



**UNIVERSITÀ
DI TORINO**

Università degli Studi di Torino

Corso di Laurea in Informatica

Titolo

Piattaforma Web Distribuita per 3D Gaussian Splatting: Architettura a Microservizi per la Ricostruzione 3D da Video

Relatore

Grangetto Marco

Candidato
Claudio Pioggia
262307

Anno Accademico 2024/2025

Indice

Introduzione	v
Ringraziamenti	ix
1 Stato dell'Arte del 3D Gaussian Splatting	1
1.1 Fondamenti di rendering 3D	1
1.1.1 Pipeline di rendering	1
1.1.2 Tecniche di rendering	2
1.1.3 Proiezione e volume di visibilità	3
1.1.4 Rasterizzazione	5
1.1.5 Considerazioni per GPU moderne	6
1.2 Panorama delle rappresentazioni 3D	7
1.2.1 Volume Rendering: verso rappresentazioni continue . .	7
1.3 Fondamenti teorici del Gaussian Splatting	12
1.3.1 Rappresentazione 3D basata su gaussiane	12
1.3.2 Pipeline di training	15
1.3.3 Strutture dati per gaussiane 3D	18
1.3.4 Posizionamento nel panorama delle tecniche attuali . .	20
1.4 Structure from Motion e COLMAP: prerequisiti per 3D Gaussian Splatting	21
1.4.1 Structure from Motion: ricostruzione 3D da immagini	21
1.4.2 COLMAP: implementazione di riferimento per SfM . .	22
1.4.3 Integrazione COLMAP-3DGS: dal preprocessing al rendering	23
1.4.4 Requisiti e limitazioni	23
1.5 Algoritmi di training utilizzati	24
1.5.1 Approccio standard (3DGS baseline)	24
1.5.2 Monte Carlo Markov Chain: miglioramento qualità tramite campionamento probabilistico	25
1.5.3 Taming 3DGS: ottimizzazione per risorse computazionali limitate	26
1.5.4 Parametri di configurazione del training	27
1.5.5 Confronto degli approcci	27

1.6	Metriche di Valutazione	28
1.6.1	Peak Signal-to-Noise Ratio (PSNR)	28
1.6.2	Structural Similarity Index Measure (SSIM)	28
1.6.3	Learned Perceptual Image Patch Similarity (LPIPS) .	29
1.6.4	Applicazione nel Gaussian Splatting	30
1.7	Applicazioni e limitazioni attuali	31
1.7.1	Campi di applicazione nella computer vision	31
1.7.2	Limiti all'adozione diffusa	32
2	Architettura del Sistema Proposto	35
2.1	Obiettivi e funzionalità del sistema	35
2.1.1	Obiettivi di progetto	35
2.1.2	Funzionalità del sistema	36
2.1.3	Benefici attesi	38
2.2	Organizzazione a Layer	39
2.2.1	Layer di Storage	40
2.2.2	Layer di Orchestrazione	40
2.2.3	Layer di Pre-elaborazione	40
2.2.4	Layer di Training	40
2.2.5	Layer Backend-for-Frontend	40
2.2.6	Layer di Presentazione	40
2.3	Principi e pattern architetturali	41
2.3.1	Principi di design	41
2.3.2	Pattern architetturali implementati	43
2.4	Containerizzazione con Docker e Docker Compose	46
2.4.1	Isolamento delle dipendenze specializzate	46
2.4.2	Orchestrazione con Docker Compose	46
2.4.3	Gestione risorse GPU	46
2.5	Gestione asincrona e code messaggi	47
2.5.1	Code leggere per workflow complessi	47
2.5.2	Stato sempre sincronizzato	47
2.5.3	Resilienza e reprocessing selettivo	47
2.6	Flusso di elaborazione end-to-end	48
2.6.1	Fase di Upload e Inizializzazione	48
2.6.2	Pipeline di Elaborazione	48
3	Implementazioni e tecnologie	53
3.0.1	Stack Tecnologico Implementato	53
3.0.2	Servizi containerizzati	55
3.1	Sistema di parametrizzazione adattiva per livelli di qualità e auto-scaling hardware	58
3.1.1	Struttura dei parametri	58
3.1.2	Processo di Calcolo Parametri	59
3.1.3	Configurabilità e Sperimentazione	59

3.2	Layer di Storage	61
3.2.1	MongoDB - Database dei metadati	61
3.2.2	Amazon S3 - Object Storage	64
3.3	Layer di Orchestrazione	67
3.3.1	Code di messaggi e configurazione RabbitMQ	67
3.3.2	Pattern Producer-Consumer ibrido	69
3.3.3	Persistenza stati intermedi	71
3.3.4	Resilienza e Fault Tolerance	72
3.4	Layer di Pre-elaborazione	73
3.4.1	Architettura del Layer	73
3.4.2	Video Preprocessing Service	74
3.4.3	Point Cloud Reconstruction Service	78
3.4.4	Integrazione e Coordinamento	79
3.5	Layer di Training	80
3.5.1	Architettura containerizzata multi-algoritmo	80
3.5.2	Handler Training	80
3.5.3	Interfaccia API REST unificata	84
3.5.4	Handler Metrics Generation	85
3.6	Layer Backend-for-Frontend	87
3.6.1	Architettura e Funzionalità Principali	87
3.6.2	Riepilogo Endpoint REST e WebSocket	88
3.6.3	Logica e Flussi Principali degli Endpoint	88
3.6.4	Autenticazione e Sicurezza	89
3.6.5	Creazione, fork e retry di un modello	90
3.6.6	Notifiche push in real-time	92
3.6.7	Considerazioni e Best Practice	92
3.7	Layer di Presentazione	92
3.7.1	Architettura dell'interfaccia utente	92
3.7.2	Sistema di notifiche real-time	94
3.7.3	Visualizzatore 3D Integrato	98
3.7.4	Ottimizzazione performance e UX	100
3.7.5	Integrazione con il Backend	100
4	Analisi Sperimentale e Valutazione Prestazioni	103
4.1	Metodologia di Testing	103
4.1.1	Obiettivi della Valutazione Sperimentale	103
4.1.2	Dataset Video e Caratteristiche di Input	103
4.1.3	Pipeline di Preprocessing e Adattamento Hardware	105
4.1.4	Ambiente di Deployment e Configurazione Hardware	107
4.1.5	Metodologia di Valutazione	108
4.2	Benchmark e Risultati Sperimentali	109
4.2.1	Benchmark 1: Confronto Algoritmi e Scalabilità Hardware	109

