Trabajo de Fin de Grado

Control y Análisis de la Densificación en 3D Gaussian Splatting

TITULACIÓN: Grado en Ingeniería Informática

AUTOR: Acaymo Jesús Granado Sánchez

TUTORIZADO POR:
Adrián Peñate Sánchez y Jorge Bustos Sánchez

Junio 2025

Agradecimientos

A mi familia, en especial a mi abuela, a mis tíos, a mis padres y a mis hermanos, por ser siempre mi mayor fuente de apoyo, paciencia y motivación. Gracias por estar presentes en cada paso, y por enseñarme que lo verdaderamente importante no es de dónde partimos, sino hasta dónde somos capaces de llegar.

A mis tutores, Adrián Peñate Sánchez y Jorge Bustos Sánchez, por su guía, paciencia y dedicación a lo largo de este trabajo. Gracias por confiar en mis ideas, por orientarme cuando más lo necesitaba y por transmitirme la pasión por la investigación.

A la crew de Fortnite, por ser mucho más que un grupo de amigos. Por estar siempre, por las risas y por acompañarme en cada partida, dentro y fuera del juego.

Resumen

El control de densificación en tecnologías, como es 3D Gaussian Splatting (3DGS), es fundamental para lograr representaciones visuales precisas y eficientes de escenas que pueden llegar a ser muy complejas. Este proceso es bastante relevante, sobre todo en entornos de trabajo donde la calidad visual y la eficiencia computacional son críticas, como en simulaciones científicas, efectos visuales y videojuegos. Sin embargo, uno de los retos más significativos en 3DGS es la gestión eficiente de la densidad de los puntos en el espacio, lo cual puede afectar considerablemente la calidad del renderizado y el rendimiento del sistema.

Este proyecto de TFG aborda este problema mediante la propuesta de un nuevo enfoque para controlar la densificación de puntos en un modelo basado en parches utilizando Python. Este enfoque no solo busca reducir el costo computacional asociado con el procesamiento de grandes volúmenes de datos sino también mantener una calidad suficiente para su correcta manejabilidad.

A través de este trabajo, se propone modificar el entrenamiento de 3DGS para que sea basado en parches, además de la adición de técnicas que modifican la densidad de los puntos en función de criterios como la cantidad de gaussianas de la escena u otras propiedades de estas, como pueden ser su posición o su opacidad.

Una vez implementado el modelo, se diseñará una serie de experimentos para evaluar y comparar el rendimiento de la modificación propuesta frente al 3D Gaussian Splatting convencional. La comparativa se centrará en medidas cuantitativas como LPIPS, SSIM, PSNR, el número de gaussianas utilizadas y el tiempo de ejecución.

Este TFG juega un papel muy importante en diversos campos en donde la capacidad para generar imágenes de alta calidad de manera rápida y eficiente no solo optimiza los flujos de trabajo, sino que también eleva el nivel de la experiencia visual ofrecida a los usuarios finales.

Abstract

The control of densification in technologies such as 3D Gaussian Splatting (3DGS) is fundamental for achieving precise and efficient visual representations of scenes that can be highly complex. This process is especially relevant in work environments where visual quality and computational efficiency are critical, such as in scientific simulations, visual effects, and video games. However, one of the most significant challenges in 3DGS is the efficient management of point density in space, which can considerably affect rendering quality and system performance.

This final year project addresses this problem by proposing a new approach to control point densification in a patch-based model using Python. This approach not only aims to reduce the computational cost associated with processing large volumes of data but also to maintain sufficient quality for proper manageability.

Through this work, it is proposed to modify the training of 3DGS to be patch-based, in addition to incorporating techniques that adjust the density of the points based on criteria such as the number of gaussians in the scene or other properties of these, such as their position or opacity.

Once the model is implemented, a series of experiments will be designed to evaluate and compare the performance of the proposed modification against conventional 3D Gaussian Splatting. The comparison will focus on quantitative measures such as LPIPS, SSIM, PSNR, the number of gaussians used, and execution time.

This final year project plays a very important role in various fields where the ability to generate high-quality images quickly and efficiently not only optimizes workflows but also enhances the visual experience offered to end users.

Índice general

Índice de Algoritmos

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Contexto y Relevancia

Actualmente, la generación de escenas 3D ha experimentado un crecimiento significativo, respaldado por la alta demanda de industrias como el cine, el marketing y los videojuegos. Según un informe de The Brainy Insights, el mercado global de 3D Rendering fue valorado en 3.11 mil millones de USD en 2022 y se espera que alcance los 34.57 mil millones de USD para 2032, con una tasa de crecimiento anual compuesta del 27.67 % entre 2023 y 2032, demostrando el auge de las escenas 3D en los distintos ámbitos del desarrollo de escenas a nivel mundial [4].

1.2. 3D Gaussian Splatting (3DGS)

Dentro de este panorama, el modelo 3D Gaussian Splatting (3DGS)[5] destaca por su capacidad para reconstruir escenas 3D a partir de un conjunto de imágenes [5] . Mediante el uso de gaussianas para representar la información espacial, este método utiliza representaciones tridimensionales que se componen de tres distribuciones gaussianas en los ejes X, Y y Z. Estas gaussianas son controladas mediante parámetros como la varianza y la covarianza, que determinan su forma, y las medias, que definen sus posiciones en el espacio. Este control parametrizado permite generar nuevas imágenes desde ángulos no capturados origpinalmente.

1.3. Limitaciones de 3DGS

A pesar de las innovaciones que ofrece 3DGS[5], este modelo presenta importantes lagunas. La etapa de densificación, encargada de detallar las representaciones, implica un alto consumo de almacenamiento, memoria de la tarjeta gráfica y tiempo de entrenamiento, en parte debido a que se fundamenta en dos operaciones principales: la clonación de gaussianas, que se utiliza

para rellenar espacios vacíos y afinar el detalle de la escena, y la división de gaussianas de gran tamaño, que permite un refinamiento adicional. Este hecho supone una limitación en equipos convencionales, restringiendo así su accesibilidad de manera masiva.

1.4. Objetivos

Ante este panorama, se hace necesario implementar una metodología que optimice o controle el proceso de densificación. El objetivo principal es reducir el consumo de recursos (almacenamiento, memoria y tiempo de entrenamiento) sin comprometer la calidad de las escenas generadas. Esto no solo facilitará el acceso a la tecnología 3DGS[5] a un público más amplio, sino que también permitirá mejorar la eficiencia de sistemas en ámbitos tan exigentes como el cine, el marketing y el desarrollo de videojuegos.

1.5. Organización del documento

El resto del documento se estructura de la siguiente manera: en el **Capítulo 2** se detalla la planificación del proyecto, incluyendo las fases de desarrollo y los recursos necesarios. El **Capítulo 3** analiza el estado actual del tema y establece los objetivos iniciales de la investigación. En el **Capítulo 4** se exponen las aportaciones y novedades que se derivan del trabajo, así como las competencias superadas a lo largo del proyecto. El **Capítulo 5** se dedica al desarrollo, describiendo la metodología, las técnicas y el proceso de implementación de la propuesta. Posteriormente, en el **Capítulo 6** se presentan los resultados parciales obtenidos durante el desarrollo. El **Capítulo 7** recoge los resultados finales alcanzados. Finalmente, en el **Capítulo 8** se ofrecen las conclusiones finales y se plantean las líneas de trabajo futuro.

Capítulo 2

Planificación del trabajo

2.1. Metodología seguida

Para el desarrollo de este TFG se ha adoptado una metodología iterativa incremental, elegida por su enfoque modular y su capacidad para realizar mejoras continuas. Esta metodología permite desarrollar y refinar el proyecto en fases consecutivas, donde cada etapa se basa en los resultados de la anterior, facilitando la incorporación progresiva de mejoras en el modelo de 3DGS[5].

En cada iteracións se lleva a cabo una serie de tareas que abarcan desde el estudio de posibles funcionalidades hasta el análisis de resultados obtenidos a partir de la implementación de estas. En primera instancia, se experimentará con 3DGS[5] para un dataset con el objetivo de establecer un caso base que servirá para posteriores comparaciones y validaciones. A partir de esta fase se implementarán mejoras incrementales, probando distintos métodos y ajustando parámetros.

Además, cada iteración incluirá la evaluación de los resultados obtenidos, lo que permitirá identificar puntos de mejora y ajustar los parámetros necesarios para optimizar tanto el rendimiento como la calidad visual del modelo. De esta forma, el proyecto evoluciona de manera progresiva y segura.

2.2. Planificación Temporal

Cuadro 2.1: Planificación temporal del proyecto

Fases	Duración Estimada (horas)	Tareas
Estudio Preliminar / Análisis	40 horas	• **Tarea 1.1:** Revisión bibliográfica de métodos existentes en 3D Gaussian Splatting y técnicas de control de densificación. • **Tarea 1.2:** Análisis de herramientas y bibliotecas disponibles en Python[6] y PyTorch[7] que podrían utilizarse en el proyecto.
Diseño / Desarrollo / Implementación	180 horas	• **Tarea 2.1:** Implementación de lógica basada en parches para la fase de entrenamiento de 3DGS[5]. • **Tarea 2.2:** Medición de la calidad de los resultados con distintos números de parches usando la lógica implementada en la tarea 2.1. • **Tarea 2.3:** Implementación de un nuevo control de densificación basado en características clave de las gaussianas. • **Tarea 2.4:** Renderizado de las escenas con las modificaciones realizadas.
Evaluación / Validación / Prueba	40 horas	• **Tarea 3.1:** Evaluación de resultados.
Documentación / Presentación	40 horas	• **Tarea 4.1:** Redacción de la memoria del Trabajo de Fin de Grado. • **Tarea 4.2:** Preparación de la presentación del TFG. • **Tarea 4.3:** Realización de ensayos para la presentación final.

2.3. Tecnologías usadas

Para el desarrollo de este TFG se han empleado diversas herramientas y tecnologías que facilitan la implementación y evaluación. Entre las principales se destacan:

- **Lenguaje de programación Python:** Python [6] es un lenguaje de programación de alto nivel, interpretado y de código abierto, el cual destaca por su sintaxis clara y su facilidad de aprendizaje. La elección de Python[6] para este proyecto se basa en la disponibilidad de un abanico de bibliotecas especializadas en procesamiento numérico, redes neuronales y visualización 3D, lo que facilita la implementación y experimentación de métodos, así como el desarrollo de pruebas para la interpretación de resultados.
- **Bibliotecas numéricas y de procesamiento:** Se utiliza PyTorch [7] para gestionar las operaciones matemáticas y el procesamiento de datos necesarios en la manipulación de gaussianas, aprovechando su estructura de tensores y su capacidad de realizar cálculos de forma optimizada en GPUs.
- **Herramientas de visualización y análisis de resultados:** Se utiliza matplotlib [8] para la generación de gráficos y la realización de pruebas en el análisis de resultados, TensorBoard [9] para la observación en tiempo real de la evolución del entrenamiento, y GeoGebra [10] para plantear y visualizar implementaciones matemáticas.
- **Gestión de entornos:** Docker [11] se emplea para la gestión de repositorios, evitando así las incompatibilidades de versiones e incorrecta organización de direcotorios.
- **Herramienta de gestión de proyectos:** Jira [12] se ha empleado para gestionar y hacer seguimiento del progreso del proyecto. Esta herramienta de gestión de tareas permite organizar el trabajo, asignar tareas específicas, y monitorizar el avance mediante un tablero visual.

Capítulo 3

Estado actual y objetivos iniciales

3.1. Introducción

Durante los últimos años, el renderizado y representación de las escenas 3D han sufrido un gran avance, gracias al desarrollo constante de modelos basados en aprendizaje automático. Entre estos avances destaca Neural Radiance Fields (NeRF) [13], un método que permite generar representaciones fotorrealistas a partir de un conjunto de imágenes 2D. Sin embargo, NeRF[13] presenta un alto coste computacional, ya que requiere evaluar una red neuronal en cada punto del espacio, lo que lo hace poco eficiente para aplicaciones en tiempo real.

Por ello, ha surgido 3D Gaussian Splatting (3DGS)[5], un enfoque alternativo que ofrece una representación más eficiente al modelar la escena mediante nubes de puntos gaussianos en vez de depender de redes neuronales profundas. Este método ha demostrado una mayor velocidad y adaptabilidad que NeRF[13], Sin embargo, su proceso de densificación introduce un crecimiento incontrolado en el número de gaussianas, lo que incrementa significativamente el consumo de memoria y almacenamiento

A lo largo del desarrollo de este trabajo se han estudiado en profundidad múltiples métodos que se encuentran relacionados con la representación de escenas 3D, los cuales han sido clave para entender un poco más en profundidad las limitaciones de los enfoques actuales y orientar las soluciones propuestas. Entre ellos destacan Instant-NGP [14], que introduce técnicas de compresión y codificación para acelerar NeRF; Mip-NeRF [15], que emplea mip-mapping para mejorar el anti-aliasing; y Mip-NeRF 360 [16], una extensión que permite representar escenas panorámicas completas.

3.2. Representación de Modelos 3D: De NeRF a 3DGS

La representación de escenas en 3D ha evolucionado a lo largo del tiempo, adoptando diferentes enfoques para almacenar y procesar la información geométrica y visual. Estos enfoques pueden clasificarse en dos grandes categorías: representaciones explícitas e implícitas, dependiendo de cómo se organizan y manipulan los datos de la escena.

3.2.1. Modelos Explícitos e Implícitos

Existen dos enfoques principales para la representación de modelos 3D:

1. **Representaciones Explícitas**: Las representaciones explícitas almacenan directamente la geometría y apariencia de la escena en estructuras discretas, lo que permite un acceso inmediato a la información sin necesidad de evaluar funciones matemáticas o redes neuronales durante el proceso de renderizado.

   Algunos ejemplos son:

   - **Mallas poligonales**: utilizadas en gráficos 3D tradicionales.
   - **Vóxeles**: empleados principalmente en simulaciones volumétricas y motores de física.
   - **Nubes de puntos**: formadas por un conjunto de puntos con información sobre su posición y, en algunos casos, su color u otras propiedades.

   Estos métodos presentan ventajas significativas, como un renderizado más rápido, ya que no dependen de redes neuronales complejas, y un menor consumo de almacenamiento en comparación con enfoques implícitos volumétricos. Sin embargo, también tienen desventajas, como la aparición de artefactos visuales en caso de baja resolución.

2. **Representaciones Implícitas**: A diferencia de las representaciones explícitas, los modelos implícitos no almacenan directamente la geometría de una escena, sino que definen una función matemática o neuronal encargada de describirla. Es decir, en lugar de contar con una nube de puntos predefinida, por ejemplo, la escena se genera evaluando una función en cada consulta para determinar su apariencia.

   Un ejemplo de este enfoque es NeRF[13], que utiliza una red neuronal para definir la densidad y el color de cada punto de la escena a partir de un conjunto de coordenadas espaciales.

   Entre sus ventajas destaca la capacidad de representar detalles finos sin necesidad de utilizar estructuras discretas complejas. Sin embargo, este método conlleva un alto costo computacional, debido a la frecuencia de llamadas para evaluaciones.

3.3. Fundamentos de Aprendizaje Automático y Redes Neuronales

3.3.1. Redes Neuronales

Una red neuronal escencialmente es un conjunto de modelos matemáticos inspirado en la estructura y funcionamiento del cerebro humano. Esta nueva estructura está compuesta por las conocidas neuronas artificiales, las cuales se encuentran organizadas en capas, encargadas de procesar la información mediante funciones de activación y la propagación de errores.

En el aprendizaje automático, las redes neuronales se usan para modelar tipos de relaciones complejas, permitiendo aprender patrones y haciendo tareas laboriosas, tales como la clasificación o generación.

En el caso de NeRF[13], se hace uso de una red nauronal multicapa (MLP)[17], capaz de aprender a representar una escena 3D a partir de imágenes 2D, codificando la densidad y el color en un escpacio de 3 coordenadas.

La Ilustración 3.1 muestra la arquitectura típica de una red neuronal MLP, donde se distinguen las capas de entrada (en verde), las capas ocultas (en amarillo) y la capa de salida (en rojo). Este tipo de estructura es la base para muchos de los modelos empleados en reconstrucción 3D mediante aprendizaje automático.

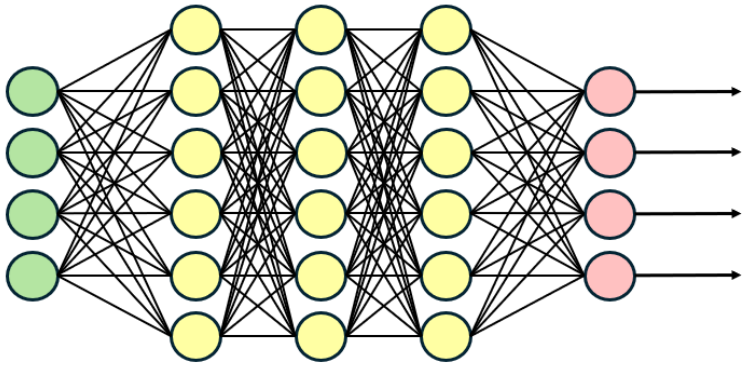


Ilustración 3.1: Esquema general de una red neuronal multicapa (MLP), donde se observa la estructura de capas de entrada, ocultas y de salida

3.3.2. Perceptrón

Una neurona(perceptrón) es conocida como la unidad básica o elemental de la red neuronal y, cuya función principal es recibir un conjunto de entradas (x1, x2, x3...), que pueden representar características, pixeles o incluseo coordenadas. A las esntradas se las asigna un

peso (w1, w2, w2...) que indican que tan relevante o importante es cada una de las entradas en el cálculo final. La neurona combina estas entradas ponderadas mediante operaciones matemáticas para posteriormente aplicar una función de activación que permite generar una salida. La salida se expande a otras neuronas de capas posteriores permitiendo así el aprendizaje y la toma inteligente de decisiones.

La Ilustración 3.2 muestra el esquema interno de una neurona artificial, destacando el proceso de combinación de entradas y la aplicación de la función de activación para producir una salida.

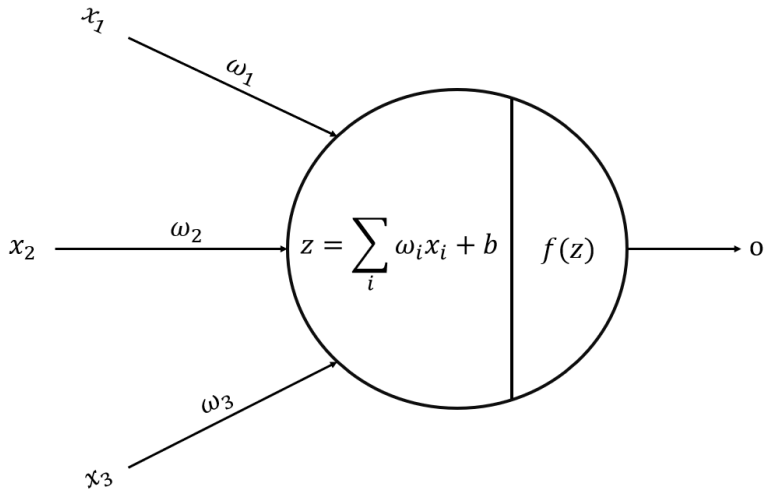


Ilustración 3.2: Representación de una neurona artificial, destacando la combinación de entradas ponderadas y la aplicación de la función de activación para generar la salida.

3.3.3. Aprendizaje Automático

El aprendizaje automático, conocido como Machine Learning en inglés, es una rama de la inteligencia artificial que permite a los sistemas aprender patrones y tomar decisiones a partir de datos de entrada, sin necesidad de estar programados explícitamente para una tarea específica.

Este proceso se compone de dos etapas fundamentales: el entrenamiento y la inferencia. Durante el entrenamiento, el modelo analiza un conjunto de datos etiquetado para aprender la relación entre entradas y salidas esperadas. Posteriormente, en la fase de inferencia, el modelo ya entrenado es capaz de recibir nuevos datos y realizar predicciones o clasificaciones basadas en lo aprendido.

La Ilustración 3.3 muestra de manera esquemática este proceso. Se parte de un conjunto de datos de entrenamiento, sobre el cual se entrena el modelo de Machine Learning. Una vez entrenado, este modelo puede recibir nuevas entradas y producir salidas o predicciones correspondientes.

Ilustración 3.3: Proceso de aprendizaje automático, donde un modelo aprende a partir de datos de entrada y es capaz de realizar predicciones o clasificaciones.

3.3.4. Aplicación del Aprendizaje Automático en la Representación de Modelos 3D

Dentro de los modelos 3D, el aprendizaje automático ha cambiado la forma en que se generan, procesan y optimizan estos, permitiendo así avances contundentes en la reconstrucción de escenas 3D.

Uno de los avances más destacados ha sido la reconstrucción y renderizado de escenas 3D a partir de imágenes en dos dimensiones. Dentro de eso se encuentran modelos tales como NeRF[13] que utiliza redes neuronales profundas para aprender una representación implícita del volumen de la escena. Por otro lado, existe 3DGS[5], modelo capaz de representar escenas mediante una nube de gaussianas, siendo así más eficiente al no necesitar una red neuronal.

La Ilustración 3.4 compara de forma visual ambos enfoques. Mientras NeRF transforma las imágenes 2D en una escena 3D mediante una red neuronal, 3DGS lo hace directamente a través de gaussianas distribuidas en el espacio, simplificando el proceso de reconstrucción.

