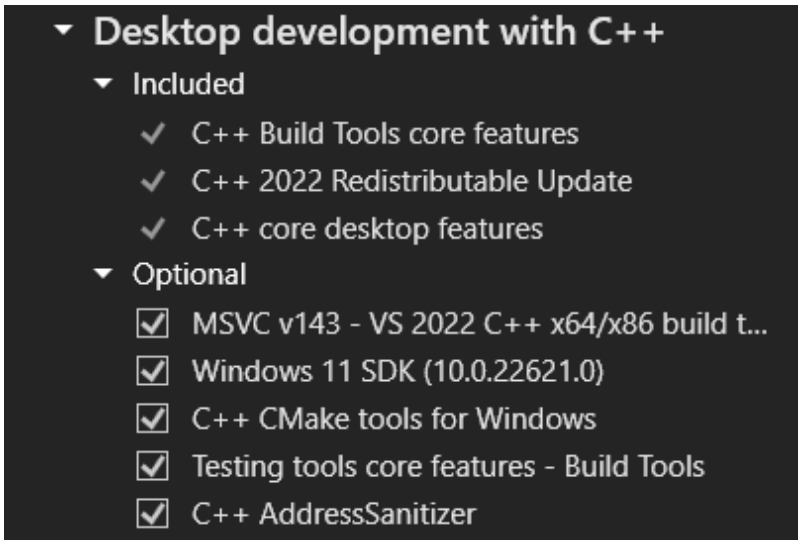

Figura 4.1: Instalación mínima de paquetes necesarios para compilar las dependencias de NeRFstudio en Windows 11 (16)

La documentación de NeRFstudio se centra en la instalación desde un entorno conda limpio, asumiendo que el usuario tiene todas las dependencias necesarias para compilar las librerías. Sin embargo, la documentación no aborda en detalle ciertos aspectos específicos de la instalación, como la instalación de los SDK necesarios para compilar las dependencias de NeRFstudio en diferentes sistemas operativos. Esto puede dar lugar a errores o problemas durante la instalación, especialmente para usuarios sin experiencia en la configuración de entornos de desarrollo.

Tras instalar el SDK de Windows 11 y las herramientas de compilación de Visual Studio 2022 (ver figura 4.1), se pudo completar la instalación de NeRFstudio sin errores.

Una vez instalado NeRFstudio, se realizó una prueba de entrenamiento (ver figura 4.2 que muestra una termina de anaconda) utilizando la escena de ejemplo ”poster” y el método «nerfacto». El entrenamiento se ejecutó en un sistema con una GPU NVIDIA GTX 1650 Ti de 4 GB de VRAM. El tiempo de entrenamiento fue de aproximadamente 5 horas. La visualización de la escena en el visor Viser presentó un bajo rendimiento, con una tasa de fotogramas (FPS) muy baja y una resolución limitada, lo que dificultaba la interacción con la escena.

(16) Visual Studio Installer >Visual Studio Build Tools 2022 >Modificar

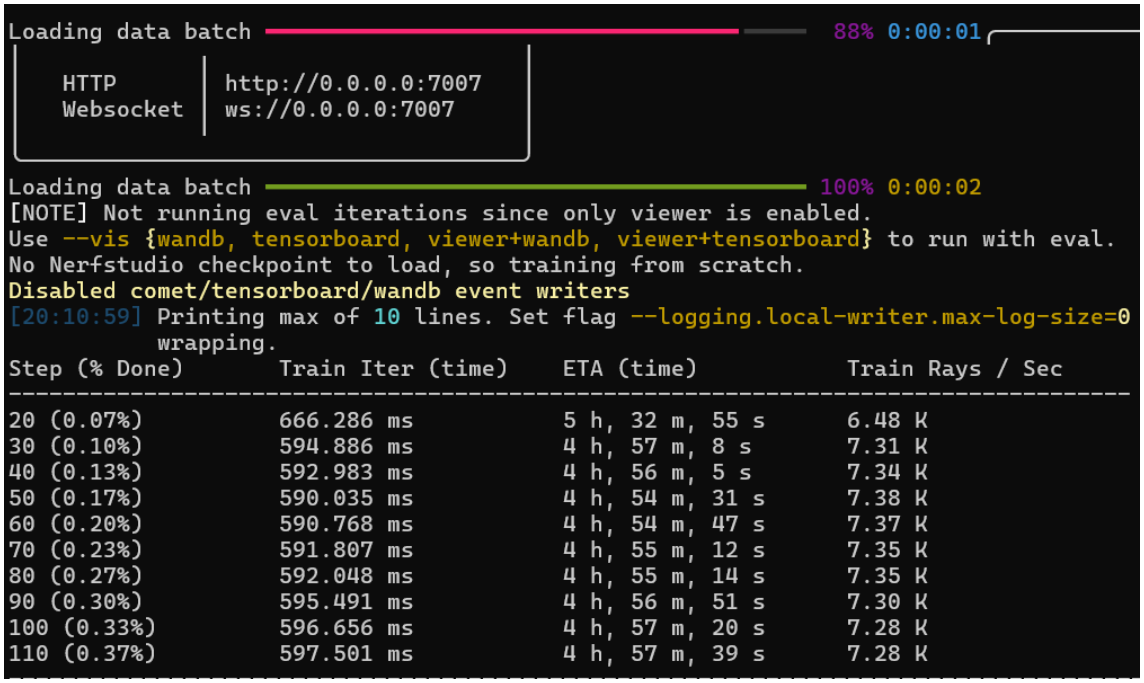

Figura 4.2: Terminal de anaconda ejecutando NeRFstudio completamente en una máquina local (17)

En base a los resultados obtenidos, se decidió descartar la opción de ejecución local. El bajo rendimiento del visor y los largos tiempos de entrenamiento, incluso con una escena de ejemplo, indican que esta solución no es adecuada para usuarios sin hardware potente. Además, la instalación en un entorno local presenta desafíos en la gestión de dependencias, la configuración del entorno y la resolución de problemas de compatibilidad, lo que dificulta su uso para usuarios no técnicos.

Además, la ejecución local no permite aprovechar las ventajas de los servicios en la nube, que son más interesantes, ya que traen ventajas como el acceso a GPUs más potentes y la posibilidad de ejecutar el entrenamiento en segundo plano, sin afectar al rendimiento del sistema del usuario.

Instalación local con virtualización de GPU en la nube

A continuación, se evaluó la opción de virtualización de GPUs en la nube. Esta solución combina la instalación local de NeRFstudio con el acceso remoto a una GPU en la nube, utilizando servicios como «Juice» [30]. Juice permite virtualizar la GPU, transmitiendo los comandos gráficos a través de la red y ofreciendo acceso a recursos de procesamiento remoto.

Elección del servicio en la nube

Para utilizar la virtualización de GPUs en la nube, es necesario seleccionar un proveedor de servicios que ofrezca GPUs potentes a un precio accesible. El servicio elegido debe ser compatible con «Juice» y proporcionar la flexibilidad necesaria para la configuración del entorno de ejecución.

Inicialmente, se consideró la opción de alquilar un servidor dedicado en la nube con un sistema operativo Linux, para tener un control total sobre el entorno de ejecución.

(17) Comando utilizado para iniciar el entrenamiento `ns-train nerfacto -data ''rutaPosterData''`

Sin embargo, los costes asociados a esta opción, especialmente para servidores con GPUs potentes, eran prohibitivos para este TFG.

Nombre	CPU	RAM	GPU	VRAM	Precio por hora
CPU Basic	2 vCPU	16 GB	-	-	Gratis
CPU Upgrade	8 vCPU	32 GB	-	-	$0.03
Nvidia T4 - small	4 vCPU	15 GB	Nvidia T4	16 GB	$0.40
Nvidia T4 - medium	8 vCPU	30 GB	Nvidia T4	16 GB	$0.60
1x Nvidia L4	8 vCPU	30 GB	Nvidia L4	24 GB	$0.80
...	...	...	...	...	...
Nvidia A10G - small	4 vCPU	15 GB	Nvidia A10G	24 GB	$1.00
Nvidia A10G - large	12 vCPU	46 GB	Nvidia A10G	24 GB	$1.50
...	...	...	...	...	...

Tabla 4.2: Instancias de Hugging Face Spaces con GPUs, con precios cercanos a 1 dólar por hora

Como se muestra en la tabla 4.2, los precios por hora de las instancias con GPUs pueden parecer asequibles. Sin embargo, el coste mensual de estas instancias puede ser elevado, especialmente si se utiliza la plataforma de forma regular. A continuación, se calcula el coste mensual de algunas de las instancias con GPUs más económicas, suponiendo un uso de 8 horas al día durante 30 días:

- El precio mensual para la instancia "Nvidia T4 - small" sería de 0.40 $/hora * 8 horas/día * 30 días/mes = 96.00 $/mes $\approx$ 88 €/mes.
- El precio mensual para la instancia "Nvidia T4 - medium" sería de 0.60 $/hora * 8 horas/día * 30 días/mes = 144.00 $/mes $\approx$ 132 €/mes.
- El precio mensual para la instancia "1x Nvidia L4" sería de 0.80 $/hora * 8 horas/día * 30 días/mes = 192.00 $/mes $\approx$ 176 €/mes.

Debido a los altos costes asociados al alquiler de servidores dedicados en la nube, se descartó esta opción para este TFG. Se buscó una solución más económica que permitiera acceder a GPUs potentes sin incurrir en gastos elevados.

Además, se priorizó la utilización de servicios que ofrecieran un buen equilibrio entre coste y rendimiento, ya que la plataforma se dirige a un público amplio, incluyendo usuarios con recursos limitados.

Se exploró la posibilidad de utilizar servicios de computación en la nube con GPUs prestadas, como Google Colab, que ofrece acceso gratuito a GPUs para la ejecución de tareas de aprendizaje automático.

Sin embargo, pensando a largo plazo, cabe destacar que Colab no ofrece ninguna API para poder controlar el servidor desde fuera de su página web, por lo que habría sido difícil desarrollar una implementación automatizada sin haber explorado no oficialmente la conexión backend-frontend propia de Colab.

Por suerte, existe un servicio muy similar a Google Colab (tan similar que Google lo adquirió en 2017 para que no compitiera con Colab) llamado Kaggle, que ofrece también GPUs prestadas, pero con mejores límites y características, además de una API sencilla para inicializar las libretas Jupyter desde scripts Python.

Características	Kaggle	Google Colab
GPU	Tesla P100, Tesla T4x2	Tesla T4
VRAM	16GB (P100), 16GB (T4)	16GB (T4)
RAM	30GB	12GB (T4)
Tiempo de ejecución	Max 12 horas (web y API)	Hasta 12 horas (web)
Disco temporal	70GB temporales	100GB temporales
Tiempo de uso de GPU	30 horas semanales	2-3 horas diarias
Disco persistente	20GB persistentes	Google Drive
Precio	Gratuito	Gratuito y de pago
Beneficios	API, flexibilidad	Google Drive, facilidad de uso
Restricciones	Contenido, inactividad	Terminal de pago, inactividad

Tabla 4.3: Comparación entre Kaggle y Google Colab para el entrenamiento de modelos NeRF

La tabla 4.3 presenta una comparación de las características de Kaggle y Google Colab.

Es importante tener en cuenta las limitaciones en el tiempo de uso de GPU que ofrece cada servicio. Kaggle permite un máximo de 30 horas de uso de GPU por semana, mientras que Google Colab ofrece entre dos y tres horas de uso diario. Esta diferencia puede ser significativa para los usuarios que necesitan entrenar escenas complejas que requieren tiempos de entrenamiento prolongados.

Kaggle también ofrece otras ventajas, como una API disponible para la automatización, una mayor flexibilidad en la configuración del entorno y una mejor compatibilidad con las librerías utilizadas por NeRFstudio.

En base a estas ventajas, se seleccionó Kaggle como el servicio en la nube para la ejecución del backend de NeRFstudio.

Elección de protocolo de conexión

Tras seleccionar Kaggle como el servicio en la nube para la ejecución de NeRFstudio, se procedió a evaluar la viabilidad de utilizar Juice para la virtualización de la GPU. Se realizó una prueba de concepto utilizando «Vkcube», una aplicación que renderiza un cubo 3D utilizando Vulkan, para verificar la correcta funcionalidad de «Juice».

Sin embargo, la API de Kaggle solo permite iniciar la ejecución de libretas Jupyter de forma asíncrona. No es posible interactuar con la libreta Jupyter en tiempo real ni acceder a la GPU virtualizada durante la ejecución. Esta limitación se debe a que la API de Kaggle está diseñada para la ejecución de tareas en segundo plano, como la automatización de procesos de análisis de datos, no para la interacción en tiempo real.

Para poder interactuar con el backend de NeRFstudio en tiempo real y acceder a la GPU virtualizada, se necesita un método para establecer una conexión con la máquina virtual de Kaggle durante la ejecución de la libreta Jupyter. Una solución es utilizar un túnel que permita acceder a los puertos de la máquina virtual de forma remota. Existen diferentes herramientas que permiten crear túneles, como Cloudflare Tunnels (18) y Ngrok vía (19). Estas herramientas permiten exponer puertos locales a Internet, incluso si el sistema no tiene una dirección IP pública.

Sin embargo se evaluaron dos herramientas menos conocidas pero también menos restrictivas para la creación de túneles: Zrok y Hypershell.

(18) Fuente: Cloudflare Tunnels (https://www.cloudflare.com/products/tunnel/)
(19) Fuente: Ngrok (https://ngrok.com/)

Zrok

Zrok (20) es una herramienta que permite crear túneles seguros entre sistemas, utilizando una red superpuesta proporcionada por OpenZiti (21). Zrok facilita la conexión entre dispositivos, incluso si se encuentran en redes restrictivas o carecen de direcciones IP públicas.

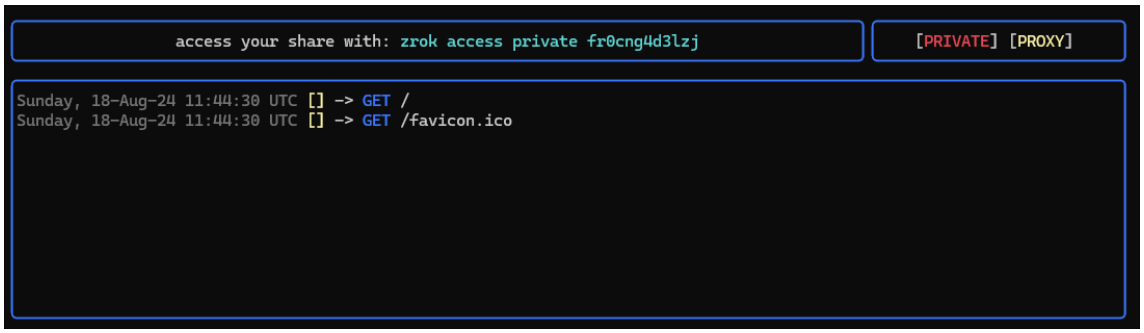

Figura 4.3: Zrok sirviendo NeRFstudio (22)

Las pruebas realizadas con Zrok visibles en la figura 4.3 que muestra una captura de la terminal de Zrok, mostraron su utilidad para exponer aplicaciones web. Sin embargo, se identificaron algunas limitaciones que dificultan su uso en este proyecto:

- **Automatización limitada:** La creación de túneles con Zrok no se puede automatizar fácilmente, ya que las direcciones de los túneles se generan de forma aleatoria. Esto dificulta la integración de Zrok en un flujo de trabajo automatizado.
- **Límites de tráfico y costes:** Zrok ofrece planes de precios con límites de tráfico de red diarios. El uso de la plataforma para el entrenamiento y la renderización de escenas NeRF puede superar fácilmente estos límites, lo que implicaría costes adicionales.

Además, Zrok presenta una segmentación en sus planes de precios, lo que introduce restricciones adicionales. Los límites de tráfico de red diarios pueden suponer un obstáculo importante para los usuarios con necesidades elevadas de ancho de banda o que utilicen aplicaciones que requieran una transmisión de datos continua.

A pesar de estas limitaciones, Zrok ofrece una solución viable para la creación de túneles seguros, especialmente para aplicaciones web. Sin embargo, las limitaciones en la automatización y los posibles costes asociados al tráfico de red hacen que Zrok no sea la opción ideal para este TFG.

Se buscó una herramienta que permitiera la creación de túneles de forma automatizada, con la posibilidad de generar claves de conexión en tiempo de ejecución y sin límites de tráfico.

Hypershell

Hypershell (23) es una herramienta que permite establecer conexiones seguras tipo shell entre sistemas, utilizando un protocolo peer-to-peer (P2P) con autenticación y cifrado de extremo a extremo. Hypershell no requiere una red centralizada, lo que facilita la conexión entre dispositivos en redes restrictivas o sin direcciones IP públicas.

(20) Fuente: Zrok (https://www.openziti.org/zrok)
(21) Fuente: OpenZiti (https://www.openziti.org/)
(22) Comando: `./zrok share private 127.0.0.1:7007`
(23) Fuente: Hypershell (https://docs.pears.com/tools/hypershell)

Las pruebas realizadas con Hypershell visibles en la figura 4.4 que muestra una captura de una terminal de Hypershell ejecutándose, demostraron su utilidad para acceder a shells remotos, transferir archivos y crear túneles seguros para exponer servicios locales. Hypershell permite la conexión entre dispositivos en redes restrictivas, sin necesidad de direcciones IP públicas.

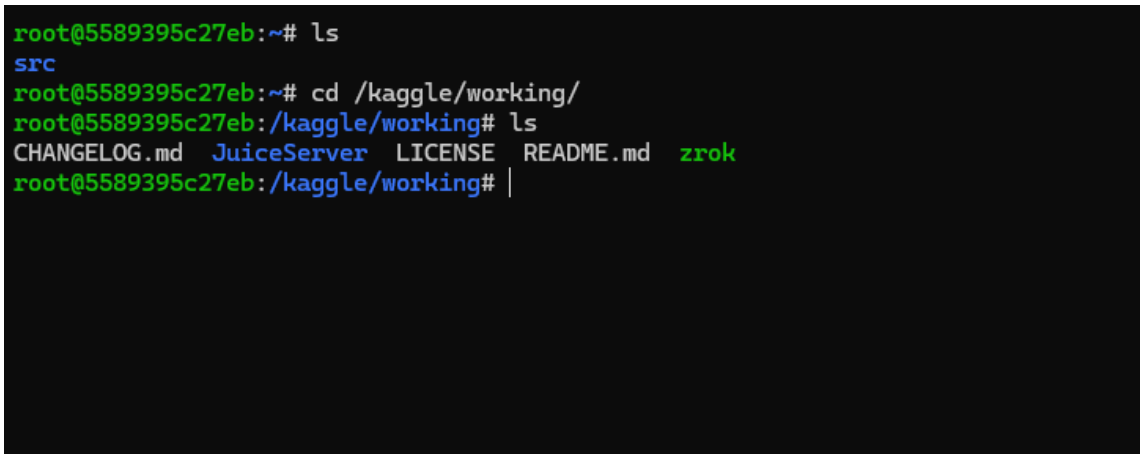

Figura 4.4: Conexión a la máquina virtual de Kaggle utilizando Hypershell (24)

Hypershell ofrece las siguientes ventajas:

- **Conexiones P2P directas:** Hypershell utiliza un protocolo P2P, lo que elimina la necesidad de un servidor intermedio y permite conexiones más rápidas y eficientes.
- **Seguridad robusta:** Hypershell implementa autenticación y cifrado de extremo a extremo, garantizando la seguridad y la privacidad de la comunicación.
- **Automatización:** Hypershell permite generar claves de conexión en tiempo de ejecución, lo que facilita la integración en flujos de trabajo automatizados.
- **Sin límites de tráfico:** una de las grandes ventajas del P2P es que, una vez establecida la conexión, no existen dependencias externas, lo que implica también que no hay restricciones en el ancho de banda ni en la cantidad de datos transferidos.
- **Flexibilidad y control:** Hypershell ofrece opciones de configuración avanzadas, como la restricción de protocolos, la configuración precisa de los túneles y la gestión de acceso a nivel de usuario.

En comparación con Zrok, Hypershell ofrece una mayor flexibilidad, seguridad y control sobre la conexión. La capacidad de generar claves de conexión en tiempo de ejecución y la ausencia de límites de tráfico lo convierten en una solución ideal para este TFG.

En base a las ventajas mencionadas anteriormente, se seleccionó Hypershell como la herramienta para la creación del túnel y conexión segura entre el servidor y el instalador.

Se implementó un prototipo que combinaba la instalación local de NeRFstudio con la virtualización de la GPU en la nube utilizando «Juice» y Hypershell. La prueba de concepto se realizó utilizando «Vkcube» para verificar la funcionalidad del sistema."

(24) Comando: `hypershell zzjwaft7fpooh5wdpmwsdxtrsgoe6mojowhh9usdejt6uzb5ezro`

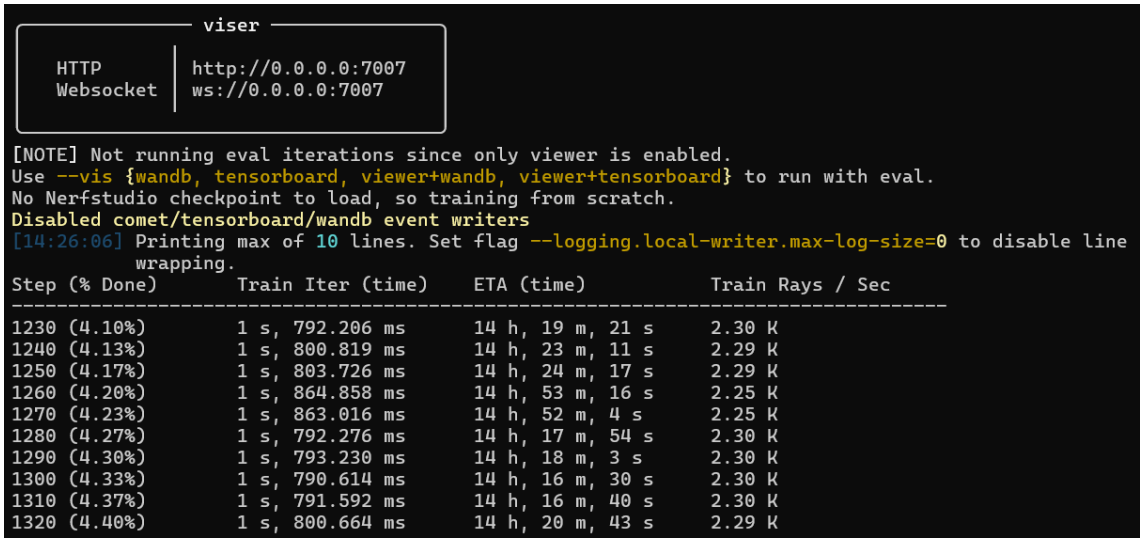

Figura 4.5: Captura de la ejecución de NeRFstudio en un sistema local, utilizando Juice para acceder a una GPU P100 virtualizada en Kaggle (25)

Como se puede ver en la figura 4.5, que muestra una captura de la ejecución de NeRFstudio en un sistema local utilizando Juice para acceder a una GPU P100 virtualizada en Kaggle, los resultados de la prueba de concepto mostraron un tiempo de entrenamiento de aproximadamente 14 horas para la escena «poster» con el método «nerfacto» en una GPU P100 virtualizada. El visor presentaba un retraso de unos 3 segundos para renderizar la escena después de mover la cámara. La prueba se realizó mediante una conexión a Internet por cable de 600 Mbps.