4.2.2	Benchmark 2: Confronto tra livelli di qualità (Taming 3DGS)	114
4.2.3	Benchmark 3: Latency Budget End-to-End (Balanced, 30k iterazioni)	117
5	Criticità, limitazioni e prospettive di miglioramento	121
5.1	Limitazioni e criticità	121
5.1.1	Acquisizione dei dati	121
5.1.2	Limitazioni hardware e scalabilità	122
5.1.3	Parametrizzazione e ottimizzazione	124
5.1.4	Stabilità e maturità degli algoritmi	125
5.1.5	Output e compatibilità	126
5.2	Sviluppi futuri	128
A	Parametri di configurazione del training	129
A.1	Parametri base (3D Gaussian Splatting)	129
A.2	Parametri aggiuntivi MCMC	129
A.3	Parametri aggiuntivi Taming 3DGS	129
Bibliografia		133

Introduzione

Motivazioni, obiettivi e contributi

Il 3D Gaussian Splatting rappresenta una delle innovazioni più promettenti nel campo della ricostruzione 3D e del novel view synthesis, offrendo un approccio efficiente per la generazione di scene tridimensionali fotorealistiche a partire da un insieme di immagini o video. Questa tecnica, introdotta nel 2023, ha rapidamente guadagnato attenzione nella comunità scientifica per la sua capacità di combinare alta qualità visiva con velocità di rendering in tempo reale una volta completato il training, superando molte delle limitazioni delle tecniche precedenti. Tuttavia, nonostante i risultati impressionanti raggiunti in ambito di ricerca, l'utilizzo del 3D Gaussian Splatting rimane confinato principalmente agli esperti del settore: le implementazioni esistenti richiedono competenze tecniche specifiche, configurazioni complesse e una conoscenza approfondita dei parametri di training. Queste barriere tecniche limitano significativamente l'adozione della tecnologia da parte di utenti non specializzati, impedendo lo sfruttamento del suo potenziale in applicazioni più ampie.

Obiettivi del progetto

Il presente lavoro si propone di colmare questo divario attraverso lo sviluppo di una piattaforma web integrata che renda il 3D Gaussian Splatting accessibile a un pubblico più ampio. Gli obiettivi specifici del progetto includono:

- **Progettazione di un'architettura distribuita:** sviluppo di un sistema basato su microservizi containerizzati che consenta scalabilità e manutenibilità del sistema.
- **Implementazione di una pipeline end-to-end:** creazione di un workflow completo che guida l'utente dal caricamento del video fino alla visualizzazione del modello 3D finale.

- **Integrazione di algoritmi multipli:** supporto per diversi approcci di training (Standard, MCMC, Taming 3DGS) per consentire confronti e ottimizzazioni in base alle esigenze specifiche.
- **Interfaccia utente intuitiva:** sviluppo di un frontend web che semplifichi l'interazione con la tecnologia, nascondendo la complessità tecnica sottostante.
- **Valutazione delle prestazioni:** implementazione di un sistema di raccolta metriche che registra i risultati di qualità (PSNR, SSIM) e tempi di training per ciascun algoritmo, creando le basi per future analisi comparative sistematiche.

Contributi originali

I principali contributi originali di questo lavoro sono:

- **Contributo architettonale:** progettazione di un'architettura a microservizi per il 3D Gaussian Splatting con implementazione di componenti specializzati, inclusi algoritmi di estrazione frame intelligente da video, sistema di preprocessing per la stima delle gaussiane iniziali per algoritmi MCMC e Taming, pipeline di gestione asincrona del workflow completo, e trasformazione adattiva dei dati di output in formati ottimizzati per la fruizione tramite web viewer interattivi.
- **Contributo di integrazione:** Sviluppo di una piattaforma unificata che integra e rende accessibili tre diversi algoritmi di training per il Gaussian Splatting (Standard, MCMC, Taming), fornendo un framework comparativo per la valutazione delle loro prestazioni relative.
- **Contributo di usabilità:** Realizzazione di un'interfaccia web user-friendly che automatizza e semplifica il processo di creazione di modelli 3D, rendendo la tecnologia accessibile anche a utenti senza competenze tecniche specifiche.

Motivazioni tecniche

La scelta di un'architettura distribuita basata su container Docker è motivata da diverse considerazioni tecniche. Il processo di training del Gaussian Splatting è computazionalmente intensivo e può richiedere da alcuni minuti a diverse ore, a seconda della complessità della scena, delle risorse hardware a disposizione e della qualità desiderata. Un'architettura monolitica, pur mantenendo operativo il frontend, limiterebbe significativamente la capacità di gestire multipli job di training concorrenti e impedirebbe l'allocazione efficiente delle risorse computazionali.

L'approccio a microservizi consente di separare le diverse responsabilità del sistema e, soprattutto, di scalare orizzontalmente i componenti più computazionalmente intensivi. I servizi di training, essendo containerizzati e stateless, possono essere deployati su multiple macchine per processare job in parallelo, mentre il frontend e l'API rimangono centralizzati. Questa architettura permette di aggiungere capacità computazionale semplicemente deployando nuove istanze dei container di training su hardware aggiuntivo.

La containerizzazione con Docker offre ulteriori vantaggi in termini di isolamento delle dipendenze, riproducibilità dell'ambiente di esecuzione e facilità di deployment. Ogni componente del sistema può essere sviluppato, testato e deployato indipendentemente, facilitando la manutenzione e l'evoluzione del sistema.