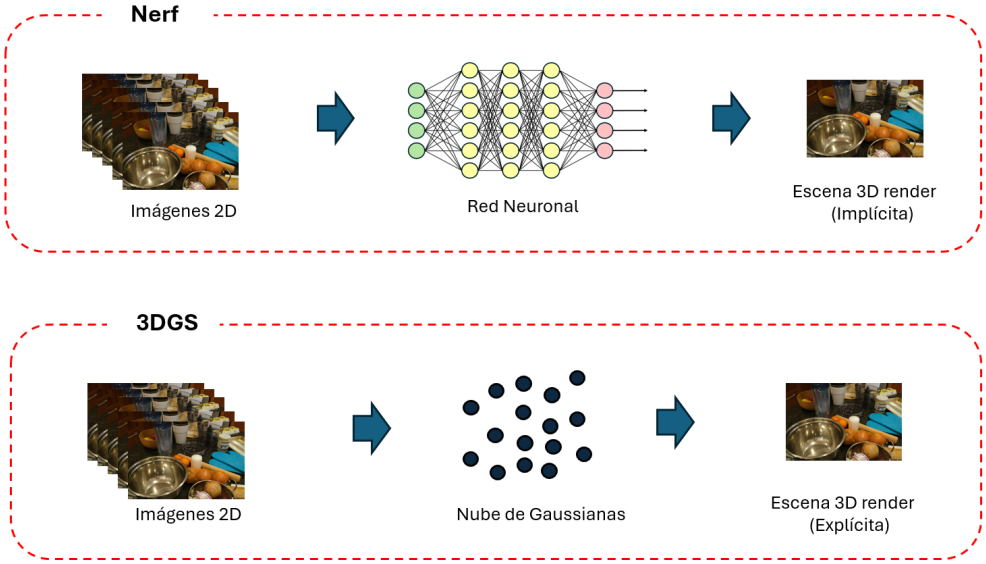


Ilustración 3.4: Aplicación del aprendizaje automático en la representación de modelos 3D, destacando las diferencias entre los enfoques basados en redes neuronales (NeRF) y representaciones explícitas (3DGS).

3.3.5. Neural Radiance Fields y sus limitaciones

A lo largo de los años, NeRF[13] se ha consolidado como una de las técnicas más destacadas para la construcción y renderizado de escenas 3D. Este modelo fue introducido en "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis", logrando reconstrucciones detalladas y fotorrealistas a partir de imágenes 2D.

El funcionamiento de NeRF[13] se basa en una red neuronal multicapa (MLP)[17] que aprende una función capaz de mapear coordenadas espaciales y direcciones de visión de las cámaras a valores de color y densidad volumétrica. Este proceso se desarrolla en los siguientes pasos:

1. Se toma una coordenada en el espacio tridimensional (x, y, z) junto con una dirección de visión de la cámara (θ, ϕ).
2. Las coordenadas espaciales se transforman mediante positional encoding para mejorar la capacidad de representación de la red.
3. Se evalúa una MLP[17] que predice dos valores clave: la densidad del punto en el espacio y su color RGB.
4. Finalmente, se aplica un proceso de integración volumétrica en el que se lanzan rayos desde la cámara a través de la escena. A lo largo de su trayectoria, estos rayos acumulan los valores de color y densidad de los puntos que intersectan, lo que permite reconstruir la imagen final.

La Ilustración 3.5 representa visualmente este flujo de datos. A partir de una entrada 5D que combina posición y dirección, el modelo neuronal predice color y densidad para cada punto consultado. Estas predicciones se utilizan para sintetizar imágenes a través de la simulación de rayos proyectados en la escena.

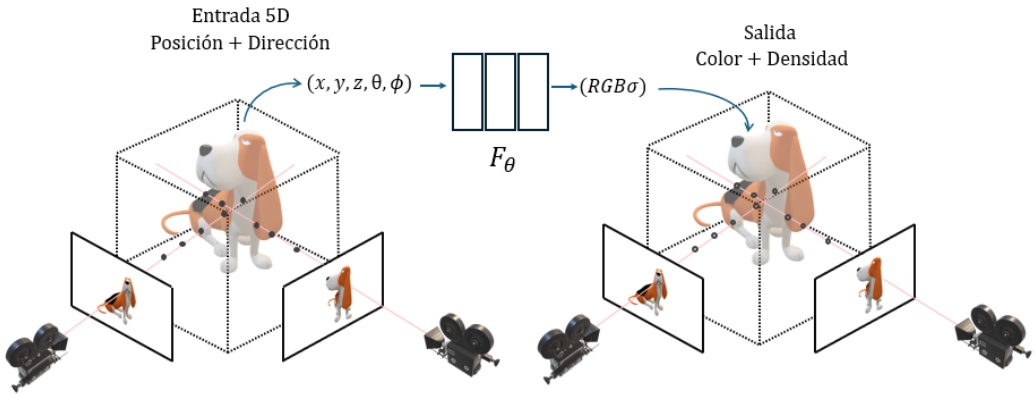


Ilustración 3.5: Funcionamiento interno de NeRF, donde se proyectan rayos a través de la escena y se evalúa una red neuronal en cada punto para obtener color y densidad.

A pesar de sus ventajas, NeRF[13] presenta varias limitaciones importantes:

- **Altos tiempos de entrenamiento:** El modelo requiere un entrenamiento prolongado para lograr reconstrucciones de calidad.
- **Renderizado lento:** La inferencia en NeRF[13] es computacionalmente costosa, lo que dificulta su uso en aplicaciones en tiempo real.
- **Dependencia de poses de cámara precisas:** La calidad de la reconstrucción depende en gran medida de la exactitud en la calibración de las vistas de entrada.

3.4. 3D Gaussian Splatting (3DGS)

3.4.1. Principios de Funcionamiento

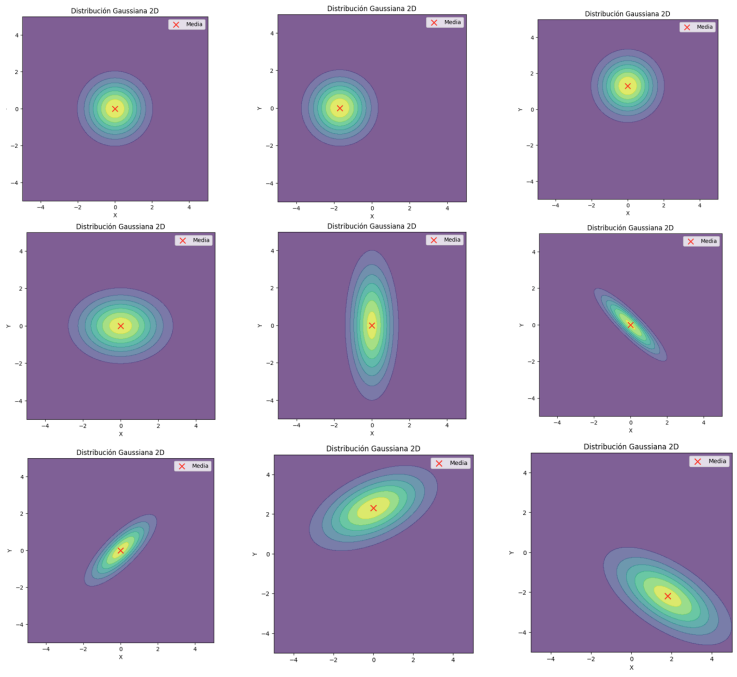
3D Gaussian Splatting (3DGS)[5] fue presentado como una opción más innovadora para representar y renderizar escenas 3D, siendo esta una alternativa más eficiente que los enfoques basados en redes neuronales, como NeRF[13]. En vez de usar una función implícita, 3DGS[5] usa una nube de puntos gaussianos, permitiendo así una representación explícita y bastante eficiente para el renderizado en tiempo real.

Cada gaussiana en la representación 3D en 3DGS[5] está definida por varios parámetros:

- Un vector tridimensional (x, y, z) que determina la ubicación del punto en el espación. Estos valores vienen dado por las las medias de las tres distribuciones gaussianas.
- Un color (r, g, b) y una opacidad α que controlan la apariencia visual del punto gaussiano.
- Una covarianza y una orientación que controlan la forma y la disperción de la gaussiana en el espacio, permitiendo así que la gaussiana se pueda ajustar a la geometría de la propia escena.

Al contrario de los modelos que se basan en mallas o vóxeles, cada punto gaussiano se proyecta sobre la imagen de salida, evitando así las reconstrucciones intermedias. Permitiendo de esta manera un renderizado más eficiente en GPUs, sin necesidad de evaluar funciones neuronales para cada píxel.

El Cuadro 3.1 se muestra una comparación visual de gaussianas 2D modificadas, reflejando cómo varían sus formas en función de los parámetros de escala, varianza y covarianza.



Cuadro 3.1: Comparación visual de Gaussianas con distintas modificaciones.

3.4.2. COLMAP y la Inicialización de Gaussianas

COLMAP[1] es una herramienta usada en 3DGS[5], ya que permite generar una nube de puntos mediante Structure from Motion. Esta nube de puntos facilitarán las posiciones para las gaussianas iniciales del modelo.

La Ilustración 3.6 se muestra un ejemplo típico de nube de puntos generada con COLMAP. En ella se pueden observar los puntos reconstruidos (aquellso que se muestran en tonos grises) y las posiciones de las cámaras estimadas (en rojo)

Ilustración 3.6: Nube de puntos generada con COLMAP a partir de un conjunto de imágenes. Imagen adaptada de [1].

3.4.3. Structure-from-Motion (SfM) en la Reconstrucción 3D

Sfm[18] es una técnica de visión por computador que permite la reconstrucción de escenas 3D a partir de múltiples imágenes que hayan sido tomadas desde distintos ángulos. Estima la posición de la cámara y la estructura tridimensional de la escena a partir de la detección y emparejamiento de puntos en las imágnees.

3.4.4. Problema de la Densificación en 3DGS

Uno de los principales problemas que presenta 3DGS[5] es el crecimiento incontrolado del número de gaussianas durante la etapa de densificación. Este hecho impacta negativamente a la eficiencia computacional y de almacenamiento del modelo, dificultando así su implementación en dispositivos que tengan recursos limitados.

Durante el entrenamiento de 3DGS[5], se realiza el proceso de densificación, en el cual el modelo ajusta de forma dinámica la cantidad de gaussianas que tiene la escena. Este proceso tiene dos operaciones claves:

1. Clonación de gaussianas: Se generan gaussianas adicionales en zonas donde la reconstrucción es escasa, rellenando espacios vacíos para mejorar la cobertura en aquellas regiones de las escenas que se encuentran poco detalladas. Sin embargo, esto puede provocar una sobrepoblación innecesaria, pues los criterios seguidos en la densificación se basan en umbrales.

2. División de gaussianas grandes: Las gaussianas que presentan un gran tamaño se subdividen para mejorar el nivel de detalle de la escena, permitiendo así una representación más precisas en aquellas regiones de la escena que presenten cambios geométricos bruscos. Por otro lado, esto puede generar un número excesivo de gaussianas, aumentando así el consumo de la memoria.

El crecimiento incontrolado de guassianas presenta varios problemas críticos:

1. Aumento del consumo de memoria VRAM: A medida que el númeor de gaussianas generadas es mayor, la cantidad de datos que se almacenan en los tensores y por ende en la GPU, es inviable.

2. Mayor costro computacionale durante el entrenamiento. Cuando se procesa y optimiza cada gaussiana se requiere actualizar los parámetros de estas, haciendo que el entrenamiento sea mas lento y supongo un mayor coste en el hardware

En la Ilustración 3.7 se muestra la relación entre el número de gaussianas y el consumo de memoria VRAM en distintas escenas. Se observa una correlación clara, donde escenas con mayor número de gaussianas presentan un mayor uso de memoria, lo que evidencia la necesidad de controlar esta densificación.

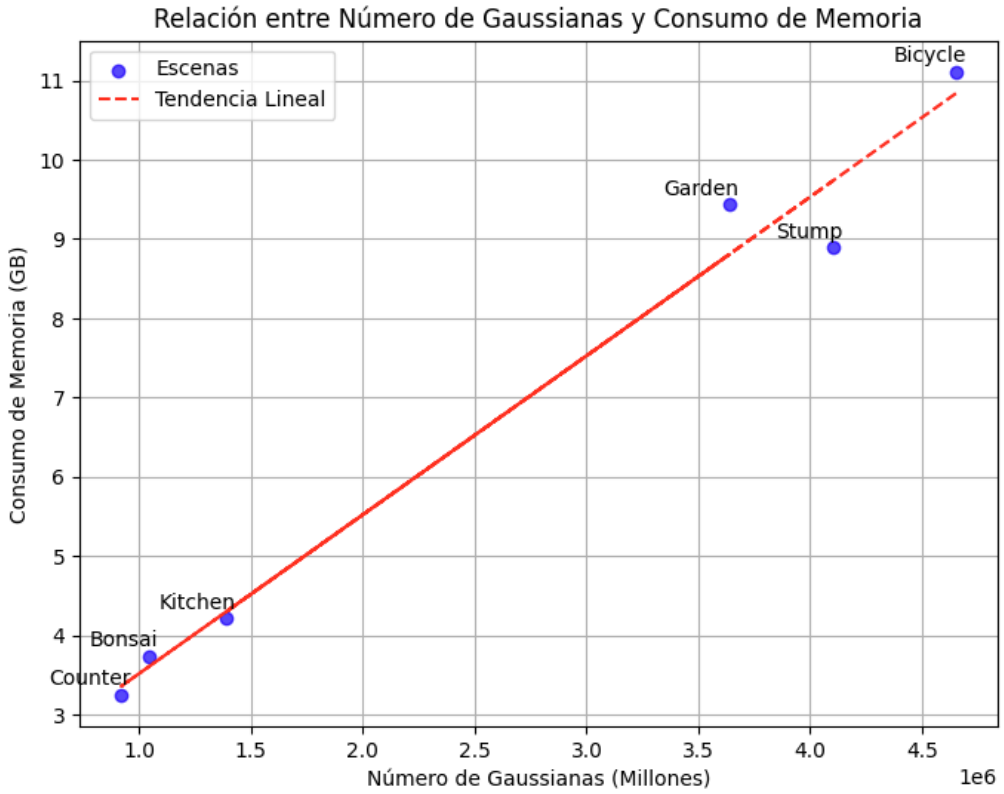


Ilustración 3.7: Relación entre el número de gaussianas generadas y el consumo de memoria VRAM durante el entrenamiento de 3DGS.

3.5. Métodos para Controlar la Densificación en 3DGS

3.5.1. Taming 3DGS: Control del Crecimiento de Gaussianas

Taming 3DGS[19] introduce un nuevo sistema de limitación del crecimiento de gaussianas, basado en una ecuación cuadrática que regula el proceso de densificación de manera controlada y predecible.

Para evitar el crecimiento incontrolado de gaussianas durante el entrenamiento, Taming 3DGS[19] establece un límite máximo de gaussianas en cada etapa del proceso, determinado mediante una ecuación cuadrática. Este límite se ajusta en función de un presupuesto de gaussianas previamente estipulado por el usuario, lo que permite distribuir de manera eficiente la cantidad de gaussianas a lo largo de todo el entrenamiento.

Además, se implementa un mecanismo de selección y poda basado en un sistema de puntuación (score-based ranking). Este sistema evalúa distintos parámetros clave de cada gaussiana para determinar cuáles deben eliminarse y cuáles deben conservarse.

La ecuación es la siguiente:

$$g(x) = \left(\frac{B - S - kN}{N^2}\right) x^2 + kx + S \tag{3.1}$$

Donde:

$k = \frac{2(B-S)}{N}$

S es el número de puntos iniciales.

B es el número de puntos finales.

x representa el paso actual.

N es el paso final.

3.5.2. Compact 3DGS: Reducción del Tamaño del Modelo

Compact 3DGS[20] surge como solución al problema del alto consumo de memoria y almacenamiento de 3DGS, proponiendo un efoque más optimo para reducir el número de gaussianas y compactar los atributos de estos puntos.

Durante el entrenamiento de 3DGS[5] se incrementa el número de gaussianas mediante clonación y subdivisión. Esto supone la introducción de gaussianas redundantes que no aportan cambios significativos al resultado final, lo que incrementa el uso de memoria GPU y el almacenamiento.

Para abordar esto, Compact 3DGS[20] propone una estraegia de máscara volumétrica basada en:

- Se eliminan las gaussianas que tienen baja opacidad .
- Se eliminan las gaussianas que son pequeñas y se pueden elimnar sin afectar de forma significativa al resultado final.

Además, Compact 3DGS[20] propone la reduccion del tamaño d elos atributos de cada gaussiana. En el modelo original, cada gaussiana almacena la posicion 3d, opacidad, color, covarianza y orientación.

Sin embargo, Compact 3DGS[20] optimiza esta representación utilizando dos estrategias:

- Se reduce el almacenamiento del color. En vez de almacenar los valores de color de cada gaussiana, se usa una red neuronal basada en grids lo que permite interpolar los colores y reducir el número de prámetros necesarios.
- Se comprimen los adributos geométricos. Para representar los atributos como escala y rotación, hace uso de una cuantización vectorial con diccionarios de código, agrupando gaussianas con características parecidas y guardando solo un índice de referencia en vez de los valores completos.

La Ilustración 3.8 resume visualmente el proceso propuesto por Compact 3DGS, desde la enmascaración hasta la compresión de atributos y el renderizado final.

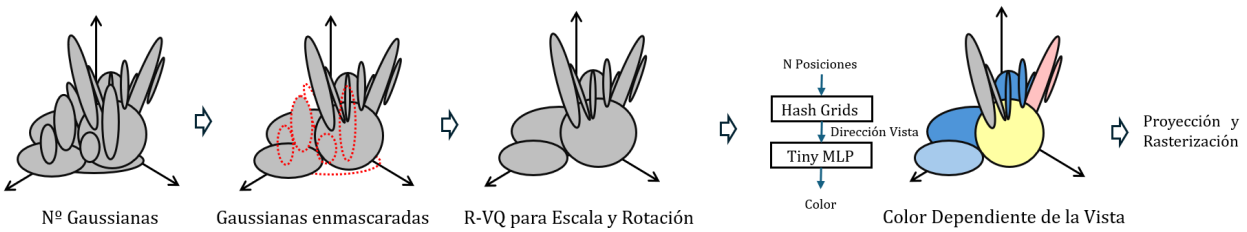


Ilustración 3.8: Esquema del proceso de reducción de gaussianas y compresión de atributos propuesto en Compact 3DGS.

3.5.3. 3D Gaussian Splatting as Markov Chain Monte Carlo (MCMC)

Por otro lado, se encuentra 3DGS MCMC[21] que reformular el proceso de densificación de 3DGS[5], basando el proceso en un muestreo probabilístico. El objetivo de este modelo es mejorar la distribución de las gaussianas en la escena y evitar la generación excesiva de gaussianas que pueden ser innecesarias, reduciendo así el consumo de memoria y mejorando la eficiencia.

En 3DGS[5] tradicional, la colocación de gaussianas se basa en la minimización de una función de pérdida mediante gradientes, lo que ayuda a proporcionar soluciones locales para la división y eliminación de gaussianas. Sin embargo, 3DGS MCMC[21] interpreta esto como un problema de muestreo probabilístico, modelando la distirbución de las gaussianas mediante cadenas de Markov Monte Carlo (MCMC)

En la Ilustración 3.9 se puede observar cómo el optimizador puede quedar atrapado en un mínimo local (región inferior izquierda) y no alcanzar el mínimo global (cima más baja a la derecha). Este problema es especialmente crítico en entornos complejos como la representación 3D.

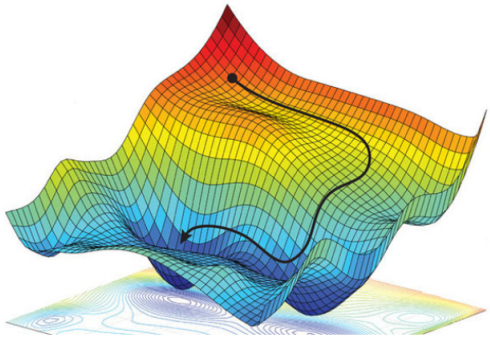


Ilustración 3.9: Representación de un paisaje de función de pérdida en un problema de optimización. Imagen adaptada de [2].

Este proceso de optimización se realiza mediante descenso por el gradiente, una técnica usada en aprendizaje automático. El objetivo es reducir al mínimo una función de pérdida que mide la diferencia entre la imagen generada (render) por las gaussianas actuales y la imagen objetivo (ground trouth). Para ello, se calcula el gradiente de dicha pérdida con respecto a los parámetros de cada gaussiana (posición, escala, orientación, color, ...etc), y se actualizan estos parámetros en la dirección que reduce la pérdida. Esto se realiza durante un proceso iterativo, donde la escala de actualización, también conocido por longitud de pasos, viene dado por la tasa de aprendizaje..

Sin embargo, el descenso por gradiente clásico puede conducir a mínimos locales 3.9, lo cual limita la capacidad del modelo para explorar configuraciones más óptimas de gaussianas.

El modelo define una distribución de probabilidad G que asigna alta probabilidad a configuaciones de gaussianas que reconstruyen de forma fiel y estable la escena

Para esto, 3DGS MCMC[21] emplea Stochastic Gradient Langevin Dynamics (SGLD)[22], que es un método que nace del descenso del gradiente estocástico[23], añadiendo un término de ruido controlado. La adición de este ruido añade aleatoriedad en la actualización de las gaussianas, ayudando a evitar mínimos locales no óptimos.

A continuación, se muestra un esquema simplificado del proceso de actualización de una gaussiana con SGLD[22]:

Algoritmo 1 Actualización de una gaussiana con SGLD

Require: Gaussiana g, tasa de aprendizaje λ_{lr}, coeficiente de ruido λ_{noise}

1: Calcular gradiente $\nabla L(g)$
2: Muestrear ruido $\epsilon \sim \mathcal{N}(0, \Sigma)$
3: $g \leftarrow g - \lambda_{lr} \cdot \nabla L(g) + \lambda_{noise} \cdot \epsilon$

El término de ruido ϵ es definido como:

$$\epsilon = \lambda_{lr} \cdot \sigma^{-k(o-t)} \cdot \Sigma\eta, \quad \eta \sim \mathcal{N}(0, I) \tag{3.2}$$

donde o es la opacidad de la gaussiana, Σ su covarianza, y k y t son hiperparámetros que controlan la transición suave del ruido.