Al utilizar una conexión WiFi de 200 Mbps, el tiempo de entrenamiento no se vio afectado significativamente. Sin embargo, la visualización en el visor se volvió inconsistente, con una tasa de fotogramas variable y un retraso notable en la renderización. La inicialización del entrenamiento también tardaba unos 20-30 segundos más.

Estos resultados indican que Juice es sensible a la latencia y la velocidad de la conexión a Internet. La virtualización de la GPU y la transmisión de los comandos gráficos a través de la red introducen un retraso que puede afectar al rendimiento del visor, especialmente en conexiones con alta latencia o un ancho de banda limitado.

Debido a los problemas de rendimiento del visor y la sensibilidad de Juice a la latencia de la conexión, se descartó la opción de virtualización de la GPU en la nube. Sin embargo, la prueba de concepto permitió verificar la funcionalidad de la API de Kaggle y la viabilidad de ejecutar el backend de NeRFstudio en la nube.

En base a los resultados de las evaluaciones anteriores, se decidió ejecutar el backend de NeRFstudio completamente en la nube, utilizando Kaggle. Esta solución ofrece acceso a GPUs potentes, la posibilidad de automatizar el entrenamiento y la renderización, y la flexibilidad de acceder a la plataforma desde cualquier dispositivo con conexión a Internet.

4.1.4. Ejecución del Backend en la nube

La solución final seleccionada consiste en ejecutar el backend de NeRFstudio en una libreta Jupyter de Kaggle. La libreta Jupyter se configura para iniciar un entorno limpio

(25) El comando utilizado para iniciar Hypershell es: `hypershell ''semilla'' -L 43210:127.0.0.1:43210` y el comando utilizado para iniciar NeRFstudio es: `ns-train nerfacto -data ''rutaPosterData''`

cada vez que se ejecuta, instalando NeRFstudio y sus dependencias. Hypershell se utiliza para establecer un túnel con una conexión segura con la libreta Jupyter y así poder controlar el backend de forma remota.

La implementación final del backend de NeRFstudio para la plataforma Cloudstudio se ha realizado utilizando una combinación de tecnologías que permiten el entrenamiento, el renderizado y la gestión de escenas NeRF en la nube.

Los componentes principales del backend son un servidor Flask que gestiona las solicitudes de los clientes y un servidor NeRFstudio que coordina el proceso de entrenamiento y renderizado.

Sin embargo, no todo ha sido trivial y han surgido varios problemas que han tenido que solucionarse:

1. Al implementar el servidor NeRFstudio en Kaggle, se encontraron problemas de compatibilidad entre las dependencias de NeRFstudio y las librerías preinstaladas en el entorno de ejecución.

   La instalación de NeRFstudio utilizando pip no siempre resolvía correctamente las dependencias, debido a la complejidad de las relaciones entre los diferentes paquetes de Python. En algunos casos, las versiones de las librerías preinstaladas en Kaggle entraban en conflicto con las versiones requeridas por NeRFstudio.

   Para solucionar estos problemas de compatibilidad, se utilizó un archivo `constraints.txt` que especifica las versiones exactas de las librerías que se deben instalar. El archivo `constraints.txt` se utiliza junto con el argumento `-c` al ejecutar `pip install`, lo que permite restringir la instalación de los paquetes a las versiones especificadas. Este método garantiza la compatibilidad entre las dependencias de NeRFstudio y el entorno de ejecución de Kaggle.

   Este es un buen ejemplo de la iteratividad del Extreme Programming. En la iteración 1 de la instalación del backend de NeRFstudio, el objetivo fue implementar por un lado la conexión del instalador con Kaggle y por otro la ejecución del backend en la nube. Se desarrolló la libreta Jupyter en Kaggle y se integró con Hypershell para la conexión remota.

   Se realizaron pruebas de conexión y se verificó la correcta ejecución de NeRFstudio en Kaggle. Las pruebas revelaron problemas de compatibilidad entre las dependencias, lo que llevó a la decisión de utilizar un archivo llamado `constraints.txt` en la iteración 2.

2. Tras resolver los problemas de compatibilidad, se ejecutó el backend de NeRFstudio en Kaggle. Se observó que el espacio de almacenamiento persistente en `/kaggle/working/` es limitado.

   Para optimizar el uso del espacio de almacenamiento, se utilizó la carpeta temporal `/kaggle/temp/` para almacenar los datos temporales generados durante el entrenamiento, como los checkpoints intermedios. El archivo de punto de control (checkpoint) del modelo entrenado se guarda en la carpeta persistente `/kaggle/working/`.

   La estructura de carpetas del servidor de Kaggle se muestra en la figura 4.6 que es un gráfico hecho con Mermaid sobre la estructura de carpetas de la libreta de Kaggle.

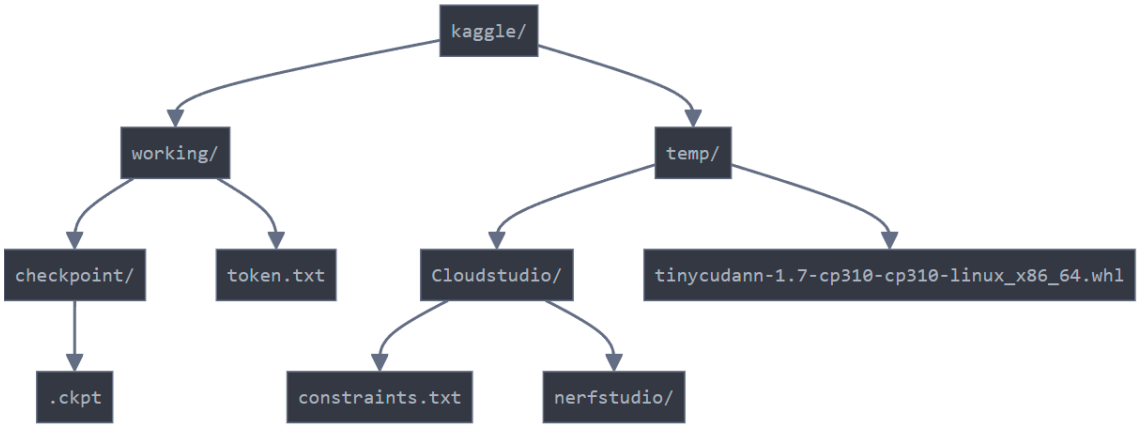

Figura 4.6: Estructura de carpetas del servidor de Kaggle

3. Se observó que el tiempo de inicialización del servidor de Kaggle, incluyendo la instalación de NeRFstudio y sus dependencias, podía ser elevado, llegando a los 20 minutos en algunos casos. Este tiempo de inicialización se percibía como una pausa en la ejecución de la libreta Jupyter, lo que afectaba a la experiencia de usuario. Para mejorar la eficiencia de la plataforma, se implementaron varias optimizaciones para reducir el tiempo de inicialización.

   Para acelerar la instalación de NeRFstudio y sus dependencias, se exploró la posibilidad de paralelizar el proceso utilizando hilos en Python. La paralelización permite aprovechar los múltiples núcleos de CPU disponibles en las instancias de Kaggle con GPUs, lo que reduce el tiempo total de instalación.

4. Inicialmente, se paralelizó la instalación de Hypershell, ya que este proceso no depende de otras librerías. Esto permitió establecer la conexión remota con la libreta Jupyter desde el inicio de la ejecución. Se intentó paralelizar la instalación de NeRFstudio, utilizando hilos para la descarga y la instalación de las dependencias. Sin embargo, se observó que la paralelización de pip, el gestor de paquetes de Python, no proporcionaba una mejora significativa en el rendimiento, debido a las limitaciones de pip en la gestión de dependencias.

   La paralelización de la instalación de Hypershell redujo el tiempo de espera en unos 10 segundos para establecer la conexión remota, pero no tuvo un impacto significativo en el tiempo total de inicialización, ya que la instalación de NeRFstudio seguía siendo un proceso secuencial.

5. Siguiendo el principio de optimización de Amdahl, se analizó el proceso de instalación para identificar los pasos que más tiempo consumían. Se observó que la instalación de NeRFstudio y sus dependencias cuando utilizabamos el comando `pip install -e Cloudstudio` era el paso que más tiempo requería.

   Para comprender las limitaciones de pip en la paralelización, es necesario analizar su funcionamiento. Pip realiza la instalación de paquetes en tres pasos secuenciales:

   - **Resolución de dependencias:** Pip analiza las dependencias especificadas en el archivo requirements.txt o en los metadatos del paquete y determina qué versiones de los paquetes son compatibles entre sí. Este paso es crucial para garantizar la correcta instalación de los paquetes y sus dependencias. La resolución de dependencias es un proceso inherentemente secuencial, ya que la instalación de un paquete puede depender de la instalación previa de otros paquetes.

- **Descarga de paquetes:** Pip descarga los archivos necesarios para cada paquete, incluyendo el código fuente (.tar.gz, .zip) o las ruedas precompiladas (.whl), que son archivos binarios específicos para una plataforma y arquitectura determinadas.
- **Instalación y/o compilación:** Pip instala los paquetes descargados. Para paquetes de Python puro, pip copia los archivos al directorio de instalación. Para paquetes que requieren compilación, como los que contienen extensiones en C o C++, pip compila el código fuente antes de la instalación:
- Si se encuentra una rueda precompilada compatible con la plataforma, pip la instala directamente.
- Si no hay un wheel disponible, pip compilará el paquete localmente. Como se puede ver en la figura 4.7 tenemos que recompilar NeRFstudio ya que hemos hecho cambios. Un pequeño truco consiste en añadir un parámetro a la compilación de binarios, que puede servir para acelerar la compilación de paquetes no precompilados. En nuestro caso, solo ayuda para la compilación de una librería llamada TinyCudaNN, que se reduce un minuto. Más tarde hablaremos de otros cambios que se han hecho en esta librería.

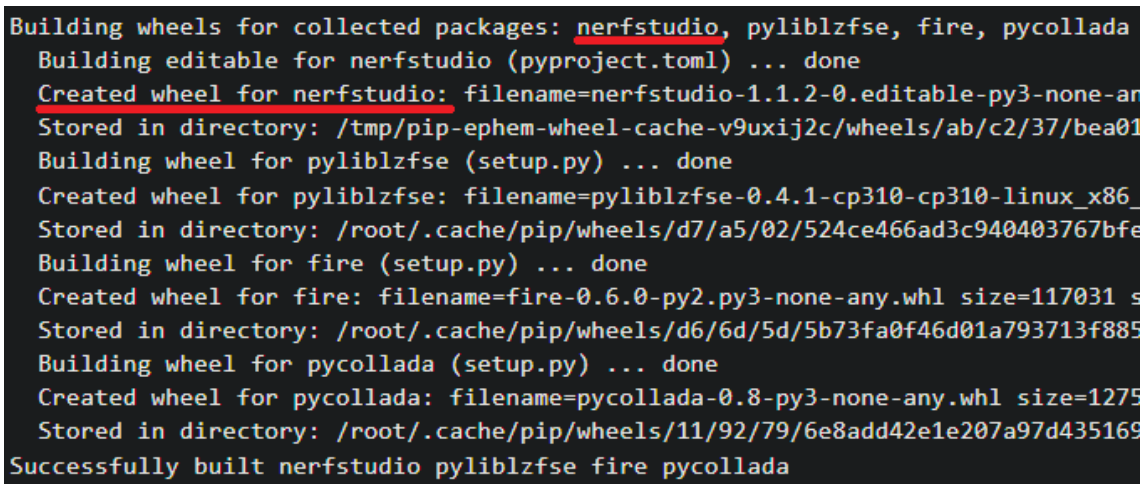

Figura 4.7: Compilación del wheels de NeRFstudio, entre otros ya que hemos hecho cambios para adaptarlo a nuestras necesidades (26)

La compilación e instalación de paquetes pueden ser procesos intensivos en recursos, especialmente si se realizan de forma paralela. La compilación simultánea de varios paquetes puede dar lugar a la competencia por recursos de CPU y memoria, lo que puede afectar negativamente al rendimiento. Además, la instalación paralela de paquetes puede causar problemas de integridad si varios procesos intentan acceder y modificar los mismos archivos o directorios al mismo tiempo.

6. Debido a las limitaciones de pip en la paralelización, se exploró la posibilidad de utilizar un gestor de paquetes alternativo que ofreciera un mejor rendimiento en la instalación de NeRFstudio y sus dependencias. Si bien existen proyectos que intentan paralelizar pip, como pip-faster (27) y propuestas para mejorar la paralelización en el propio pip (28), estas soluciones aún se encuentran en desarrollo o no ofrecen la mejora de rendimiento deseada.

(26) Salida de la terminal de Kaggle ejeuctando: `pip install -e Cloudstudio -c Cloudstudio/constraints.txt`
(27) Fuente: pip-faster (https://github.com/jolynch/pip-faster)
(28) Fuente: Issue abierto en el GitHub oficial de pip (https://github.com/pypa/pip/issues/8187)

7. Se seleccionó uv (29), un gestor de paquetes de Python escrito en Rust, como alternativa a pip. uv está diseñado para ser un reemplazo directo de pip, ofreciendo un mayor rendimiento en la instalación de paquetes, gracias a su capacidad de paralelizar la descarga, la compilación y la instalación de paquetes.

   Por eso lo instalamos normalmente con:
   `pip install uv` o `curl -LsSf https://astral.sh/uv/install.sh | sh`

   Para utilizar uv en el entorno de Kaggle, se configuró para utilizar la instalación de Python actual, mediante el argumento `-python`. Esto garantiza que uv pueda encontrar las librerías locales y las instaladas con pip.

   También, se utilizó el argumento `-e` al ejecutar `uv install` para instalar NeRFstudio en modo editable. Esto permite modificar el código fuente de NeRFstudio y ver los cambios reflejados en la plataforma sin necesidad de reinstalar el paquete.

8. Una vez resuelto esto, el tiempo de ejecución baja a unos 7 minutos, lo cual es bastante mejor, pero sigue siendo inviable para una aplicación diseñada para usuarios sin conocimientos del sistema. En realidad, ya hemos hecho todo lo que podíamos hacer de manera oficial. Sin embargo, todavía podemos probar un truco más específico para esta plataforma, que se podría considerar un «hack».

9. Se analizó el proceso de instalación y se identificó que la compilación de tinycudann, una librería que utiliza NeRFstudio para la aceleración por GPU, consumía una cantidad significativa de tiempo (aproximadamente 5 minutos). Para optimizar este paso, se utilizó un repositorio (30) que proporciona ruedas precompiladas de tinycudann para los entornos de ejecución de Kaggle y Colab. La utilización de las ruedas precompiladas eliminó la necesidad de compilar tinycudann, lo que redujo significativamente el tiempo de inicialización.

10. En resumen, la utilización de uv como gestor de paquetes permitió paralelizar la descarga y la instalación de los paquetes, reduciendo el tiempo de instalación del backend en un 30 %. La utilización de ruedas precompiladas de tinycudann eliminó la necesidad de compilar la librería, lo que redujo el tiempo de inicialización en un 40 % adicional.

Estas optimizaciones implementadas en el proceso de inicialización, incluyendo la paralelización de Hypershell, la utilización de uv y las ruedas precompiladas de tinycudann, permitieron reducir el tiempo de instalación del backend de 15 minutos a 50 segundos. La tabla 4.4 muestra el impacto de las optimizaciones en el tiempo de inicialización de la libreta Jupyter (Los tiempos relativos indican la reducción del tiempo de inicialización con respecto a la implementación anterior, el tiempo total corresponde al tiempo de inicialización acumulado) y la figura 4.8 confirma que el backend tarda 1:15 en terminar toda la configuración, como se ve en la marca roja incluida la descarga de la escena de prueba y como se ve en el progreso de la descarga, que llega al 100 % sobre el segundo 75.

(29) Fuente: uv (https://github.com/astral-sh/uv)
(30) Fuente: tiny-cuda-nn-wheels (https://github.com/OutofAi/tiny-cuda-nn-wheels)

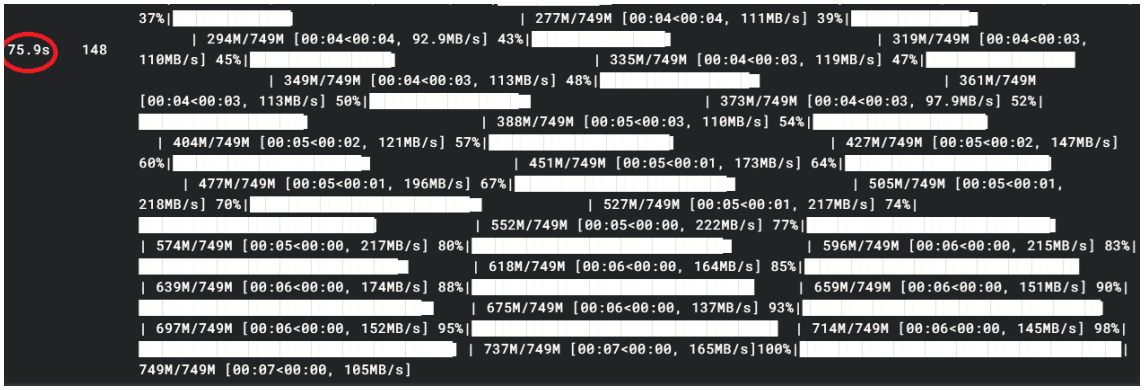

Figura 4.8: El backend tarda alrededor de 1:15 en ejecutar toda la instalación (31)

Para verificar la correcta instalación y configuración del backend de NeRFstudio, también se incluyó la descarga de una escena de prueba en la libreta Jupyter. El entrenamiento de esta se puede ejecutar para comprobar que todo funciona bien.

Descripción	Tiempo relativo	Tiempo de Ejecución total
Aprovisionamiento de recursos	20 seg	∼20 seg
Tiempo de instalación	+ 60 seg	∼1:20 min
Con Escena de prueba	+ 10 seg	∼1:30 min
Sin Wheel tinycudann	+ 4:20 min	∼5:50 min
Sin UV install	+ 6:30 min	∼12:20 min
Sin MAKEFLAGS	+ 2:10 min	∼14:30 min
Sin Hypershell paralelizado	+ 10 seg	∼14:50 min
Implementación mínimamente funcional	-	∼15 min

Tabla 4.4: Impacto de las optimizaciones en el tiempo de inicialización del backend de NeRFstudio en Kaggle

Tras implementar las optimizaciones mencionadas anteriormente, se logró un tiempo de inicialización satisfactorio para el backend de NeRFstudio en Kaggle. El backend se ejecuta en la nube y se controla de forma remota utilizando Hypershell. Los siguientes apartados se centran en el desarrollo del visor y el instalador, con el objetivo de proporcionar una interfaz de usuario intuitiva y fácil de usar que permite el control remoto del entrenamiento de escenas NeRF.

Una vez terminada la instalación y configuración del backend de NeRFstudio, necesitamos una forma de controlarlo, ya que Kaggle no ofrece estos servicios en su API. Para ello, usamos Flask para crear un servidor a través del túnel que Hypershell crea, que nos permita controlar la libreta Jupyter.

4.1.5. Backend Flask en Kaggle

Esta parte de la libreta Jupyter es el núcleo del control remoto del servidor (implementado como una celda en la libreta) e implementa una aplicación «Flask» junto con «Socket.IO» que permite ejecutar comandos de forma remota y monitorizar métricas del sistema. A continuación, se detallan los pasos principales:

(31) Backend ejecutado con el comando desde el instalador: `ns-train nerfacto -pipeline.datamanager.masks-on-gpu True -pipeline.datamanager.images-on-gpu True -viewer.make-share-url True -data data/nerfstudio/poster`

1. **WebSockets:** se utiliza Flask-SocketIO para establecer conexiones WebSocket con el cliente instalador, lo que permite el control remoto en tiempo real y el envío de actualizaciones sobre el estado del sistema. El sistema de websockets solo se utiliza para enviar métricas del sistema, que también se emplean como un "ping" para saber el estado de la conexión con el cliente.

   En las primeras iteraciones no existía un sistema de websockets, ya que todo se hacía con peticiones HTTP que funcionaban bien, pero con lentitud. Sin embargo, con el fin de que el sistema fuera más rápido y eficiente y de que las métricas tardaran demasiado en mostrar el estado del sistema, se ha implementado una parte de websockets que permite el envío más rápido de estos datos.

2. **Métricas del sistema:** el servidor recopila métricas del sistema cada segundo, como el porcentaje de uso de la CPU, el porcentaje de memoria RAM utilizada, el porcentaje de uso de la GPU y el porcentaje de memoria VRAM utilizada, y las envía periódicamente a los clientes conectados lo que permite monitorizar el rendimiento. Estas también se utilizan como latidos, lo que permite informar al instalador de cuándo se ha completado la instalación, entre otras cosas.

3. **Ejecución de comandos:** el servidor proporciona dos endpoints que utilizan la API de Flask para ejecutar comandos, visualizar sus resultados y detener los procesos en ejecución:

   a) El primero se utiliza como la entrada/salida de una terminal. Recibe los comandos enviados mediante una solicitud POST a /run, los ejecuta con una terminal emulada y envía infinitamente los resultados hasta que el proceso ha terminado.

   b) El segundo se utiliza para comprobar si hay algún proceso en ejecución y, en ese caso, bloquear el acceso para poder eliminarlo. Funciona con una solicitud POST a /stop y es muy útil en caso de que haya un problema.

4. **Gestión de procesos:** el servidor utiliza la clase `PtyProcessUnicode`(32) para ejecutar comandos en una pseudo-terminal, lo que permite la interacción con los procesos de entrenamiento y renderizado. Este método, similar al famoso «pexpect», es necesario porque nos permite crear una terminal emulada en el servidor, que se encarga de todo y permite que cualquier programa funcione, y solo enviar los segmentos de texto que necesitamos de vuelta al instalador.

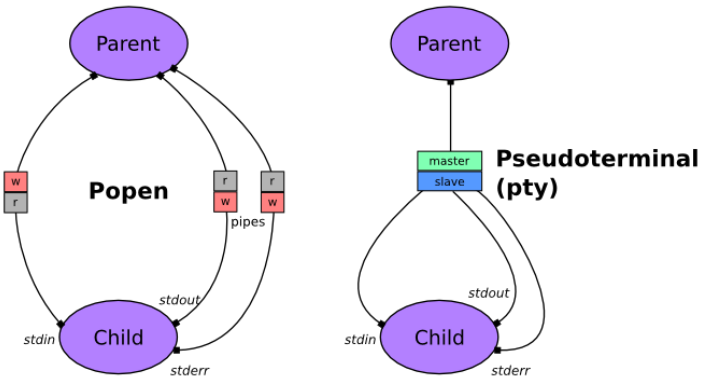

Figura 4.9: Gráfico explicando la diferencia entre Popen, utilizado en las primeras implementaciones y Ptyprocess (33)

(32) Fuente: PtyProcessUnicode (https://github.com/pexpect/ptyprocess)

Las primeras implementaciones de la gestión de la terminal del servidor no funcionaban bien, ya que hay ciertas salidas que no se pueden redirigir por una PIPE ni se envían por «stdout», que es lo que hace «Popen» (véase figura 4.9). Con esta implementación, tenemos un control completo y podemos eliminar los caracteres innecesarios mientras mantenemos los caracteres de control.