Metodologia di sviluppo

Il progetto ha seguito un approccio di sviluppo esplorativo, caratteristico della ricerca applicata. Partendo dall'analisi delle implementazioni esistenti di 3D Gaussian Splatting, sono stati identificati i requisiti funzionali principali e progettata un'architettura modulare che potesse integrarli efficacemente.

Lo sviluppo è proceduto in modo iterativo, con l'implementazione graduale dei diversi componenti del sistema e la loro integrazione progressiva. Particolare attenzione è stata dedicata alla valutazione delle prestazioni, attraverso la raccolta di metriche oggettive (PSNR, SSIM, tempi di processing) che permettessero una valutazione quantitativa della qualità dei risultati ottenuti.

Struttura della tesi

La tesi è organizzata come segue:

Il **Capitolo 1** presenta lo stato dell'arte del 3D Gaussian Splatting, fornendo il contesto teorico necessario alla comprensione del lavoro. Inizia con una panoramica delle rappresentazioni 3D esistenti, dalle tecniche tradizionali agli approcci neurali emergenti, per poi approfondire i fondamenti teorici del Gaussian Splatting. Vengono descritti i diversi algoritmi di training disponibili (Standard, MCMC, Taming) e analizzate le applicazioni attuali. Il capitolo si conclude identificando le limitazioni che ostacolano l'adozione diffusa della tecnologia.

Il **Capitolo 2** descrive l'architettura del sistema proposto, dettagliando gli obiettivi funzionali e le scelte progettuali adottate. Viene presentato l'approccio a microservizi containerizzati, i diversi componenti del sistema e le

loro interazioni. Il capitolo include il flusso di elaborazione end-to-end e conclude con le motivazioni delle scelte tecnologiche effettuate.

Il **Capitolo 3** illustra i dettagli implementativi dei vari componenti del sistema. Ogni sezione si concentra su un aspetto specifico dell’implementazione: dal frontend con Three.js per la visualizzazione 3D, al backend per la gestione delle API, dal sistema di processing video al motore di training che integra i diversi algoritmi, fino al sistema di code messaggi per la comunicazione asincrona tra servizi.

Il **Capitolo 4** riporta l’analisi sperimentale e la valutazione delle prestazioni del sistema sviluppato. Vengono presentate le metodologie di testing adottate, le metriche utilizzate per la valutazione (PSNR, SSIM, tempi di training), i risultati del confronto tra i diversi algoritmi e l’analisi delle prestazioni dell’architettura containerizzata, evidenziando i punti di forza e i risultati raggiunti.

Il **Capitolo 5** discute le criticità emerse durante lo sviluppo e le limitazioni del sistema attuale. Vengono analizzate le limitazioni tecniche, le sfide architettoniche legate alla gestione distribuita, le limitazioni intrinseche del Gaussian Splatting e le considerazioni relative alla scalabilità e ai costi operativi.

Il **Capitolo 6** presenta le conclusioni del lavoro, riassumendo i risultati ottenuti e il raggiungimento degli obiettivi prefissati. Vengono delineati i possibili miglioramenti futuri, dalle ottimizzazioni algoritmiche all’integrazione di nuove tecniche come il 4D Gaussian Splatting, fino alle estensioni funzionali e alle considerazioni commerciali. Il capitolo si conclude con le lezioni apprese durante lo sviluppo del progetto.

Ringraziamenti

Acknowledgements text

Capitolo 1

Stato dell'Arte del 3D Gaussian Splatting

1.1 Fondamenti di rendering 3D

Prima di analizzare le tecniche di rappresentazione 3D e il Gaussian Splatting, è necessario introdurre alcuni concetti fondamentali del rendering tridimensionale che costituiscono la base teorica per la comprensione degli argomenti trattati in questo lavoro.

1.1.1 Pipeline di rendering

Il rendering 3D è il processo di generazione di immagini bidimensionali a partire da scene tridimensionali. La pipeline di rendering tradizionale può essere suddivisa in fasi principali:

1. **Trasformazione geometrica:** conversione delle coordinate degli oggetti 3D dallo spazio mondiale al sistema di coordinate della camera
2. **Proiezione:** mappatura dei punti 3D sul piano immagine 2D attraverso la camera virtuale
3. **Rasterizzazione:** conversione delle primitive geometriche in pixel discreti
4. **Shading:** calcolo del colore finale di ogni pixel considerando illuminazione e materiali

1.1.2 Tecniche di rendering

Esistono due paradigmi principali per il rendering 3D:

- **Rasterizzazione:** Tecnica che processa le primitive geometriche (triangoli, punti) **proiettandole** direttamente sul piano immagine. È altamente efficiente e ottimizzata per hardware GPU, permettendo rendering real-time. Tuttavia, gestisce con difficoltà effetti ottici complessi come riflessioni accurate e trasparenze multiple.
- **Ray tracing:** Metodo che simula il percorso fisico della luce **lanciando** raggi virtuali dalla camera attraverso ogni pixel dell'immagine. Offre realismo superiore nella gestione di riflessioni, rifrazioni e ombre, ma richiede maggiori risorse computazionali.