A continuación, se detallan los procedimientos específicos para actualizar los distintos parámetros que definen una gaussiana. El ruido estocástico solo se añade a la posición (media), mientras que la opacidad y la escala se actualizan de forma determinista, ya que el ruido, según los autores, afecta negativamente a su estabilidad durante el entrenamiento.

Algoritmo 2 Actualización de la posición (media) con SGLD

Require: Media actual μ, gradiente $\nabla L(\mu)$, tasa de aprendizaje λ_{lr}, opacidad o, covarianza Σ

1: Calcular factor de ruido con la función sigmoide: $\sigma = \text{sigmoid}(-k(o - t))$
2: Muestrear ruido: $\eta \sim \mathcal{N}(0, I)$
3: Calcular ruido: $\epsilon = \lambda_{lr} \cdot \sigma \cdot \Sigma\eta$
4: Actualizar posición: $\mu \leftarrow \mu - \lambda_{lr} \cdot \nabla L(\mu) + \epsilon$

Algoritmo 3 Actualización de opacidad y escala sin ruido

Require: Opacidad o, escala s, gradientes $\nabla L(o)$ y $\nabla L(s)$, tasa de aprendizaje λ_{lr}

1: $o \leftarrow o - \lambda_{lr} \cdot \nabla L(o)$
2: $s \leftarrow s - \lambda_{lr} \cdot \nabla L(s)$

El proceso de densificación de 3DGS[5] emplea una heurística basada en la clonación de gaussianas en áreas poco definidas, la división de gaussianas grandes para mejorar la precisión y la poda de gaussianas con poca opacidad.

Sin embargo, estas no consideran de forma explícita la probabilidad de contibución de cada gaussiana a la calidad del renderizado. Por ello, 3DGS MCMC[21] define de nuevo estas operaciones dentro de la distribución de probabilidad, para mantener así una coherencia estadística del modelo.

3.5.4. Group Training: Acelerando y Mejorando 3DGS

Recientemente, se ha propuesto el método *Group Training*[24] como una estrategia para mejorar tanto la velocidad como la calidad del entrenamiento en 3DGS[5]. Esta técnica, que fue introducida por Wang et al, consiste en dividir el conjunto total de gaussianas en dos grupos: el *Grupo Activo* (o *Under-training*) y el *Grupo en Caché* (*Cached Group*). Durante el entrenamiento, solo las gaussianas del primer grupo juegan un papel en los procesos de densificación y optimización, mientras que las gaussianas almacenadas en caché se excluyen de forma temporal para así reducir el coste computacional. Tras ciertas iteraciones, ambos grupos se combinan y se vuelve a realizar la segementación para organizar de nuevo las gaussianas, asegurando así que todas las gaussianas contribuyan en el entrenamiento.

La Ilustración 3.10 muestra un esquema general del funcionamiento de esta técnica. El grupo activo se actualiza mediante muestreo, entrenamiento y densificación, mientras que el grupo en caché queda temporalmente bloqueado (representado con candados). Posteriormente, ambos grupos se fusionan para iniciar un nuevo ciclo de entrenamiento.

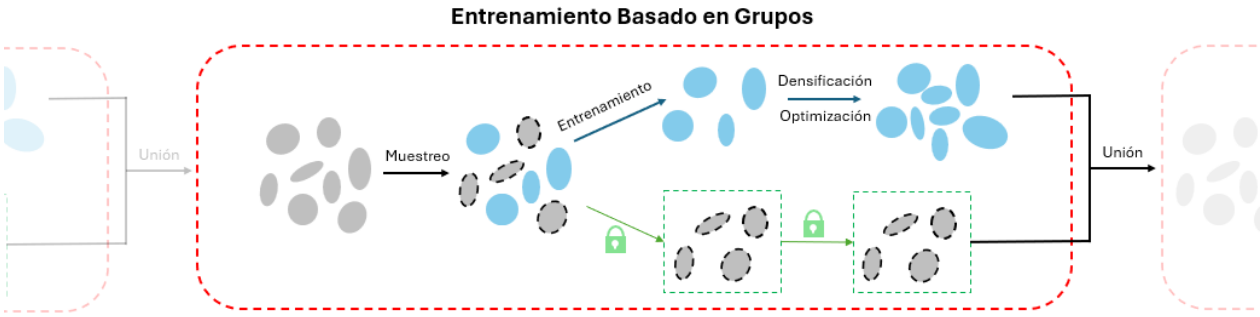


Ilustración 3.10: Esquema del método Group Training aplicado a 3DGS. Las gaussianas se dividen en un grupo activo, que participa en la densificación y optimización, y un grupo en caché, que se excluye temporalmente para reducir el coste computacional.

3.6. Objetivos del Trabajo

El objetivo principal de este trabajo es analizar y desarrollar técnicas para controlar la densificación de gaussianas en 3DGS[5], con el fin de eliminar el crecimiento inctrolado del número de gaussianas. Esto provoca un alto consumo de memoria, ralentiza el entrenamiento y afecta de forma negativa a recursos como el almacenamiento.

Para ello, se plantean los siguientes objetivos:

- Estudio y análisis del problema de densificación en 3DGS[5]: Revisar la documentación actual sobre modelos basados en Gaussian Splatting e identificar factores claves que influyan en el crecimiento de gaussianas durante el entrenamiento del modelo.
- Implementación de técnicas para el control de la densificación: Implementar y evaluar técnicas para eliminar la sobreproblación de gaussianas sin perder calidad del resultado y probar a implementar técnicas anteriormente leídas.
- Optimización de los recursos: Reducir el tiempo de ejecución del entrenamiento y minimizar el consumo de memoria VRAM, así como el almacenamiento.
- Evaluación y validación: Comparar los resultados de las nuevas técnicas frente a la implementación base de 3DGS[5].

Capítulo 4

Aportaciones del trabajo

4.1. Principales aportaciones

Este Trabajo de Fin de Grado contibuye de forma significativa al campo de la representación de escenas 3D mediante 3DGS[5], aportando mejora a una de sus limitaciones, el crecimiento incontrolado de gaussianas durante el proceso de densificación. Las principales aportaciones han sido:

- **Optimización del proceso de densificación en 3DGS:** Se propone un nuevo enfoque para controlar el número de gaussianas generadas en cada etapa de entrenamiento. De esta manera, se regula la densificación para evitar el crecimiento incotrnolado, mejorando así el rendimiento del sistema sin afectar a la calidad del resultado.
- **Reducción del consumo de memoria y almacenamiento:** Uno de los principales problemas de 3DGS[5] es el alto uso de memoria VRAM y almacenamiento debido a la generación masiva de gaussianas. La solución presentada en este TFG busca minimizar estos requisitos, permitiendo la implementación del modelo en dispositivos con capacidades gráficas más limitadas.
- **Disminución del tiempo de entrenamiento:** Al limitar la cantidad de gaussianas, se logra reducir el tiempo de entrenamiento para generar una escena sin perder calidad en el resultado.
- **Facilitación del uso de 3DGS[5] en distintas industrias:** Con estas mejoras, adoptar 3DGS[5] como herramienta para secotres como el cine y el desarrollo de los videojuegos se vuelve más accesible, no solo para las propias empresas sino para los usuarios aficionados con hardware menos sofisticado.

4.2. Alineamiento con los objetivos de desarrollo sostenible

Cuadro 4.1: Grado de relación del TFG con los Objetivos de Desarrollo Sostenible.

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

A continuación, se justifica la relación del TFG con los Objetivos de Desarrollo Sostenible marcados en la tabla 4.1:

- **ODS 1, 2, 5, 6, 14, 15, 16 (Grado 0 - No proceden):** Estos objetivos se centran en problemas sociales, medioambientales o políticas (como pobreza, hambre, igualdad de género, ecosistemas, paz ... etc), que no están directamente relacionados con la parte técnica del TFG.
- **ODS 3 - Salud y bienestar (Grado 1 - Bajo):** Aunque no es el objetivo principal del trabajo, la optimización de la generación de escenas 3D puede tener aplicaciones indirectas en campos como la medicina (por ejemplo, simulaciones médicas o tratamientos innnovadores).
- **ODS 4 - Educación de calidad (Grado 2 - Medio):** El uso de menos recursos computacionales permite que instituciones educativas con recursos más limitados accedan a tecnologías de representación 3D, facilitando así la formación de sus alumnos en áreas como visión artificial o IA.
- **ODS 7 - Energía asequible y no contaminante (Grado 1 - Bajo):** El trabajo puede influir de forma indirecta al ahorro energético, al reducir la cantidad de recursos

necesarios para entrenar y ejecutar modelos 3DGS[5], lo que se puede traducir como un menor consumo energético.

- **ODS 8 - Trabajo decente y crecimiento económico (Grado 2 - Medio):** Este trabajo puede favorecer a la aparición de nuevos perfiles laborales especializados en ello, para ssectores como los videojuegos, la animación o la realidad virtual.
- **ODS 9 - Industria, innovación e infraestructuras (Grado 3 - Alto):** Esta es la opción más relacionado con el TFG. Se fomenta la innovación tecnológica al proponer una mejora en un método relativamente reciente como es 3D Gaussian Splatting, facilitando su integración en infraestructuras industriales o tecnológicas.
- **ODS 10 - Reducción de las desigualdades (Grado 1 - Bajo):** Al facilitar el uso de herramientas gráficas avanzadas en hardware con menores características (menos potentes), se contribuye a reducir las desigualdades tecnológicas entre distintas regiones con menor acceso a recursos.
- **ODS 11 - Ciudades y comunidades sostenibles (Grado 2 - Medio):** La generacioñon de escenas 3D podría aplicarse a la mapeo urbano, simulaciones de movilidad o representación de espacios públicos, haciendo de estos, proyectos más sostenibles.
- **ODS 12 - Producción y consumo sostenibles (Grado 3 - Alto):** Se reduce el uso innecesario de memoria y almacenamiento, por lo que impacta de forma positiva en la eficiencia del consumo de recursos.
- **ODS 13 - Acción por el clima (Grado 1 - Bajo):** De forma indirecta, la optimización de proceso relacionados con la generación de escenas puede suponer una menor huella de carbono asociada al entrenamiento de modelos, al reducir el consumo de energía.
- **ODS 17 - Alianzas para lograr objetivos (Grado 1 - Bajo):** Puesto que el TFG se basa en artículos científicos, se promuebe la colaboración dentro de esta comunicad, en busca del avance colectivo a una mejor solución..

4.3. Competencias específicas

Según lo establecido en la Memoria del Plan de Estudios del plan 41 del Grado de Ingeniería Informática, durante el desarollo de este Trabajo de Fin de Grado se han cubierto las diversas competencias, entre las que destacan:

- **CI6:** *Conocimiento y aplicación de los procedimientos algorítmicos básicos de las tecnologías informáticas para diseñar soluciones a problemas, analizando la idoneidad y complejidad de los algoritmos propuestos.*
    - Se aplica al diseño de estrategias para controlar la densificación en 3D Gaussian Splatting (3DGS)[5], analizando el impacto computacional de diferentes enfoques y si es idóneo en términos de eficiencia y consumo de recursos.

- **CI15:** *Conocimiento y aplicación de los principios fundamentales y técnicas básicas de los sistemas inteligentes y su aplicación práctica.*
    - Este TFG profundiza en la optimización de 3DGS[5] mediante técnicas avanzadas de aprendizaje automático, asegurando un mejor control del proceso de densificación.
- **CI16:** *Conocimiento y aplicación de los principios, metodologías y ciclos de vida de la ingeniería del software.*
    - A lo largo del proyecto, se han seguido metodologías de experimentación y desarrollo para implementar y validar soluciones optimizadas en los resultados del modelofd.
- **TFG:** *Ejercicio original a realizar individualmente y presentar y defender ante un tribunal universitario, consistente en un proyecto en el ámbito de las tecnologías específicas de la Ingeniería en Informática de naturaleza profesional en el que se sinteticen e integren las competencias adquiridas en las enseñanzas.*

Capítulo 5

Desarrollo

La etapa de desarrollo de este TFG se centrará principalmente en analizar en profundidad el comportamiento del método de densificación empleado en 3DGS. Este análisis permitirá determinar la importancia y el impacto que tiene dicha etapa en la calidad final de las escenas renderizadas, así como estudiar los efectos emergentes al modificar diferentes parámetros del modelo.

Posteriormente, se abordará el estudio, implementación y validación de resultados del método de densificación basado en parches. Finalmente, se propondrán nuevas hipótesis metodológicas, cuya implementación se llevará a cabo con el propósito de evaluar su eficacia mediante un análisis de los resultados obtenidos.

5.1. Entrenamiento 3DGS

En primer lugar, se llevará a cabo el entrenamiento de múltiples escenas del dataset Mip-NeRF 360 [25], comparando los resultados obtenidos con la densificación activada y desactivada, con el objetivo de analizar la relevancia e impacto que tiene este proceso en los resultados generales del modelo.

Es importante destacar que la calidad de los resultados dependerá principalmente de cuatro aspectos clave: la calidad de imagen, el número de gaussianas de la escena, el consumo de memoria gráfica y el tiempo requerido para el entrenamiento.

Todos los experimentos se entrenaron durante 30 000 iteraciones, utilizando los mismos parámetros base para garantizar una comparación justa entre las distintas configuraciones.

Dado que gran parte de los experimentos fueron realizados en una GPU NVIDIA RTX 3090 con 24 GB de memoria VRAM, se aplicó un *downscaling* por un factor de 4 a todas las imágenes de entrada. Esta reducción fue necesaria para asegurarse de que los procesos pudieran ejecutarse dentro de la memoria disponibles.

En ciertos experimentos más exigentes, donde las escenas superaban las capacidades de la 3090, se trasladaron los experimentos a una GPU NVIDIA A100 con 80 GB de VRAM.

La calidad de imagen se evaluará mediante distintas métricas:

- **SSIM (Structural Similarity Index Measure)**[26]: Esta métrica mide la similitud estructural entre dos imágenes, considerando factores como la luminancia, el contraste y la estructura. Su valor se correlaciona estrechamente con la percepción visual humana.
- **PSNR (Peak Signal-to-Noise Ratio)** [27]: Esta métrica mide la relación entre la potencia máxima de una señal (la imagen original o ground truth) y la potencia del ruido generado por las diferencias con la imagen reconstruida, ofreciendo así una valoración objetiva de la calidad general de la imagen.
- **LPIPS (Learned Perceptual Image Patch Similarity)** [28]: Esta métrica utiliza redes neuronales previamente entrenadas para evaluar la distancia perceptual entre imágenes, proporcionando una medida precisa de la similitud visual percibida desde la perspectiva humana.

El proceso de entrenamiento en 3DGS[5] sigue un proceso iterativo de optimización, en el que se ajustan los parámetros de cada gaussiana con el objetivo de minimizar la diferencia entre las imágenes generadas por el modelo y las imágenes reales del conjunto de entrenamiento.

Tal y como se representa en la Imagen 5.1, en cada iteración del entrenamiento se realiza lo siguiente:

1. Se selecciona aleatoriamente una cámara del conjunto de vistas disponibles.
2. Se genera una imagen renderizada de la escena desde esa vista, utilizando el estado actual del modelo.
3. Se compara la imagen generada con la imagen real (Ground Truth), obteniendo así una medida de error o pérdida.
4. Se calcula el gradiente de esa pérdida respecto a los parámetros de las gaussianas, y se actualizan en la dirección que reduce dicho error.
5. Si la densificación está habilitada y se cumplen las condiciones necesarias, se realiza una fase de densificación: se añaden, dividen o eliminan gaussianas en función de su escala, opacidad o cobertura.

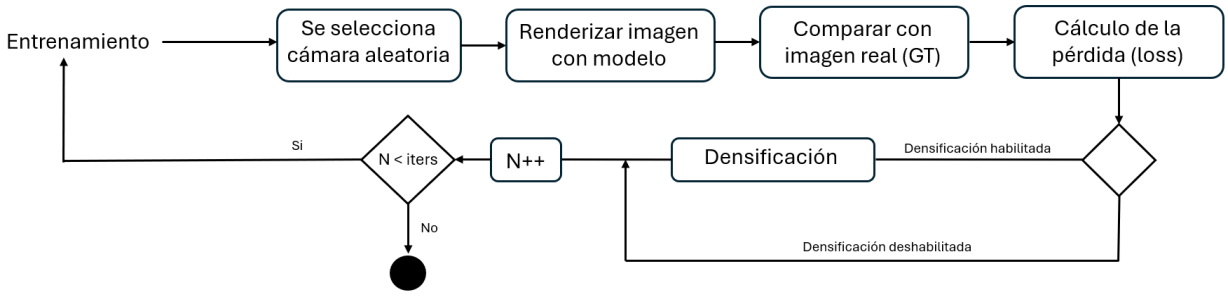


Ilustración 5.1: Esquema general del proceso de entrenamiento en 3D Gaussian Splatting

Este bucle se repite durante un número determinado de iteracionesa. El proceso de densificación, permite afinar la representación de la escena adaptando el número, en base a una serie de umbrales aplicados a las propiedades de las gaussianas.

5.1.1. Resultados sin densificación

En primer lugar, se abordará el método sin densificación. Para desactivar la densificación basta con modificar un parámetro del modelo llamado densify_until_iter, estableciendo su valor en cero, o comentar directamente la llamada a la función densify_and_prune() durante el proceso de entrenamiento. De esta manera, es posible evaluar fácilmente el impacto de este proceso sobre los resultados.

La Tabla 5.1 muestra los resultados obtenidos al entrenar las escenas sin aplicar el proceso de densificación. Se incluyen métricas como SSIM, PSNR y LPIPS, así como el número de gaussianas utilizadas, la memoria máxima consumida y el tiempo de entrenamiento.

Además, en la Figura 5.2 se presentan visualmente las reconstrucciones obtenidas sin densificación, lo que permite una comparación directa con las reconstrucciones que se mostrarán posteriormente cuando la densificación está habilitada.

Escena	**SSIM**	**PSNR**	**LPIPS**	**Nº Gaussianas**	**Max Mem (GB)**	**Tiempo**
Bicycle	0,57	22,75	0,46	54.275	3,68	14m 9s
Bonsai	0,94	31,28	0,13	206.613	2,29	8m 57s
Counter	0,91	28,61	0,14	155.767	1,89	8m 29s
Garden	0,74	25,22	0,31	138.766	3,74	15m 16s
Kitchen	0,94	30,65	0,09	241.367	2,25	9m 35s
Stump	0,56	23,04	0,50	32.049	2,48	13m 33s

Cuadro 5.1: Resultados sin densificación

(a) Bicycle

(b) Bonsai

(c) Counter

(d) Garden

(e) Kitchen

(f) Stump

Ilustración 5.2: Reconstrucciones sin densificación.

5.1.2. Resultados con densificación habilitada

Una vez obtenidos los resultados en el escenario sin densificación, se procede a evaluar el comportamiento del modelo con la densificación activada.

En la Tabla 5.2 se presentan los resultados cuantitativos obtenidos con la densificación activada. Se muestran las métricas SSIM, PSNR y LPIPS, junto con el número total de gaussianas generadas, la memoria máxima utilizada y el tiempo de entrenamiento requerido para cada escena.

Asimismo, en la Figura 5.3 se muestran las reconstrucciones visuales obtenidas tras entrenar cada escena con la densificación habilitada. Estas imágenes permiten comparar visualmente el impacto de la densificación con respecto a las reconstrucciones previas sin este proceso.

Escena	**SSIM**	**PSNR**	**LPIPS**	**Nº Gaussianas**	**Max Mem (GB)**	**Tiempo**
Bicycle	0,78	25,69	0,20	4.652.308	11,11	37m 8s
Bonsai	0,96	33,01	0,08	1.040.856	3,74	12m 50s
Counter	0,93	29,59	0,10	919.028	3,25	13m 53s
Garden	0,87	27,83	0,10	3.638.595	9,43	35m 35s
Kitchen	0,95	32,53	0,06	1.385.668	4,21	16m 39s
Stump	0,78	26,88	0,20	4.103.252	8,89	30m 59s

Cuadro 5.2: Resultados con la densificación activada

(a) Bicycle

(b) Bonsai

(c) Counter

(d) Garden

(e) Kitchen

(f) Stump

Ilustración 5.3: Reconstrucciones con densificación activada.

La Figura 5.3 presenta las reconstrucciones obtenidas tras entrenar las escenas con la densificación activada. A simple vista, se puede observar una mejora visuall significativa respecto a los resultados obtenidos sin densificación (Figura 5.2), especialmente en escenas con mayor complejidad geométrica.

La densificación consiste en añadir nuevas Gaussianas durante el entrenamiento, en regiones donde se detecta poca cobertura espacial o fotométrica; dividir gaussianas cuando estas son extremadamente grandes para así rellenar más espacios sin información y eliminarlas cuando la opacidad es tan baja como para no contibuir en la reconstrucción. Este proceso permite que la representación gane en precisión y en capacidad de reconstrucción, al incrementar la cantidad de gaussianas disponibles para aproximarse a la forma y el color de la escena.

Como se observa en la Tabla 5.2, esta mejora visual se refleja también en métricas objetivas como SSIM[29], PSNR y LPIPS[3], que experimentan una mejora notable respecto a sus valores sin densificación (Tabla 5.1). Además, el número de Gaussianas se incrementa de forma drástica, lo que supone un mayor uso de memoria y un incremento considerable en el tiempo de entrenamiento.

Este pequeño experimento evidencia que el proceso de densificación es clave dentro de 3DGS[5], siendo determinante para obtener una reconstrucciones más realistas al igual que detalladas. Tener la densificación activada permite al modelo adaptarse mejor a la escenas y capturar detalles más finos, lo cual es beneficioso en escenarios con estructuras más complejas.