5. **Streaming de salida:** La salida de los comandos se transmite en tiempo real a los clientes, lo que permite hacer un seguimiento en vivo del progreso del entrenamiento. La conexión cliente-servidor no se interrumpe hasta que el proceso actual se complete y es persistente entre desconexiones. Esta conexión no es bloqueante, lo que significa que permite seguir sirviendo los endpoints y entrenando la escena.

6. **Transferencia de archivos:** Mantiene una conexión con el servidor que permita enviar como el vídeo de la escena y recibir archivos de este, como la escena entregada.

4.2 Visor

NeRFstudio ofrece dos opciones para la visualización de las escenas NeRF: el visor "legacy", que está obsoleto, y el visor Viser (34). Viser es una implementación más reciente, diseñada para ser más modular, extensible y ofrecer un mejor rendimiento. En base a estas ventajas, se seleccionó Viser como base para el desarrollo del visor de la plataforma. Sin embargo, Viser requiere algunas modificaciones para adaptarlo a los requisitos específicos del TFG, como la traducción al español, la simplificación de la interfaz de usuario y la implementación de la sincronización de cámara entre clientes.

El visor se conecta al backend de NeRFstudio utilizando WebSockets, un protocolo de comunicación que permite la transmisión de datos en tiempo real entre el cliente y el servidor. La conexión WebSocket se establece a través de una URL pública generada por el backend de NeRFstudio, utilizando un servicio externo que proporciona un proxy inverso. Este servicio proporciona una dirección web temporal que permite al visor conectarse al backend, incluso si el backend no tiene una dirección IP pública. La URL se genera al iniciar el entrenamiento de la escena y se muestra al usuario a través de la interfaz del instalador.

El visor Viser, en su configuración por defecto, ofrece las siguientes funcionalidades:

- **Visualización en tiempo real:** El visor permite visualizar la escena 3D renderizada por el backend de NeRFstudio en tiempo real, con una tasa de fotogramas fluida y baja latencia.

  Las pruebas realizadas mostraron una tasa de fotogramas de hasta 120 FPS al mover la cámara, con una latencia de aproximadamente medio segundo entre el movimiento de la cámara y la actualización de la vista en el visor. La latencia incluye el tiempo de transmisión de los datos, el renderizado de la escena en el backend y la actualización de la imagen en el visor.

- **Control de la cámara:** El visor permite al usuario controlar la posición y la orientación de la cámara, navegar por la escena y utilizar herramientas de visualización, como la selección de diferentes modos de renderizado y la visualización de información sobre la escena.

(33) Fuente: Primera foto documentación Ptyprocess (https://ptyprocess.readthedocs.io/en/latest/)
(34) Fuente: Viser (https://github.com/nerfstudio-project/viser)

- **Interfaz de usuario:** Viser proporciona una interfaz de usuario (UI) intuitiva para acceder a las funcionalidades del visor. La interfaz de usuario permite controlar la cámara, seleccionar el modo de renderizado, visualizar información sobre la escena y grabar vídeos con keyframes. La interfaz de usuario está en inglés por defecto.

Figura 4.10: Escena `redwoods2` mediante el Visor "legacy" de NeRFstudio (35)

La figura 4.10 muestra el visor «legacy» de NeRFstudio con el menú de «Render» desplegado, visualizando la escena `redwoods2` durante el entrenamiento. Si bien este visor ofrece una buena base para el desarrollo del visor de la plataforma, se han identificado algunas áreas de mejora para adaptarlo a los requisitos del TFG:

- **Traducción al español:** La interfaz de usuario de Viser está en inglés por defecto. Se requiere traducir la interfaz de usuario al español para hacerla accesible a un público más amplio.
- **Simplificación de la interfaz de usuario:** La interfaz de usuario de Viser ofrece una gran cantidad de opciones de configuración y visualización, lo que puede ser abrumador para usuarios no técnicos. Se propone simplificar la interfaz de usuario, ocultando las opciones avanzadas y presentando una interfaz más limpia e intuitiva.
- **Sincronización de cámara:** Viser no implementa la sincronización de cámara entre clientes por defecto. Se requiere desarrollar esta funcionalidad para permitir la colaboración multiusuario en la exploración de la escena.

Para abordar estas áreas de mejora, se aprovechará la arquitectura modular de Viser para extender su funcionalidad y adaptarlo a los requisitos del TFG.

4.2.1. Cambios propuestos

El código fuente del visor Viser se encuentra en la librería Viser de Python, que se instala como parte de NeRFstudio. La inicialización del visor se realiza en el módulo

(35) Fuente: Fotograma extraído de NeRFstudio ejecutándose en la nube (https://huggingface.co/spaces/SpacesExamples/nerfstudio)

Viewer de NeRFstudio. Para implementar las mejoras mencionadas anteriormente, se modificó el script `Viewer.py`, que inicializa el visor Viser.

Sincronización de la cámara

Para implementar la sincronización de la cámara entre clientes, se realizaron las siguientes modificaciones en el script `Viewer.py`, que se pueden apreciar en la figura 4.12:

1. **Variables globales:** Se añadieron dos variables globales al script `Viewer.py`:
   - `sincronizacion`: Esta variable booleana controla si la sincronización de la cámara está activa (True) o inactiva (False). Por defecto, la sincronización está desactivada.
   - `control`: Esta variable numérica almacena el ID del cliente que tiene el control de la cámara sincronizada. Inicialmente, el control se asigna al primer cliente que se conecta (ID 0).
2. **Funciones de control:** Se implementaron dos funciones para gestionar la sincronización de la cámara:
   - `toggle_sincronizacion()`: Esta función activa o desactiva la sincronización de la cámara, actualizando el valor de la variable sincronizacion. La función también muestra un mensaje en la consola indicando el estado actual de la sincronización.
   - `toggle_control(num_clientes)`: Esta función permite cambiar el control de la cámara sincronizada al siguiente cliente en la secuencia. El argumento num _clientes indica el número total de clientes conectados. La función actualiza el valor de la variable control y muestra un mensaje en la consola indicando qué cliente tiene el control.
3. **Botones de control:** Se añadieron dos botones a la interfaz de usuario del visor para controlar la sincronización de la cámara como se pueden ver en la figura 4.11 los botones de control de la sincronización de la cámara en el visor Viser:
   - **Sincronización:** Este botón alterna entre "Activar sincronización" y "Desactivar sincronización", activando o desactivando la sincronización de la cámara al llamar a la función `toggle_sincronizacion()`.
   - **Cambiar control:** Este botón permite cambiar manualmente el control de la cámara sincronizada al siguiente cliente en la secuencia, al llamar a la función `toggle_control(num_clientes)`.

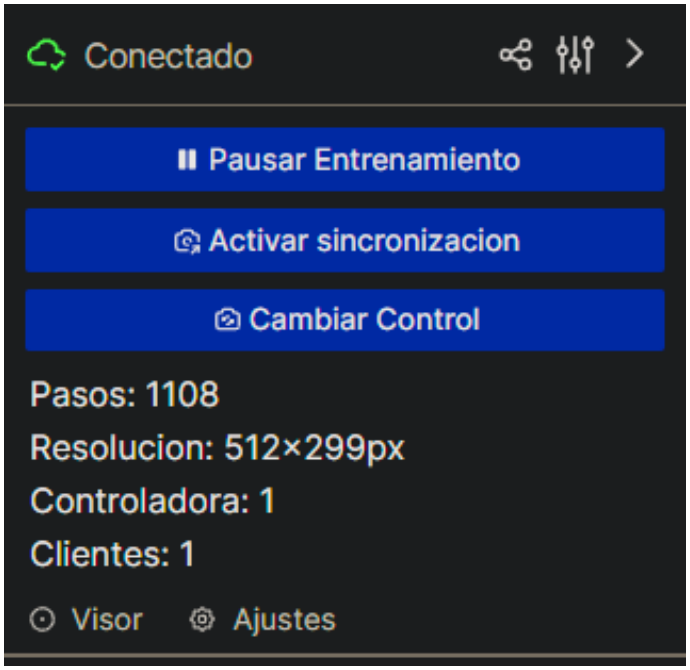

Figura 4.11: Botones de control de la sincronización de la cámara en el visor Viser (36)

4. **Identificación de los clientes:** Se añadió una variable de instancia clientN a la clase Viewer para asignar un ID único a cada cliente conectado. Esta variable se inicializa en 1 y se incrementa cada vez que un nuevo cliente se conecta. La información sobre el ID del cliente que controla la cámara sincronizada y el número total de clientes conectados se muestra en la interfaz de usuario del visor, modificando la función `make_stats_markdown()`.

5. **Función de sincronización:** Se implementó la función `sync_camera()`, que se ejecuta periódicamente para sincronizar la cámara entre los clientes. Cuando la variable sincronización está activa (True), la función `sync_camera()` copia la posición y la rotación de la cámara del cliente que tiene el control (control) a las vistas de los demás clientes conectados.

6. **Gestión de conexiones y desconexiones:** Se modificaron las funciones `handle_disconnect()` y `handle_new_client()` para gestionar las conexiones y desconexiones de los clientes.

    - `handle_disconnect()`: Esta función se ejecuta cuando un cliente se desconecta del visor. Si el cliente que se desconecta tenía el control de la cámara sincronizada, la función reasigna el control al siguiente cliente en la secuencia.
    - `handle_new_client()`: Esta función se ejecuta cuando un nuevo cliente se conecta al visor. La función asigna un ID único al nuevo cliente, utilizando la variable clientN, y actualiza la interfaz de usuario para mostrar la información del nuevo cliente.

(36) Captura recortada del menú de control de Viser ejecutada con el comando: `ns-train nerfacto -viewer.make-share-url True -data data/nerfstudio/poster`

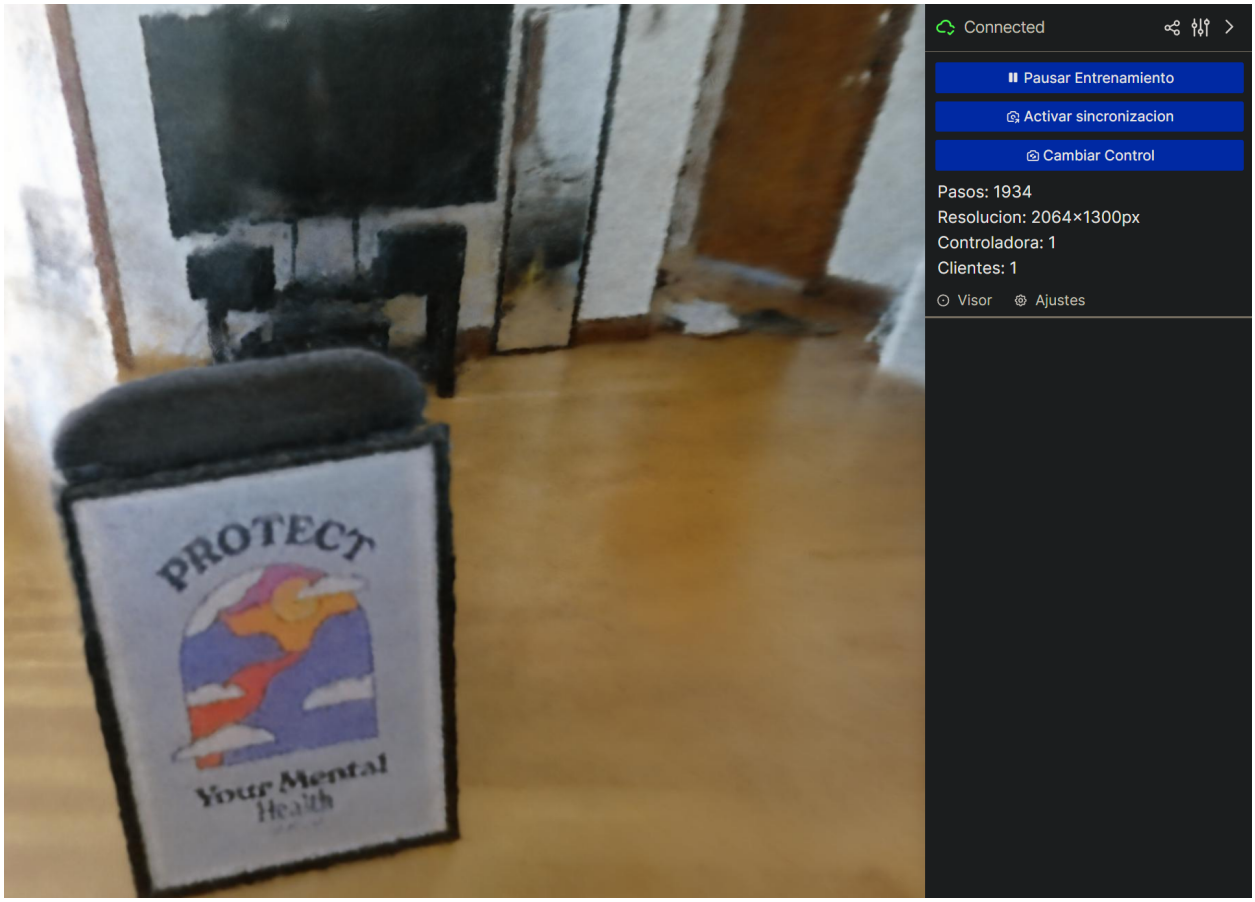

Figura 4.12: Visor Viser modificado, mostrando la escena «poster» durante el entrenamiento (37)

Para mejorar la usabilidad del visor, se realizaron las siguientes modificaciones (visibles en la figura 4.12) en la interfaz de usuario:

- **Traducción al español:** Se tradujo la interfaz de usuario al español, incluyendo los textos de los botones, las etiquetas y los mensajes de la consola. La traducción se realizó modificando las funciones que generan la interfaz de usuario en el script `viewer.py`.
- **Simplificación de la interfaz de usuario:** Se simplificó la interfaz de usuario, ocultando las opciones que no son relevantes para el caso de uso de la plataforma, como la exportación a malla 3D y la grabación de vídeos. Esto se logró modificando el archivo `control_panel.py`, que define los elementos de la interfaz de usuario.
- Las opciones avanzadas se ocultaron utilizando un sistema de pestañas, lo que permite acceder a ellas si es necesario, pero no satura la interfaz de usuario con una gran cantidad de opciones.

En resumen, las modificaciones realizadas en el visor Viser permitieron implementar la sincronización de cámara entre clientes, la traducción de la interfaz de usuario al español y la simplificación de la interfaz de usuario, mejorando la usabilidad y las capacidades de colaboración de la plataforma.

(37) Viser ejecutado con el comando: `ns-train nerfacto -viewer.make-share-url True -data data/nerfstudio/poster/`

4.3 Instalador

El instalador de «Cloudstudio» es una aplicación de escritorio que facilita la configuración y el uso de la plataforma. El instalador se encarga de instalar las dependencias necesarias, configurar la conexión con Kaggle, abrir un túnel directo al servidor con Hypershell, iniciar el entrenamiento de escenas en la nube, controlar remotamente el servidor y proporciona una interfaz de usuario para todo esto, sin necesidad de escribir nada de código.

El instalador se desarrolló principalmente utilizando Python, la librería customTkinter para la interfaz gráfica de usuario y flask, para la conexión con el backend Flask. El instalador se distribuye como un archivo ejecutable (.exe) utilizando PyInstaller, lo que permite su uso en sistemas Windows sin necesidad de instalar Python u otras librerías adicionales. El instalador también se puede ejecutar en otros sistemas operativos, como Linux o Mac, utilizando el código fuente en Python proporcionado en el repositorio del TFG (38).

Para su funcionamiento, el instalador requiere una conexión a Internet constante, el archivo de configuración de la API de Kaggle (kaggle.json) y el video de la escena 3D que se desea procesar y entrenar. Como se puede ver en la figura 4.13 el instalador puede funcionar de forma portable en una sola carpeta, siempre y cuando se tenga el vídeo y el archivo de la API de kaggle en el mismo directorio.

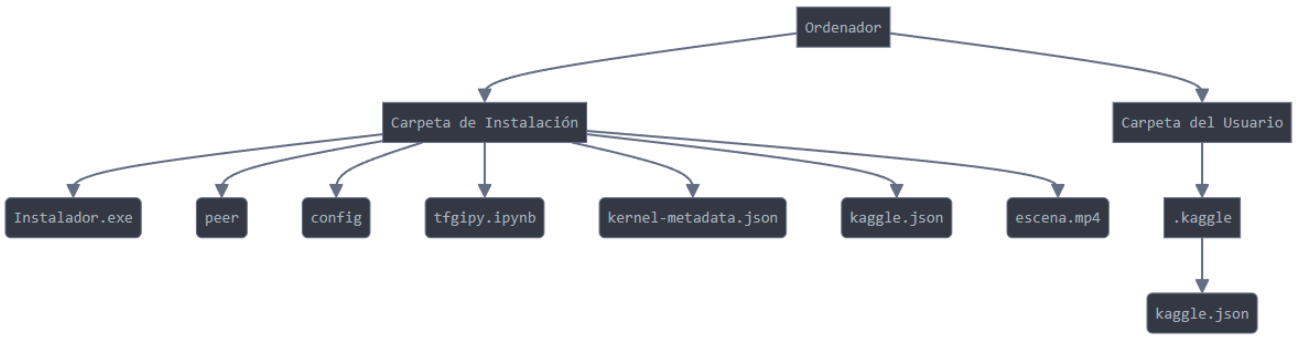

Figura 4.13: Estructura final de las carpetas del instalador. Cabe destacar que peer y config se crean sobre la marcha (39)

Las funciones del instalador son las siguientes:

4.3.1. Características principales del instalador

- **Instalación de dependencias:** el instalador importa todas las dependencias necesarias para ejecutar el instalador como nodejs-bin, flask, socketIO y la librería de la API de Kaggle. En el caso de Windows, el instalador incluye los binarios de estas dependencias en el archivo ejecutable, lo que permite instalarlas sin necesidad de descargarlas e instalarlas por separado.
- **Consistencia entre ejecuciones:** el instalador es consistente entre ejecuciones, ya que utiliza archivos de configuración para almacenar los pares de claves previamente generados entre ejecuciones. Esta característica permite, además, hacer por-

(38) Fuente: Código fuente del instalador en Python disponible en el Repositorio Github de Cloudstudio (https://github.com/pablogranell/Cloudstudio/)
(39) Gráfico realizado con Mermaid sobre la estructura de carpetas final del instalador

table la instalación entre diferentes sistemas. También, permite actualizar la libreta Jupyter utilizada en el servidor simplemente colocándola al lado del ejecutable.

- **Configuración y conexión con la API de Kaggle:** el instalador comprueba la existencia del archivo de configuración de la API de Kaggle (kaggle.json) en la ubicación predeterminada o junto al archivo ejecutable. Si no se encuentra el archivo, el instalador permite al usuario elegir la ubicación en la que tiene el archivo y realiza la transferencia a la ruta correcta. Una vez hecho esto el instalador gestiona la autenticación y la conexión con la API de Kaggle, permitiendo la ejecución de la libreta Jupyter en la nube.

- **Verificación de los datos de entrada:** el instalador comprueba que los vídeos de la escena 3D que se desea entrenar estén disponibles en la ubicación correcta. Después, permite al usuario transferir cómodamente los vídeos a la carpeta correcta del servidor para que el backend de NeRFstudio los procese y así pueda entrenar una escena de estos.

- **Disposición de accesos directos:** el instalador presenta comandos predefinidos al usuario para ejecutar comandos comunes de NeRFstudio. Algunos de estos comandos son: un comando para transferir los vídeos al servidor, un comando para procesar los datos y transformarlos al formato compatible con NeRFstudio, un comando para iniciar el entrenamiento de una escena NeRF, un comando para recuperar el punto de control de una escena a medias o totalmente entrenada y un comando para detener el backend en la nube.

```
PREDEFINED_COMMANDS = {
    "Transferir video al servidor": {
        "command": f"hypershell-copy {os.path.join(SCRIPT_DIR, 'escena.mp4')} token:/kaggle/temp/Cloudstudio/data/nerfstudio/escena.mp4",
        "local": True
    },
    "Preparar escena": {
        "command": "ns-process-data video --data /kaggle/temp/Cloudstudio/data/nerfstudio/escena --output-dir /kaggle/temp/Cloudstudio/dat
        "local": False
    },
    "Entrenamiento": {
        "command": "ns-train nerfacto --pipeline.datamanager.masks-on-gpu True --pipeline.datamanager.images-on-gpu True --logging.local-w
        "local": False
    },
    "Devolver escena (semi)entrenada": {
        "command": "hypershell-copy token:/kaggle/working/checkpoint.ckpt /kaggle/temp/Cloudstudio/data/nerfstudio/poster",
        "local": True
    },
```

Figura 4.14: Recorte de la estructura de los comandos predefinidos del instalador (40)

Como se puede ver en la figura 4.14, esta parte es altamente modular, ya que permite elegir si se quiere que el comando se ejecute en el servidor o en local y permite añadir cualquier comando siguiendo las convenciones de la terminal de Linux.

- **Control remoto de la libreta Jupyter:** utiliza Hypershell para establecer una conexión segura con el backend de Flask y controlar la ejecución de la libreta Jupyter. Permite iniciar, detener el entrenamiento y gestionar la ejecución de comandos en el servidor remoto. La interfaz proporciona botones y campos de entrada para enviar comandos personalizados al backend de manera cómoda.

- **Monitoreo en tiempo real:** muestra las métricas del sistema (CPU, RAM, GPU y VRAM) y el estado de la conexión en tiempo real. Utiliza gráficos y medidores ac-

(39) Para descargar el archivo kaggle.json, el usuario debe iniciar sesión en su cuenta de Kaggle y generar una nueva clave API en la sección «Account»
(40) Fuente: Estructura disponible (https://github.com/pablogranell/Cloudstudio/blob/main/FINAL.py#L26)

tualizados dinámicamente para visualizar el uso de recursos del servidor. También incluye indicadores de estado para el estado de ejecución de la libreta Jupyter, estado de conexión con la libreta de Kaggle y el estado del backend de NeRFstudio.

- **Tolerancia a fallos:** el instalador implementa mecanismos de tolerancia a fallos para detectar y recuperarse de errores en la conexión con el backend en la nube, como la pérdida de conexión o la desconexión del servidor. El instalador monitoriza el estado del servidor verificando el túnel del proceso de Hypershell y el backend de Flask utiliza un mecanismo de latido para comprobar la disponibilidad de la libreta Jupyter. Si se detecta un error en la conexión, el instalador intenta reconectarse automáticamente al backend tras un tiempo determinado.
- **Terminal integrada:** Incluye una consola que muestra la salida de los comandos ejecutados, lo que permite al usuario seguir el progreso de las operaciones. Permite el desplazamiento y busca la URL del visor automáticamente para ponerla en un apartado específico para mayor comodidad del usuario. Además, indica claramente cuando el servidor está ocupado ejecutando comandos.