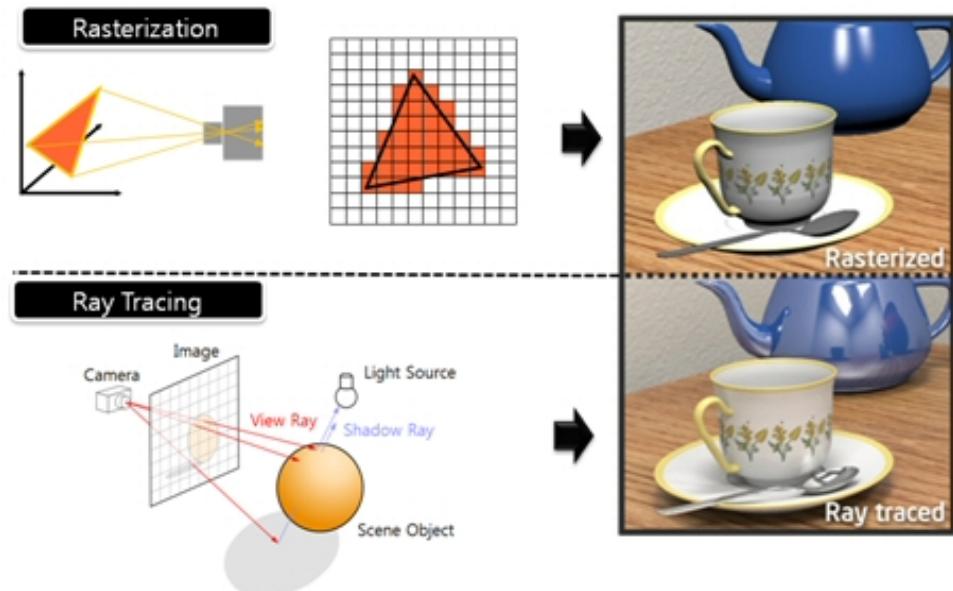


Figura 1.1: Rasterizzazione vs Ray tracing

Fonte: [1]

1.1.3 Proiezione e volume di visibilità

La trasformazione da spazio 3D a immagine 2D avviene attraverso il concetto di **view frustum**, che definisce precisamente quale porzione dello spazio 3D sarà visibile nell'immagine finale.

1.1.3.1 Il view frustum

Il view frustum è un volume tronco-piramidale con vertice nel punto di osservazione della camera (eye point). È delimitato da sei piani:

- **Near plane:** Il piano più vicino alla camera, definisce la distanza minima di rendering
- **Far plane:** Il piano più lontano, oltre il quale gli oggetti non vengono renderizzati
- **Left/Right planes:** Delimitano l'estensione orizzontale del campo visivo
- **Top/Bottom planes:** Definiscono l'estensione verticale del campo visivo

Il frustum è caratterizzato da parametri chiave: il **field of view** (FOV) che determina l'ampiezza angolare della visione, l'**aspect ratio** che definisce il rapporto larghezza/altezza dell'immagine, e le distanze **near** e **far** che stabiliscono l'intervallo di profondità renderizzabile.

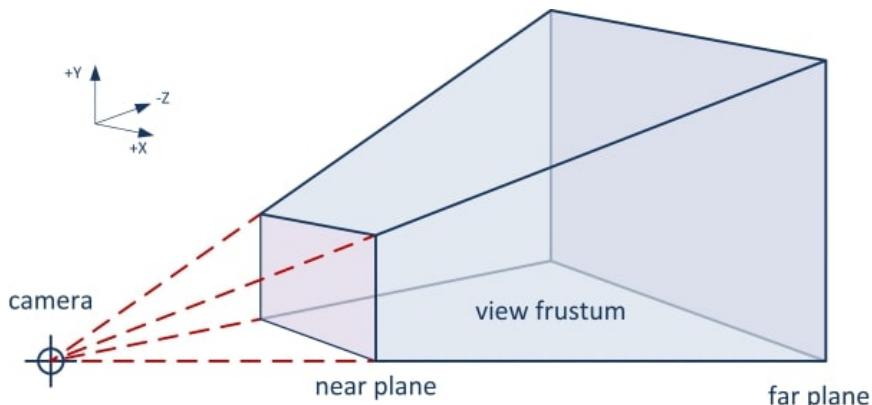


Figura 1.2: Rappresentazione del view frustum

Fonte: [2]

1.1.3.2 Tipi di proiezione

I due tipi principali di proiezione determinano forme diverse del volume di visibilità:

- **Proiezione parallela (o ortografica)** utilizza raggi paralleli per proiettare gli oggetti sul piano immagine, mantenendo le dimensioni relative costanti indipendentemente dalla distanza. In questo caso, il volume di visibilità non è più un frustum ma diventa un **parallelepipedo retto** con pareti parallele, poiché non c'è convergenza dei raggi verso un punto di osservazione. Sebbene sia utile per applicazioni CAD e disegno tecnico dove è necessario preservare proporzioni e misure esatte, non è adatta per il rendering 3D realistico poiché non simula la percezione visiva umana.
- **Proiezione prospettica** mantiene invece la forma tronco-piramidale del view frustum, facendo convergere i raggi di proiezione verso il punto di osservazione della camera. Gli oggetti più distanti appaiono più piccoli a causa della convergenza geometrica del frustum, creando un senso naturale di profondità e realismo visivo.

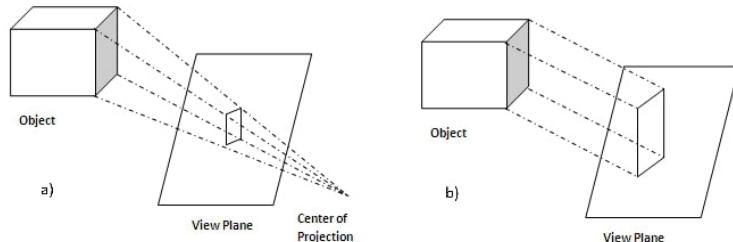


Figura 1.3: Tipi di proiezione: a) proiezione prospettica b) proiezione parallela

Fonte: [3]

Le rispettive **matrici di proiezione**, costruite a partire dai parametri del frustum (FOV, aspect ratio, near/far), trasformano le coordinate 3D in coordinate di clip normalizzate, permettendo contemporaneamente la corretta proiezione e l'eliminazione automatica (culling) di geometrie esterne al volume visibile.

$$P_{ortho} = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & -\frac{2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (1.1.1)$$

Matrice di proiezione ortografica

$$P_{persp} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix} \quad (1.1.2)$$

Matrice di proiezione prospettica

dove n = near plane, f = far plane, l = left, r = right, t = top, b = bottom. La differenza fondamentale risiede nella terza riga: la proiezione prospettica introduce il fattore -1 nell'elemento (4, 3) che causa la divisione prospettica (trasformazione da coordinate omogenee), mentre la proiezione ortografica mantiene $w = 1$, preservando le proporzioni.