5.2. Estudio paramétrico del método de densificación en 3DGS

La etapa de densificación incluye varios parámetros internos que afectan de forma directa al comportamiento del proceso. En concreto, existen tres hiperparámetros clave que inicialmente no están expuestos para su ajuste, pero que al ser modificados de forma interna en el propio código permiten estudiar el resultado de nuevas configuraciones del modelo. Estos parámetros son:

- **Número de Gaussianas Clonadas (N)**: Este valor determina en cuántas gaussianas va a ser dividida una gaussiana cuando tenga las condiciones necesarias para ser dividida. A mayor valor de N, más agresiva es la división, lo que a priori puede hacer más precisa la representación de la escena pero, también generar más puntos y por lo que, puede consumir un mayor almacenamiento y un valor de memoria VRAM mayor.
- **Umbral de Escala (Scale Threshold)**: Este umbral define a partir de qué valor de escala una gaussiana se considera lo suficientemente grande como para ser dividida. Una reducción supone una mayor sensibilidad al tamaño de la gaussiana, pues a menor valor mayor frecuencia de subdivisiones, por lo que afecta de forma directa al número de gaussianas generadas durante la densificación.
- **Opacidad Mínima (Min Opacity)**: Durante el entrenamiento, aquellas gaussianas cuya opacidad está por debajo de este umbral son candidatas para ser eliminadas. Un valor alto provoca un filtrado más estricto, reduciendo el número de gaussianas en la escena, y arriesgando de este modo el detalle de la escena, pues se va a eliminar un mayor número de gaussianas.

Estos tres parámetros tienen un gran peso en la precisión de los resultado en la reconstrucción de la escenala. Sin embargo, el impacto real no ha sido estudiado en profundidad, por lo que se propondrá a continuación.

5.2.1. Diseño experimental

Con el objetivo de analizar en profundidad el impacto de los parámetros clave en el proceso de densificación, y su relación tanto con la calidad final del modelo como con la eficiencia computacional, se ha diseñado un conjunto de experimentos.

Para ello, se han seleccionado las seis escenas pertenecientes al conjunto de datos utilizado (Bicycle, Bonsai, Counter, Garden, Kitchen y Stump). Sobre cada una de estas escenas se han realizado múltiples entrenamientos, variando de forma combinada los siguientes parámetros:

Cuadro 5.3: Parámetros evaluados durante los experimentos y valores considerados.

Parámetro	**Descripción**
N	Número de gaussianas generadas durante el proceso de clonación. Valores evaluados: $\{2, 3\}$.
scale_threshold	Umbral mínimo de escala necesario para permitir la clonación de una gaussiana. Valores evaluados: $\{15, 20, 25\}$.
opacity_threshold	Umbral mínimo de opacidad requerido para considerar una gaussiana como activa. Valores evaluados: $\{0{,}001, 0{,}005, 0{,}01\}$.

El número total de combinaciones posibles asciende a 18, como resultado de combinar los valores considerados para cada parámetro:

```
2 (N) x 3 (scale_threshold) x 3 (opacity_threshold) = 18 combinaciones
```

Cada una de estas configuraciones ha sido aplicada a las seis escenas seleccionadas, resultando en un total de 108 experimentos independientes.

La tabla 5.4 recoge todas las combinaciones de parámetros evaluadas:

N	scale_threshold	opacity_threshold
2	15	0.001
2	15	0.005
2	15	0.01
2	20	0.001
2	20	0.005
2	20	0.01
2	25	0.001
2	25	0.005
2	25	0.01
3	15	0.001
3	15	0.005
3	15	0.01
3	20	0.001
3	20	0.005
3	20	0.01
3	25	0.001
3	25	0.005
3	25	0.01

Cuadro 5.4: Combinaciones de parámetros evaluadas

Para cada combinación de parámetros y escena, se han registrado métricas de calidad visual (SSIM[29], PSNR y LPIPS). Además, en aquellos casos que han presentado mejores resultados en términos de calidad, se ha analizado también el consumo de memoria y el tiempo total de entrenamiento, con el objetivo de evaluar la eficiencia global de cada configuración y su impacto sobre los recursos computacionales disponibles.

5.2.2. Análisis exploratorio de métricas y parámetros

Con el objetivo de entender de forma visual y estadística el comportamiento de las métricas de calidad (SSIM, PSNR, LPIPS) y la relación que existe con los parámetros seleccionados del modelo, se llevó a cabo un análisis de los datos recogidos tras los entrenamientos.

En primer lugar, se analizaron las distribuciones globales de las tres métricas de calidad mediante histogramas con suavizado por densidad. Tal y como se muestra en la Figura 5.4, se observa una gran dispersión en los valores de SSIM, PSNR y LPIPS entre las diferentes combinaciones de parámetros evaluadas.

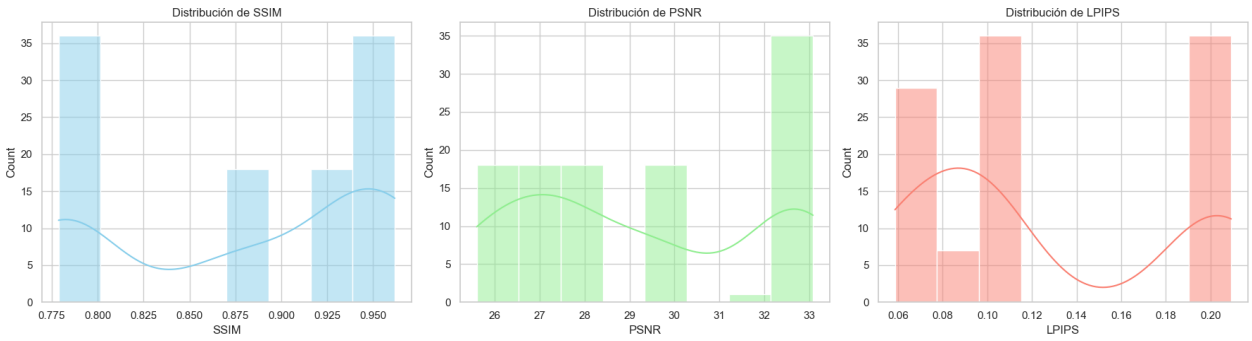


Ilustración 5.4: Distribución de las métricas SSIM, PSNR y LPIPS[3] para todas las combinaciones evaluadas.

En particular, se puede observar que las métricas SSIM y PSNR no siguen una única tendencia, sino que se agrupan en dos zonas principales. Esto puede sugerir que hay combinaciones de parámetros que generan resultados muy distintos entre sí, en cuanto a calidad. En el caso de `LPIPS`, los valores se concentran sobre todo en los extremos, lo que puede indica que algunas configuraciones generan reconstrucciones muy parecidas a la original, mientras que otros resultados presentan un claro empeoramientto

Como segundo paso, se representaron las métricas de calidad mediante diagramas de caja (*boxplots*) por escena, con el objetivo de analizar cómo varía el resultado del modelo en función del dataset utilizado. Como se observa en la Figura 5.5, escenas como Bonsai y Kitchen presentan valores de SSIM y PSNR consistentemente altos, junto con valores bajos de LPIPS, lo que puede indicar que estas escenas son más fáciles de reconstruir con buenos resultados y estar relacionado con la geometría o la cantidad de puntos presentes en la inicialización.

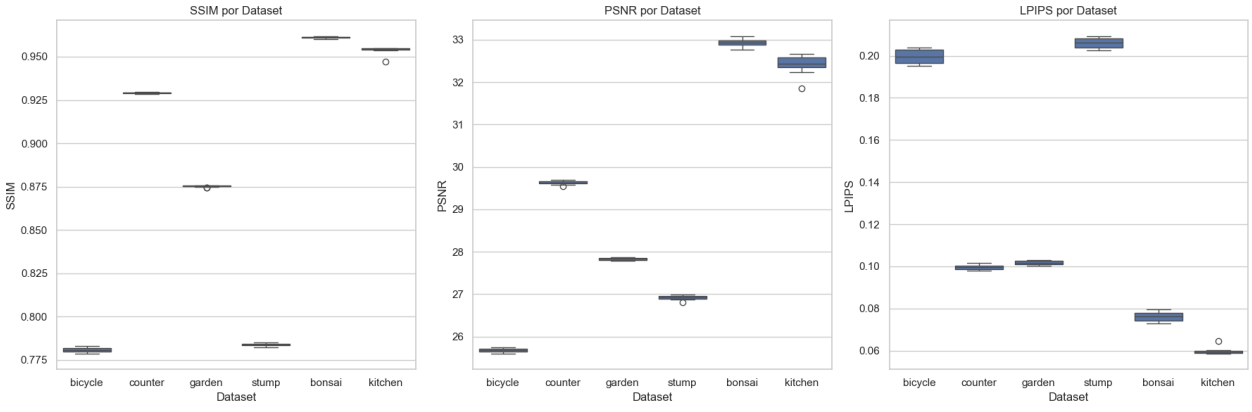


Ilustración 5.5: Distribución de las métricas SSIM, PSNR y LPIPS para cada escena.

5.2.3. Selección de mejores combinaciones por escena

Con el objetivo de identificar la mejor combinación de parámetros para cada escena, se prestará atención a las métricas de calidad obtenidas (PSNR, SSIM y LPIPS), teniendo en cuenta que un menor valor de LPIPS indica una cuan similar es la percepción entre la imagen reconstruida y la original.

Finalmente, se seleccionó la mejor configuración para cada escena, identificando aquella que obtuvo con la mejor combinación de valores.

Se observa que en la Tabla 5.5 las mejores configuraciones obtenidas corresponden al caso en que el valor de N es igual a 3. Esto tiene sentido, ya que un mayor número de gaussianas generadas tras cada división permite al modelo capturar con mayor precisión los detalles de la escena, afinando así los pequeños detalles, y con ello, la propia escena.

Escena	**Opacity**	**N**	**Scale**	**SSIM**	**PSNR**	**LPIPS**	**Max Mem (GB)**	**Tiempo**
Bicycle	0.001	3	15	0.78	25.75	0.2	11.7	52m 13s
Bonsai	0.001	3	20	0.96	33.08	0.07	4.02	22m 10s
Counter	0.001	3	20	0.93	29.66	0.1	3.42	20m 02s
Garden	0.005	3	20	0.88	27.85	0.1	9.8	47m 12s
Kitchen	0.001	3	20	0.95	32.67	0.06	4.78	27m 13s
Stump	0.005	3	20	0.78	26.99	0.20	9.12	50m 51s

Cuadro 5.5: Mejores combinaciones por escena según promedio de calidad visual.

Sin embargo, al comparar estos resultados con los obtenidos en la configuración base de 3DGS[5] (sin variación de parámetros), se aprecia un incremento notable en el consumo de memoria VRAM y en el tiempo total de entrenamiento. Esto también es esperable, dado que al generar más gaussianas se incrementa la cantidad de datos que deben ser almacenados y procesados durante cada iteración.

Por este motivo, se decide llevar a cabo un análisis más equilibrado, por lo que se procederá a filtrar nuevamente los resultados considerando únicamente aquellas combinaciones en las que el valor de N es igual a 2.

En la Tabla 5.6 se muestran las mejores configuraciones obtenidas para cada escena bajo esta restricción. Si bien los valores de las métricas de calidad son, en general, ligeramente inferiores a los alcanzados con N = 3, representan configuraciones mucho más eficientes, tanto desde el punto de vista de recursos, como el almacenamiento y la memoria VRAM, debido al descenso en el número de gausianas, como en el propio tiempo de entrenamiento.

Escena	Opacity	N	Scale	SSIM	PSNR	LPIPS	Max Mem (GB)	Tiempo
Bicycle	0.001	2	15	0.77	25.71	0.20	11.2	38m 12s
Bonsai	0.005	2	25	0.96	33.07	0.07	3.62	12m 02s
Counter	0.001	2	20	0.92	29.69	0.09	3.18	13m 35s
Garden	0.001	2	20	0.87	27.86	0.10	9.38	35m 27s
Kitchen	0.005	2	25	0.95	32.56	0.05	4.18	16m 18
Stump	0.010	2	25	0.78	26.92	0.20	8.78	30m 38s

Cuadro 5.6: Mejores combinaciones por escena considerando únicamente $N = 2$.

5.2.4. Conclusiones del análisis paramétrico

Los resultados que se han obtenido, refleja el gran impacto que tienen estos parámetros menos accesibles a nivel usuario para la densificaicón en 3DGS[5]. En particular, se confirma que un mayor número de gaussianas por división (N = 3) permite afinar los detalles y obtener métricas superiores, aunque a costa de un aumento considerable en el uso de memoria VRAM y el tiempo de entrenamiento.

Por otro lado, filtrar el análisis por N = 2, se observan combinaciones de parámetros más eficientes en términos de recursos, con una pérdida asumible de calidad visual en muchos casos.

Un punto interesante que se puede concluir a partir de las Tablas 5.5 y 5.6 es que se presentan todos los valores posibles para los parámetros `opacity_threshold` y `scale_threshold`. No existe una única combinación para todas las escenas; por el contrario, cada dataset parece obtener mejores resultados a partir de valores distintos para estos umbrales. Este hecho puede sugerir que el uso de umbrales estáticos y deterministas podría no ser lo más adecuada para un buen resultado general en todos los casos.

Por tanto, una posible mejora futura podría consistir en explorar mecanismos más dinámicos para determinar estos valores, basados en características específicas de cada escena. Esto podría permitir mantener la calidad de las reconstrucciones reduciendo el sobrecoste computacional.

5.3. Hipótesis 1: Entrenamiento en 3DGS basado en parches

5.3.1. Introducción

A lo largo de la historia de la informática, y más concretamente de la inteligencia artificial, se ha demostrado que el uso de fragmentos de datos de conjutnos mayores es una estrategia que ha arrojado buenos resultados en cuanto a la eficiencia y la capacidad de generalización. Esta idea ha sido aplicado en campos tan convencionales como el procesamiento de imágenes, el análisis de audio y texto.

Si se profundiza en la visión por computador, por ejemplo, es muy común deividr una imagen en parches más pequeños para poder ser procesados de manera independientes. De este modo, los modelos aprenden patrones locales con una mayor precisión y pueden entrenarse en dispositivos cuyos recursos son mñas limitados. Esta técnica también es usada en campos como videojuegos, donde se cargan chunks de información dependiendo de la posición del jugador o, incluso en la medicina, donde se analizan fragmentos de una radiografías para poder detectar anomalías.

5.3.2. Uso de los parches en campos más específicos

En del aprendizaje profundo, el uso de subconjuntos de datos durante el entrenamiento ha jugado un papel fundamental dando lugar a avances significativos, como se recoge artículos como An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale [Dosovitskiy et al., 2020] [30], donde se introduce el modelo Vision Transformer (ViT), que divide las imágenes en parches fijos de 16×16 píxeles para procesarlas como secuencias de tokens, logrando resultaods prometedores sin necesidad de convoluciones.

Otro caso destacado es el de los métodos de aprendizaje auto-supervisado, como SimCLR [Chen et al., 2020] [31], que utilizan crops aleatorios (subimágenes) como vistas alternativas de una misma imagen para aprender representaciones robustas sin etiquetas.

Asimismo y un ejemplo más resonado en los últimos años, en la reconstrucción de escenas 3D, modelos como NeRF [13] [Mildenhall et al., 2020] no procesan las imágenes completas, sino que seleccionan rayos individuales (píxeles muestreados) en cada iteración.

5.3.3. Aplicación en 3D Gaussian Splatting

Basandose en los enfoques descritos, se propone como primera hipótesis plantear un entrenamiento alternativo basado en parches para 3DGS[5], mediante el uso de parches aleatorios de las imagenes usadas para el proceso de entrenamiento. En vez de renderizar la imagen completa en cada iteración se selecciona una región de la imagen, y se calcula la pérdida únicamente sobre esa región.

Con esto, se intentará reducir el coste de los recursos, evitar que se generen tantas gaussianas durante el proceso de densificación y forzar al modelo a aprender a partir de menos información.

5.3.4. Implementación técnica

A partir de la hipótesis planteada, se implementa un entrenamiento basado en parches aleatorios dentro del pipeline original de 3DGS[5]. El objetivo es que el modelo aprenda a representar la escena a partir de fragmentos más pequeños de las imágenes, en lugar de procesar cada imagen completa en cada iteración.

División en parches Cada imagen utilizada para el entrenamiento, de dimensiones $H \times W$, se divide en n parches. Esta división tiene lugar al comienzo de cada iteración, haciendo uso de la función divide_and_select_patch(width, height, num_patches), y posteriomente seleccionándose un parche de forma aleatoria para cada iteración con el fin de calcular la pérdida correspondiente.

La Figura 5.6 muestra un ejemplo visual del proceso de división para distintas configuraciones: cuando $n = 1$ la imagen se procesa completa, mientras que para $n = 2$ y $n = 4$ se subdivide en partes más pequeñas, cada una identificada numéricamente.

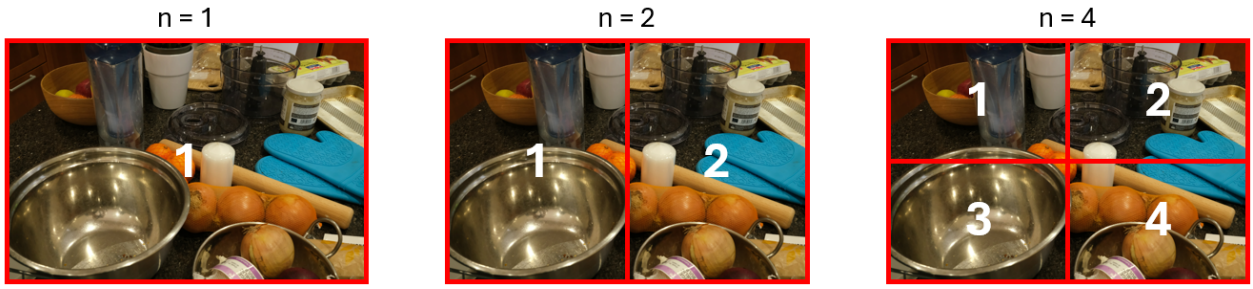


Ilustración 5.6: Ejemplo de división de una imagen en diferentes configuraciones de parches.

Por ejemplo, si $n = 4$, la imagen se divide en cuatro parches, y únicamente se utiliza uno de ellos por iteración.

Implementación del sistema de parches La lógica de división de las imágenes en parches está implementada en la función divide_and_select_patch(width, height, num_patches), incluida en el módulo patches_utils.py del proyecto. Esta función toma como entrada el ancho y alto de la imagen, junto con el número de parches deseado ($n \in \{1, 2, 4\}$), y devuelve una lista de coordenadas que definen los distintos parches.

```python
def divide_and_select_patch(width, height, num_patches):
    if num_patches not in [1, 2, 4]:
        raise ValueError("num_patches debe ser uno de los siguientes valores: 1, 2, 4")

    if num_patches == 1:
        patches = [[0, 0, width, height]]
    elif num_patches == 2:
        patches = [[0, 0, width // 2, height], [width // 2, 0, width, height]]
    elif num_patches == 4:
        patches = [
            [0, 0, width // 2, height // 2],
            [width // 2, 0, width, height // 2],
            [0, height // 2, width // 2, height],
            [width // 2, height // 2, width, height]
        ]

    return patches
```

Ilustración 5.7: Codigo del utils

Internamente, la función divide la imagen en regiones de igual tamaño. Por ejemplo:

- **Para** $n = 1$: se devuelve un único parche que abarca toda la imagen.
- **Para** $n = 2$: se divide la imagen verticalmente en dos mitades iguales.
- **Para** $n = 4$: se genera una cuadrícula de 2×2, dividiendo la imagen en cuatro regiones.

Este proceso se ilustra en la Figura 5.8, donde se compara una imagen completa con el parche seleccionado para el cálculo de la pérdida. Como se puede observar, solo se utiliza una parte de la imagen renderizada en cada iteración.

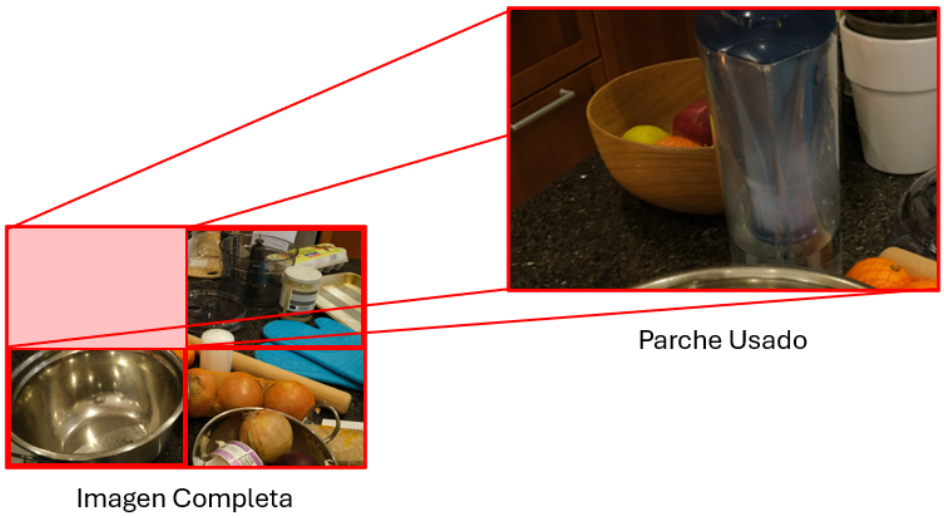


Ilustración 5.8: Comparativa entre una imagen renderizada completa (izquierda) y el parche seleccionado para calcular la pérdida (derecha).

Proceso de renderizado y cálculo de pérdida Durante cada iteración del entrenamiento, se sigue un flujo bien, el cual se muestra en la Figura 5.9. Este flujo muestra el proceso

que permite calcular la pérdida sobre un parche de la imagen.