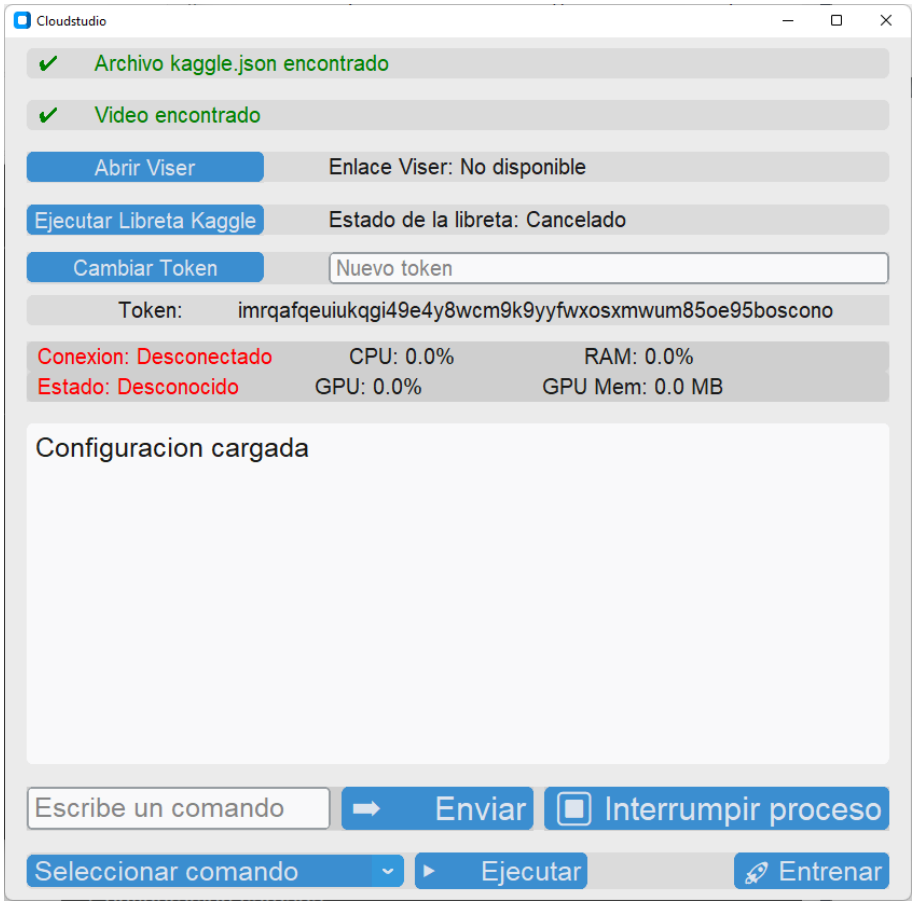

Figura 4.15: Interfaz gráfica del instalador con el modo claro activado (41)

Todo esto se presenta con una interfaz sencilla e intuitiva con «customTkinter», que proporciona una experiencia coherente y ofrece feedback constante. Incluye botones intuitivos, barras de progreso para mostrar el avance de las operaciones y un diseño limpio

(41) Captura de la ejecución del Instalador iniciado mediante Cloudstudio.exe y con el modo claro de Windows activado

y organizado que guía al usuario a través del proceso de configuración y uso de la plataforma. La interfaz también se adapta al modo oscuro y modo claro del sistema, como se ve en la figura 4.15 y la figura 4.16.

4.3.2. Interfaz gráfica

Por encima, el instalador implementa una interfaz gráfica de usuario mediante «customTkinter» que facilita la interacción del usuario con la plataforma. «customTkinter» es una extensión de Tkinter, la librería estándar de Python para la creación de interfaces gráficas, por lo que se integra fácilmente en este y con el backend del instalador, que gestiona el túnel y la conexión con el servidor.

La idea es que el usuario no tenga que hacer muchos clics ni escribir nada, así que, eliminando la terminal integrada que sirve para revisar el progreso, las reglas de diseño que hemos utilizado se basan en reducir la interacción con el instalador al mínimo ya que gran parte de las tareas se pueden automatizar. Esta interfaz, que se muestra en la figura 4.16, es un ejemplo de ello y guía al usuario en el proceso de entrenamiento de escenas NeRF con tan solo 3 clics.

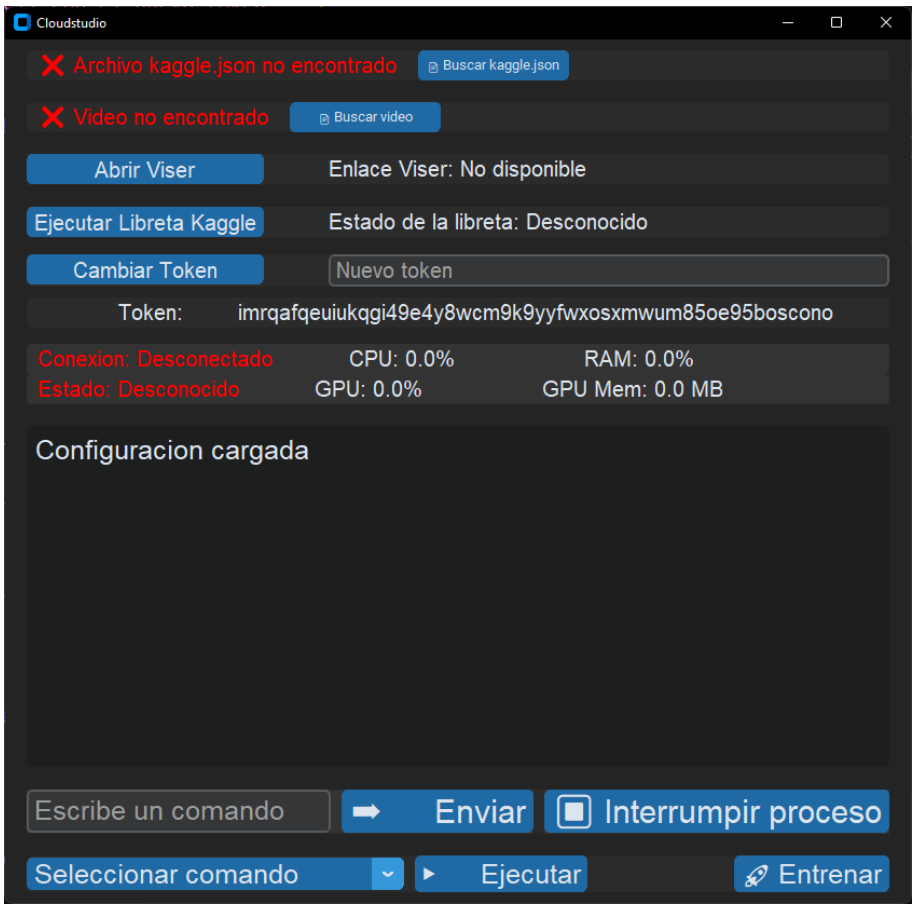

Figura 4.16: Interfaz gráfica del instalador nada más abrirlo (42)

En resumen, el instalador simplifica la configuración y el uso de la plataforma, abstrayendo la complejidad técnica del usuario. Su interfaz intuitiva, la automatización de las tareas y la gestión de la conexión con el backend en la nube permiten a los usuarios, incluso sin conocimientos técnicos avanzados, entrenar y visualizar escenas NeRF de forma sencilla.

(42) Captura de la ejecución del Instalador iniciado mediante Cloudstudio.exe

4.4 Conexión cliente-servidor

La plataforma utiliza una arquitectura cliente-servidor: el backend de NeRFstudio y el backend de Flask se ejecutan en la nube (Kaggle), mientras que los clientes (visor e instalador) se ejecutan en el sistema del usuario. Esta conexión entre el cliente y el servidor en nuestra plataforma se basa en una arquitectura compleja que combina tres tecnologías para lograr una comunicación eficiente y segura:

- **WebSockets:** Se utiliza para la comunicación entre el visor y el backend de NeRFstudio, y para el envío de métricas entre el servidor y el instalador. Esto permite la transmisión de datos y métricas en tiempo real, como la imagen renderizada de la escena, la información sobre la posición de la cámara y la información sobre el uso de la GPU.
- **HTTP:** Se utiliza para enviar los comandos al backend Flask y que este envíe la respuesta al instalador hasta que termina la ejecución de dicho comando. También se usa para solicitar la parada del comando en ejecución.
- **Hypershell:** Se utiliza para la comunicación entre el instalador y la libreta de Kaggle, ya que crea un túnel entre ellos mediante UDP holepunching que encapsula las 2 previas tecnologías. Esto permite redirigir el puerto del servidor a uno local, para así poderle hacer peticiones desde el instalador.

El proceso de conexión se divide en varias etapas y componentes clave, y cada uno aporta una funcionalidad concreta al protocolo de conexión, el cual es crucial para el funcionamiento general del sistema. Estas etapas en orden son:

4.4.1. Establecimiento del túnel con Hypershell

El primer paso en la conexión con el backend Flask es el establecimiento de un túnel seguro entre el cliente y el servidor utilizando Hypershell. Para ello, el instalador implementa un entorno portátil de Node en instalador.exe, que inicia Hypershell para crear un túnel con un comando como `hypershell token -L puerto:localhost:puerto`. Simultáneamente, el servidor ejecuta `hypershell-server token -disable-firewall` para aceptar la conexión entrante. Hypershell utiliza UDP holepunching para establecer una conexión directa entre el cliente y el servidor, incluso cuando ambos están detrás de NAT o firewalls. Este enfoque es crucial ya que permite la conexión entre sistemas sin direcciones IP públicas, como las redes mas restrictivas del cliente y la de los servidores de Kaggle.

4.4.2. Comunicación a través de Flask y SocketIO

Una vez establecido el túnel, la comunicación entre el cliente y el servidor se realiza principalmente a través de Flask y SocketIO. El servidor Flask, implementado en server.py, se ejecuta en el lado del servidor, exponiendo endpoints HTTP y WebSocket. El cliente se conecta al servidor Flask a través de SocketIO, utilizando el túnel Hypershell previamente establecido. Las métricas del sistema, como uso de CPU, memoria y GPU, se transmiten en tiempo real utilizando WebSockets, permitiendo actualizaciones rápidas y eficientes de la interfaz de usuario. Las peticiones HTTP, como la ejecución de comandos y la obtención de su respuesta, se realizan a través del mismo túnel.

4.4.3. Manejo de comandos y terminal remota

El sistema implementa una funcionalidad de terminal remota y ejecución de comandos predefinidos. Los comandos predefinidos se almacenan en la variable `PREDEFINED_COMMANDS` en FINAL.py. Dependiendo de si un comando es local o remoto, se ejecuta utilizando `subprocess.run()` en el cliente o se envía al servidor mediante una petición POST. La función `run_command()` en FINAL.py maneja la ejecución de comandos, enviando una solicitud POST al servidor y recibiendo la salida en chunks. La salida de la terminal se muestra en la interfaz de usuario utilizando un cuadro de texto y una cola para manejar la actualización asíncrona.

4.4.4. Gestión de errores y reconexión

El sistema implementa mecanismos robustos para manejar errores de conexión y reconexiones. Si se produce un error de conexión, el sistema intenta reconectarse automáticamente después de un breve intervalo. La función `restart()` en la clase Hypershell limpia el socket, el túnel y todos los componentes relacionados antes de iniciar una nueva conexión. Durante los períodos de reconexión, el sistema no intenta enviar datos a través del túnel, evitando así errores adicionales.

Los componentes claves que ayudan al proceso de conexión son:

4.4.5. Interacción con la API de Kaggle

La interacción con la API de Kaggle es un componente crucial del sistema. El cliente descarga la plantilla de la libreta de Kaggle junto con el archivo de metadatos utilizando la API, actualiza la clave privada en la libreta Jupyter descargada y, a continuación, la sube de nuevo a Kaggle de forma segura mediante HTTPS. Se inicia la ejecución de la libreta Jupyter en la plataforma Kaggle y, a continuación, el cliente monitoriza periódicamente su estado. Este proceso permite la ejecución remota de código en los servidores de Kaggle, aprovechando sus recursos computacionales.

4.4.6. Integración del visor Viser

El visor Viser se integra en el sistema para la visualización de resultados. Se inicia como un componente separado durante el proceso de entrenamiento. El cliente busca constantemente la URL del visor Viser en la salida del proceso de entrenamiento. Una vez detectada, la URL se muestra en la interfaz de usuario, lo que permite al usuario acceder fácilmente a la visualización.

4.4.7. Editor de libretas Jupyter

El instalador se encarga de crear y guardar un par de claves para posteriormente incrustarlas en la libreta Jupyter mediante el comando `echo "token» /root/.hypershell/peer`. Esto facilita la conexión, ya que el instalador ya sabe cuál será la clave pública del servidor y simplemente envía peticiones hasta que este esté preparado y empiece a devolverlas. Además, de esta forma, puedes tener varias libretas Jupyter diferentes abiertas siempre y cuando los instaladores estén en distintas carpetas.

4.4.8. Seguridad y autenticación

La seguridad en la comunicación cliente-servidor se maneja mediante un sistema de clave pública y privada de tipo ed25519 generadas con `hypershell-keygen`, diseñado para su uso con el protocolo hypercore (43). La conexión es extremo a extremo (E2E) debido a la naturaleza P2P de la comunicación, lo que proporciona un alto nivel de seguridad. Las credenciales se manejan de forma segura, sin necesidad de transmitirlas en texto plano a través de la red, se guardan en la libreta Jupyter del servidor y se cargan al servidor de Kaggle con HTTPS.

4.4.9. Depuración

El sistema implementa un mecanismo sencillo de depuración. La función `log_to_terminal()` se utiliza para mostrar mensajes importantes al usuario. Estos mensajes ayudan al usuario a comprender el estado actual del túnel, del backend Flask, el estado del entrenamiento de la escena y cualquier otro problema que pueda surgir. Aunque no se trata de un sistema de registro de errores formal, esta función proporciona una forma efectiva de comprender el protocolo de conexión.

4.4.10. Concurrencia y manejo de múltiples clientes

Aunque el sistema está diseñado principalmente para un solo cliente que controla de forma remota, tiene capacidades de concurrencia. El servidor puede gestionar varias conexiones Hypershell, Flask y del visor Viser simultáneamente. Flask permite gestionar solicitudes de métricas y comandos de forma concurrente. El visor Viser puede gestionar múltiples clientes conectados simultáneamente para la visualización colaborativa de las escenas entrenadas.

4.4.11. Actualizaciones del sistema

El proceso de actualización del sistema es bastante flexible. Para actualizar el instalador (cliente), se modifica el código fuente directamente, que además es bastante modular. Para actualizar el servidor, se modifica el archivo `tfgipy.ipynb` y el instalador lo sube automáticamente a Kaggle. Este enfoque permite una flexibilidad considerable en la actualización de componentes individuales del sistema, como las claves o el orden de ejecución.

4.4.12. Flujo de comunicación de NeRFstudio

Para NeRFstudio no hemos hecho ningún cambia en su protocolo de comunicaciones ya que presenta un proxy inverso público mediante `https://share.viser.studio/?request_forward` que nos permite compartir el visor entre varios clientes sin tener que configurar el instalador en cada uno de ellos. El flujo de información del backend de NeRFstudio y el visor es:

1. El instalador envía un comando de ejecución del backend para iniciar el entrenamiento y la visualización de una escena.
2. El backend configura NeRFstudio para entrenar la escena especificada y genera una URL pública para el visor Viser, utilizando un servicio externo.

(43) Fuente: Hypercore Github (`https://github.com/holepunchto/hypercore`)

3. El usuario abre la URL del visor en su navegador web.
4. El visor establece una conexión WebSocket con el backend utilizando la URL proporcionada.
5. **Interacción con la escena:** El usuario interactúa con la escena 3D en el visor, enviando comandos de control al backend (posición y orientación de la cámara, zoom, etc.) a través de WebSockets. Para gestionar el renderizado y la visualización:
   - El usuario interactúa con la escena 3D en el visor, controlando la cámara, el zoom y otras opciones de visualización.
   - El visor envía la información sobre la posición de la cámara y las opciones de visualización al backend a través de WebSockets.
   - El backend renderiza la escena 3D de acuerdo con la posición de la cámara y las opciones de visualización recibidas del visor.
   - El backend envía la imagen renderizada al visor mediante WebSockets.
   - El visor recibe la imagen, la renderiza con three.js y la muestra al usuario, permitiendo la visualización de la escena en tiempo real.
6. **Sincronización Multi-Cliente:** Cuando la sincronización está activada, el visor, junto con el backend, se organizan para replicar la posición de la cámara en todos los clientes.
   - Si la sincronización está activada, el visor del cliente «controlador» envía información adicional sobre la posición y orientación de la cámara.
   - El backend retransmite esta información al resto de visores conectados.
   - Los visores del resto de clientes actualizan la posición y orientación de sus cámaras, sincronizando la vista con la del cliente «controlador».

4.5 Herramientas y librerías utilizadas

El desarrollo de la plataforma se basó en las siguientes herramientas y librerías, para cada una de las cuales se adjunta en el pie de página la documentación leída:

- **NeRFstudio [9]:** NeRFstudio es la librería principal utilizada para el entrenamiento, la renderización y la visualización de las escenas NeRF. Su arquitectura modular y extensible facilita la implementación de nuevas funcionalidades y la integración de diferentes métodos NeRF.
- **Viser (44):** Viser es el framework de visualización web utilizado por NeRFstudio. Viser proporciona una interfaz de usuario intuitiva para explorar e interactuar con las escenas NeRF renderizadas. Se modificó Viser para traducir la interfaz de usuario al español, simplificarla y añadir la funcionalidad de sincronización de cámara.
- **Kaggle (45):** Kaggle es la plataforma de aprendizaje automático en la nube utilizada para ejecutar el backend de NeRFstudio. Kaggle proporciona acceso gratuito a GPUs y una API que permite automatizar la ejecución de libretas Jupyter.

(44) Fuente: Repositorio Viser (https://github.com/nerfstudio-project/viser)
(45) Fuente: Kaggle (https://www.kaggle.com/)

- **Flask (46):** Flask es un micro-framework de Python utilizado para crear el servidor web. Desde el cliente hasta el servidor, Flask maneja las peticiones HTTP y proporciona los endpoints necesarios para la comunicación entre el cliente y el servidor.
- **SocketIO** (47): SocketIO se utiliza para establecer una comunicación bidireccional en tiempo real entre el cliente y el servidor. Lo usamos principalmente para transmitir las métricas del sistema, ya que son datos que requieren actualizaciones frecuentes.
- **PyInstaller (48):** PyInstaller se utiliza para empaquetar el instalador de Python y sus dependencias en un archivo ejecutable independiente. Esto permite distribuir el instalador para sistemas Windows sin necesidad de que el usuario tenga Python o las librerías adicionales instaladas.
- **nodejs-bin (49):** nodejs-bin es un paquete de Python que proporciona una distribución binaria de Node.js para su uso en proyectos Python. Se utiliza para ejecutar Hypershell, que está escrito en Node.js, desde el instalador de Python.
- **customTkinter (50):** customTkinter es una extensión de Tkinter, la librería estándar de Python para la creación de interfaces gráficas de usuario. Se utiliza para desarrollar la interfaz gráfica del instalador, proporcionando una apariencia moderna y una experiencia de usuario intuitiva.
- **uv (51):** uv es un gestor de paquetes de Python escrito en Rust, que ofrece un mayor rendimiento que pip en la instalación de paquetes. Se utiliza para acelerar la instalación de NeRFstudio y sus dependencias en la libreta Jupyter.
- **Hypershell (52):** Hypershell es una herramienta que permite establecer conexiones seguras de tipo shell entre sistemas, utilizando un protocolo P2P con autenticación y cifrado de extremo a extremo. Se utiliza para controlar el backend de NeRFstudio en Kaggle de forma remota.
- **GitHub (53):** GitHub se utiliza para el control de versiones del código fuente de la plataforma y para alojar el repositorio de la libreta Jupyter que contiene el backend de NeRFstudio.
- **Requests (54):** Librería empleada para realizar peticiones HTTP desde el cliente al servidor, facilitando la comunicación entre ambos componentes.
- **NBFormat (55):** Librería empleada para leer, escribir y manipular notebooks de Jupyter, utilizada para actualizar la clave privada en el notebook de Kaggle.
- **Threading (56):** Módulo de Python utilizado para implementar concurrencia, permitiendo que múltiples tareas se ejecuten simultáneamente sin bloquear la interfaz de usuario.

(46) Fuente: Flask docs (https://flask.palletsprojects.com/en/3.0.x/)
(47) Fuente: Flask-SocketIO docs (https://flask-socketio.readthedocs.io/en/latest/)
(48) Fuente: PyInstaller (https://pyinstaller.org/)
(49) Fuente: Repositorio nodejs-bin (https://github.com/samwillis/nodejs-pypi)
(50) Fuente: Repositorio customTkinter (https://github.com/TomSchimansky/CustomTkinter)
(51) Fuente: Repositorio uv (https://github.com/astral-sh/uv)
(52) Fuente: Repositorio Hypershell (https://github.com/holepunchto/hypershell)
(53) Fuente: Repositorio GitHub (https://github.com/pablogranell/Cloudstudio)
(54) Fuente: Request docs (https://requests.readthedocs.io/en/latest/)
(55) Fuente: NBFormat docs (https://nbformat.readthedocs.io/en/latest/format_description.html)
(56) Fuente: Threading docs (https://docs.python.org/es/3.8/library/threading.html)

- **Subprocess (57):** Utilizado para ejecutar comandos del sistema operativo y procesos externos desde el script de Python, esencial para ciertas operaciones del sistema.

(57) Fuente: Subprocess docs (https://docs.python.org/3/library/subprocess.html)

CAPÍTULO 5
Resultados

Una vez que la plataforma *Cloudstudio* ha sido implementada con éxito, podemos hacer pruebas para comprobar si permite a usuarios sin conocimientos técnicos sobre NeRF ni servidores, entrenar y visualizar escenas 3D utilizando la tecnología NeRF satisfactoriamente.

Esta plataforma ha sufrido algunas modificaciones desde su primera implementación, que se pueden ver en la figura 5.1, lo que refleja la metodología de desarrollo iterativo propia del Extreme Programming. La figura es útil para poder comparar los cambios que se han hecho hasta la implementación final.