1.1.4 Rasterizzazione

La **rasterizzazione** è il processo di conversione delle primitive geometriche continue (triangoli, linee, punti) in pixel discreti su una griglia 2D. Rappresenta la fase finale della pipeline di rendering dove le forme geometriche vengono "disegnate" nell'immagine.

Il processo avviene attraverso questi passaggi:

- **Scan conversion:** Determinazione di quali pixel dell'immagine sono coperti da ogni primitiva geometrica
- **Interpolazione:** Calcolo dei valori degli attributi (colore, coordinate texture, normali) per ogni pixel attraverso interpolazione baricentrica nei triangoli
- **Z-buffering:** Gestione della visibilità tramite buffer di profondità per determinare quale superficie è più vicina alla camera
- **Shading:** Applicazione di modelli di illuminazione per calcolare il colore finale di ogni pixel

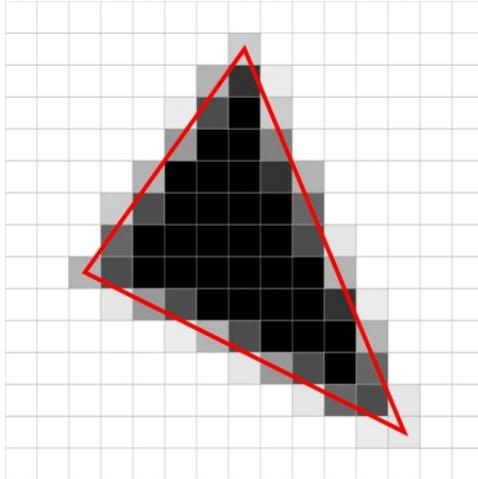


Figura 1.4: Triangolo 3D rappresentato su uno schermo

Fonte: [4]

1.1.5 Considerazioni per GPU moderne

La rasterizzazione è altamente ottimizzata per le GPU moderne grazie alla sua struttura adatta alle pipeline grafiche tradizionali: processa primitive geometriche in sequenza attraverso stadi fissi. Questo la rende ideale per applicazioni real-time.

Il ray tracing, pur essendo anch'esso intrinsecamente parallelo (ogni raggio può essere tracciato indipendentemente), richiede strutture dati complesse per l'accelerazione spaziale e accessi di memoria meno prevedibili, risultando tradizionalmente più costoso computazionalmente. Tuttavia, le GPU moderne con unità RT (Ray Tracing) dedicate stanno rendendo il ray tracing sempre più competitivo anche per applicazioni real-time. Le GPU moderne sono ottimizzate per parallelismo massivo e throughput elevato. Tra le caratteristiche e gli aspetti più importanti abbiamo:

- **Parallelizzazione:** Migliaia di thread processano simultaneamente diversi pixel o primitive
- **Memory bandwidth:** La velocità di accesso alla memoria è spesso il fattore limitante
- **Coerenza spaziale:** Algoritmi che sfruttano la località spaziale dei dati ottengono prestazioni superiori

Questi principi fondamentali influenzano profondamente la progettazione di tecniche moderne come il 3D Gaussian Splatting, che combina l'efficienza della rasterizzazione con la flessibilità di rappresentazioni alternative alle mesh tradizionali.

1.2 Panorama delle rappresentazioni 3D

1.2.0.1 Tecniche tradizionali

Le rappresentazioni 3D tradizionali si basano principalmente su mesh poligonali e point clouds, che differiscono fondamentalmente nella loro organizzazione dei dati.

Le **mesh poligonali** rappresentano dati strutturati: utilizzano triangoli interconnessi per definire superfici continue, dove ogni vertice è connesso ad altri vertici per formare spigoli, che a loro volta si collegano per creare facce triangolari. Questa struttura interconnessa garantisce rappresentazioni geometriche precise e compatibilità con le pipeline di rendering tradizionali, ma richiede maggiore complessità computazionale per la gestione delle connessioni. Inoltre, presentano limitazioni nella gestione di geometrie complesse come capelli, fumo o superfici trasparenti.

Le **point clouds**, d'altra parte, rappresentano dati non strutturati: consistono in insiemi discreti di punti nello spazio 3D, ciascuno con posizione e colore associati, ma senza informazioni di connessione tra di essi. Questa caratteristica le rende più semplici da renderizzare dal punto di vista computazionale, poiché ogni punto può essere processato indipendentemente. Sebbene siano efficaci per la cattura di dati da sensori come LiDAR¹, soffrono di discontinuità visive e difficoltà nel rendering di superfici smooth. Le limitazioni delle rappresentazioni discrete appena citate hanno portato allo sviluppo di approcci volumetrici, che modellano lo spazio 3D come un continuum di proprietà ottiche piuttosto che attraverso elementi geometrici separati.

1.2.1 Volume Rendering: verso rappresentazioni continue

Le limitazioni delle mesh poligonali e dei point clouds hanno portato allo sviluppo di approcci alternativi basati su rappresentazioni volumetriche. Il volume rendering rappresenta un paradigma fondamentale che modella fenomeni difficili da rappresentare con primitive geometriche tradizionali.

1.2.1.1 Principi fondamentali

Come definito da [6], "Direct volume rendering methods generate images of a 3D volumetric data set without explicitly extracting geometric surfaces from the data". Questa tecnica utilizza un modello ottico per mappare i valori dei dati a proprietà ottiche come colore e opacità.

¹Il *LiDAR* (Light Detection and Ranging) è una tecnologia di telerilevamento che misura la distanza dagli oggetti usando impulsi laser. Viene comunemente impiegata per ottenere mappe 3D precise di ambienti, sia in ambito terrestre che aereo.