- Se selecciona aleatoriamente una cámara del conjunto de entrenamiento.
- Se renderiza la imagen desde esa vista, obteniendo la imagen renderizada `image_rendered`.
- Se selecciona aleatoriamente un parche $P = (x_1, y_1, x_2, y_2)$ dentro de la imagen.
- Se recorta el parche renderizado: image_rendered[:, y_1:y_2, x_1:x_2].
- Se extrae el parche correspondiente de la imagen real (ground truth): gt_image[:, y_1:y_2, x_1:x_2].
- Se calcula la pérdida entre ambos parches, por ejemplo con una combinación de L1 y SSIM:

```
patch_rendered = image_rendered[:, y1:y2, x1:x2]
patch_gt = gt_image[:, y1:y2, x1:x2]
loss = L1(patch_rendered, patch_gt)
```

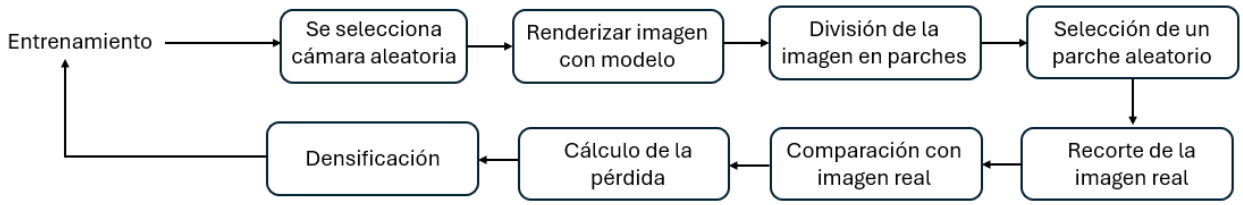


Ilustración 5.9: Flujo de trabajo del sistema de entrenamiento por parches en 3D Gaussian Splatting.

Parámetro `patch_number` El número de parches utilizado durante el entrenamiento se controla mediante un nuevo parámetro –patch_number, que se pasa como argumento desde la línea de comandos al lanzar el script de entrenamiento. Este valor se recoge en el archivo principal, se almacena en los argumentos globales del programa y se utiliza posteriormente para invocar a la función divide_and_select_patch. De esta forma, el usuario puede determinar en cuantos parches se dividen las imágenes de entrenamiento sin modificar el código fuente.

```
parser.add_argument("--densification", type=str, default="true") # TFG: Se añaden el parametro densification para indicar si se desea o no densificacion
parser.add_argument("--patch_number", type=int, default=1) # TFG: Se añaden el parametro patch_number para indicar el numero de parches a usar
```

Ilustración 5.10: Parámetro que indicar el número de parches

5.3.5. Resultados

Los resultados obtenidos reflejan un aumento significativo en el número de Gaussianas, respecto al entrenamiento base. En lugar de limitarse, el modelo produjo una explosión del número de estos puntos, incluso más que en 3D Gaussian Base.

En la Tabla 5.7 se muestran los valores promedios obtenidos en función del número de parches utilizados durante el entrenamiento. Se comparan las métricas de calidad visual (SSIM, PSNR y LPIPS), junto con el número total de gaussianas generadas, la memoria VRAM utilizada y el tiempo de entrenamiento.

Patch Number	**SSIM**	**PSNR**	**LPIPS**	**Nº Gaussianas**	**Max Mem (GB)**	**Tiempo**
3DGS	0.88	29.26	0.12	2.623.200	6.77	24m 24s
1	0.88	29.25	0.12	2.623.500	6.8	24m 55s
2	0.88	28.90	0.13	3.272.000	7.75	26m 30s
4	0.87	28.19	0.15	3.855.000	8.68	28m 52s

Cuadro 5.7: Promedios globales obtenidos en función del número de parches usados durante el entrenamiento.

A partir de los resultados obtenidos, se observa un incremento generalizado en el número de gaussianas generadas, así como un mayor consumo de memoria VRAM y un aumento en el tiempo de entrenamiento. Por otro lado, aunque las métricas de calidad visual como SSIM[29], PSNR y LPIPS[3] muestran un descenso, esta no es especialmente drástico. Esto puede sugerir que, a pesar de trabajar con fragmentos cada vez más pequeños de la imagen en lugar de la imagen completa, el modelo mantiene, dentro de lo que cabe, una capacidad razonable de generalización.

Hipótesis Sin embargo, volviendo al aumento que se observa en el número de gaussianas, así como en el uso de memoria VRAM y el tiempo de entrenamiento, se plantean tres posibles hipótesis que podrían explicar este comportamiento:

1. **Priorización del detalle local**: al trabajar con secciones más pequeñas de la imagen (parches), el modelo podría estar forzando una representación más precisa de los detalles locales, lo que provoca una generación mayor de gaussianas para cubrir con precisión esas zonas limitadas.
2. **Densificación innecesaria**: es posible que se estén densificando gaussianas que no contribuyen significativamente a la pérdida, es decir, gaussianas que no han recibido gradiente en la vista actual, pero que aún así participan en el proceso de densificación. Este fenómeno se representa en la Figura 5.11, donde se observa cómo ciertas gaussianas ajenas al parche activo acumulan estadísticas sin haber sido optimizadas directamente.
3. **Superposición entre parches**: dado que una misma gaussiana puede ocupar un área grande, podría estar jugando un papel en múltiples parches a lo largo del entrenamiento. Esto puede provocar acumulación de gradiente redundante y, por tanto, un aumento innecesario del número de gaussianas. Un ejemplo visual se presenta en la Figura 5.12, donde se ilustra cómo una gaussiana puede abarcar varias regiones de la imagen.

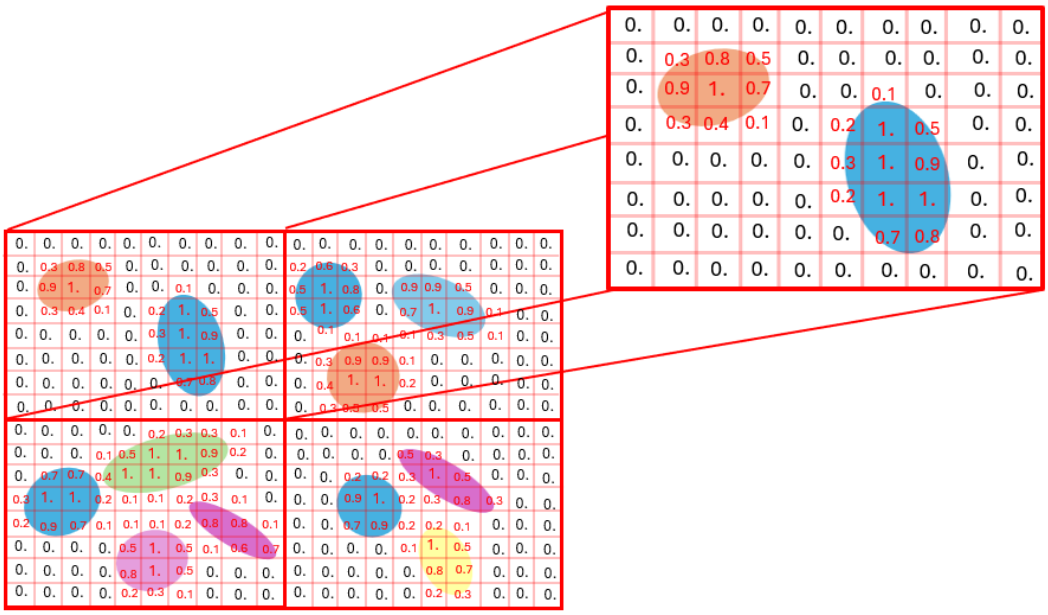


Ilustración 5.11: Hipótesis 2 — Densificación innecesaria: gaussianas fuera del parche visible acumulan estadísticas de densificación a pesar de no haber recibido gradiente significativo.

Ilustración 5.12: Hipótesis 3 — Superposición entre parches: las gaussianas grandes pueden abarcar varios parches, contribuyendo a la redundancia en el entrenamiento.

5.4. Hipotesis 2: Diferentes mecanismos de densificación

5.4.1. Introducción

Como segunda hipótesis, se plantea el estudio de métodos alternativos ya existentes que abordan problemas similares, con el objetivo de conseguir mecanismos que permitan implementar una solución al problema del crecimiento excesivo del número de gaussianas durante la generación de escenas. Esto conlleva un aumento significativo en el consumo de memoria gráfica, almacenamiento y tiempo de entrenamiento, lo que dificulta su adopción en entornos con recursos mas casuales.

5.4.2. Ecuación para limitar el crecimiento de las gaussianas

En el artículo de *Taming 3DGS*[19] se introduce un mecanismo determinista para controlar el crecimiento del número de gaussianas a lo largo del proceso de entrenamiento. Este enfoque no se basa únicamente en establecer un límite, sino en distribuir de forma progresiva y predecible la cantidad máxima de gaussianas permitidas en cada paso del entrenamiento, asegurando así que no supere presupuesto total de gaussianas para todo el entrenamiento.

Para ello, se define una función cuadrática que actúa como guía para la densificación, asignando en cada iteración un número máximo de gaussianas que el sistema puede generar o densificar. Esta fórmula determinaría el número máximo de gaussianas que debe haber presente en cada uno de los pasos para llegar de forma progresiva al objetivo final.

La ecuación propuesta es la siguiente:

$$g(x) = \left(\frac{B - S - kN}{N^2} \right) x^2 + kx + S \tag{5.1}$$

donde:

- $g(x)$: número máximo de gaussianas permitidas en la iteración x.
- S: número de gaussianas iniciales.
- B: número total de gaussianas permitidas al final del entrenamiento (presupuesto global).
- N: número total de pasos (iteraciones) de entrenamiento.
- x: paso actual del entrenamiento.
- k: pendiente inicial del crecimiento, que se define como:

$$k = \frac{2(B - S)}{N} \tag{5.2}$$

Deducción de la ecuación

La función de crecimiento se obtiene a partir de un polinomio cuadrático general:

$$g(x) = ax^2 + bx + c \tag{5.3}$$

Se aplican las siguientes condiciones:

1. En el paso inicial $x = 0$, debe cumplirse que $g(0) = S$, pues en el primer paso solo pueden existir las gaussianas primitivas designadas como puntos inicializados:

$$g(0) = a \cdot 0^2 + b \cdot 0 + c = S \Rightarrow c = S \tag{5.4}$$

2. En el paso final $x = N$, debe cumplirse que $g(N) = B$, pues el valor de las gaussianas debe ser igual al presupeusto de gaussianas designado para el entrenamiento:

$$g(N) = aN^2 + bN + S = B \Rightarrow a = \frac{B - S - bN}{N^2} \tag{5.5}$$

3. Se define $b = k$ para obtener una expresión en función de un parámetro de crecimiento ajustable:

$$a = \frac{B - S - kN}{N^2} \tag{5.6}$$

4. El valor de k se elige como:

$$k = \frac{2(B - S)}{N} \tag{5.7}$$

   El valor de k asegura que la curva de crecimiento comience con una pendiente inicial elevada, permitiendo al modelo densificar rápidamente durante las primeras fases del entrenamiento, donde la capacidad de representación aún es limitada. A medida que el entrenamiento avanza, esta pendiente decrece de forma natural gracias al componente cuadrático, lo que ofrece un comportamiento suave y progresivo.

Sustituyendo a, $b = k$ y $c = S$ en la ecuación original se obtiene la expresión final de la función de crecimiento (Ec. 5.1).

Esto proporciona un control más fino y predecible sobre el número de gaussianas, asegurando que se mantenga dentro del límite designado por el usuario y distribuyéndolas con lógica creciente a lo largo del entrenamiento.

5.4.3. Influencia del parámetro k sobre la curva de crecimiento

La ecuación cuadrática planteada en la Sección 5.1 permite controlar de forma progresiva el número máximo de gaussianas que el modelo puede generar en cada paso del entrenamiento. Esta función depende de un parámetro de crecimiento k, que condiciona la pendiente inicial de la curva y, por tanto, determina cuán rápido se permite densificar la escena.

Para comprender cómo influye este parámetro en la evolución del número de gaussianas, se ha realizado un graficado experimental para distintos valores de k, manteniendo fijos el número de pasos (N), las gaussianas iniciales (S) y el presupuesto total (B). La Figura 5.13 muestra la evolución del número máximo de gaussianas permitidas en función del paso de entrenamiento, comparando distintos valores de k.

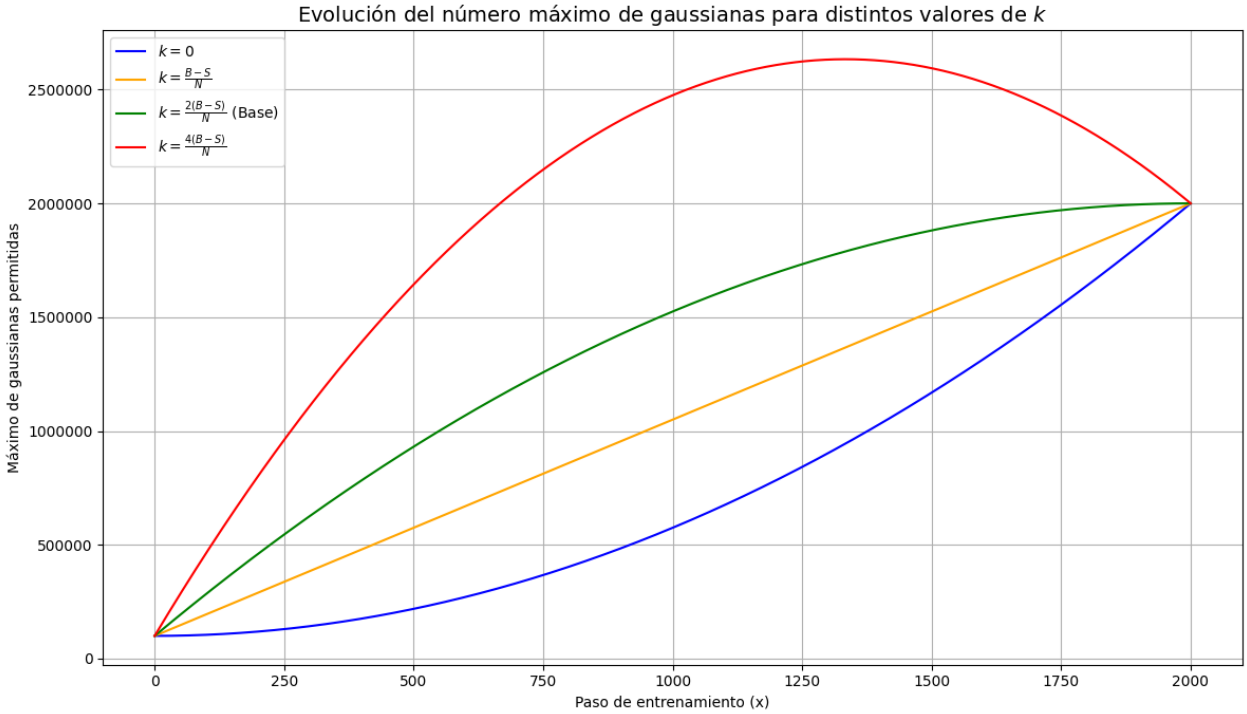


Ilustración 5.13: Evolución del número máximo de gaussianas permitidas en función del paso de entrenamiento para distintos valores de k.

A partir de la Figura 5.13 se pueden extraerse las siguientes conclusiones:

- $k = 0$: genera una curva puramente cuadrática con crecimiento lento al inicio. Esto puede dificultar que el modelo capture la estructura inicial de la escena, ya que dispone de pocas gaussianas en las primeras etapas. Solo aquellas escenas con un mayor porcentaje de puntos iniciales, saldrán beneficiadas.
- $k = \frac{B-S}{N}$: ofrece un crecimiento más moderado y lineal. Es útil para entornos donde se desea evitar una explosión temprana de gaussianas, aunque puede no ser suficiente en escenas complejas.

- $k = \frac{2(B-S)}{N}$ **(base)**: se muestra como una opción equilibrada. Refleja un crecimiento progresivo que alcanza el presupuesto de gaussianas al final del entrenamiento sin saturar el sistema en etapas iniciales.
- $k = \frac{4(B-S)}{N}$: provoca un crecimiento muy agresivo, alcanzando el máximo de gaussianas demasiado pronto. Esto puede hacer explotar el consumo memoria y producir gaussianas antes de que el modelo haya convergido lo suficiente como para optimizarlas eficazmente.

Por tanto, se concluye que el valor $k = \frac{2(B-S)}{N}$ representa una elección razonable para la mayoría de casos.

5.4.4. Simulación del comportamiento de la función

Con el objetivo de analizar cómo se comporta la función propuesta en diferentes escenarios, se ha desarrollado un pequeño script en Python[6] que permite simular la evolución del número máximo de gaussianas permitidas a lo largo del entrenamiento. Este programa implementa la ecuación 5.1 y grafica su evolución para diferentes combinaciones de parámetros S, B y N

La Figura 5.14 muestra una comparativa entre distintas curvas generadas al modificar el presupuesto total de gaussianas B, el número de puntos iniciales S y la duración del entrenamiento N.

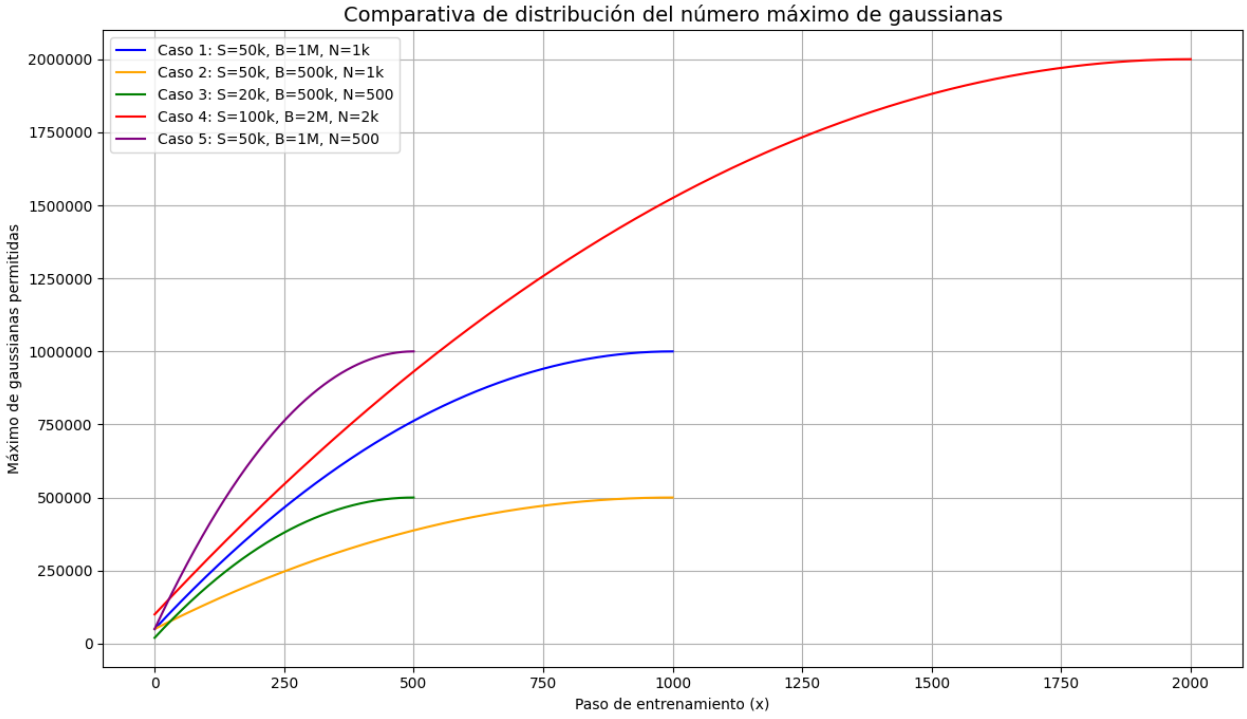


Ilustración 5.14: Comparativa entre diferentes curvas de crecimiento del número máximo de gaussianas para distintos presupuestos y duraciones de entrenamiento.

- **Caso 1:** con $S = 50\,000$, $B = 1\,000\,000$, y $N = 1000$, se observa un crecimiento progresivo y equilibrado.
- **Caso 2:** con $S = 50\,000$, $B = 500\,000$ y $N = 1000$, el crecimiento es más limitado, lo que reduce significativamente el número de gaussianas finales. Este caso representa un escenario en el que se busca reducir el uso de recursos manteniendo la misma duración de entrenamiento.
- **Caso 3:** con un valor inicial reducido de $S = 20\,000$, $B = 500\,000$ y $N = 500$, se simula un entrenamiento más corto con presupuesto bajo. El crecimiento es rápido desde el inicio, útil para dispositivos con limitaciones más estrictas.
- **Caso 4:** con $S = 100\,000$, $B = 2\,000\,000$ y $N = 2000$, se permite mayor grado de crecimiento. Esta configuración es ideal para entornos de alto rendimiento, donde se prioriza la calidad final por encima del tiempo o el consumo de memoria.
- **Caso 5:** con $S = 50\,000$, $B = 1\,000\,000$ y $N = 500$, se mantiene el mismo presupuesto que en el Caso 1, pero en la mitad de iteraciones, lo que da como resultado un crecimiento mucho más agresivo del número de gaussianas.

5.4.5. Integración de la ecuación en el código de 3DGS

La ecuación comentada anteriomente ha sido implementada dentro del proceso de densificación. Para ello, se ha añadido una función específica llamada calculate_gaussian_budget(), encargada de calcular el número máximo de gaussianas permitidas en cada paso del entrenamiento según la ecuación 5.1.

Esta función tiene como parámetros el paso actual (current_step), el número total de pasos de entrenamiento (total_steps), el número de puntos iniciales (initial_points) y el presupuesto global de gaussianas (budget). A partir de estos valores, calcula el límite de cada paso de gaussianas que deben estar presentesf.

```python
def calculate_gaussian_budget(self, current_step, total_steps, initial_points, budget):
    if current_step > total_steps:
        return budget
    x = current_step
    N = total_steps
    B = budget
    S = initial_points
    k = (2 * (B - S)) // N
    target = int((B - S - (k*N) ) / N**2) * x**2 + k * x + S
    return target
```

Ilustración 5.15: Función calculate_gaussian_budget() implementando la ecuación de crecimiento progresivo de gaussianas.