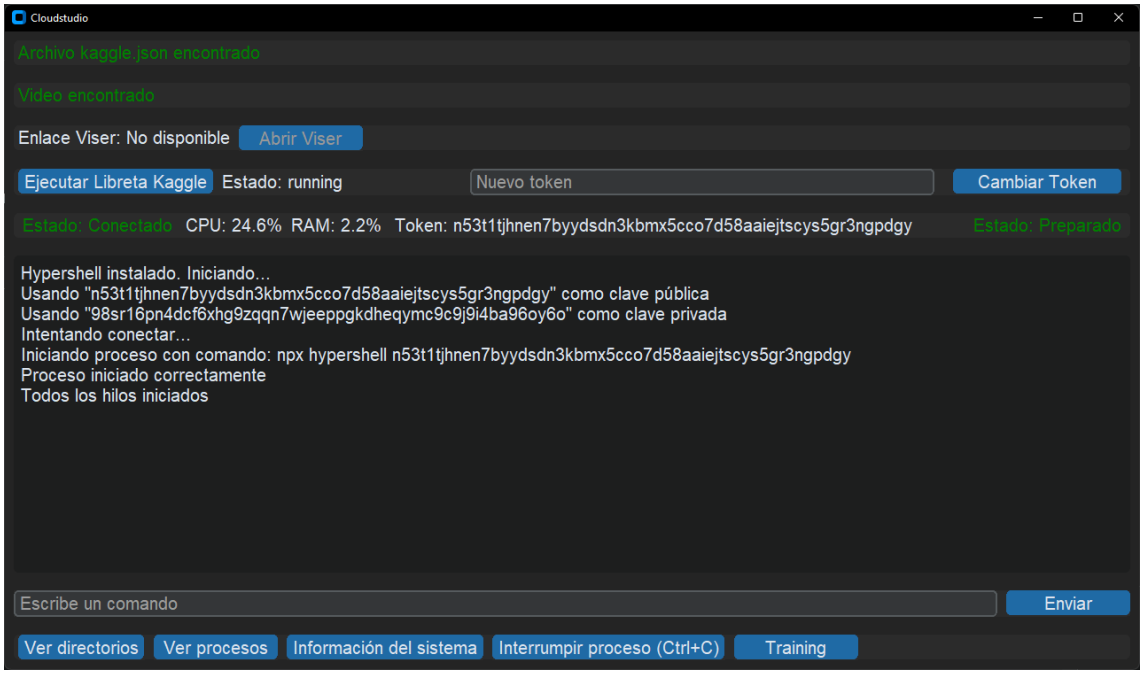

Figura 5.1: Primera implementación de la interfaz gráfica del instalador

5.1 Requisitos y especificaciones cumplidos

A continuación tenemos una lista que explica los cambios e implementaciones que se han hecho para cumplir los requisitos y especificaciones establecidos en el capítulo 3:

- **Facilidad de uso:** tanto el instalador como el visor se diseñaron con interfaces gráficas de usuario intuitivas. Con la implementación final del instalador, toda la puesta

en marcha de dependencias, la creación del túnel, la conexión con el backend en la nube y la apertura del visor se podían realizar con solo tres clics. El primero para encender la libreta Jupyter y conectarse, el segundo para ejecutar NeRFstudio con los parámetros elegidos y el tercero para acceder a la URL del visor.

Es decir, los usuarios pueden iniciar el entrenamiento de una escena NeRF con solo unos pocos clics y esto se ha mantenido a lo largo de las distintas iteraciones de instalador como un principio de diseño. Ejemplo de ello son los 3 mismos botones de la figura 5.1 y 5.2.

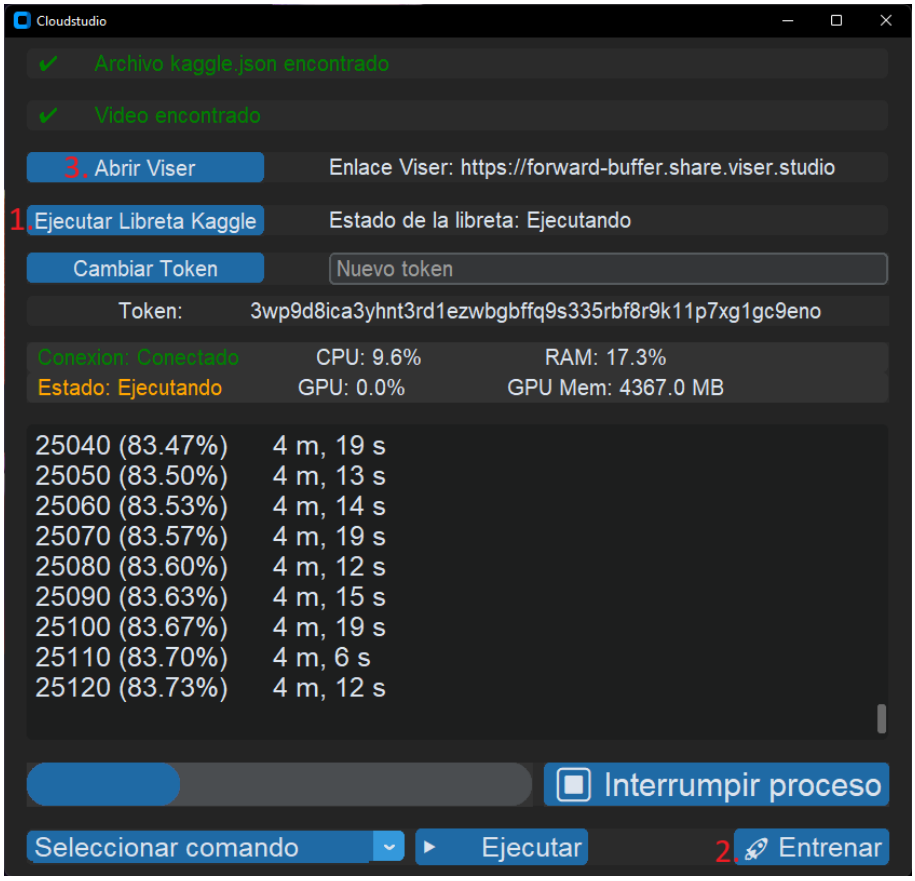

Figura 5.2: Instalador final marcando en rojo los clics necesarios para entrenar una escena (58)

Otro aspecto a destacar, como se puede ver en la figura 5.1, es que hay partes del instalador que siguen en inglés ya que reciben estos datos externamente. Así que, a lo largo del desarrollo de la interfaz, se han realizado numerosos ajustes para que la interfaz actúe como intermediaria y traduzca automáticamente todos los mensajes de estado. Este subsistema nos permite mantener la consistencia y la facilidad de uso del instalador.

- **Simplicidad:** Durante la implementación de la plataforma, se ha pensado principalmente en la simplicidad y, posteriormente, en la funcionalidad. Así que, se ha reducido el número de opciones disponibles a primera vista para no abrumar al usuario nada más abre las diferentes partes de la plataforma.

  Puede verse la diferencia de esta medida en los instaladores de la figura 5.1 y la figura 5.2. La diferencia de esta simplificación también se puede ver en el visor en la figura 4.12, previamente explicada, y la figura 5.13, de la que hablaremos más adelante y que ahora solo precisa atención en la cantidad de opciones que muestra.

(58) Captura reproducible ejecutando el instalador y pulsando los botones marcados como 1 y 2

- **Rendimiento:** El backend, ejecutado en Kaggle con una GPU Tesla P100, logró entrenar escenas interiores de tamaño medio (22 m^2) en menos de 30 minutos con el modelo nerfacto como se puede ver en la figura 5.3. Al utilizar el modelo «nerfacto-big», el tiempo estimado de entrenamiento era de casi ocho horas.

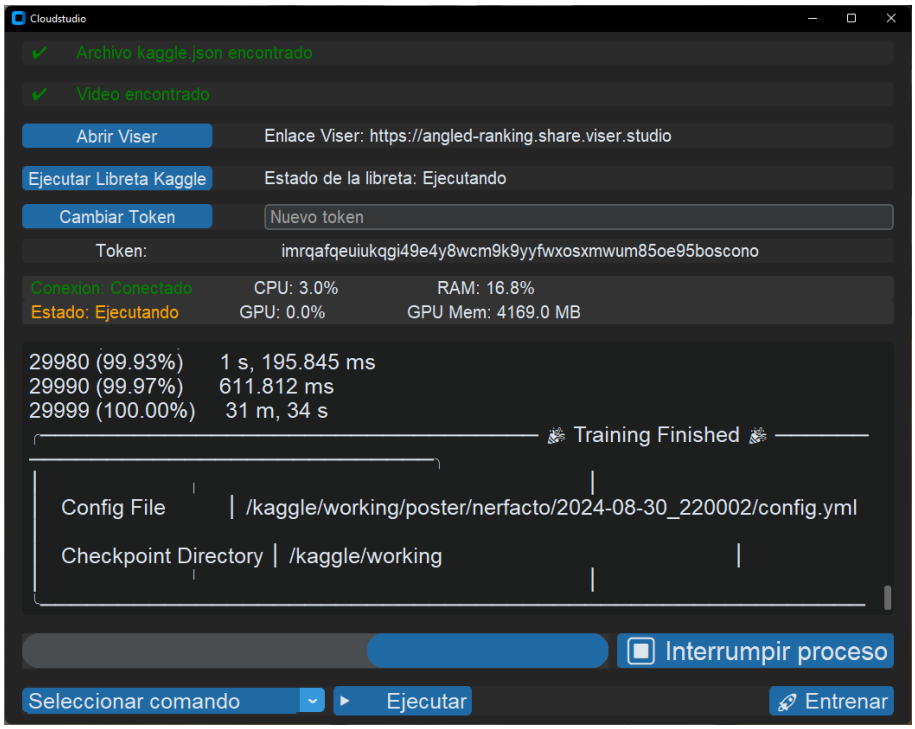

Figura 5.3: Pantalla del instalador con la escena «poster» totalmente entrenada (59)

Un detalle importante a destacar es que abrir una instancia del visor hace que el entrenamiento tarde un 5 % más, ya que tiene que ir optimizando las poses de la cámara a medida que se entrena la escena. Por ello se recomienda dejar que la escena se entrene unos 10-15 minutos antes de empezar a visualizarla para un correcto funcionamiento.

El visor mantuvo una tasa de fotogramas cercana a la cantidad máxima especificada por cada sistema durante la interacción en tiempo real con la escena, prueba de ello es la figura 5.4.

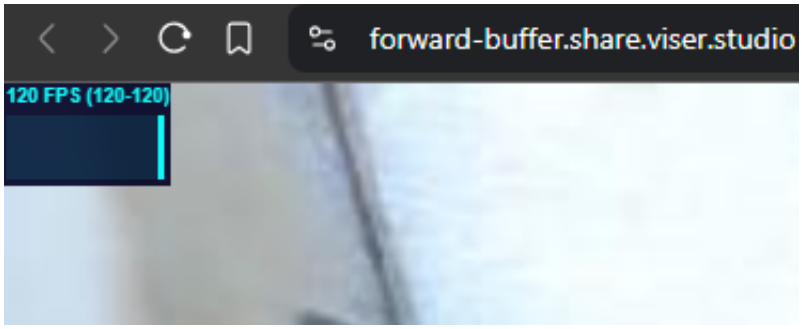

Figura 5.4: Métricas del visor mostrando 120 FPS constantes en un navegador Chromium (60)

(59) Estado del instalador obtenido tras pulsar los tres clics del instalador y dejar que el backend de NeRFstudio entrene la escena durante cerca de 32 minutos

(60) Tan solo hace falta abrir el visor pulsando los 3 clics de entrenamiento de NeRF y abrir el menú de configuración de la esquina superior derecha, para luego activar "WebGL stats"

La latencia de renderizado en el visor desde que movía la cámara fue de aproximadamente medio segundo y, cuando se conectaba el segundo cliente, la latencia empeoraba ligeramente, hasta los 0.8 segundos por cada cliente. En total, el viaje de vuelta entre un cliente que movía la cámara y otro cliente conectado a la misma red que estaba visualizando la escena en la posición de la cámara sincronizada por el otro cliente fue de unos dos segundos.

Este tiempo de respuesta mejoraba notablemente cuando la escena ya estaba entrenada y solo se servía la escena renderizada a los clientes.

- **Claridad visual:** una vez que la escena se entrena por completo, el backend permite subir la resolución del renderizado hasta 2000x1200 sin afectar enormemente al tiempo de respuesta (hasta 5 segundos si la escena no está entrenada). Como se ve en la figura 5.5 los objetos con muchas muestras como el poster tienen una gran calidad visual, mientras que los objetos con menos muestras como la habitación pasada la puerta no resuelven correctamente por falta de información.

  Las escenas renderizadas mostraron una calidad visual aceptable en el modelo navegado desde las primeras iteraciones del entrenamiento. Tras 6000 iteraciones (el 20 % del modelo entrenado), ya era posible distinguir las formas y los contornos de los objetos en la escena de una manera más sencilla. La calidad visual mejoró progresivamente a medida que avanzaba el entrenamiento, alcanzando un alto nivel de detalle en sus últimas etapas.

Figura 5.5: Calidad visual representativa de una escena totalmente entrenada con «nerfacto» (61)

- **Modularidad:** La integración de NeRFstudio en el backend permite la configuración de diversos parámetros del modelo, la selección de diferentes métodos NeRF y la incorporación de nuevos métodos a medida que se desarrollan. Esta modularidad es aprovechada por el instalador para adaptar la funcionalidad al caso de uso.

  Por ejemplo, una vez se había probado todo y se había implementado «nerfacto», se pensó en añadir un modelo reciente que permite el movimiento en la escena, llamado «K-Planes» [31]. Para ello la modularidad del instalador y NeRFstudio fueron

(61) Escena obtenida habiendo dejado al backend de NeRFstudio entrenar la escena completamente

extremadamente convenientes, estando el modelo ya disponible en el backend de NeRFstudio y pudiéndose añadir al instalador en un minuto.

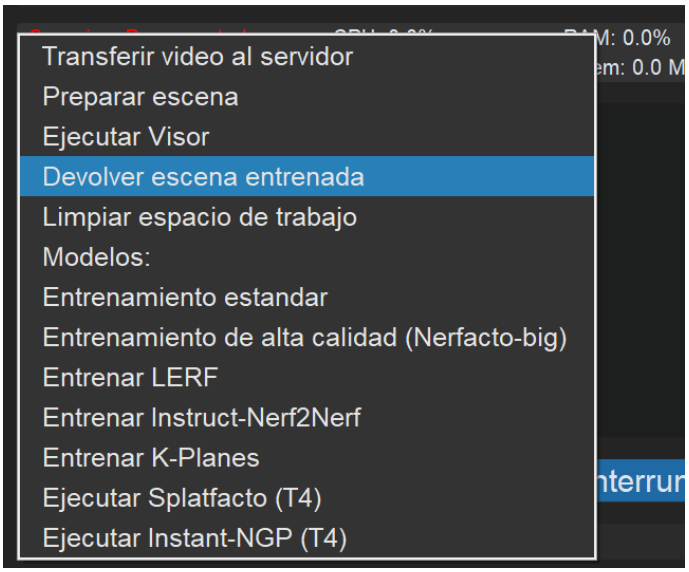

Figura 5.6: Sistema de comandos predefinidos del instalador que muestra todos los comandos disponibles mediante una interfaz sencilla (62)

Como se ve en la figura 5.6, se muestran los comandos disponibles mediante una interfaz sencilla que permite evitar por completo tener que escribirlos en la terminal.

- **Flexibilidad:** el instalador se distribuye como un archivo ejecutable en Windows, pero el código fuente de Python permite ejecutarlo en otros sistemas operativos con solo unas adaptaciones de rutas necesarias. El visor, al ser una aplicación web, es compatible con los navegadores web más comunes con soporte para WebGL como se puede ver en la figura 5.7.

Figura 5.7: Navegador Microsoft Edge ejecutando el Visor

- **Colaboración y control remoto:** la sincronización de la cámara implementada en el visor permite la colaboración multiusuario en la exploración de la escena NeRF. Las pruebas realizadas con dos clientes conectados simultáneamente, mostraron una sincronización precisa de la cámara, sin afectar al rendimiento del sistema, exceptuando un incremento de la latencia de medio segundo.

(62) Recorte del menú desplegable de la esquina inferior izquierda del instalador, ejecutable mediante Instalador.py

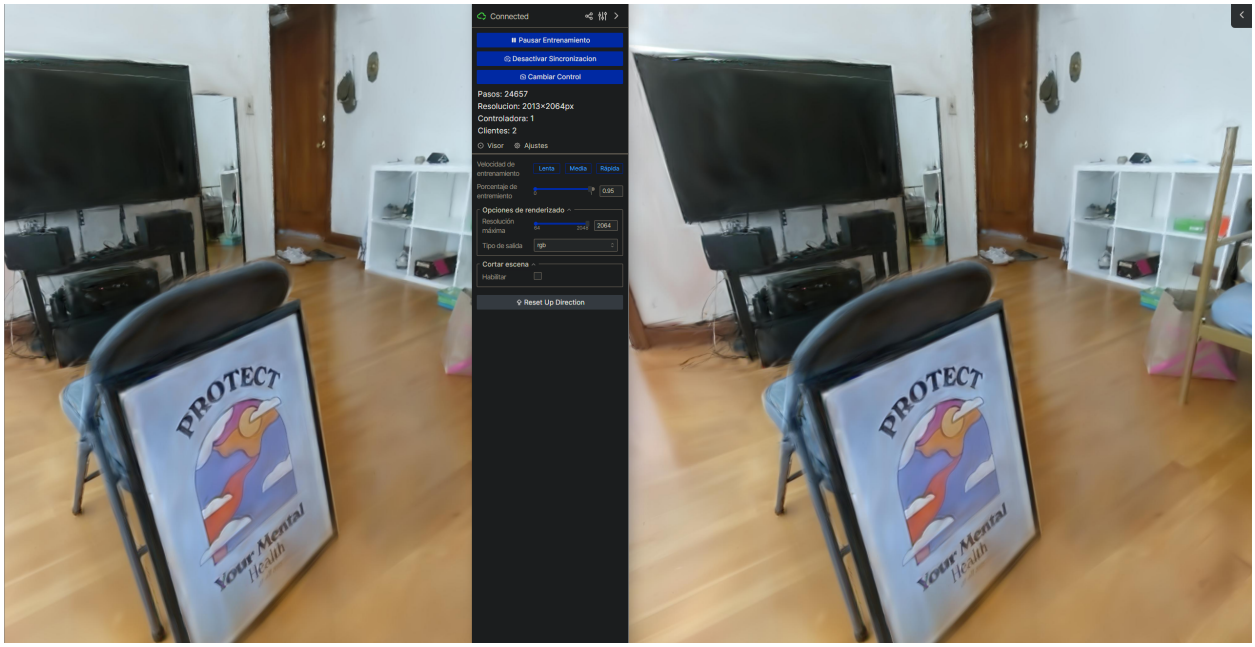

Figura 5.8: Dos pestañas del visor de NeRFstudio ejecutando una escena con la escena de sincronización activada (63)

La figura 5.8, en la que la primera pestaña tiene el control por lo que la segunda pestaña cambia la posición de la cámara para seguirla, es prueba del correcto funcionamiento de la sincronización.

- **Automatización:** El instalador automatiza la configuración del entorno de ejecución, la conexión con Kaggle, la ejecución del backend y otras tareas, lo que simplifica el uso de la plataforma para usuarios no técnicos.

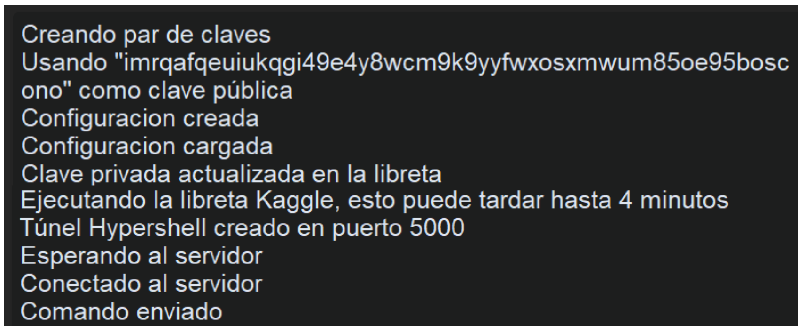

Figura 5.9: Terminal integrada del instalador que muestra cómo se configura todo el proceso automáticamente (64)

Como se puede ver en la figura 5.9, el usuario solo ejecuta la libreta Jupyter y el instalador hace la creación de claves, la configuración de archivos, la inyección de claves en la libreta Jupyter, la conexión con Kaggle, la creación del túnel y la conexión con el servidor.

(63) Splatfacto ejecutado manualmente con `ns-train splatfacto -pipeline.datamanager.masks-on-gpu True -pipeline.datamanager.images-on-gpu True -viewer.make-share-url True -logging.local-writer.max-log-size=0 -data data/nerfstudio/poster`

(64) Esta parte del flujo de trabajo se puede conseguir borrando todos los archivos de configuración presentes en la carpeta del instalador

5.2 Resultados Cuantitativos

Se realizaron una serie de pruebas para evaluar el rendimiento de la plataforma *Cloudstudio* en diferentes escenarios y con distintas configuraciones. A continuación, se presentan los resultados obtenidos.

5.2.1. Nerfacto

El modelo nerfacto tiene una gran cantidad de opciones que se pueden configurar, lo que permite modular un amplio espectro entre la calidad de la escena y el tiempo de entrenamiento. Además, el modelo también se presenta en diferentes tamaños, lo que complica aún más la elección correcta en el entorno en el que se ejecuta el backend de NeRFstudio.

Métrica	**Nerfacto**	**Nerfacto-big**	**Optimizado**	**Impacto**
num_rays_per_batch	4096	8192	4096	nerfacto-big procesa el doble de rayos por lote, lo que consigue una mayor calidad, pero requiere más VRAM. No es relevante a la hora de elegir modelos, ya que se puede configurar en ambos.
nerf_samples_per_ray	48	128	48	nerfacto-big muestrea más puntos por rayo, mejorando la calidad pero aumentando el tiempo de renderizado.
proposal_samples_per_ray	(256, 96)	(512, 256)	(256, 96)	nerfacto-big utiliza más muestras para cada red, mejorando la eficiencia del muestreo pero aumentando la complejidad y el tiempo de entrenamiento.
hidden_dim	64	128	64	nerfacto-big tiene el doble de dimensiones en las capas ocultas, aumentando la capacidad de la red pero requiriendo más memoria y cómputo.
appearance_embed_dim	32	128	32	nerfacto-big mejora la representación de las variaciones de apariencia pero aumentando la complejidad, requiriendo más espacio de almacenamiento.
max_res	2048	4096	8192	Se valoró la posibilidad de establecer una resolución máxima de 8192, que no afectaría al rendimiento, ya que el visor incluye un ajuste para configurarlo sobre la marcha.
Optimizador	Adam	RAdam	RAdam	RAdam puede converger más rápido y ser más estable.

Tabla 5.1: Diferencias entre «nerfacto» y «nerfacto-big»

La tabla 5.1 muestra una comparación entre Nerfacto, Nerfacto-Big y los ajustes optimizados elegidos en función de las pruebas realizadas con ellos. Como se puede ver en la tabla, se comparan configuraciones destinadas a la calidad, la velocidad, la memoria VRAM y el almacenamiento, exponiendo una explicación de lo que hace cada una y argumentando a favor de los ajustes optimizados. Estas recomendaciones se tienen en cuenta para seleccionar el modelo por defecto que le mostramos al usuario en el instalador.