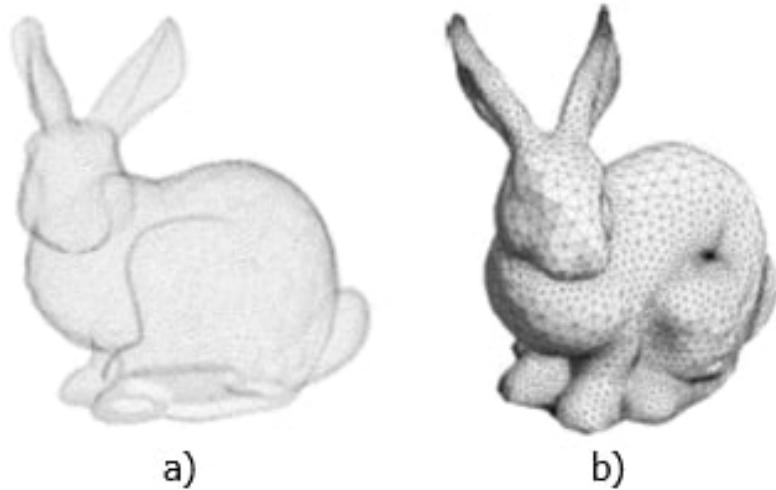


Figura 1.5: Esempi di modelli 3D: a) point cloud b) mesh poligonale
Fonte: [5]

Il volume rendering assume che il volume sia composto da particelle che simultaneamente emettono e assorbono luce [6]. Durante il rendering, le proprietà ottiche vengono accumulate lungo ogni raggio di vista per formare un'immagine dei dati. Ogni elemento di volume (voxel) corrisponde a una posizione nello spazio dei dati e ha uno o più valori associati.

Il processo di generazione dell'immagine avviene attraverso il campionamento del volume lungo tutti i raggi di vista e l'accumulo delle proprietà ottiche risultanti [6]. Per il modello di emissione-assorbimento, il colore e l'opacità accumulati vengono calcolati iterativamente, ordinando i campioni lungo il raggio di vista.

1.2.1.2 Fenomeni volumetrici

Molti effetti visivi sono di natura volumetrica [6]. Fluidi, nuvole, fuoco, fumo, nebbia e polvere sono difficili da modellare con primitive geometriche. I modelli volumetrici sono meglio adatti per creare tali effetti, poiché assumono che la luce venga emessa, assorbita e diffusa da un gran numero di particelle nel volume.

1.2.1.3 Applicazioni scientifiche e ingegneristiche

Oltre alla modellazione di fenomeni volumetrici, il volume rendering è essenziale per applicazioni scientifiche e ingegneristiche che richiedono la visualizzazione di dataset tridimensionali [6]. Gli esempi includono:

- **Imaging medico:** visualizzazione di dati acquisiti da dispositivi di imaging medicale

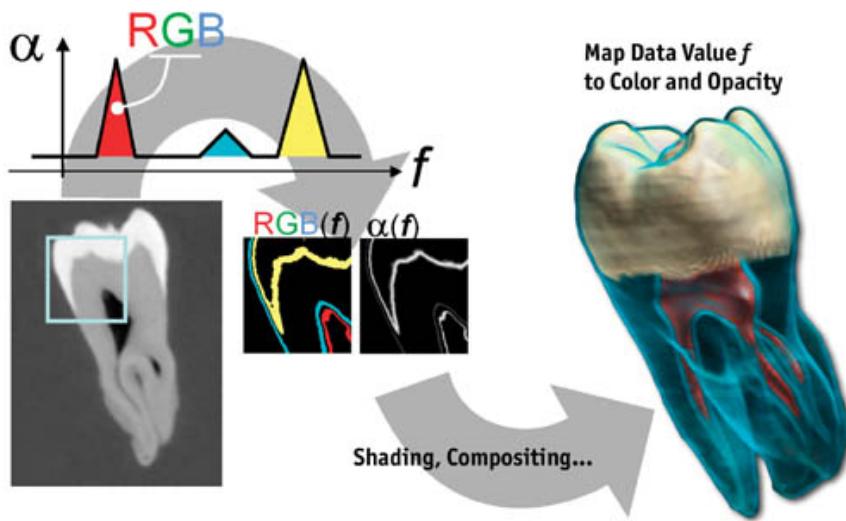


Figura 1.6: Il processo del volume rendering

Fonte: [7]

- **Simulazioni fluidodinamiche:** visualizzazione di risultati da simulazioni computazionali
- **Esplorazione scientifica:** analisi interattiva di dati volumetrici complessi

Gli utenti di applicazioni di volume rendering interattivo si affidano alle prestazioni degli acceleratori grafici moderni per un'esplorazione efficiente dei dati e la scoperta di caratteristiche.

1.2.1.4 Limitazioni tecniche

Il volume rendering presenta diverse sfide computazionali significative [6]:

- **Richieste hardware elevate:** necessita di computer molto potenti per ottenere risultati in tempo reale
- **Velocità di rendering limitata:** il processo è più lento rispetto al rendering di oggetti solidi tradizionali
- **Consumo di memoria:** i dati volumetrici richiedono grandi quantità di memoria per scene dettagliate
- **Complessità algoritmica:** gli algoritmi necessari sono matematicamente e computazionalmente complessi

1.2.1.5 Evoluzione verso tecniche moderne

Il volume rendering ha costituito la base teorica per lo sviluppo di tecniche moderne di **novel view synthesis**, cioè la generazione di immagini realistiché di una scena da angolazioni o posizioni di osservazione non presenti nel set originale di immagini. Le tecniche texture-based descritte da [6] hanno dimostrato come sia possibile combinare facilmente algoritmi poligonali con volume rendering, richiedendo solo pochi pass di rendering e offrendo un alto livello di interattività senza sacrificare la qualità del rendering.