El valor que devuleve esta función se usa durante la ejecución de la función densify_and_prune(), que se invoca en cada paso de entrenamiento dentro de la densificación. Primero se calcula

el paso actual y el número total de pasos válidos para la densificación, y luego se llama a la función junto con el presupuesto introducido.

```
if iteration > opt.densify_from_iter and iteration % opt.densification_interval == 0:
    size_threshold = 20 if iteration > opt.opacity_reset_interval else None

    total_steps = opt.densify_until_iter - opt.densify_from_iter # TFG: Numero total de pasos
    current_step = max(0, iteration - opt.densify_from_iter) # TFG: Numero de paso actual

    gaussians.densify_and_prune(
        opt.densify_grad_threshold,
        min_opacity=0.005,
        extent=scene.cameras_extent,
        max_screen_size=20,
        radii=radii,
        current_step=current_step, # TFG: Paso en el que me encuentro ahora mismo
        total_steps=total_steps, # TFG: Pasos totales
        budget=introduced_budget # TFG: Numero de Guassianas máximo
    )
```

Ilustración 5.16: Fragmento de código donde se calcula el paso actual e invoca la función densify_and_prune() con el valor de presupuesto.

Finalmente, dentro de la función densify_and_prune() se obtiene el número de puntos iniciales y se calcula el número objetivo de gaussianas mediante calculate_gaussian_budget().

```
def densify_and_prune(self, max_grad, min_opacity, extent, max_screen_size, radii,  current_step, total_steps, budget):
    grads = self.xyz_gradient_accum / self.denom
    grads[grads.isnan()] = 0.0

    self.tmp_radii = radii

    initial_points = self.get_xyz.shape[0]
    target_gaussians = self.calculate_gaussian_budget(current_step, total_steps, initial_points, budget)

    self.densify_and_clone(grads, max_grad, extent)
    self.densify_and_split(grads, max_grad, extent)

    prune_mask = (self.get_opacity < min_opacity).squeeze()
    if max_screen_size:
        big_points_vs = self.max_radii2D > max_screen_size
        big_points_ws = self.get_scaling.max(dim=1).values > 0.1 * extent
        prune_mask = torch.logical_or(torch.logical_or(prune_mask, big_points_vs), big_points_ws)

    tmp_radii = self.tmp_radii
    self.tmp_radii = None

    torch.cuda.empty_cache()
```

Ilustración 5.17: Implementación del presupuesto de gaussianas dentro de la función densify_and_prune()

5.5. Parametrización del valor máximo de gaussianas

Para asegurar que el sistema tenga un mayor controls sobre el número total de gaussianas que dbee contener la escena generada tras el entrenameinto, se añade un nuevo parámetro denominado –budget. Este parámetro representa el número máximo de gaussianas que el modelo puede llegar a tener al finalizar el proceso de entrenamiento y densificación. De esta

manera, el usuario podrá determinar de forma explícita el número de gaussianas máximo, determinando este la prioridad entre uso de recursos y calidad del resultado.

La adición de este nuevo parámetro dse hizo de la siguiente forma:

```
parser.add_argument("--budget", type=int, default=1000000)
```

El valor definido por este parámetro se utiliza como presupuesto global de gaussianas (B) en la ecuación cuadrática presentada anteriormente en la Sección 5.1, que regula el crecimiento progresivo de gaussianas durante el entrenamiento.

5.6. Control del crecimiento: experimentación con presupuesto fijo

Con el objetivo de comprobar cuál es el comportamiento del modelo al incorporar la ecuación de crecimiento de gaussianas, se ha llevado a cabo una serie de experimentos, utilizando como presupuesto B los valores máximos de gaussianas generadas en las ejecuciones originales de 3DGS[5] con densificación activada Estos valores, que servirán como límite superior para el número de gaussianas permitidas durante el entrenamiento, se detallan en la Tabla 5.8.

Escena	**Budget**
Bicycle	4.652.308
Bonsai	1.040.856
Counter	919.028
Garden	3.638.595
Kitchen	1.385.668
Stump	4.103.252

Cuadro 5.8: Presupuesto máximo de gaussianas por escena, basado en los valores generados por 3DGS base.

A pesar de establecer un número máximo de gaussianas, los resultados reflejan que el número real de gaussianas generadas sobrepasa el valor objetivo. Esto indica que, si bien el modelo es capaz de calcular el límite teórico, como se observó en los experimentos básicos, no existe aún un mecanismo efectivo que limite la creación de nuevas gaussianas una vez alcanzado dicho presupuesto.

La Tabla 5.9 resume los resultados obtenidos al aplicar únicamente la ecuación cuadrática para controlar el crecimiento, sin incorporar técnicas de poda adicionales. Como se puede observar, en todas las escenas el número final de gaussianas excede el presupuesto previamente definido, acompañado de un mayor consumo de memoria y tiempo de entrenamiento.

Escena	SSIM	PSNR	LPIPS	Nº Gaussianas	Memoria Máxima (GB)	Tiempo
Bicycle	0.78	25.72	0.20	5.648.547	12.81	44m 24s
Bonsai	0.96	33.08	0.08	1.330.141	4.19	15m 34s
Counter	0.93	29.67	0.10	1.167.766	3.63	15m 50s
Garden	0.88	27.97	0.10	4.057.787	10.14	40m 12s
Kitchen	0.96	33.06	0.06	1.493.873	4.40	18m 9s
Stump	0.79	26.99	0.21	4.516.967	9.64	34m 35s

Cuadro 5.9: Resultados tras aplicar únicamente la ecuación cuadrática para controlar el crecimiento, sin mecanismos adicionales de poda.

Este fenómeno se debe a que la ecuación cuadrática únicamente indica el número máximo deseado de gaussianas en cada iteración, pero no impide explícitamente que el sistema siga creando más. La densificación continúa añadiendo nuevas gaussianas en cada paso sin un control estricto que verifique si se ha excedido el objetivo actual.

En otras palabras, el modelo conoce el valor de referencia $g(x)$, pero no actúa en consecuencia cuando lo sobrepasa, lo cual provoca un crecimiento descontrolado en las etapas del entrenamiento.

Para ilustrar mejor esta hipótesis, se ha generado una figura que muestra el comportamiento teórico del modelo cuando no existe un mecanismo de poda. En ella se representa el crecimiento deseado (presupuesto), el crecimiento real del sistema, y el exceso acumulado que se produce cuando se ignora el límite establecido.

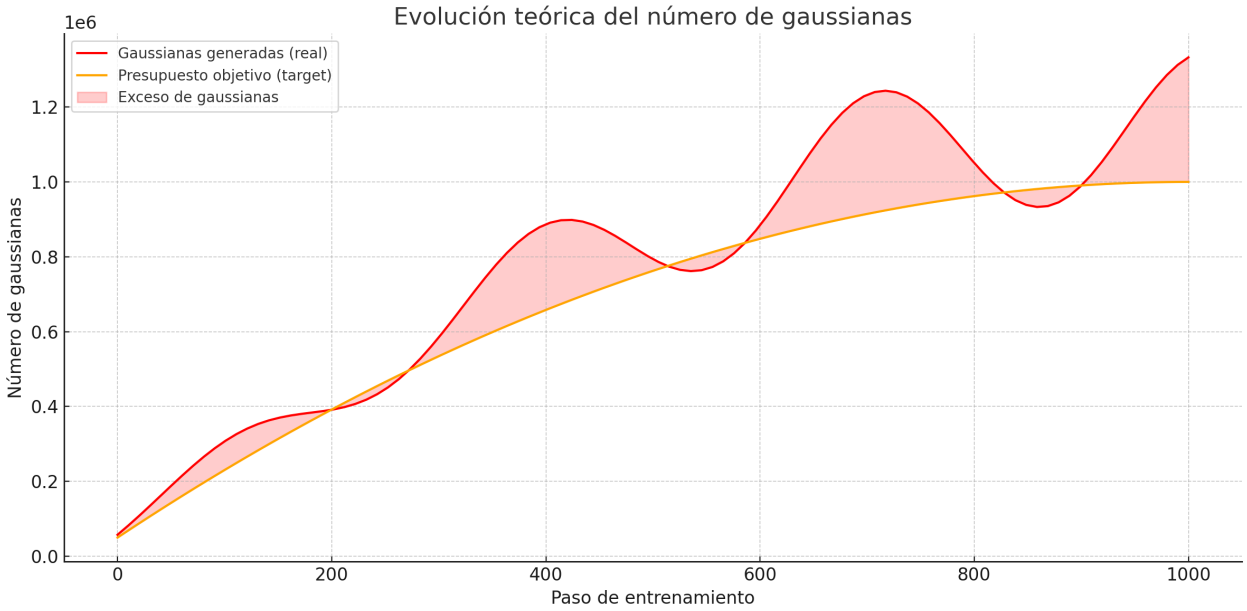


Ilustración 5.18: Representación teórica del crecimiento del número de gaussianas sin control de presupuesto. El área sombreada en rojo representa el exceso de gaussianas acumuladas que deberían ser podadas.

Por ello, es conveniente desarrollar un mecanismo adicional de control, que elimine de forma activa el exceso de gaussianas generadas para ajustarse al límite. Esta idea se abordará en el siguiente apartado.

5.7. Planteamiento de la poda

En el aprendizaje automático, la poda es una técnica utilizada para reducir la complejidad de los modelos, eliminar redundancia y mejorar la eficiencia computacional.

Si se aplica esta idea a 3DGS[5], la poda hace referencia a eliminar gaussianas que se consideran innecesarias o excedentes, ya sea porque se tiene que lograr un objetivo de gaussianas o porque no son suficientemente significativas para la reconstrucción de la escena.

En el apartado anterior se observó que, tras implementar un mecanismo de crecimiento controlado mediante una función cuadrática, el número real de gaussianas generadas puede superar el objetivo establecido en cada iteración. Esto ocurre porque, aunque se calcula correctamente el presupuesto $g(x)$ en cada paso, el sistema no elimina el exceso de gaussianas ya existentes, provocando que el número total siga creciendo.

Por ello, se debe introducir un mecanismo adicional de poda que actúe cuando se detecte un exceso de gaussianas respecto al valor objetivo. Este mecanismo podría consistir en:

1. Calcular cuántas gaussianas hay actualmente en el modelo.
2. Obtener el número máximo deseado según el paso actual del entrenamiento (target).
3. Determinar el exceso:

$$\texttt{exceso\_gaussianas} = \texttt{número\_actual} - \texttt{target\_gaussians}$$

4. Si hay exceso, eliminar ese número de gaussianas del sistema.

Este proceso se puede llevar a cabo utilizando distintos criterios de selección para decidir qué gaussianas eliminar.

Dado que el modelo no dispone de un mecanismo para eliminar gaussianas excedentes, en el siguiente apartado se propone una primer estrategia de poda basada en la opacidad de las gaussianas.

Esta decisión surge a raíz de los análisis realizados en secciones anteriores, donde se observó que el parámetro `min_scale` tiene una gran influencia en los resultados, pero su valor óptimo varía entre escenas. Por tanto, en lugar de utilizar este parámetro como un umbral fijo durante la densificación, se plantea emplearlo como criterio dinámico para seleccionar las gaussianas menos relevantes (aquellas con menor escala) cuando sea necesario reducir su número.

5.8. Implementación de un mecanismo de poda basado en escala

El primer intento para limitar el número de gaussianas durante el entrenamiento es un mecanismo de poda basado en la escala de cada gaussiana. El planteamiento es que las gaussianas de menor escala (es decir, que ocupan un espacio 3D más pequeño) podrían tener una menor contribución visual, y por tanto, podrían ser candidatas para eliminarlas.

```
# PODA POR ESCALA
if excess_gaussians > 0:
    # TFG: Se calcula la escala máxima por gaussiana
    scales = self.get_scaling.max(dim=1).values

    # TFG: Se ordenan las gaussianas por escala de menor a mayor
    low_scale_indices = torch.argsort(scales)[:excess_gaussians]

    # TFG: Se crea una máscara booleana, las gaussianas seleccionadas se eliminarán
    mask = torch.ones(self.get_xyz.shape[0], dtype=bool, device="cuda")
    mask[low_scale_indices] = False

    # TFG: Se aplica la poda
    self.prune_points(~mask)"""
```

Ilustración 5.19: Codigo Poda basada en escala

La Figura 5.20 ilustra visualmente este procedimiento, donde se muestra cómo, tras la densificación, se seleccionan las gaussianas con menor escala y se eliminan para respetar el presupuesto.

El procedimiento es el siguiente:

1. Se calcula el número de gaussianas excedentes como:

$$\texttt{exceso\_gaussianas} = \texttt{número\_actual} - \texttt{target\_gaussians}$$

2. Si hay exceso, se calcula la escala máxima de cada gaussiana (valor máximo entre sus tres componentes de escala).
3. Se ordenan las gaussianas por este valor maximo de escala, de menor a mayor.
4. Se eligen tantas gaussuanas con menor escalar como exceso exista.
5. Se genera una máscara booleana en la que estas gaussianas quedan marcadas como `False`.
6. Finalmente, se aplica esta máscara al método prune_points(), eliminando así las gaussianas seleccionadas.

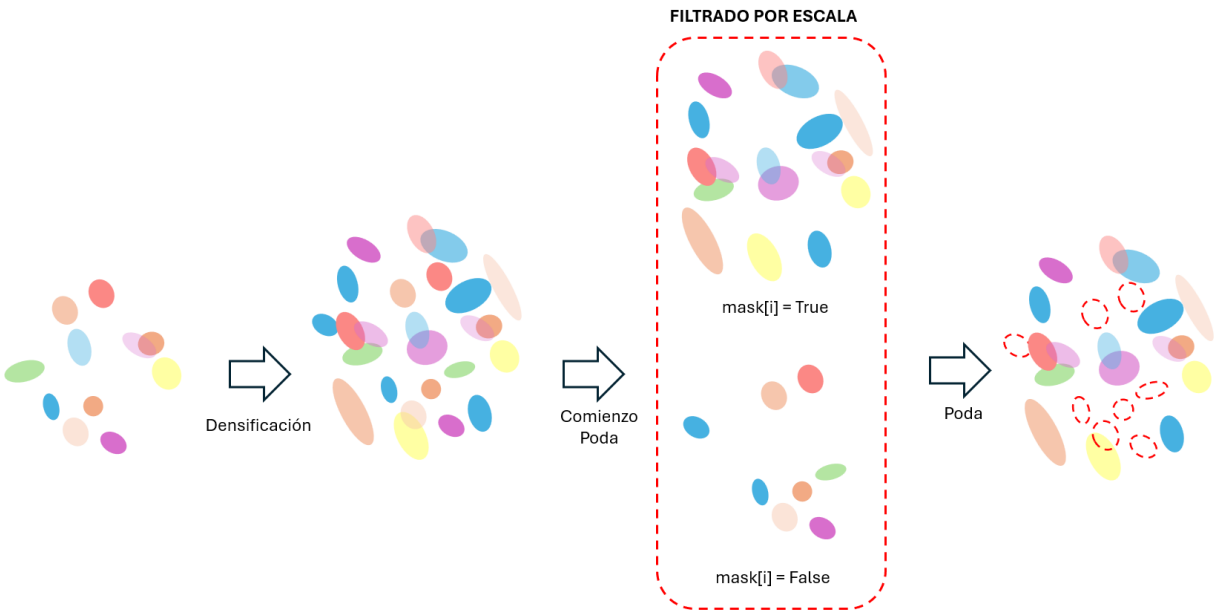


Ilustración 5.20: Ejemplo ilustrativo del mecanismo de poda basado en escala. Las gaussianas con menor escala (en azul) son eliminadas para cumplir el presupuesto.

5.9. Implementación de un mecanismo de poda basado en opacidad

La estrategia de poda consiste en eliminar el número exacto de gaussianas necesarias para volver al límite deseado en cada uno de los pasos del entrenamiento. Para seleccionar cuáles eliminar, se utiliza como criterio la opacidad: se asume que las gaussianas con menor opacidad tienen una menor relevancia en la reconstrucción de la escena y, por tanto, pueden ser descartadas con menor impacto.

```
# PRUNEAR POR OPACIDAD
if excess_gaussians > 0:

    # TFG: Se ordenan las gaussianas por opacidad de menor a mayor
    low_opacity_indices = torch.argsort(self.get_opacity.squeeze())[:excess_gaussians]

    # TFG: Se crea una máscara booleana, las gaussianas seleccionadas se eliminarán
    mask = torch.ones(self.get_xyz.shape[0], dtype=bool, device="cuda")
    mask[low_opacity_indices] = False

    # TFG: Se aplica la poda
    self.prune_points(~mask)
```

Ilustración 5.21: Codigo Poda basada en opacidad

El procedimiento completo puede verse ilustrado en la Figura 5.22, donde se muestra cómo se eliminan las gaussianas menos opacas para respetar el límite de presupuesto definido.

1. Se calcula el exceso de gaussianas como:

$$\texttt{excess\_gaussians} = \texttt{número\_actual} - \texttt{target\_gaussians}$$

2. Si el exceso es positivo, se ordenan todas las gaussianas según su opacidad en orden ascendente.
3. Se seleccionan los índices de las gaussianas con menor opacidad, correspondientes al número en exceso.
4. Se construye una máscara booleana en la que estas gaussianas quedan marcadas como `False` y el resto como `True`.
5. Finalmente, se aplica esta máscara inversa al método prune_points(), que elimina las gaussianas seleccionadas del modelo.

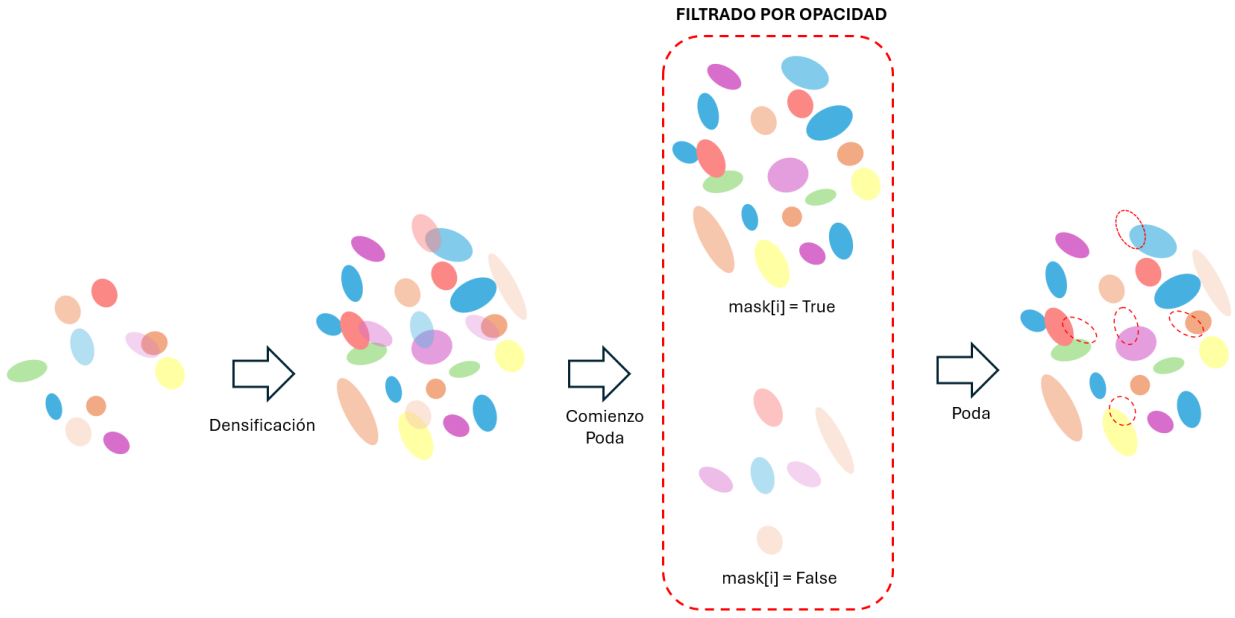


Ilustración 5.22: Ejemplo ilustrativo del mecanismo de poda basado en opacidad. Las gaussianas con menor opacidad son eliminadas para mantener el presupuesto.

Capítulo 6

Resultados Parciales

Este capítulo presenta los resultados parciales obtenidos tras implementar y evaluar distintos mecanismos de control del crecimiento de gaussianas en 3DGS[5]. En concreto, se analizan los efectos de la poda basada en la escala, la poda basada en la opacidad y el uso de un optimizador acelerado. Para cada estrategia, se explora el impacto de aplicar diferentes presupuestos máximos (100 %, 75 % y 50 %) sobre métricas de calidad visual, consumo de memoria y tiempo de entrenamiento.

6.1. Resultados con distintos presupuestos de gaussianas para poda basada en escala

Una vez implementado el mecanismo de poda basado en escala descrito en el apartado anterior, se ha procedido a evaluar su eficacia en la limitación del crecimiento del número de gaussianas. Para ello, se realizaron varios experimentos utilizando distintos presupuestos máximos (*budgets*) de gaussianas para cada escena. Estos presupuestos se definieron como un porcentaje del número total de gaussianas generadas en la versión base de 3DGS[5] con la densificación activada.

En concreto, se probaron tres configuraciones:

- **100 %**: presupuesto igual al número de gaussianas de la versión base.
- **75 %**: reducción del presupuesto al 75 % del total original.
- **50 %**: reducción del presupuesto al 50 % del total original.

6.1.1. Presupuesto del 100 %

Para evaluar la efectividad del mecanismo de poda basado en la escala, se ha fijado un presupuesto equivalente al 100 % del número de gaussianas generadas por 3DGS[5] en su

versión base. Los resultados de esta evaluación se presentan en la Tabla 6.1.

Cuadro 6.1: Resultados con poda por escala al 100 % del presupuesto.