Al profundizar más en estas métricas, el ajuste más relevante para elegir el modelo es la «cantidad de rayos por lote», ya que afecta a todas las métricas que lo hacen viable como modelo. Al profundizar más en ella, descubrimos que afecta a las siguientes métricas:

- La calidad de la escena, que es subjetiva, pero siempre nos interesará aumentar.
- La cantidad de rayos en escena que también es importante, ya que afecta a la exactitud de la transmitancia predicha por el modelo.
- El cómputo requerido, que nos interesa para poder ocupar equitativamente un número suficiente de unidades de procesamiento paralelo, con lo que se aumenta la ocupación y se maximiza la utilización de los recursos.
- La memoria VRAM, que siempre nos interesará maximizar, sin activar la hiperpaginación de esta.
- El espacio utilizado, que preferiblemente nos interesa reducir al máximo posible, durante o post entrenamiento.
- El tiempo de entrenamiento, que nos interesa siempre reducir todo lo que sea posible pero que es dependiente de todo lo anterior.

Se considera que el modelo, con su configuración por defecto, está lo suficientemente optimizado para ocupar equitativamente todas las unidades de procesamiento paralelo y el almacenamiento no supone un problema.

En cuanto al resto de métricas, sabemos que nos interesa encontrar un equilibrio aumentando la calidad y disminuyendo el tiempo de almacenamiento, mientras nos mantenemos por debajo de los niveles de hiperpaginación de la VRAM (algo menos de 16 GB). Para ello se han realizado pruebas para ver en qué punto del espectro calidad-tiempo debería posicionarse.

Como se ve en la gráfica 5.10, a medida que vamos aumentando el número de rayos por lote, el uso de VRAM también aumenta de manera significativa. La gráfica muestra una relación clara entre estos dos factores, con el eje X representando los rayos por lote en escala logarítmica, y el eje Y mostrando el uso de VRAM en GBs.

La curva comienza con un uso relativamente bajo de VRAM, alrededor de una base de 0.6 GB para 2 rayos por lote. A medida que el número de rayos aumenta, el consumo de VRAM crece de manera gradual al principio, pero luego se observa un incremento más pronunciado. Hay un punto de inflexión notable alrededor de los 4096 rayos por lote, marcado en la gráfica como la configuración por defecto, a partir del cual el aumento en el uso de VRAM se vuelve mucho más pronunciado.

Cabe destacar que con un rayo por lote se produce un «error de tensor», lo que sugiere que este es el límite inferior para el funcionamiento del backend de NeRFstudio. También hay que señalar que con más de 44 000 rayos por lote se producen errores de memoria insuficiente, que marcan el límite superior de la capacidad del sistema.

Es decir, el límite inferior es técnico y el límite superior es de recursos, lo que demuestra la gran escalabilidad de NeRFstudio.

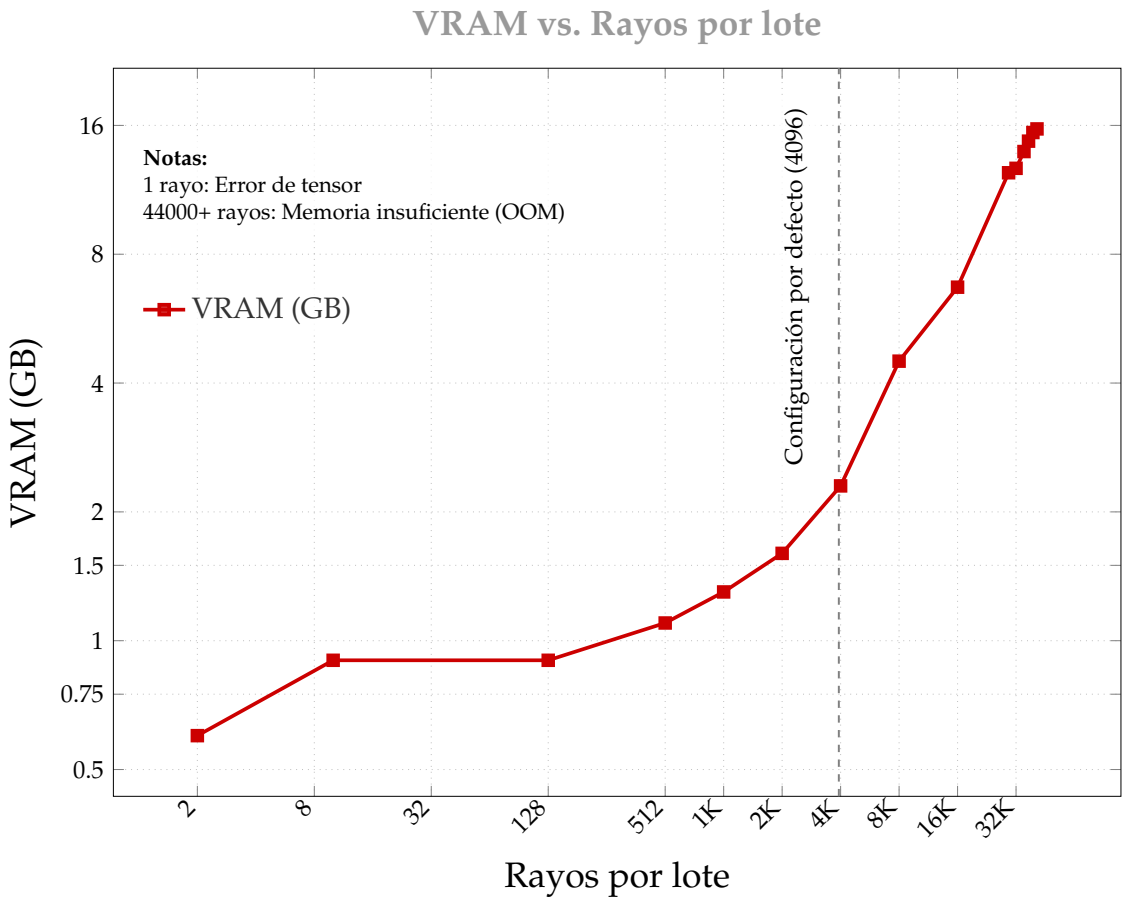

Figura 5.10: Análisis de la relación entre el número de rayos por lote y uso de VRAM

Como se observa en la figura 5.11, a medida que aumentamos el número de rayos por lote, el tiempo de entrenamiento muestra una tendencia interesante. Inicialmente, para cantidades pequeñas de rayos por lote, el tiempo de entrenamiento se mantiene relativamente constante e incluso muestra una ligera disminución. Este comportamiento sugiere que, para estas cantidades menores, el sistema no puede procesar eficientemente los rayos, ya que, en cantidades tan pequeñas, la GPU no puede paralelizar el entrenamiento.

Sin embargo, a partir de 512 rayos por lote, se observa un punto de inflexión claro en la curva. El tiempo de entrenamiento comienza a aumentar de manera exponencial. Este incremento se vuelve particularmente pronunciado a partir de 4 K rayos por lote, donde la curva asciende de forma muy empinada. La gráfica vuelve a mostrar un punto de inflexión notable alrededor de los 4096 rayos por lote, lo que sugiere que este punto también podría ser un umbral importante en términos de eficiencia computacional.

Para las cantidades más altas de rayos por lote, como 16K, 32K y 36K, el tiempo de entrenamiento alcanza valores extremadamente elevados, llegando a más de 4 horas para 36K rayos. Este aumento drástico en el tiempo de procesamiento podría indicar, a primera vista, que el sistema tiene dificultades significativas para gestionar eficientemente cantidades tan grandes de rayos por lote. Sin embargo, en este sentido, el sistema encuentra las mayores cantidades de números de rayos por segundo, con un máximo de 100 K rayos/s.

Esto se debe a que entre estos valores se encuentra la mayor cantidad de rayos que el sistema puede procesar y almacenar a la vez, lo que maximiza la ocupación de todas las unidades de procesamiento y VRAM. Esto no quita para que el tiempo de entrenamiento aumente, ya que a más rayos, más cálculos son necesarios y mejor es la calidad.

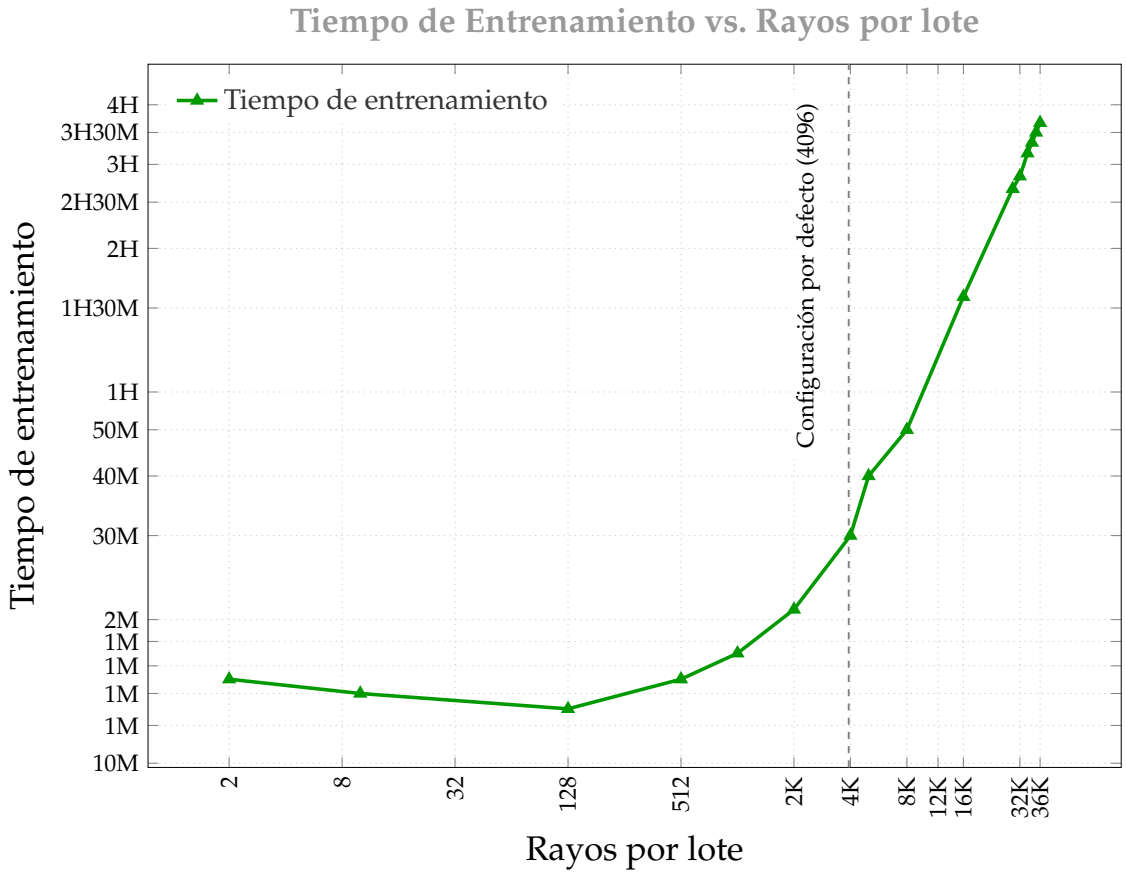

Figura 5.11: Análisis de la relación entre el número de rayos por lote y tiempo de entrenamiento

Ahora nos gustaría hacer una consideración conjunta de estas métricas, que nos permita observar la relación entre el uso de VRAM de la gráfica 5.10 y el tiempo de entrenamiento de la gráfica 5.11 en función del número de rayos por lote. Esta visualización conjunta nos permite identificar patrones y puntos de inflexión comunes en ambas métricas. Para ello, unificamos las tablas y obtenemos la figura 5.12.

Como se puede ver en esta figura, en la región de bajos rayos por lote (2-512), vemos que tanto el uso de VRAM como el tiempo de entrenamiento se mantienen relativamente estables. Sin embargo, a partir de 512 rayos por lote, ambas curvas comienzan a mostrar un incremento notable.

El punto más significativo en ambas curvas se encuentra alrededor de los 4096 rayos por lote, marcado como la configuración por defecto. En este punto, observamos un cambio notable en ambas métricas. Para la VRAM, se produce un cambio en la pendiente de la curva, pasando de un crecimiento gradual a uno más pronunciado. Simultáneamente, en el tiempo de entrenamiento, se observa el inicio de un aumento exponencial en el tiempo requerido.

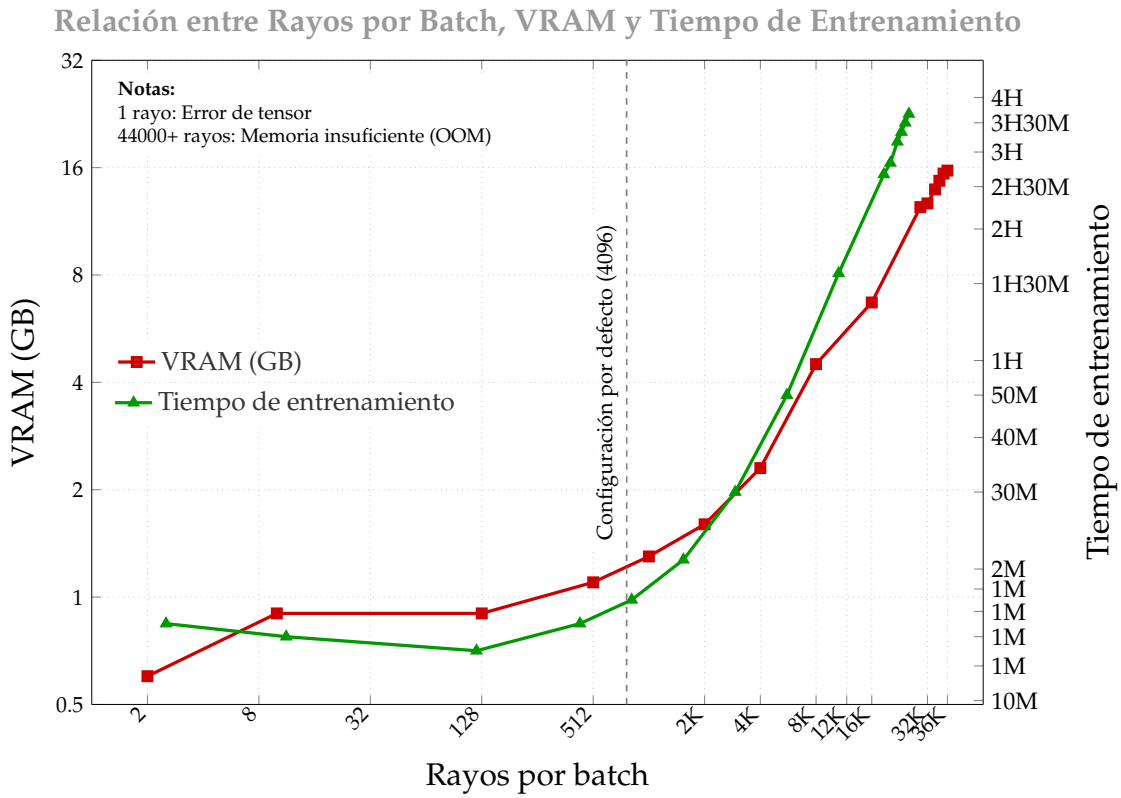

Figura 5.12: Análisis de la relación entre el número de rayos por batch, uso de VRAM y tiempo de entrenamiento

Este punto de 4096 rayos por lote parece representar un equilibrio óptimo entre el uso de recursos y la eficiencia del entrenamiento. Después de este punto, ambas métricas muestran un crecimiento acelerado, lo que sugiere una disminución en la eficiencia del sistema a medida que se aumenta el número de rayos por lote.

Es interesante notar que, aunque el tiempo de entrenamiento aumenta significativamente para valores altos de rayos por lote (16K-42K), el sistema alcanza su máxima eficiencia en términos de rayos procesados por segundo en este rango. Esto indica que el sistema está utilizando al máximo sus recursos de procesamiento y memoria, aunque a costa de tiempos de entrenamiento más largos.

Teniendo en cuenta esta gráfica combinada, podemos concluir que, para ambos valores (VRAM y tiempo de entrenamiento), el punto de inflexión más significativo y óptimo se encuentra en los 4096 rayos por lote. Este punto parece ofrecer el mejor equilibrio entre eficiencia de procesamiento, uso de recursos y tiempo de entrenamiento, lo que justifica su elección como configuración por defecto en NeRFstudio y su validez para elegir `Cloudstudio`.

Además, cabe destacar que el modelo con esta configuración aprovecha bien el resto de métricas mencionadas anteriormente, como el cómputo disponible, los checkpoints de escena terminada (que no superan los 190 MB) y la calidad visual, que es excelente una vez que se entrena la escena.

Este ajuste, junto con la activación del almacenamiento y la gestión de máscaras e imágenes en la memoria de la GPU (65), permitió aprovechar al máximo la memoria y la capacidad de cómputo de la GPU P100 que Kaggle proporciona.

(65) Comando: `-pipeline.datamanager.masks-on-gpu True -pipeline.datamanager.images-on-gpu True`

5.2.2. Pruebas con Gaussian Splatting

Para evaluar el rendimiento de Gaussian Splatting, se realizaron pruebas similares, pero menos exhaustivas, debido al uso de software en fases tempranas de desarrollo.

Se utilizaron GPU Tesla T4, ya que las GPUs que Kaggle asigna automáticamente al utilizar la API, como la P100, tienen un "Compute Capability" de 6.0, y Gaussian Splatting requiere una GPU con "Compute Capability" de 7.0 o superior.

Dado que la API de Kaggle no permite seleccionar esta GPU, se utilizará la interfaz web de Kaggle para ejecutar manualmente una libreta Jupyter. Este motivo fue el que impidió utilizar este método por defecto en la plataforma, ya que limitaba la automatización del inicio de la libreta Jupyter.

Así pues, se realizaron pruebas con «splatfacto», que es el modelo por defecto de NeRFstudio para Gaussian Splatting [10]. Como se puede ver en la figura 5.13, es un método NeRF que ofrece una mayor eficiencia en el entrenamiento y la renderización que «nerfacto». Los resultados obtenidos con Gaussian Splatting fueron prometedores, ya que mostraron un tiempo de entrenamiento reducido y una alta calidad visual incluso con pocos pasos.

Figura 5.13: Escena «poster» utilizando el método Gaussian Splatting. Como se puede ver, con unos pocos pasos se consigue una gran claridad visual. (66)

Para recopilar números y comparar «nerfacto» y el «splatfacto» cuantitativamente, se entrenó la misma escena. La escena «poster» se entrenó en 30 minutos con «nerfacto» y en 15 minutos con Gaussian Splatting, y mostró una calidad visual comparable.

La figura 5.13 muestra la escena «poster» en una fase inicial de entrenamiento sobre los 6000 pasos utilizando el método Gaussian Splatting. Los resultados obtenidos son prometedores, ya que presentan una mayor velocidad de convergencia, como se puede

(66) Escena entrenada en Kaggle utilizando T4x2 con el comando: `ns-train splatfacto -pipeline.datamanager.masks-on-gpu True -pipeline.datamanager.images-on-gpu True -viewer.make-share-url True -data data/nerfstudio/poster`

ver en la figura 5.14, y una reducción de artefactos visuales en comparación con Instant NGP y «nerfacto» con la misma cantidad de pasos.

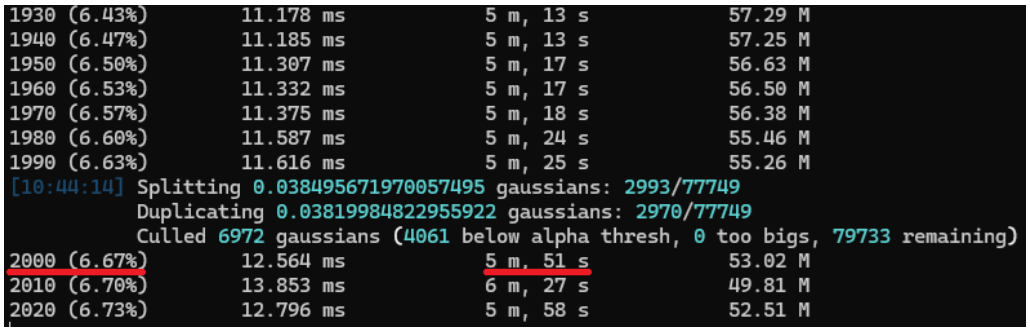

Figura 5.14: Escena «Poster» del terminal NeRFstudio entrenada con «splatfacto», que converge más rápido que nerfacto, como se puede ver en el número de pasos a los 5 minutos de entrenamiento. (67)

Además, con la misma cantidad de pasos se reconocen de forma más clara las siluetas y las formas. Además, en ambas escenas se puede aumentar la resolución hasta 2000x2000, lo que incrementa el tiempo de respuesta durante el entrenamiento a unos 2 segundos en el caso de nerfacto y a 0.7 segundos en splatfacto.

Característica	**Nerfacto**	**Gaussian Splatting**
Tiempo de entrenamiento	30 minutos	15 minutos
Velocidad de convergencia	60 ms de media	20 ms de media
Calidad visual inicial (<500)	Inutilizable	Baja
Calidad visual inicial (<2000)	Baja	Media
Calidad visual inicial (>6000)	Media	Alta
Tiempo de respuesta (2000x2000)	2 segundos	0.7 segundos
Tiempo de respuesta (Resolución automática)	0.5 segundos	0.5 segundos
Compatibilidad con Tesla P100	Sí	No
Capacidad de cómputo mínima requerida	<6.1	7.0+
Uso de memoria GPU	3 GB	2 GB

Tabla 5.2: Comparación de aspectos técnicos entre nerfacto y splatfacto en la escena «poster»

La tabla 5.2 muestra claramente las diferencias entre estos modelos al comparar su compatibilidad y su rendimiento. Cabe recordar que para evaluar el rendimiento de «splatfacto», se ejecutó la libreta Jupyter de forma manual en Kaggle, seleccionando una instancia con una GPU Tesla T4, que tiene una capacidad de cómputo de 7.5.

Estos resultados demuestran que la GPU T4 ofrece un rendimiento significativamente mejor en términos de tiempo de entrenamiento y velocidad de renderizado, además de una ligera mejora en la calidad de imagen. Sin embargo, es importante señalar que no es posible considerarlo, ya que no permite la automatización de la plataforma, lo que incumple el primer requisito de esta: la facilidad de uso.