Questa flessibilità ha aperto la strada a rappresentazioni ibride che combinano i vantaggi del volume rendering con l'efficienza di primitive esplicite. Prima i Neural Radiance Fields (NeRF), poi il 3D Gaussian Splatting, possono essere visti come un'evoluzione naturale di questi concetti, mantenendo la capacità di rappresentare fenomeni volumetrici complessi.

L'eredità del volume rendering è particolarmente evidente nella gestione delle proprietà ottiche e nell'accumulo lungo i raggi di vista, principi che rimangono centrali nelle tecniche moderne di neural rendering e rappresentazioni gaussiane.

1.2.1.6 Neural Radiance Fields (NeRF)

L'evoluzione del volume rendering verso approcci basati su machine learning ha trovato la sua espressione più innovativa nei Neural Radiance Fields (NeRF), introdotti da [8]. Questa tecnica rappresenta un salto paradigmatico nella rappresentazione di scene 3D, utilizzando reti neurali per codificare implicitamente volumi continui attraverso funzioni differenziabili.



Figura 1.7: Ottimizzazione di una rappresentazione 5D continua
Fonte: [9]

Rappresentazione 5D continua

NeRF modella una scena statica come una funzione continua 5D che mappa coordinate spaziali 3D (x, y, z) e direzioni di vista 2D (θ, ϕ) a proprietà ottiche: densità volumetrica σ e colore emesso dipendente dalla vista

(r, g, b) . Questa rappresentazione viene approssimata attraverso una rete neurale Multi-Layer Perceptron (MLP)² completamente connessa F_Θ : $(x, d) \rightarrow (c, \sigma)$, dove Θ rappresenta i pesi ottimizzabili della rete.

L’architettura impone vincoli di consistenza multi-vista: la densità volumetrica σ dipende esclusivamente dalla posizione spaziale x , mentre il colore RGB c può variare in funzione sia della posizione che della direzione di vista d . Questa separazione consente di rappresentare effetti non-Lambertiani³ come riflessi speculari e variazioni di colore dipendenti dall’angolo di osservazione.

Rendering volumetrico differenziabile

Il processo di rendering utilizza principi classici del volume rendering per accumulare colore e opacità lungo raggi camera. Per un raggio $r(t) = o + td$ con limiti near e far t_n e t_f , il colore atteso $C(r)$ viene calcolato attraverso l’integrale:

$$C(r) = \int_{t_n}^{t_f} T(t)\sigma(r(t))c(r(t), d)dt \quad (1.2.3)$$

dove $T(t) = \exp(-\int_{t_n}^t \sigma(r(s))ds)$ rappresenta la trasmittanza accumulata lungo il raggio. L’integrale viene approssimato numericamente attraverso campionamento stratificato, evitando le limitazioni di risoluzione delle griglie voxel discrete.

Innovazioni tecniche

NeRF introduce due miglioramenti fondamentali per ottenere risultati di qualità elevata:

- **Positional Encoding:** Le coordinate di input vengono mappate in uno spazio dimensionale superiore utilizzando funzioni sinusoidali ad alta frequenza. Questo approccio, ispirato ai Transformer⁴, consente alla rete di rappresentare dettagli ad alta frequenza che altrimenti verrebbero smussati dalla tendenza delle reti neurali verso funzioni a bassa frequenza.

²Un *Multi-Layer Perceptron* (MLP) è una rete neurale artificiale composta da più strati di neuroni completamente connessi. È in grado di apprendere relazioni non lineari tra input e output attraverso l’addestramento supervisionato.

³Gli *effetti non-Lambertiani* sono fenomeni ottici in cui la luce riflessa da una superficie varia in base all’angolo di osservazione, come accade con i riflessi speculari, le superfici lucide o traslucide. Diversamente dai materiali Lambertiani, che riflettono la luce in modo uniforme, questi effetti dipendono dalla direzione della visuale.

⁴I *Transformer* sono un’architettura di rete neurale basata sull’attenzione, progettata per elaborare sequenze di dati (come testo o tempo) senza l’uso di strutture ricorrenti. Introdotti nel contesto del Natural Language Processing, usano il *positional encoding* per incorporare informazioni sulla posizione degli elementi nella sequenza.

- **Campionamento gerarchico:** Il sistema utilizza due reti distinte (“coarse” e “fine”) per allocare efficientemente i punti di campionamento. La rete coarse fornisce una distribuzione di probabilità che guida un campionamento più informato per la rete fine, concentrando le risorse computazionali nelle regioni più rilevanti del volume.

Vantaggi e limitazioni

I Neural Radiance Fields hanno dimostrato capacità superiori nella sintesi di nuove viste rispetto ai metodi precedenti, particolarmente nella rappresentazione di scene complesse con geometrie intricate e materiali non-Lambertiani. La rappresentazione continua elimina gli artefatti di discretizzazione tipici delle griglie voxel, mentre i requisiti di memoria sono significativamente ridotti rispetto agli approcci volumetrici tradizionali.

Tuttavia, NeRF presenta limitazioni significative in termini di efficienza computazionale. Il training richiede tipicamente 1-2 giorni su GPU high-end per singole scene, mentre il rendering è computazionalmente intensivo, richiedendo centinaia di milioni di query alla rete per singola immagine. Questa inefficienza limita l’applicabilità in contesti real-time o interattivi.

Impatto e eredità

NeRF ha stabilito i fondamenti teorici per una nuova generazione di tecniche di novel view synthesis, influenzando lo sviluppo di rappresentazioni ibride che combinano i vantaggi del rendering volumetrico con primitive più efficienti. I principi di rappresentazione continua, rendering differenziabile e ottimizzazione basata su immagini costituiscono l’eredità concettuale che ha ispirato successive innovazioni come il 3D Gaussian Splatting.

1.3 Fondamenti teorici del Gaussian Splatting

1.3.1 Rappresentazione 3D basata su gaussiane

Il 3D Gaussian Splatting introduce un paradigma innovativo per la rappresentazione di scene tridimensionali, utilizzando primitive gaussiane 3D come elementi fondamentali in alternativa alle tradizionali mesh poligonali o ai Neural Radiance Fields impliciti.