Escena	SSIM	PSNR	LPIPS	Gaussianas	Max Mem (GB)	Tiempo
Bicycle	0.77	25.60	0.23	4.650.215	7.98	40m 40s
Bonsai	0.95	31.53	0.10	1.036.104	2.96	13m 41s
Counter	0.92	29.51	0.12	915.634	2.50	14m 37s
Garden	0.87	27.88	0.11	3.637.795	6.93	37m 36s
Kitchen	0.96	32.88	0.06	1.378.580	3.22	17m 26s
Stump	0.78	26.98	0.22	4.101.465	6.27	33m 10s

A pesar de mantener el mismo presupuesto de gaussianas que 3DGS[5] base, se observa una ligera caída en métricas como PSNR y SSIM en varias escenas. En particular, Bonsai, Bicycle y Counter presentan descensos notables en la métrica PSNR con respecto a los valores originales. Esto puede sugerir que la eliminación basada en la escala puede afectar a gaussianas relevantes para la reconstrucción, sobre todo en aquellas que tienen geometría muy compleja y necesita de las gaussianas pequeñas para afinar el detalle.

6.1.2. Presupuesto del 75 %

Para analizar el impacto de una reducción más agresiva del presupuesto de gaussianas, se evaluó el rendimiento del modelo aplicando un recorte al 75 % del presupuesto original. Los resultados obtenidos se resumen en la Tabla 6.2.

Cuadro 6.2: Resultados con poda por escala al 75 % del presupuesto.

Escena	SSIM	PSNR	LPIPS	Gaussianas	Max Mem (GB)	Tiempo
Bicycle	0.74	25.07	0.27	3.483.177	9.48	35m 53s
Bonsai	0.94	30.89	0.11	774.691	3.33	12m 32s
Counter	0.91	29.20	0.14	682.355	2.87	13m 00s
Garden	0.86	27.67	0.13	2.725.224	8.10	33m 15s
Kitchen	0.95	32.40	0.06	1.034.968	3.68	15m 27s
Stump	0.71	25.70	0.32	3.071.245	7.41	28m 58s

Al reducir el presupuesto al 75 %, se evidencia aún más la pérdida de caldidad, especialmente en Bicycle y Stump, donde el PSNR y LPIPS empeoran significativamente. Si bien es cierto que el tiempo y la memoria muestran un claro ahorro con respecto a 3DGS[5] Base, la calidad visual se comienza a ver comprometida.

6.1.3. Presupuesto del 50 %

Para evaluar el límite inferior del presupuesto sin comprometer excesivamente la calidad, se exploró un escenario de poda al 50 % del presupuesto original. Los resultados obtenidos se muestran en la Tabla 6.3.

Cuadro 6.3: Resultados con poda por escala al 50 % del presupuesto.

Escena	**SSIM**	**PSNR**	**LPIPS**	**Gaussianas**	**Max Mem (GB)**	**Tiempo**
Bicycle	0.70	24.42	0.31	2.320.719	7.55	29m 45s
Bonsai	0.91	28.88	0.15	515.413	2.93	11m 18s
Counter	0.89	28.66	0.16	452.867	2.51	11m 29s
Garden	0.84	27.18	0.17	1.812.736	6.64	28m 18s
Kitchen	0.93	30.90	0.10	686.367	3.16	13m 44s
Stump	0.52	22.53	0.47	2.046.523	5.78	24m 18s

Cuando se recorta al 50 % de las gaussianas, decae de forma drástica la calidad visual, especialmente escenas como Stump, que pierde más de 4 puntos en PSNR y alcanza un valor de LPIPS cercano al 0.5. El resto de escenas también presentan descensos en SSIM y PSNR, lo puede resultar excesiva para aquellos casos en lo que sea necesario la calidad visual.

6.1.4. Conclusiones

Las métricas como PSNR y LPIPS se ven afectadas negativamente en la mayoría de escenas, reflejando así que el criterio de escala no siempre prioriza correctamente las gaussianas más relevantes.

Si bien es cierto que mejora la parte de eficiencia, reflejado en el tiempo, la memoria usada y el numero de gaussianas generadas, esta mejora va acompañada de una pérdida objetiva de calidad, especialmente en configuraciones más restrictivas (75 % y 50 %). Esto limita lo útil que puede llegar a ser la poda por escala donde se requiere precisión visual.

6.2. Resultados con distintos presupuestos de gaussianas para poda basada en opacidad

Al igual que la poda basada en escalaa desarrollada anteriormente, se probaron tres presupuestos distintos para la poda basada en opacidad: 100 %, 75 % y 50 % del número de gaussianas.

6.2.1. Presupuesto del 100 %

En primer lugar, se utiliza como límite el mismo número de gaussianas generadas en el entrenamiento original con densificación activada, es decir, un presupuesto del 100 %. La Tabla 6.4 muestra los resultados obtenidos al aplicar la poda basada en opacidad bajo esta condición.

Cuadro 6.4: Resultados tras aplicar la poda con un presupuesto del 100 %.

Escena	**SSIM**	**PSNR**	**LPIPS**	**Gaussianas**	**Max Mem (GB)**	**Tiempo**
Bicycle	0.78	25.75	0.20	4.645.749	11.29	40m 51s
Bonsai	0.96	33.04	0.08	1.033.780	3.73	14m 3s
Counter	0.93	29.75	0.10	916.420	3.23	14m 59s
Garden	0.88	27.95	0.10	3.635.310	9.50	38m 21s
Kitchen	0.95	32.48	0.06	1.379.504	4.23	17m 49s
Stump	0.79	26.95	0.21	4.099.971	9.03	33m 39s

Como puede observarse, la aplicación de la poda permite mantener un número de gaussianas muy cercano al presupuesto deseado, con una ligera mejora en el uso de memoria y el tiempo respecto al 3DGS[5] base. Las métricas de calidad visual se mantienen prácticamente intactas.

6.2.2. Presupuesto del 75 %

En el segundo experimento, se reduce el número máximo de gaussianas permitidas al 75 % del presupuesto original. El objetivo es analizar hasta qué punto se puede compactar el modelo sin comprometer la calidad visual de las reconstrucciones. Los resultados obtenidos tras aplicar la poda basada en opacidad con esta restricción se resumen en la Tabla 6.5.

Cuadro 6.5: Resultados tras aplicar la poda con un presupuesto del 75 %.

Escena	SSIM	PSNR	LPIPS	Gaussianas	Max Mem (GB)	Tiempo
Bicycle	0.78	25.79	0.20	3.484.847	9.30	34m 50s
Bonsai	0.96	33.04	0.08	773.723	3.30	12m 30s
Counter	0.93	29.70	0.10	684.763	2.84	13m 33s
Garden	0.88	27.96	0.10	2.724.573	7.98	33m 19s
Kitchen	0.96	32.74	0.06	1.032.618	3.66	15m 51s
Stump	0.79	27.02	0.21	3.074.099	7.37	29m 50s

Los resultados muestran que, con un recorte del 25 %, el modelo logra mantener una calidad casi idéntica, mientras que se reducen los recursos necesarios. Es especialmente destacable la estabilidad de las métricas en escenas como `Bonsai` y `Kitchen`.

6.2.3. Presupuesto del 50 %

Finalmente, se evalúa un escenario más agresivo, reduciendo el presupuesto al 50 % del valor original. El objetivo es comprobar si esta simplificación impacta de forma significativa en la reconstrucción de las escenas, tanto en calidad visual como en recursos computacionales. Los resultados de este experimento se recogen en la Tabla 6.6.

Cuadro 6.6: Resultados tras aplicar la poda con un presupuesto del 50 %.

Escena	SSIM	PSNR	LPIPS	Gaussianas	Max Mem (GB)	Tiempo
Bicycle	0.78	25.76	0.22	2.321.167	7.37	27m 18s
Bonsai	0.96	32.81	0.09	515.413	2.88	10m 22s
Counter	0.93	29.56	0.11	452.867	2.46	10m 53s
Garden	0.87	27.87	0.12	1.814.211	6.50	26m 3s
Kitchen	0.95	32.79	0.06	686.163	3.09	13m 10s
Stump	0.79	27.00	0.22	2.048.184	5.71	24m 4s

Es cierto que se observa una leve caída en métricas como LPIPS, la calidad visual global sigue siendo sorprendentemente buena. A cambio, se consigue una mejora significativa en rendimiento, memoria y tiempo.

6.2.4. Conclusiones

Los resultados de los experimentos reflehan que la poda basada en opacidad es aparentemente eficaz para controlar el crecimiento de gaussianas durante el entrenamiento de 3DGS[5].

Además, el mecanismos logra una reducción seignificativa en el uso de memoria y en el tiempo de entrenameinto, sin comprometer en exceso las métricas de calidad de imagen.

Se concluye que la combinación de una función de crecimiento controlado con una poda basada en opacidad representa una solución acertada que permite equilibrar calidad y eficiencia en la reconstrucción de escenas 3D.

6.3. Resultados con optimizador acelerado y poda basada en opacidad

Con el objetivo de acelerar el proceso de entrenamiento sin comprometer la calidad, se ha decidido utilizar el optimizador acelerado incluido en el repositorio de 3DGS. Esta versión agrega un rasterizador optimizado y un nuevo tipo de optimizador llamado sparse_adam, el cual ofrece una mejora significativa en el rendimiento.

Según los autores del repositorio, este optimizador puede alcanzar hasta una aceleración de entrenamiento de 2.7x en comparación con la versión estándar.

Este ha sido utilizado para realizar los experimentos con poda basada en opacidad. A continuación se muestran los resultados para tres presupuestos diferentes de gaussianas: 100 %, 75 % y 50 %.

6.3.1. Resultados con presupuesto del 100 % y optimizador acelerado

Con el objetivo de evaluar el impacto del optimizador acelerado en combinación con un presupuesto limitado, se ha ejecutado un experimento manteniendo el 100 % del presupuesto original de gaussianas, junto con el mecanismo de poda por opacidad. La Tabla 6.7 muestra los resultados obtenidos.

Cuadro 6.7: Resultados con optimizador acelerado y poda por opacidad al 100 %.

Escena	**SSIM**	**PSNR**	**LPIPS**	**Gaussianas**	**Max Mem (GB)**	**Tiempo**
Bicycle	0.77	25.66	0.23	4.497.435	10.81	20m 21s
Bonsai	0.96	33.03	0.08	1.034.734	3.75	8m 4s
Counter	0.93	29.71	0.01	911.972	3.45	9m
Garden	0.87	27.90	0.11	3.423.012	9.10	20m 52s
Kitchen	0.95	32.75	0.06	1.380.211	4.40	11m 37s
Stump	0.78	26.95	0.22	3.980.451	8.52	16m 36s

Los resultados muestran que, al mantener el presupuesto original de gaussianas y combinarlo con poda basada en opacidad y el optimizador acelerado, se consigue preservar la

calidad visual de las escenas prácticamente intacta. Métricas como SSIM[29] y LPIPS[3] se mantienen estables, mientras que el tiempo de entrenamiento se reduce drásticamente en comparación con el modelo original, demostrando la eficiencia del nuevo enfoque sin comprometer calidad visual.

6.3.2. Resultados con presupuesto del 75 % y optimizador acelerado

A continuación, se presenta un nuevo experimento donde se reduce el presupuesto de gaussianas al 75 % del valor original, combinando esta limitación con el uso del optimizador acelerado y un mecanismo de poda basado en opacidad. La Tabla 6.8 resume los resultados obtenidos en las distintas escenas.

Cuadro 6.8: Resultados con optimizador acelerado y poda por opacidad al 75 %.

Escena	**SSIM**	**PSNR**	**LPIPS**	**Gaussianas**	**Max Mem (GB)**	**Tiempo**
Bicycle	0.77	25.65	0.23	3.488.527	9.44	18m 35s
Bonsai	0.96	32.91	0.08	773.813	3.32	7m 8s
Counter	0.93	29.65	0.10	684.289	2.96	8m 2s
Garden	0.87	27.88	0.11	2.727.018	8.07	19m 5s
Kitchen	0.95	32.72	0.06	1.033.967	3.80	9m 54s
Stump	0.78	26.96	0.22	3.075.811	7.29	15m 25s

Al reducir el presupuesto de gaussianas al 75 %, se mantienen métricas de calidad casi iguales a las del caso del 100 %, con una ganancia adicional en eficiencia, por lo que es posible disminuir el número de gaussianas sin afectar de forma notable la calidad visual.

6.3.3. Resultados con presupuesto del 50 % y optimizador acelerado

Para explorar aún más la eficiencia del sistema, se ha llevado a cabo un experimento reduciendo el presupuesto de gaussianas al 50 %, manteniendo tanto el optimizador acelerado como la poda por opacidad. La Tabla 6.9 recoge los resultados obtenidos en este escenario más restrictivo.

Cuadro 6.9: Resultados con optimizador acelerado y poda por opacidad al 50 %.

Escena	SSIM	PSNR	LPIPS	Gaussianas	Max Mem (GB)	Tiempo
Bicycle	0.77	25.66	0.23	2.322.018	7.46	15m 15s
Bonsai	0.96	32.71	0.09	515.413	2.92	7m 1s
Counter	0.92	29.47	0.11	452.867	2.56	7m 5s
Garden	0.87	27.77	0.12	1.812.716	6.60	15m 54s
Kitchen	0.95	32.27	0.06	686.367	3.20	8m 7s
Stump	0.78	26.96	0.23	2.046.729	5.69	13m 15s

Incluso con solo el 50 % del presupuesto original de gaussianas, el modelo es capaz de mantener una calidad visual muy similar. Aunque se observan ligeros cambios en algunas métricas, como PSNR o LPIPS, estas se mantienen dentro de márgenes aceptables. Además, la reducción en el tiempo de entrenamiento y el uso de memoria es significativa.

6.4. Resultados generales

Como se puede observar en la Tabla 6.10, la estrategia de poda basada en la escala consigue reducir bastante el número de gaussianas y el consumo de memoria VRAM. No obstante, esta reducción también supone una pérdida considerable en las métricas de calidad, especialmente cuando se reduce el presupuesto al 50 %. En concreto, la métrica SSIM cae hasta de un valor de 0.888 a un valor de 0,80 y el PSNR desciende hasta los 27,10, lo que refleja descenso en la calidad visual de las escenas reconstruidas bastante importante.

Presupuesto	SSIM	PSNR	LPIPS	Nº Gaussianas	Max Mem (GB)	Tiempo
3DGS Base	0,88	29,26	0,12	2.623.284	6,78	25m
100 % Escala	0,88	29,06	0,14	2.619.965	4,98	26m 20s
75 % Escala	0,85	28,49	0,17	1.961.943	5,81	23m 20s
50 % Escala	0,80	27,10	0,23	1.305.770	4,76	19m 50s

Cuadro 6.10: Resultados con poda basada en escala para diferentes presupuestos de gaussianas.

Por otro lad, la Tabla 6.11 muestra que la poda basada solamente en opacidad presenta un mejor equilibrio entre calidad y eficiencia. A diferencia del método anterior, esta técnica consigue mantener las métricas de calidad casi intactas, incluso al reducir el número de gaussianas al 75 %. Es importante destacar que, aunque al 50 % se empieza a observar un ligero descenso en las métricas, los valores obtenidos siguen siendo aceptables y mucho mejores que los alcanzados mediante la poda por escala. De hecho, el valor del PSNR se matinene superior al del caso base.

Presupuesto	SSIM	PSNR	LPIPS	Nº Gaussianas	Max Mem (GB)	Tiempo
3DGS Base	0,88	29,26	0,12	2.623.284	6,78	25m
100 % Opacidad	0,88	29,32	0,14	2.618.455	6,84	26m 40s
75 % Opacidad	0,88	29,38	0,13	1.962.437	5,74	23m 10s
50 % Opacidad	0,88	29,29	0,14	1.306.334	4,67	18m 40s

Cuadro 6.11: Resultados con poda basada en opacidad para diferentes presupuestos de gaussianas.

Por último, la Tabla 6.12 recoge los resultados obtenidos al combinar la poda por opacidad con el optimizador acelerado propuesto en 3DGS. Esta combinación parece ser el método más eficiente de todos las evaluados, consiguiendo reducir de forma significativa los tiempos de entrenamiento, llegando incluso a entrenar escenas completas en menos de la mitad del tiempo necesario en la versión base. Además, en cuanto a calidad, se mantienen prácticamente idénticas, incluso con un presupuesto del 50 % de gaussianas, lo que demuestra que el método implantado es un éxito.

Presupuesto	SSIM	PSNR	LPIPS	Nº Gaussianas	Max Mem (GB)	Tiempo
3DGS Base	0,88	29,26	0,12	2.623.284	6,78	25m
100 % + Opt	0,88	29,33	0,12	2.537.969	6,68	14m 15s
75 % + Opt	0,88	29,30	0,13	1.963.904	5,81	13m
50 % + Opt	0,88	29,14	0,14	1.306.018	4,74	11m

Cuadro 6.12: Resultados con optimizador acelerado y poda basada en opacidad para diferentes presupuestos de gaussianas.

En resumen, a pesar de que las configuraciones con un presupuesto del 75 % en la poda por opacidad presentan métricas ligeramente superiores, el objetivo de este Trabajo de Fin de Grado se centra en reducir al máximo el consumo de recursos sin comprometer drásticamente la calidad visual. Por ello, se considera que el presupuesto del 50 % en combinación con la poda basada en opacidad y el optimizador acelerado representa la opción más adecuada, al permitir una reducción significativa de recursos manteniendo una calidad visual prácticamente indistinguible de la versión base.

Capítulo 7

Resultados

En este capítulo recoge los principales resultados obtenidos tras aplicar distintas estrategias de entrenamiento en 3D Gaussian Splatting (3DGS)[5]. El objetivo es evaluar, tanto de forma cuantitativa como cualitativa, el impacto de técnicas como el control de presupuesto, la poda de gaussianas y la aceleración del proceso de optimización. Se busca identificar combinaciones que mantengan una calidad visual alta mientras reducen significativamente el uso de recursos computacionales.

7.0.1. Resultados cuantitativos y comparativa de estrategias

A continuación se presentan los resultados obtenidos con distintas configuraciones del pipeline de entrenamiento de 3D Gaussian Splatting (3DGS)[5], incluyendo entrenamientos con y sin parches, con limitación por presupuesto (budget), técnicas de poda y optimizaciones para acelerar el entrenamiento. El objetivo de estos experimentos fue encontrar una solución que mantuviera una calidad visual elevada reduciendo los recursos computacionales necesarios.

Se decidió fijar el budget en un 50 % del modelo base, ya que supuso una reducción significativa en el número de gaussianas y en el tiempo de entrenamiento, sin llegar a degradar de forma notable la calidad visual. A partir de valores menores, se observó una pérdida de fidelidad perceptible en las imágenes reconstruidas.

La Tabla 7.1 muestra un resumen de los resultados más representativos. Se incluyen experimentos con número variable de parches, así como distintas técnicas de poda (por escala y por opacidad), con y sin optimización acelerada.

Cuadro 7.1: Resumen de resultados con distintas configuraciones del pipeline 3DGS.

Nombre	SSIM	PSNR	LPIPS	Gaussianas	Max Mem (GB)	Tiempo
3DGS Base	0,88	29,26	0,12	2.623.284	6,78	25m
3DGS 2 Patches	0,88	28,90	0,13	3.269.690	7,74	26m 35s
3DGS 4 Patches	0,87	28,19	0,15	3.857.668	8,66	28m 35s
3DGS Budget (50 %) + Poda Escala	0,80	27,10	0,23	1.305.770	4,76	17m 40s
3DGS Budget (50 %) + Poda Opacidad	0,88	29,30	0,14	1.306.334	4,66	18m 35s
3DGS Budget (50 %) + Poda Opacidad + Aceleración	0,88	29,14	0,14	1.306.017	4,74	11m

Entrenamiento por Parches. Al entrenar con múltiples parches, se observa un ligero empeoramiento en la calidad de reconstrucción conforme aumenta su número, especialmente a partir de 4, donde la PSNR cae a 28,19 y LPIPS[3] aumenta a 0,15, lo que se traduce en una ligera pérdida de nitidez.

Control del presupuesto. El uso del parámetro –densify_budget con un límite del 50 % permitió reducir las gaussianas generadas casi a la mitad (de más de 2,6 millones a unos 1,3 millones), lo cual disminuyó la memoria máxima usada y el tiempo de entrenamiento sin comprometer en exceso la calidad.

Poda por escala vs. opacidad. La poda por escala resultó ser demasiado agresiva, reduciendo considerablemente la PSNR y aumentando el LPIPS[3]. Por el contrario, la poda por opacidad logró mantener niveles de calidad muy similares al modelo original, siendo por tanto una técnica más efectiva.

Aceleración del optimizador. Finalmente, al aplicar la optimización propuesta en el repositorio 3dgs-accel, que sustituye el rasterizador original y acelera el cómputo del gradiente, se consiguió reducir el tiempo total de entrenamiento a sólo 11 minutos, manteniendo métricas de calidad similares al modelo base. Esta opción representa la mejor relación entre rendimiento y coste computacional.

7.1. Análisis cualitativo de las reconstrucciones

Además de las métricas numéricas presentadas anteriormente, se realiza a continuación un análisis visual de las reconstrucciones obtenidas para distintas escenas. Este análisis permite evaluar mejor el impacto perceptual de las técnicas propuestas más prometedoras de las que han sido desarrolladas y observar de manera directa las diferencias más relevantes entre los distintos modelos.

A continuación, se analizan de forma individual las algunas de las escenas utilizadas.

7.1.1. Escena 1 — Bonsai

En esta primera escena se aprecia cómo las métodos que usan poda no sólo consiguen reducir de forma drástica el número de gaussianas, memoria y tiempo de entrenamiento, sino que además logran mantener o incluso mejorar algunos detalles específicos respecto al modelo base.

Tal y como se indica en la Ilustración 7.1, el modelo base genera aproximadamente 1.040.856 gaussianas, mientras que las configuraciones con poda al 50 % reducen esta cifra a tan solo 515.413, lo que supone un ahorro del 50 % en primitivas. Este cambio también se traduce en una reducción sustancial del uso de memoria (de 3.74 GB a 2.88–2.92 GB) y en tiempos de entrenamiento que pasan de 12m 50s a tan solo 7m 1s con el optimizador acelerado.