(67) Argumento añadido para mejorar la legibilidad en terminal: `-logging.local-writer.max-log-size=0`

5.3 Flujo de Trabajo de la Plataforma Final

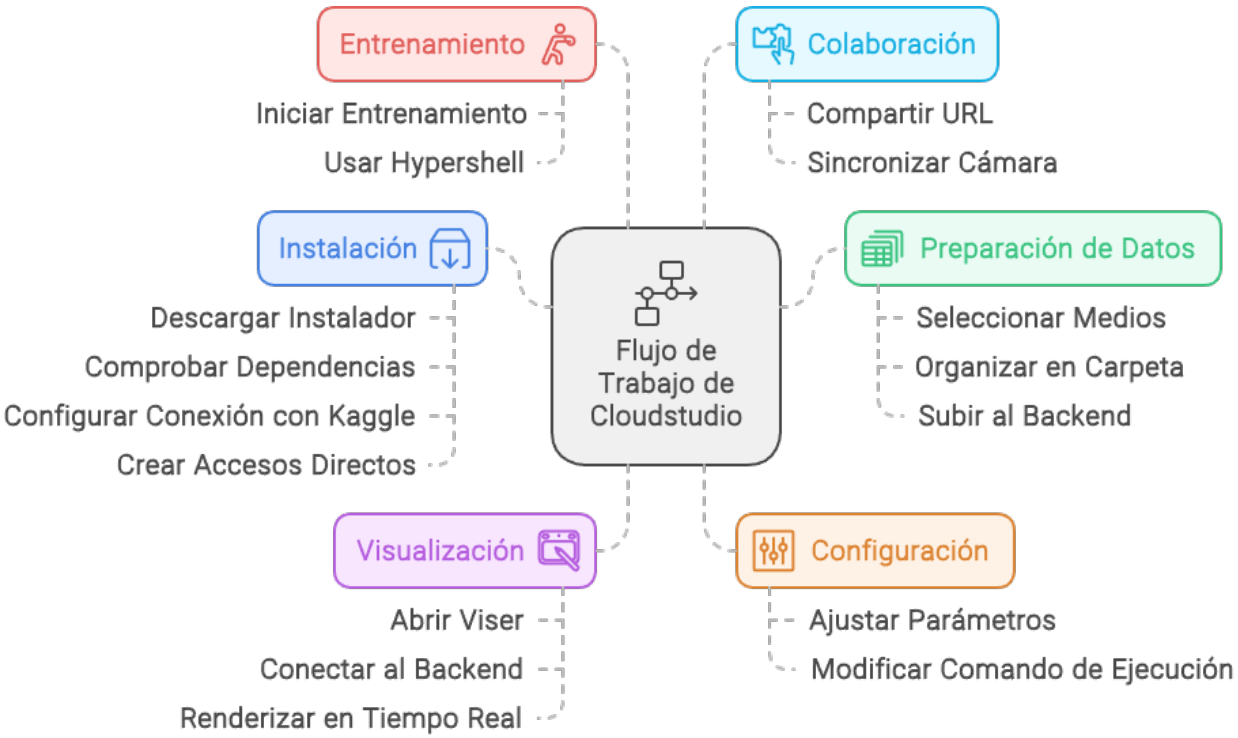

Figura 5.15: Diagrama del flujo de trabajo de *Cloudstudio*, mostrando las etapas de instalación, preparación de datos, configuración, entrenamiento, visualización y colaboración (68)

El flujo de trabajo de la plataforma *Cloudstudio* (figura 5.15) en su implementación final se resume en los siguientes pasos:

1. **Instalación:** el usuario descarga el instalador desde el repositorio del TFG y lo ejecuta en su sistema. El instalador se encarga de comprobar las dependencias necesarias, como el archivo kaggle.json, y ayuda al usuario a posicionarlo en la ruta correcta. A continuación, configura la conexión con Kaggle automáticamente, crea las claves y las guarda en el archivo de configuración.

2. **Preparación de los datos de entrada:** el usuario selecciona los vídeos o imágenes de la escena 3D que desea entrenar y los coloca en una carpeta a la que puede acceder la plataforma. Después, el instalador pone a disposición del usuario un comando que puede subir los archivos para que el backend disponga de ellos.

3. **Configuración:** el usuario puede ajustar los parámetros del modelo NeRF modificando los comandos predefinidos o escribiendo el comando con los argumentos que necesite (opcional).

4. **Entrenamiento:** el usuario inicia el entrenamiento de la escena NeRF utilizando un comando predefinido en la interfaz del instalador, con argumentos establecidos para facilitar y optimizar el entrenamiento del modelo NeRF. El instalador envía el comando al backend Flask en la nube para iniciar el entrenamiento, utilizando el túnel Hypershell.

5. **Visualización:** el usuario abre el visor Viser en su navegador web utilizando la URL que el instalador lee de la salida de la terminal de NeRFstudio y facilita al usuario.

(68) Gráfico plantilla que muestra los pasos de cada componente de la plataforma

El visor se conecta al backend en la nube y muestra la escena NeRF renderizada en tiempo real.

6. **Colaboración:** otros usuarios pueden unirse a la sesión de visualización abriendo la URL del visor en sus navegadores web sin necesidad de utilizar el instalador. Los usuarios pueden colaborar en la exploración de la escena NeRF, utilizando la funcionalidad de sincronización de la cámara. Además, ambos pueden elegir si quieren controlar la cámara, hasta un máximo de 32 clientes.

5.4 Resultados Cualitativos

Para evaluar la usabilidad y la experiencia de usuario de la plataforma *Cloudstudio*, se realizó un estudio cualitativo con un grupo de 5 personas con una edad media de 30 años. Los participantes probaron el instalador y el visor, interactuando indirectamente con el backend. Después de la prueba, se les pidió que completaran un cuestionario relacionado con los requisitos y especificaciones establecidos en el capítulo 3. Cabe destacar que previamente recibieron unas pequeñas instrucciones sobre la pestaña principal del instalador.

5.4.1. Feedback de los usuarios

Las principales observaciones y comentarios de los usuarios se pueden resumir en los siguientes puntos:

- **Facilidad de uso:** los usuarios destacaron la simplicidad del proceso de instalación y configuración y en ningún momento se sintieron perdidos a lo largo del proceso. Uno de ellos mencionó que había demasiados botones en la parte inferior del instalador.

  Palabras clave mencionadas: ”intuitivo”, ”sencillo”, ”simple”, ”fácil de usar”,”directo”.

- **Rendimiento:** la mayoría de los usuarios quedaron impresionados con la fluidez de la interacción con las escenas 3D. Además, una vez explicado todo el proceso de entrenamiento de una escena NeRF, los 30 minutos que se tarda en entrenar la escena les parecieron razonables.

  De uno de los usuarios en esta pregunta se sacó la información necesaria para no utilizar la barra de progreso desde la ejecución de la libreta hasta su conexión, ya que la barra de progreso daba la impresión de un proceso corto.

  Palabras clave: ”fluido”, ”sin retrasos notables”, ”ajustado”, ”comprensible”,”eficaz”.

- **Calidad visual:** los participantes elogiaron la calidad de las escenas renderizadas una vez entrenadas. Algunos de ellos especificaron que la escena se veía borrosa, lo cual era cierto dado que el visor había comenzado cuando la escena llevaba pocas iteraciones.

  Palabras clave: ”claro”, ”detallado”, ”borroso”,”realista”,”práctico”.

- **Colaboración:** la función de sincronización de cámara fue bien recibida, aunque algunos usuarios sugirieron mejoras en la interfaz para hacerla más evidente.

  Palabras clave: ”útil”, ”innovador”, ”potencial para trabajo en equipo”, ”divertida”, ”práctica”.

- **Automatización:** los usuarios apreciaron no tener que hacer nada más que esperar y pulsar clics. Además, no entendían del todo las tareas complejas, como la configuración del entorno y la conexión con el backend, y además consideraban que el sistema no ofrecía suficiente feedback.

  Palabras clave: ”poco feedback”, ”todo en uno”, ”poco indicativo”, ”directo”, ”automático”.

- **Interfaz gráfica:** los usuarios apreciaron que la letra de la interfaz fuera grande y que estuviera traducida. Además, especificaron que la idea de que la ventana fuera pequeña y no ocupara mucho tenía mucho sentido, ya que no era la parte principal.

  Palabras clave: ”simple”, ”consciente”, ”de un vistazo”, ”útil”, ”compacta”.

La tabla 5.3 muestra la frase más representativa de cada apartado.

Aspecto	**Comentario representativo**
Facilidad de uso	”La interfaz es intuitiva y simple.”
Rendimiento	”Es muy interesante moverse por la escena en tiempo real.”
Calidad visual	”Esto tiene que ser súper útil para ver pisos.”
Colaboración	”La sincronización de cámara es muy divertida.”
Automatización	”No he tenido que hacer casi nada para ver la escena.”

Tabla 5.3: Comentarios representativos de los usuarios

5.4.2. Áreas de mejora identificadas

A partir del feedback de los usuarios, se identificaron algunas áreas de mejora:

- **Documentación:** algunos usuarios sugirieron que la explicación previa también estuviera disponible por escrito. También se mencionó la existencia de algún tutorial o feedback más explicativo dentro de la aplicación para explicar conceptos avanzados.

- **Opciones de personalización:** se solicitó más flexibilidad en la configuración de parámetros del modelo NeRF para usuarios avanzados. También se pidió algún menú de personalizaron de la interfaz.

- **Integración:** se sugirió unificar la interfaz de Viser y la del instalador en una sola, o, por lo menos, que se pudiera hacer todo con el navegador web.

De estas ideas ya hay una implementación planteada como trabajo futuro que debería solucionarlas todas y que se basa en «Gradio». Al estar basada en «Gradio», dispone de amplia documentación, una gran personalización de la interfaz, se puede utilizar en el navegador web y tiene una mejor integración con NeRFtudio.

5.4.3. Cambios implementados

En respuesta al feedback de los usuarios, se realizaron las siguientes mejoras en la plataforma:

- Se implementó un sistema de notificaciones en el visor para informar a los usuarios sobre eventos importantes. Además, existe un apartado en los comandos predefinidos que proporciona información sobre los modelos de NeRFstudio. Hubo procesos que, igualmente, eran confusos en los mensajes, así que se ocultaron y automatizaron por completo.
- Se implementaron métodos más avanzados en el instalador que permiten a los usuarios más curiosos experimentar y ajustar parámetros adicionales de diferentes modelos NeRF.
- Se ha rediseñado la interfaz del instalador para mostrar de forma más clara los comandos predefinidos, incluyendo un desplegable que ayuda a no atosigar al usuario.
- Se añadió una barra de progreso en el instalador para proporcionar información al usuario, mostrar el avance del entrenamiento y el estado de la conexión con el backend.

Para terminar con los resultados cualitativos vamos a cuantificarlos con notas del 1 al 10 separadas por apartado.

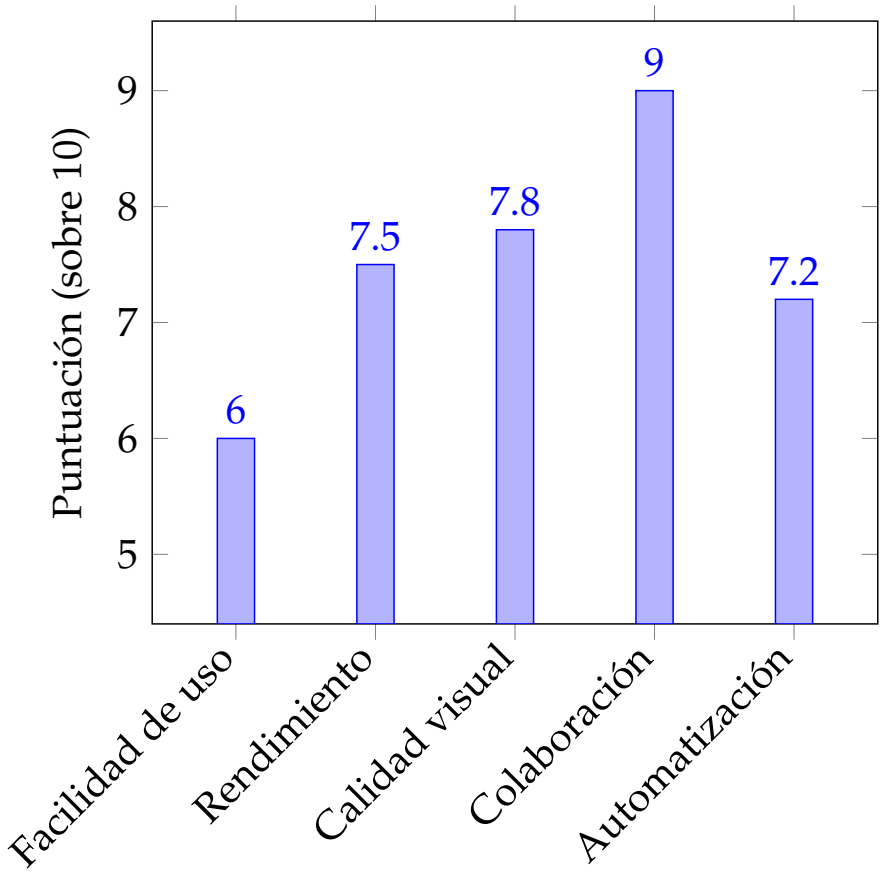

Figura 5.16: Satisfacción de usuarios por apartados antes de exponer si feedback e implementar las mejoras

En la gráfica 5.16 se pueden ver las medias sobre 10 para cada apartado entre el total de usuarios con la primera implementación funcional de la plataforma. La nota media de todos los apartados es un 7.5. Se puede ver cómo los puntos débiles de la plataforma son la automatización y la facilidad de uso.

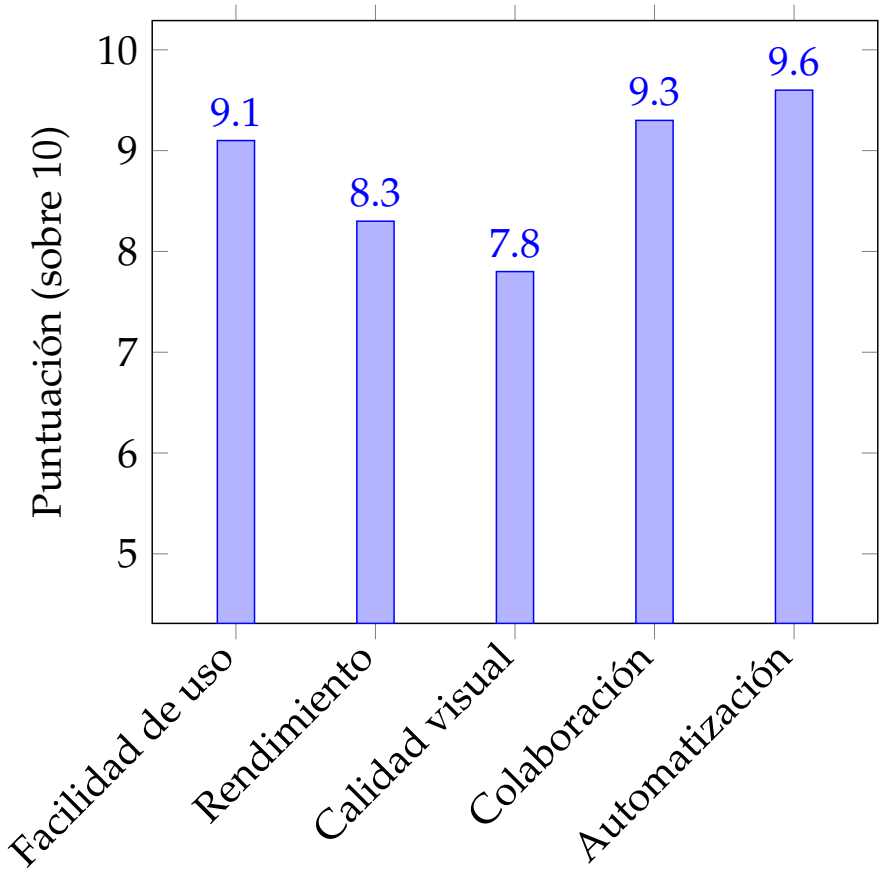

Figura 5.17: Satisfacción de usuarios por aspecto

En la gráfica 5.17 se muestran las puntuaciones medias sobre 10 para cada apartado, calculadas a partir del total de usuarios. La nota media de todos los apartados es de 8.82, lo que representa una mejora significativa en comparación con los datos previos recopilados. Estas puntuaciones se obtuvieron después de implementar los cambios pertinentes basados en el feedback de los usuarios. A pesar de la mejora general, algunos resultados aún necesitan mejorar considerablemente.

Estos resultados cualitativos y las mejoras implementadas refuerzan las fortalezas de la plataforma identificadas anteriormente, como la facilidad de uso, el alto rendimiento y la colaboración multiusuario. Al mismo tiempo, abordan algunas de las debilidades, como la necesidad de más opciones de personalización para usuarios avanzados y la mejora de la interfaz de usuario.

5.5 Fortalezas

Según la información recibida en las pruebas cualitativas y cuantitativas, las principales fortalezas de la plataforma `Cloudstudio` son:

1. **Facilidad de uso:** la plataforma abstrae la complejidad técnica del usuario y ofrece una interfaz intuitiva y sencilla. El flujo de trabajo que implementa permite no tener que interactuar con código ni con la terminal.

2. **Alto rendimiento:** la ejecución del backend en la nube, que utiliza una GPU potente, permite un entrenamiento y una renderización eficientes de las escenas NeRF. El

tiempo de entrenamiento es de alrededor de 30 minutos para escenas interiores de tamaño medio.

3. **Colaboración multiusuario:** la sincronización de la cámara del visor facilita la colaboración entre usuarios a la hora de explorar la escena. Esto permite que un usuario guíe a otro alrededor de la escena, lo que resulta de gran utilidad para dirigir a alguien a través de un espacio.

4. **Automatización:** el instalador automatiza la configuración del entorno, la conexión con Kaggle, el inicio del backend y otras tareas, minimizando la intervención del usuario. Este simplemente debe descargar el instalador, conseguir la clave de la API de Kaggle y ver un vídeo de la escena.

5. **Seguridad extremo a extremo:** Hypershell implementa un protocolo de cifrado de extremo a extremo, asegurando que todas las comunicaciones entre el cliente y el backend en la nube sean completamente seguras y privadas.

   La autenticación segura y el protocolo peer-to-peer de Hypershell minimizan los riesgos de seguridad, haciendo de *Cloudstudio* una plataforma altamente segura para el entrenamiento y la visualización de escenas 3D.

5.6 Debilidades

De acuerdo a nuestras pruebas, las principales debilidades detectadas en la plataforma `Cloudstudio` implementada son:

1. **Limitaciones de la API de Kaggle:** La API de Kaggle presenta limitaciones significativas. La más importante, además de la escasa flexibilidad que ofrece, es la falta de un comando que permita apagar la libreta Jupyter. Por tanto, en caso de problemas o si se ha terminado de visualizar la escena, la libreta Jupyter solo se puede apagar entrando en Kaggle y seleccionando «Stop Session», como se muestra en la figura 5.18.

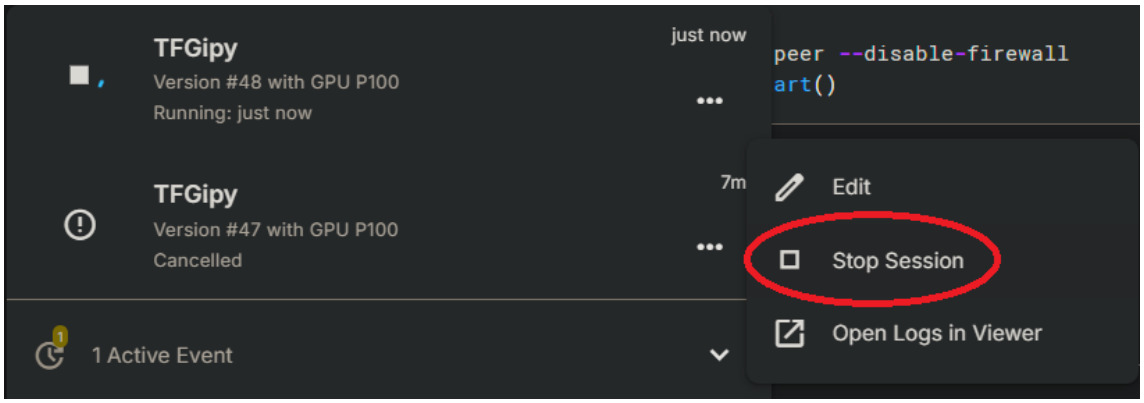

Figura 5.18: El círculo rojo muestra la única forma de apagar las libretas de Kaggle, «Stop Session», ya que no permite hacerlo mediante la API

2. **Limitación en la selección de la GPU:** La API de Kaggle no permite seleccionar la GPU que se utilizará al ejecutar una libreta Jupyter de forma remota. Esto limita la posibilidad de utilizar métodos NeRF que requieren una GPU con una capacidad de cómputo específica, como Gaussian Splatting.

3. **Dependencia de la API de Kaggle:** La plataforma depende de la API de Kaggle para la ejecución del backend, lo que puede afectar a su disponibilidad y rendimiento en caso de fallos o cambios en la API.

4. **Dependencia de paquetes externos:** la plataforma depende de la disponibilidad de los paquetes de Python y de las ruedas precompiladas de las librerías utilizadas. Si alguno de estos paquetes no estuviera disponible o no se pudiera descargar, la instalación del backend fallaría.

5. **Posibles problemas de conexión con Hypershell:** Hypershell puede tener dificultades para establecer la conexión en redes con firewalls que aplican restricciones de UDP o tienen una configuración de red compleja al utilizar un protocolo P2P.

CAPÍTULO 6
Conclusiones

La plataforma *Cloudstudio*, desarrollada en este TFG, ha demostrado ser una solución innovadora y efectiva para democratizar el acceso a la tecnología de campos de radiancia neuronal. Al cumplir con los objetivos, requisitos y especificaciones establecidos en este TFG, *Cloudstudio* ofrece una herramienta accesible que simplifica significativamente el proceso de entrenamiento, renderización y visualización de escenas 3D realistas. La combinación de facilidad de uso, alto rendimiento y capacidades de colaboración posiciona a *Cloudstudio* como una herramienta valiosa en el campo de la creación y exploración de contenido 3D.

Cloudstudio representa un avance significativo en la democratización de la tecnología NeRF. Al simplificar el proceso de entrenamiento y visualización, la plataforma hace que esta tecnología avanzada sea accesible para usuarios sin conocimientos técnicos profundos en aprendizaje profundo o gráficos por computadora. Esto podría acelerar la adopción de NeRF en diversos campos como arquitectura, diseño de productos, entretenimiento y educación.

La facilidad de uso de *Cloudstudio* abre la puerta a la innovación interdisciplinar. Los profesionales de campos no técnicos pueden ahora experimentar con la creación de escenas 3D realistas, lo que podría dar lugar a aplicaciones novedosas en áreas como el arte digital, la conservación del patrimonio cultural o la planificación urbana. Además, se presenta como un buen punto de partida para la formación en campos relacionados con la visualización 3D.

Como fieles defensores del código abierto, todo lo utilizado en este TFG ha sido de código abierto, al igual que el legado de *Cloudstudio*, que queda como código abierto y su legado durará todo lo que la comunidad desee. Además, *Cloudstudio* no solo se beneficia de la comunidad de desarrollo existente, sino que también contribuye a ella. Esto fomenta un ciclo virtuoso de innovación y mejora continua en el campo de la representación 3D basada en IA.

A pesar de las ventajas de *Cloudstudio*, existen desventajas, principalmente relacionadas con la API de Kaggle y la poca granularidad y control que ofrece. Esto limita las posibilidades que se podrían haber implementado en la plataforma. Sin embargo, es importante destacar que prestan GPUs de forma gratuita y que este TFG no habría sido posible sin ello. La dependencia de paquetes externos y la posible inestabilidad de la conexión con Hypershell en redes restrictivas también son aspectos que se deben mejorar en el futuro.