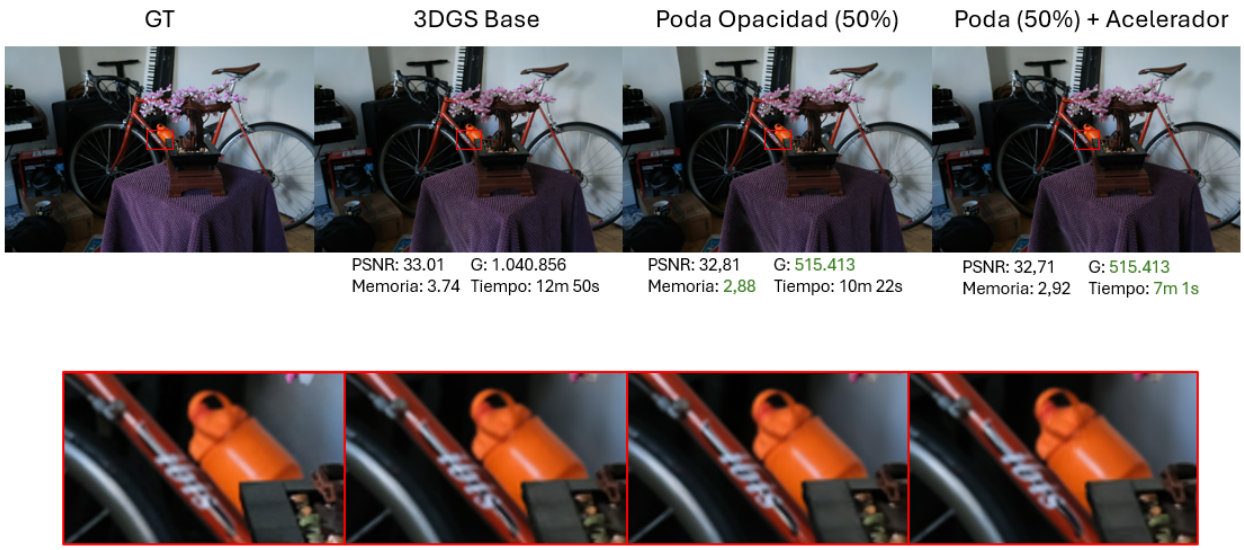


Ilustración 7.1: Comparativa visual para la escena *Bonsai*. Se muestran las reconstrucciones obtenidas con el modelo base y las distintas configuraciones de poda y aceleración. En la parte inferior se amplían detalles de la zona marcada en rojo.

En cuanto a las métricas, los métodos con poda (50 %) presentan un PSNR ligeramente inferior al modelo base. Sin embargo, la reducción de gaussianas es muy significativa, pasando de más de un millón a apenas unas 515.000, lo cual se puede traducir en un menor uso de memoria y tiempos de entrenamiento mucho más reducidos.

A nivel visual, si se compara el detalle ampliado en la parte inferior de la figura, se observa que los métodos con poda son capaces de preservar mejor ciertos reflejos y brillos en el hierro de la bicicleta, aportando un aspecto más limpio y definido en comparación con el método base, que tiende a difuminar ligeramente estos detalles.

7.1.2. Escena 2 — Garden

En esta escena se aprecia una ligera ventaja de las técnicas con poda en cuanto a eficiencia y calidad visual. A nivel numérico, las técnicas de poda por opacidad y aceleración presentan un menor número de gaussianas (alrededor de 1,8 millones frente a los más de 3,6 millones del método base), una reducción considerable en el uso de memoria (6,5-6,6 GB frente a 9,43 GB) y un tiempo de entrenamiento notablemente menor, especialmente cuando se utiliza aceleración (15 minutos frente a 35 minutos del método base).

En cuanto a calidad, las métricas PSNR se mantienen estables respecto al modelo base, incluso ligeramente superiores para la poda de opacidad.

Desde el punto de vista perceptual, observando la ampliación de la zona metálica del suelo, se aprecia que el método base introduce unos reflejos o tonos más claros que no están presentes en la imagen original (GT). Este efecto parece ser una reconstrucción excesivamente brillante, lo que puede deberse a una mayor cantidad de gaussianas que buscan sobreajustarse a la zona local. Por el contrario, las técnicas con poda consiguen una apariencia más realista y ajustada a la referencia, evitando estos comportamiento de reflección.

La Figura 7.2 muestra una comparativa visual completa de la escena *Garden*, incluyendo una ampliación de la zona metálica del suelo. En dicha ampliación se aprecian diferencias relevantes: el modelo base introduce un brillante alrededor del objeto, que no está presente en la imagen original (GT). Por el contrario, las técnicas con poda eliminan este artefacto, generando una reconstrucción más realista y fiel a la escena, sin sobreajustes ni reflejos artificiales.

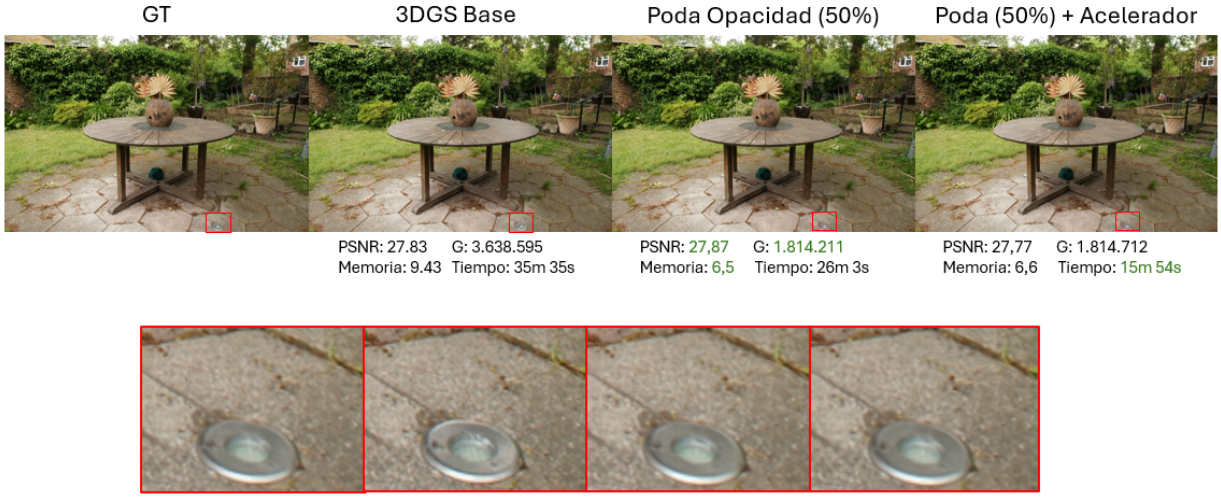


Ilustración 7.2: Comparativa visual de la escena Garden. Las técnicas con poda evitan artefactos brillantes o reflejos añadidos por el modelo base.

7.1.3. Escena 3 — Kitchen

En la escena Kitchen se observan de nuevo mejoras significativas en cuanto a eficiencia al aplicar técnicas de poda y aceleración. El número de gaussianas se reduce aproximadamente a la mitad (de 1,38 millones a 686 mil), disminuyendo el consumo de memoria (de 4,21 GB a unos 3,1 GB) y el tiempo de entrenamiento (de 16 minutos a tan solo 8 minutos con aceleración).

En términos de calidad visual, las métricas PSNR se mantienen en niveles similares al modelo base, aunque en el caso de la poda con opacidad aumenta un poco, lo que indica que el proceso de poda no afecta de manera significativa a la fidelidad global de la reconstrucción.

Sin embargo, observando la ampliación de la zona seleccionada, se aprecia un comportamiento interesante respecto al patrón de la valla o estructura metálica situada al fondo. En este caso, los métodos que aplican poda (tanto por opacidad como con poda y acelerador) logran representar de forma más clara y definida el patrón de líneas y la forma de la estructura.

La Figura 7.3 muestra una comparativa visual entre las distintas configuraciones. En la ampliación de la zona seleccionada, se aprecia un comportamiento interesante en el patrón de la valla metálica al fondo: las técnicas con poda (ya sea con o sin aceleración) logran representar de forma más clara y definida las líneas y formas de dicha estructura.

Ilustración 7.3: Comparativa visual de la escena Kitchen. Las técnicas con poda simplifican algunas estructuras de alta frecuencia como el patrón de la valla.

Por el contrario, el modelo base presenta mayores dificultades para reproducir este detalle fino, generando un resultado más difuminado, poco definidos y en alguna ocasión, bastante ruidoso en dicha región. Esto podría indicar que los mecanismos de poda favorecen una representación más precisa en las zonas de mayor importancia visual o contraste, evitando el solapamiento de gaussianas irrelevantes.

En cualquier caso, este resultado refuerza la idea de que las técnicas de poda no solo permiten reducir recursos computacionales, sino que en algunos escenarios pueden incluso potenciar la calidad visual percibida en determinadas estructuras.

7.1.4. Escena 4 — Bicycle

En esta escena se puede apreciar un claro ejemplo de cómo las técnicas de poda y aceleración permiten reducir de forma significativa los recursos computacionales manteniendo una calidad visual decente.

A nivel de métricas, la poda por opacidad (50 %) muestra los mejores resultados globales en cuanto a PSNR, número de gaussianas y uso de memoria. Esto confirma que, para esta escena, la reducción controlada mediante opacidad permite conservar mejor la calidad de la reconstrucción.

La combinación de poda y acelerador muestra un PSNR ligeramente inferior al método base, pero mantiene una calidad visual global muy similar, consiguiendo además de esto, un tiempo de entrenamiento considerablemente menor, lo cual supone una mejora significativa en eficiencia.

La Figura 7.4 muestra una comparativa visual detallada. En la parte inferior, se observa que las técnicas de poda son capaces de afinar mejor los detalles pequeños, como las hojas y ramas en la parte superior de la imagen. A pesar de tener un PSNR algo inferior, los métodos con poda por opacidad consiguen preservar mejor las texturas finas y los elementos del fondo respecto al método base, lo que evidencia ciertas limitaciones de las métricas numéricas para capturar la calidad perceptual.

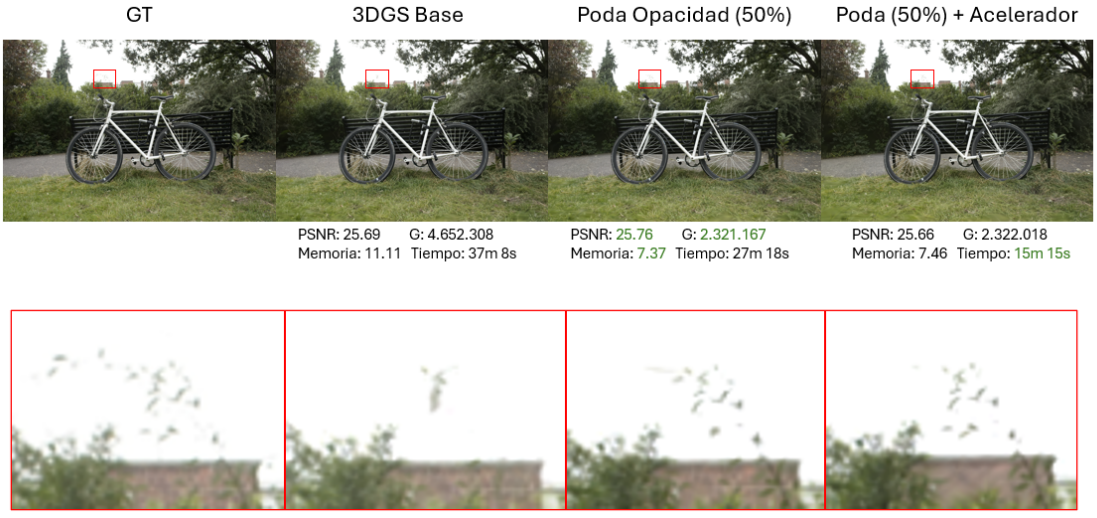


Ilustración 7.4: Comparativa visual para la escena *Bicycle*. Se muestran las reconstrucciones obtenidas con el modelo base y las distintas configuraciones de poda y aceleración. En la parte inferior se amplían detalles de la zona marcada en rojo.

Capítulo 8

Conclusiones y trabajo futuro

8.1. Conclusiones

Durante este proyecto se ha llevado a cabo un estudio sobre la optimización del proceso de densificación dentro de 3DGS[5], comenzando por un análisis paramétrico de los hiperparámetros más relevantes del proyecto base para llegar a una comprensión del funcionamiento de los mismo y, finalizando con el desarrollo de diferentes mecanismos orientados a controlar y mejorar el crecimiento de este modelo.

Los principales ojetivos planteados inicialmente se han cumplido satisfactoriamente. Se ha conseguido controlar de maneara efectiva el número de gaussianas generadas durante el entrenamiento del modelo mediante la implementación de presupuestos de gaussianas y mecanismos de poda, reduciendo así el tamaño final del modelo sin afectar drásticamente a la calidad visual de las escenas.

Por otro lado, se han planteado y evaluado distintas estrategias de poda basadas en propiedades de las propias gaussianas, como la escala o la opacidad, evidenciando que es posible eliminar un porcentaje elevado de gaussianas sin provocar un descenso significativo en las métricas de calidad (SSIM, PSNR, LPIPS).

Adicionalemnte, se hizo uso de recursos y mejoras que los propios autores del repositorio de 3DGS[5] añaden, como puede ser el acelerador del optimizador, permitiendo reducir más el tiempo de entrenamiento de las escenas, a pesar de que haya que pagar un mínimo precio de calidad.

Este trabajo desarrollado no solo ha permitido alcanzar los objetivos iniciales, sino que sienta las bases para futuras líneas de investigación orientadas a seguir mejorando la eficiencia y la escalabilidad de este tipo de representaciones.

8.2. Trabajo futuro

A partir de los resultados y las metodologías desarrolladas previamente, se abren diversas líneas de trabajo futuro que permitirían continuar mejorando el control, la eficiencia y la calidad de las representaciones generadas en 3DGS[5].

8.2.1. Análisis avanzado del parámetro *k* en la función de presupuesto

El parámetro k actúa como pendiente inicial dentro de la ecuación cuadrática que regula el crecimiento del número de gaussianas durante el entrenamiento. Un análisis futuro interesante consistiría en evaluar experimentalmente diferentes valores de k, más allá del valor base $k = \frac{2(B-S)}{N}$ propuesto en este trabajo, con el objetivo de estudiar su influencia sobre la calidad final, el consumo de recursos y la velocidad de convergencia del modelo.

Además, se podría explorar un mecanismo de ajuste dinámico de k durante el entrenamiento, haciendo que su valor no sea constante sino que dependa de la complejidad de la escena, la densidad local de gaussianas o el ritmo de mejora de la función de pérdida.

Una posible mejora podría ser emplear una función $k(x)$, dependiente del paso x, o incluso definir k en función de la tasa de gradiente medio en las gaussianas:

$$k(x) = \alpha \cdot \text{mean}\left(|\nabla_\theta \mathcal{L}|\right) + \beta \tag{8.1}$$

Donde:

- $\nabla_\theta \mathcal{L}$ representa el gradiente de la función de pérdida respecto a los parámetros del modelo.
- mean$(\cdot)$ indica que se calcula el promedio de la magnitud de los gradientes de todas las gaussianas.
- α es un parámetro de escala que regula la sensibilidad del sistema al valor del gradiente.
- β es un término independiente que actúa como crecimiento mínimo, incluso cuando los gradientes son bajos.

La raiz de este pensamiento es que cuánto más alto es el valor de los gradientes más está aprendiendo el modelo y, por tanto, se debería permitir un mayor crecimiento del número de gaussianas. Por el contrario, cuando los gradientes decrecen y tienden a valores bajos, esto suele indicar que el modelo ha convergido o que las nuevas gaussianas tienen poco impacto en la pérdida, por lo que conviene frenar el crecimiento y evitar la generación innecesaria de puntos.

8.2.2. Presupuesto local adaptativo

Otra línea de mejora podría combinar la propuesta de parches y la propuesta del presupuesto, donde el concepto de presupuesto global se mueve a un presupuesto local adaptativo, donde diferentes regiones de la escena dispongan de un número máximo de gaussianas diferente, en función de su complejidad geométrica o visual.

Por ejemplo, si hay zonas planas, monótonas o de baja variación podrían tener un presupuesto más bajo, mientras que regiones con alta curvatura, fuertes gradientes de color o cambios de tonalidad requerirían un presupuesto superior.

8.2.3. Mecanismos de poda multi-parámetro

El sistema de poda actual está basado en un solo parámetro (escala u opacidad). Un trabajo futuro interesante podría consistir en definir un *score* ponderado que combine diversos atributos de las gaussianas: escala, opacidad, gradiente acumulado, cercanía a otras gaussianas o incluso información de los coeficientes de color.

Este score podría ser de la forma:

$$Score_i = \lambda_1 \cdot \alpha_i + \lambda_2 \cdot Scale_i + \lambda_3 \cdot Grad_i + \lambda_4 \cdot Density_i \tag{8.2}$$

donde los pesos λ_i serían hiperparámetros o valores que se puedan aprender.

Bibliografía

[1] Johannes L. Schönberger. Colmap: Structure-from-motion and multi-view stereo. https://demuc.de/colmap/. Accedido: 11 de abril de 2025.

[2] R. Arizan and B. Hassibi. Descenso de gradiente estocástico (sgd), 2019. Imagen consultada en ResearchGate, subida por Ángel Sánchez Ruiz. Disponible en: https://www.researchgate.net/figure/Figura-3511-Descenso-de-gradiente-estocastico-SGD-Arizan-R-y-Hassibi-B-2019_fig9_344388136.

[3] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. *arXiv preprint arXiv:1801.03924*, 2018. Accedido: 11 de abril de 2025.

[4] The Brainy Insights. 3d rendering market size by deployment type (cloud and on-premise), application (animation, product design & modelling, visualization & simulation and others), organization size (smes and large enterprise), regions, global industry analysis, share, growth, trends, and forecast 2023 to 2032. https://www.thebrainyinsights.com/report/3d-rendering-market-13936#:~:text=The%20global%203D%20Rendering%20market,USD%2034.57%20Billion%20by%202032, 2023. Accedido: 11 de abril de 2025.

[5] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *arXiv preprint arXiv:2306.00988*, 2023. Accedido: 11 de abril de 2025.

[6] Python Software Foundation. Python documentation. https://www.python.org/doc/, 2025. Accedido: 11 de abril de 2025.

[7] PyTorch Team. Pytorch documentation. https://pytorch.org/docs/stable/index.html. Accedido: 11 de abril de 2025.

[8] Matplotlib Team. Matplotlib 3.10.1 documentation. https://matplotlib.org/stable/index.html. Accedido: 11 de abril de 2025.

[9] TensorFlow Team. Tensorboard: el kit de herramientas de visualización de tensorflow. https://www.tensorflow.org/tensorboard?hl=es-419. Accedido: 11 de abril de 2025.

[10] GeoGebra Team. Geogebra. https://www.geogebra.org/. Accedido: 11 de abril de 2025.

[11] Docker documentation. https://www.docker.com/. Accedido: 11 de abril de 2025.

[12] Jira - software de gestión de proyectos. https://www.atlassian.com/. Accedido: 11 de abril de 2025.

[13] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *arXiv preprint arXiv:2003.08934*, 2020. Accedido: 11 de abril de 2025.

[14] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics (TOG)*, 41(4):102, 2022.

[15] Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5855–5864, 2021.

[16] Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5470–5479, 2022.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[18] Johannes L. Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4104–4113, 2016. Accedido: 11 de abril de 2025.

[19] Saswat Subhajyoti Mallick, Rahul Goel, Bernhard Kerbl, Francisco Vicente Carrasco, Markus Steinberger, and Fernando De La Torre. Taming 3dgs: High-quality radiance fields with limited resources. *arXiv preprint arXiv:2406.15643*, 2024. Accedido: 11 de abril de 2025.

[20] Joo Chan Lee, Daniel Rho, Xiangyu Sun, Jong Hwan Ko, and Eunbyung Park. Compact 3d gaussian representation for radiance field. *arXiv preprint arXiv:2311.13681*, 2023. Accedido: 11 de abril de 2025.

[21] Shakiba Kheradmand, Daniel Rebain, Gopal Sharma, Weiwei Sun, Yang-Che Tseng, Hossam Isack, Abhishek Kar, Andrea Tagliasacchi, and Kwang Moo Yi. 3d gaussian splatting as markov chain monte carlo. *arXiv preprint arXiv:2404.09591*, 2024. Accedido: 11 de abril de 2025.

[22] Max Welling and Yee Whye Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, 2011.

[23] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[24] Chengbo Wang, Guozheng Ma, Yifei Xue, and Yizhen Lao. Faster and better 3d splatting via group training. *arXiv preprint arXiv:2412.07608*, 2024. Accedido: 11 de abril de 2025.

[25] Jonathan T. Barron. Mip-nerf 360 dataset. https://jonbarron.info/mipnerf360/. Accedido: 23 de abril de 2025.

[26] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *arXiv preprint arXiv:2006.13846*, 2020. Accedido: 23 de abril de 2025.

[27] ScienceDirect. Peak signal-to-noise ratio (psnr). https://www.sciencedirect.com/topics/computer-science/peak-signal-to-noise-ratio#:~:text=Peak%20Signal%2Dto%2DNoise%20Ratio,better%20quality%20by%20reducing%20noise. Accedido: 23 de abril de 2025.

[28] Richard Zhang. Lpips: Learned perceptual image patch similarity. https://github.com/richzhang/PerceptualSimilarity. Accedido: 23 de abril de 2025.

[29] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004. Accedido: 11 de abril de 2025.

[30] Kai Zhang, Gernot Riegler Li, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields. *arXiv preprint arXiv:2010.11929*, 2020. Accedido: 23 de abril de 2025.

[31] Lingjie Liu, Ben Mildenhall, Matthew Tancik, Vincent Sitzmann, Srinath Sridhar, Stephen Lombardi, and Matthias Nießner. Nsvf: Neural sparse voxel fields. *arXiv preprint arXiv:2002.05709*, 2020. Accedido: 23 de abril de 2025.