El desarrollo de *Cloudstudio* ha demostrado el potencial de la tecnología NeRF para la creación de escenas 3D realistas y la democratización del acceso a esta tecnología. La plataforma, al simplificar el proceso de entrenamiento, renderización y visualización de escenas NeRF, facilita la creación de contenido 3D y la colaboración entre usuarios,

abriendo nuevas posibilidades para la investigación, la educación, el entretenimiento y otras áreas.

6.1 Trabajo futuro

Existen varias líneas de trabajo futuro para mejorar la plataforma *Cloudstudio*, incluyendo:

- **Integración de herramientas de segmentación y edición 3D:** Aunque la plataforma ya integra herramientas como LERF [32] y Instruct-NeRF2NeRF [33] que permiten segmentar semánticamente la escena y editarla, respectivamente, una mejor integración con herramientas de segmentación panóptica y edición 3D en Cloudstudio permitiría a los usuarios entender e interactuar con la escena y sus objetos, y les daría la posibilidad de realizar modificaciones y crear contenido 3D más complejo.
- **Integración de Gaussian Splatting:** Se explorará la posibilidad de integrar Gaussian Splatting en la plataforma, utilizando un servicio en la nube que permita la selección de la GPU o desarrollando un mecanismo para ejecutar Gaussian Splatting en la GPU Tesla P100 de Kaggle.
- **Evaluación de servicios en la nube alternativos:** Se evaluarán servicios en la nube alternativos a Kaggle que ofrezcan una mayor flexibilidad en la selección de la GPU, una API más robusta y la posibilidad de interactuar con las libretas Jupyter en tiempo real.
- **Adaptación a realidad virtual:** Se investigará la viabilidad de integrar *Cloudstudio* en entornos de realidad virtual al igual que lo hace «Inmersive NGP» [22] con «Instant NGP», lo que permitiría a los usuarios experimentar las escenas NeRF de forma inmersiva.
- **Implementación en Gradio:** Se explorará la posibilidad de implementar la interfaz de usuario de *Cloudstudio* utilizando Gradio, una librería de Python para crear interfaces web interactivas para modelos de aprendizaje automático. Esta implementación permitiría unificar aún más la plataforma, ofreciendo una interfaz web completa que integre tanto el control del backend como la visualización de las escenas NeRF. Gradio facilitaría la creación de una interfaz más intuitiva y personalizable, mejorando la experiencia del usuario y simplificando el flujo de trabajo.
- **Optimización del rendimiento:** Se investigarán técnicas avanzadas de optimización para reducir aún más el tiempo de entrenamiento y mejorar la calidad de renderización en tiempo real. Esto podría incluir la implementación de técnicas de pruning (véase [34]) de redes neuronales, la exploración de arquitecturas NeRF más eficientes y la optimización de los algoritmos de renderización como Bilagrid [35].

Cloudstudio es un paso en la dirección correcta hacia la democratización del acceso a la tecnología NeRF, ya que facilita la creación de contenido 3D realista para un público más amplio. Las futuras mejoras y expansiones de la plataforma tienen el potencial de convertirla en una herramienta aún más poderosa y versátil para la creación, exploración y manipulación de escenas 3D, y de impulsar la innovación en diferentes ámbitos.

Cloudstudio no finaliza aquí ni su implementación ni su estudio, y se presenta públicamente como una plataforma que pone en manos de todos el entrenamiento, la optimización y el renderizado de escenas 3D mediante técnicas de aprendizaje profundo.

Bibliografía

[1] Ayush Tewari, Ohad Fried, Justus Thies, Vincent Sitzmann, Stephen Lombardi, Kalyan Sunkavalli, Ricardo Martin-Brualla, Tomas Simon, Jason Saragih, Matthias Nießner et al. (2022). «Advances in Neural Rendering». *Computer Graphics Forum*, 41(2): 703-735. DOI: 10.1111/cgf.14507.

[2] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi y Ren Ng. (2020). «NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis». *Communications of the ACM*, 65(1): 99-106. DOI: 10.1145/3503250.

[3] Xi Wang, Robin Courant, Jinglei Shi, Eric Marchand y Marc Christie. (2023). «JAWS: Just A Wild Shot for Cinematic Transfer in Neural Radiance Fields». *CVPR*, 23(1): 99-106. DOI: 10.1109/CVPR52729.2023.01624.

[4] Clément Jambon, Bernhard Kerbl, Georgios Kopanas, Stavros Diolatzis, Thomas Leimkühler y George Drettakis. (2023). «NeRFshop: Interactive Editing of Neural Radiance Fields». *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 6(1). DOI: 10.1145/3585499.

[5] María de la Cinta Sánchez Prades. (2016). «Prueba piloto : mantener la estimulación cognitiva de adultos y mayores dependientes en su hogar mediante actividades adaptadas y con apoyo de sus cuidadores». *Trabajo Fin de Máster. Tortosa: UNED*. URL: http://hdl.handle.net/11162/171076.

[6] Jennifer Sweetman, Peter Knapp, Danielle Varley, Rebecca Woodhouse, Dean McMillan y Peter Coventry. (2021). «Barriers to attending initial psychological therapy service appointments for common mental health problems: A mixed-methods systematic review». *Journal of affective disorders*, 284(1): 44-63. DOI: 10.1016/j.jad.2021.01.089.

[7] *Jawset Postshot*. End-to-end software for Radiance Fields. Designed and optimized for production. Easily create photorealistic 3D scenes and objects with any camera in minutes. URL: https://www.jawset.com/ (visitado 29-06-2024).

[8] Massimiliano Pepe, Vincenzo Saverio Alfio y Domenica Costantino. (2023). «Assessment of 3D Model for Photogrammetric Purposes Using AI Tools Based on NeRF Algorithm». *Heritage*, 6(8): 5719-5731. DOI: 10.3390/heritage6080301.

[9] Matthew Tancik, Ethan Weber, Evonne Ng, Ruilong Li, Brent Yi, Justin Kerr, Terrance Wang, Alexander Kristoffersen, Jake Austin, Kamyar Salahi, Abhik Ahuja, David McAllister y Angjoo Kanazawa. (2023). «Nerfstudio: A Modular Framework for Neural Radiance Field Development». *SIGGRAPH '23*, 72. DOI: 10.1145/3588432.3591516.

[10] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler y George Drettakis. (2023). «3D Gaussian Splatting for Real-Time Radiance Field Rendering». *ACM Trans. Graph.*, 42(4): 1-139. DOI: 10.1145/3592433.

[11] Noah Snavely, Steven M. Seitz y Richard Szeliski. «Photo tourism: exploring photo collections in 3D». En: *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*. 1.ª ed. Association for Computing Machinery, 2023. DOI: 10.1145/3596711.3596766.

[12] Kourosh Khoshelham y Sander Oude Elberink. (2012). «Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications». *Sensors*, 12(2): 1437-1454. DOI: 10.3390/s120201437.

[13] Seamus Coveney y A. Stewart Fotheringham. (2011). «Terrestrial laser scan error in the presence of dense ground vegetation». *The Photogrammetric Record*, 26(135): 307-324. DOI: https://doi.org/10.1111/j.1477-9730.2011.00647.x.

[14] Dominik Zimny, Artur Kasymov, Adam Kania, Jacek Tabor, Maciej Zięba y Przemysław Spurek. (2023). «MultiPlaneNeRF: Neural Radiance Field with Non-Trainable Representation». *arXiv*. eprint: 2305.10579.

[15] Ricardo Martin-Brualla, Noha Radwan, Mehdi S. M. Sajjadi, Jonathan T. Barron, Alexey Dosovitskiy y Daniel Duckworth. «NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections». En: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021, 7206-7215. DOI: 10.1109/CVPR46437.2021.00713.

[16] Thomas Müller, Alex Evans, Christoph Schied y Alexander Keller. (2022). «Instant Neural Graphics Primitives with a Multiresolution Hash Encoding». *ACM Trans. Graph.*, 41(4): 102:1-102:15. DOI: 10.1145/3528223.3530127.

[17] Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla y Pratul P Srinivasan. (2021). «Mip-NeRF: A Multiscale Representation for Anti-Aliasing Neural Radiance Fields». *IEEE/CVF international conference on computer vision*: 5835-5844. DOI: 10.1109/ICCV48922.2021.00580.

[18] Zirui Wang, Shangzhe Wu, Weidi Xie, Min Chen y Victor Adrian Prisacariu. (2021). «NeRF–: Neural Radiance Fields Without Known Camera Parameters». *arXiv preprint*. arXiv: 2102.07064.

[19] Dor Verbin, Peter Hedman, Ben Mildenhall, Todd Zickler, Jonathan T. Barron y Pratul P. Srinivasan. (2022). «Ref-NeRF: Structured View-Dependent Appearance for Neural Radiance Fields». *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*: 5481-5490. DOI: 10.1109/CVPR52688.2022.00541.

[20] Abhijit Kundu, Kyle Genova, Xiaoqi Yin, Alireza Fathi, Caroline Pantofaru, Leonidas Guibas, Andrea Tagliasacchi, Frank Dellaert y Thomas Funkhouser. (2022). «Panoptic Neural Fields: A Semantic Object-Aware Neural Scene Representation». *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*: 12861--12871. DOI: 10.1109/CVPR52688.2022.01253.

[21] Steven Liu, Xiuming Zhang, Zhoutong Zhang, Richard Zhang, Jun-Yan Zhu y Bryan Russell. (2021). «Editing Conditional Radiance Fields». *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*: 5753-5763. DOI: 10.1109/ICCV48922.2021.00572.

[22] Ke Li, Tim Rolff, Susanne Schmidt, Reinhard Bacher, Simone Frintrop, Wim P. Leemans y Frank Steinicke. (2022). «Immersive Neural Graphics Primitives». *ArXiv*, abs/2211.13494. arXiv: 2211.13494.

[23] Kaiyun Yang, Yunqi Cheng, Zonghai Chen y Jikai Wang. (2024). «SLAM Meets NeRF: A Survey of Implicit SLAM Methods». *World Electric Vehicle Journal*, 15(3): 85. DOI: 10.3390/wevj15030085.

[24] G. Mazzacca, A. Karami, S. Rigon, E. M. Farella, P. Trybala y F. Remondino. (2023). «NeRF for Heritage 3D Reconstruction». *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLVIII-M-2-2023: 1051-1058. DOI: 10.5194/isprs-archives-XLVIII-M-2-2023-1051-2023.

[25] Saeejith Nair, Yuhao Chen, Mohammad Javad Shafiee y Alexander Wong. (2023). «NAS-NeRF: Generative Neural Architecture Search for Neural Radiance Fields». *ArXiv*, abs/2309.14293. arXiv: 2309.14293.

[26] Christian Reiser, Songyou Peng, Yiyi Liao y Andreas Geiger. (2021). «KiloNeRF: Speeding up Neural Radiance Fields with Thousands of Tiny MLPs». *ICCV*: 14315--14325. DOI: 10.1109/ICCV48922.2021.01407.

[27] Haithem Turki, Deva Ramanan y Mahadev Satyanarayanan. (2021). «Mega-NeRF: Scalable Construction of Large-Scale NeRFs for Virtual Fly-Throughs». *CVPR 2022*: 12912-12921. DOI: 10.1109/CVPR52688.2022.01258.

[28] Deborah Levy, Amit Peleg, Naama Pearl, Dan Rosenbaum, Derya Akkaynak, Simon Korman y Tali Treibitz. (2023). «SeaThru-NeRF: Neural Radiance Fields in Scattering Media». *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*: 56-65.

[29] Thomas Müller, Fabrice Rousselle, Jan Novák y Alexander Keller. (2021). «Real-time Neural Radiance Caching for Path Tracing». *ACM Trans. Graph.*, 40(4): 36. DOI: 10.1145/3450626.3459812.

[30] *Juice Community Version*. Juice is GPU-over-IP: client/server software that allows GPUs to be used over a standard TCP/IP network. Run the Juice server on a machine with a physical GPU, and then immediately access that GPU from any machine with the Juice client software. URL: https://github.com/Juice-Labs/Juice-Labs (visitado 08-08-2024).

[31] Sara Fridovich-Keil, Giacomo Meanti, Frederik Rahbæk Warburg, Benjamin Recht y Angjoo Kanazawa. «K-Planes: Explicit Radiance Fields in Space, Time, and Appearance». En: *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2023, 12479-12488. DOI: 10.1109/CVPR52729.2023.01201.

[32] Justin Kerr, Chung Min Kim, Ken Goldberg, Angjoo Kanazawa y Matthew Tancik. «LERF: Language Embedded Radiance Fields». En: *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2023, 19672-19682. DOI: 10.1109/ICCV51070.2023.01807.

[33] Ayaan Haque, Matthew Tancik, Alexei A. Efros, Aleksander Holynski y Angjoo Kanazawa. «Instruct-NeRF2NeRF: Editing 3D Scenes with Instructions». En: *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2023, 19683-19693. DOI: 10.1109/ICCV51070.2023.01808.

[34] Víctor Vivancos Serrano. (2023). «Estudio de los efectos de métodos de pruning y dispersión de redes neuronales preentrenadas». *B.S. thesis. Universitat Politècnica de Catalunya*.

[35] Jiawen Chen, Sylvain Paris y Frédo Durand. (2007). «Real-time edge-aware image processing with the bilateral grid». *ACM Trans. Graph.*, 26(3): 103. DOI: 10.1145/1276377.1276506.

APÉNDICE A
Anexo I: Objetivos de Desarrollo Sostenible

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenible	**Alto**	**Medio**	**Bajo**	**No procede**
ODS 1. Fin de la pobreza.				✓
ODS 2. Hambre cero.				✓
ODS 3. Salud y bienestar.			✓	
ODS 4. Educación de calidad.		✓		
ODS 5. Igualdad de género.				✓
ODS 6. Agua limpia y saneamiento.			✓	
ODS 7. Energía asequible y no contaminante.			✓	
ODS 8. Trabajo decente y crecimiento económico.		✓		
ODS 9. Industria, innovación e infraestructuras.	✓			
ODS 10. Reducción de las desigualdades.			✓	
ODS 11. Ciudades y comunidades sostenibles.	✓			
ODS 12. Producción y consumo responsables.		✓		
ODS 13. Acción por el clima.		✓		
ODS 14. Vida submarina.				✓
ODS 15. Vida de ecosistemas terrestres.				✓
ODS 16. Paz, justicia e instituciones sólidas.			✓	
ODS 17. Alianzas para lograr objetivos.		✓		

Este trabajo de fin de grado se centra en el desarrollo de la plataforma *Cloudstudio* que facilita el entrenamiento, renderizado y visualización de escenas 3D mediante campos de radiancia neuronal. Si bien la naturaleza del trabajo es principalmente tecnológica, se puede relacionar con varios Objetivos de Desarrollo Sostenible de forma directa o indirecta.

ODS 4: Educación de calidad *Cloudstudio* puede facilitar el acceso a la educación en áreas como la visualización 3D y los gráficos computacionales. La plataforma ofrece una herramienta educativa que permite a los estudiantes explorar conceptos complejos de forma visual e interactiva.

- **Impacto:** Al permitir a los estudiantes experimentar con la tecnología NeRF, *Cloudstudio* puede contribuir a la formación de profesionales más capacitados en el ámbito de la tecnología, la innovación y el diseño.

ODS 9: Industria, innovación e infraestructuras La plataforma *Cloudstudio* refleja la esencia de la innovación tecnológica, particularmente en el campo de la inteligencia artificial y los gráficos computacionales. Al utilizar tecnologías de vanguardia como NeRF y NeRFstudio, y centrarse en mejorar la eficiencia y accesibilidad de estas herramientas, *Cloudstudio* contribuye directamente al avance de la infraestructura digital y la innovación en la industria del 3D.

- **Impacto:** el potencial de *Cloudstudio* para impulsar el desarrollo de nuevas industrias y sectores es significativo. En áreas como la realidad virtual y aumentada, el diseño arquitectónico, la producción cinematográfica y la creación de contenido multimedia, *Cloudstudio* puede ser un factor de cambio. Al democratizar el acceso a la tecnología NeRF, la plataforma abre nuevas vías para la innovación, fomenta el crecimiento económico en estos sectores y potencialmente crea nuevas oportunidades de empleo en la economía digital.

ODS 11: Ciudades y comunidades sostenibles La combinación de la plataforma *Cloudstudio* con la tecnología NeRF puede ser útil para planificar y visualizar ciudades y comunidades de forma más eficiente. La tecnología NeRF permite crear modelos tridimensionales detallados de entornos urbanos, lo que facilita la planificación de infraestructuras, el diseño de espacios públicos y la evaluación del impacto ambiental de los proyectos.

- **Impacto:** La capacidad de visualizar y simular el desarrollo urbano de forma más precisa puede contribuir a la construcción de ciudades más sostenibles y resilientes.

ODS 17: Alianzas para lograr objetivos *Cloudstudio* se basa en tecnologías de código abierto como NeRFstudio, lo que fomenta la colaboración entre desarrolladores y la creación de una comunidad activa que comparte conocimientos y recursos. La colaboración entre usuarios y la posibilidad de trabajar en proyectos conjuntos utilizando la plataforma también contribuyen al logro de este objetivo.

- **Impacto:** La plataforma *Cloudstudio*, al ser un TFG de código abierto, promueve la colaboración y la cooperación entre diferentes actores, lo que puede contribuir a la solución de problemas globales y al logro de los ODS.

En resumen, aunque este trabajo de fin de grado se centra en el desarrollo de una plataforma tecnológica, *Cloudstudio* tiene el potencial de impactar en varios ODS. Su relación más directa es con el ODS 9 (Industria, innovación e infraestructuras), pero también se puede relacionar con el ODS 4 (Educación de calidad), el ODS 11 (Ciudades y comunidades sostenibles) y el ODS 17 (Alianzas para lograr objetivos). Al promover la innovación tecnológica y la colaboración, *Cloudstudio* puede contribuir a la construcción de un futuro más sostenible y equitativo.

APÉNDICE B
Glosario

Aceleración por hardware (GPU): Uso de unidades de procesamiento gráfico (GPUs) para acelerar tareas computacionalmente intensivas, como el entrenamiento de modelos NeRF.

Ley de Amdahl: Principio que establece que la mejora máxima en el rendimiento de un sistema debido a la paralelización está limitada por la fracción del código que no se puede paralelizar.

Backend: Componente del sistema que se ejecuta en segundo plano y se encarga de las tareas más complejas, como el entrenamiento, la renderización y la gestión de la comunicación con los clientes.

Cámara RGB-D: Cámara que captura imágenes en color (RGB) y datos de profundidad (D). Las cámaras RGB-D se utilizan en aplicaciones como la realidad virtual, para capturar información sobre el entorno.

Conda: Gestor de paquetes y entornos de Python. Conda permite instalar, gestionar y actualizar paquetes de Python y sus dependencias.

Conexión WebSocket: Protocolo de comunicación que permite la comunicación bidireccional y persistente de datos en tiempo real entre un cliente y un servidor.

Control de versiones: Los sistemas de control de versiones, como Git, permiten a los desarrolladores colaborar en el código fuente, revertir cambios y gestionar diferentes versiones del software.

CUDA: Plataforma de computación paralela desarrollada por NVIDIA. CUDA permite utilizar las GPUs para acelerar el entrenamiento de modelos de aprendizaje profundo.

Descenso de gradiente: Algoritmo de optimización utilizado para entrenar redes neuronales. El descenso de gradiente ajusta los parámetros de la red neuronal para minimizar la función de pérdida.

Función de activación: Función matemática utilizada en redes neuronales para introducir no linealidad en el modelo. Las funciones de activación permiten a las redes neuronales aprender relaciones no lineales entre los datos.

Función de pérdida: Función matemática utilizada para evaluar el rendimiento de un modelo de aprendizaje automático durante el entrenamiento. La función de pérdida mide la diferencia entre las predicciones del modelo y los valores reales.

Hiperparámetro: Parámetro de un modelo de aprendizaje automático que se ajusta antes del entrenamiento. Los hiperparámetros controlan el proceso de aprendizaje y afectan al rendimiento del modelo.

Kinect: Sensor de profundidad desarrollado por Microsoft. Kinect se utiliza en aplicaciones como la realidad virtual y la realidad aumentada, para capturar información sobre el entorno.

Latencia: Retardo entre una acción y su respuesta, se mide en milisegundos (ms). En nuestro caso, es similar al tiempo de respuesta al mover la cámara por la escena.

Libreta Jupyter: Entorno interactivo que permite ejecutar código Python y visualizar los resultados en un navegador web.

LiDAR: Sensor que utiliza luz láser para medir distancias y crear modelos 3D.

NAT (Network Address Translation): Técnica que permite compartir una dirección IP pública entre varios dispositivos en una red privada.

NeRF (Neural Radiance Fields): Técnica avanzada de representación de escenas 3D que utiliza redes neuronales para modelar la radiancia y la densidad volumétrica de una escena, permitiendo renderizaciones fotorrealistas desde cualquier ángulo de visión.

Node.js: Entorno de ejecución de JavaScript del lado del servidor.

Oclusión: Bloqueo de la vista de un objeto por otro. La oclusión es un fenómeno visual que se produce cuando un objeto opaco se interpone entre la cámara y otro objeto.

Peer-to-peer (P2P): Modelo de computación distribuida en el que los dispositivos se comunican directamente entre sí. En una red P2P, no hay un servidor centralizado.

PyTorch: librería de aprendizaje automático de Python. PyTorch se utiliza para desarrollar y entrenar modelos de aprendizaje profundo, con soporte para la aceleración por GPU.

Retropropagación: Algoritmo utilizado para entrenar redes neuronales. Permite ajustar los parámetros para minimizar la función de pérdida.

Segmentación panóptica: Técnica de visión artificial que combina la segmentación semántica y la detección de instancias para etiquetar todos los píxeles de una imagen.

Tasa de fotogramas (FPS): Número de imágenes que se muestran por segundo en una animación o vídeo. La tasa de fotogramas afecta a la fluidez y el realismo de la animación.

Tesla P100: Modelo de GPU desarrollado por NVIDIA. La Tesla P100 es una GPU de alto rendimiento diseñada para aplicaciones de aprendizaje profundo y computación científica.

Tesla T4: Modelo de GPU desarrollado por NVIDIA. La Tesla T4 es una GPU de bajo consumo diseñada para aplicaciones de inferencia de aprendizaje profundo y computación en la nube.

Token: Cadena de caracteres de tipo ed25519 que se utiliza para autenticar y autorizar el acceso a un sistema.

UDP holepunching: Técnica que permite a dos dispositivos detrás de NATs establecer una conexión P2P directa. UDP holepunching utiliza una serie de mensajes UDP para negociar una conexión directa entre los dispositivos.

Unidad lineal rectificada (ReLU): Función de activación utilizada en redes neuronales. La función ReLU devuelve 0 para valores negativos y el propio valor para valores positivos.

Virtualización de GPU: Creación de una versión virtual de una GPU. La virtualización permite ejecutar múltiples aplicaciones en un único servidor físico.

Vkcube: Aplicación de ejemplo que renderiza un cubo 3D utilizando Vulkan.