Mentre le metodologie consolidate si basano su mesh poligonali progettate da artisti 3D o su nuvole di punti generate tramite tecniche di fotogrammetria classica, il 3D Gaussian Splatting adotta un approccio radicalmente diverso. Il dataset di partenza consiste in foto o video acquisiti direttamente da oggetti o scene del mondo reale, oppure in scene sintetiche generate artificialmente tramite motori grafici o dataset simulati. Questi dati vengono successivamente elaborati attraverso algoritmi specializzati per gene-

rare una rappresentazione tridimensionale fotorealistica basata su primitive gaussiane.

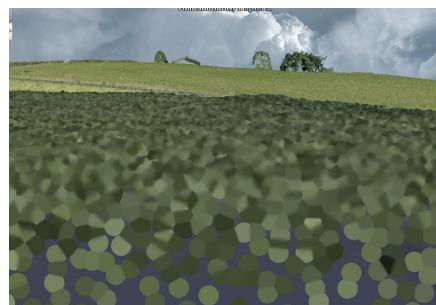
Questa metodologia offre il vantaggio significativo di catturare scene complesse del mondo reale con un livello di fedeltà visiva superiore, preservando dettagli come riflessi accurati ed effetti anisotropi direzionali che risultano difficili da riprodurre con le tecniche tradizionali di computer grafica.

Il concetto di "Splats" nella computer grafica

Per comprendere meglio il concetto di primitiva gaussiana, è utile introdurre il termine "splats", una tecnica di computer grafica introdotta nel 2001 che ha trovato diverse applicazioni nel rendering 3D. Nel contesto del rendering di point cloud, gli splats hanno dimostrato la loro efficacia nel migliorare significativamente la qualità visiva delle rappresentazioni basate su punti. Quando si renderizza una point cloud tradizionale, ogni punto viene semplicemente colorato con il proprio colore specifico. Tuttavia, la tecnica degli splats introduce un approccio più sofisticato: durante il processo di shading di ciascun punto primitivo sullo schermo, il sistema tiene conto anche del colore dei punti posizionati dietro di esso. Questo permette di renderizzare "splats" che fondono i punti in modo armonioso dal punto di vista visivo, creando una rappresentazione molto più fluida e fotorealistica della point cloud. La differenza visiva tra una point cloud tradizionale e una renderizzata con splats è particolarmente evidente: mentre la prima appare spesso frammentata e discontinua, la seconda presenta superfici continue e naturali che risultano molto più accattivanti anche per chi non ha familiarità con il rendering 3D. Questo principio di blending intelligente tra primitive adiacenti costituisce la base concettuale su cui si fonda il 3D Gaussian Splatting.



(a) Point cloud tradizionale



(b) Point cloud con splatting

Figura 1.8: Confronto tra point cloud tradizionale e splatting
[10], [11] ,

Primitive gaussiane come building blocks

Una **gaussiana 3D** può essere concettualizzata come un ellissoide tridimensionale che rappresenta un "pennellata" nello spazio 3D con proprietà variabili. A differenza dei triangoli utilizzati nelle mesh tradizionali o dei punti discreti delle point cloud, le gaussiane offrono una rappresentazione continua e flessibile che può adattarsi a diverse forme geometriche.

Ogni gaussiana 3D è definita da un insieme di parametri fondamentali:

- **Posizione media μ :** Le coordinate XYZ del centro della gaussiana nello spazio tridimensionale
- **Matrice di covarianza Σ :** Una matrice 3×3 che definisce la forma, le dimensioni e l'orientamento dell'ellissoide gaussiano
- **Opacità α :** Il grado di trasparenza della primitiva, cruciale per il blending tra gaussiane sovrapposte
- **Colore dipendente dalla vista:** Valori RGB che possono essere view-dependent attraverso l'uso di armoniche sferiche

La funzione gaussiana 3D è espressa come:

$$G(x) = \exp\left(-\frac{1}{2}(x - \mu)^T \Sigma^{-1} (x - \mu)\right) \quad (1.3.4)$$

Vantaggi della rappresentazione gaussiana

La scelta delle gaussiane come primitive presenta diversi vantaggi significativi:

- **Compattezza rappresentativa:** Scene complesse possono essere rappresentate con 1-5 milioni di gaussiane, mentre point cloud equivalenti potrebbero richiedere decine di milioni o miliardi di punti per la stessa qualità visiva.
- **Fotorealismo:** Le gaussiane permettono di ottenere risultati visivi estremamente realistici, catturando dettagli sottili di illuminazione, ombre e riflessi che spesso mancano in altre rappresentazioni 3D.
- **Rappresentazione di geometrie complesse:** A differenza delle mesh che faticano con superfici non-manifold⁵ o topologie complesse, le gaussiane possono rappresentare naturalmente geometrie intricate, dettagli fini e strutture organiche.
- **Proprietà anisotrope:** Le gaussiane possono avere valori diversi quando osservate da direzioni diverse. Questa anisotropia, implementata attraverso armoniche sferiche, consente sia di rappresentare strutture geometriche sottili (come fili d'erba o capelli) che di catturare effetti ottici realistici come riflessi che cambiano con il punto di vista.

1.3.2 Pipeline di training

Il processo di creazione di una rappresentazione Gaussian Splatting segue una pipeline strutturata che trasforma immagini o video di input in un insieme ottimizzato di gaussiane 3D. La pipeline è così strutturata:

1. **Structure from Motion (SfM):** Il processo inizia con l'analisi delle immagini di input utilizzando tecniche di Structure from Motion⁶. È

⁵Una superficie è *manifold* se l'intorno di ogni punto è topologicamente equivalente a un disco e ogni spigolo è condiviso da esattamente due facce. Esempi di geometrie *non-manifold* includono: spigoli condivisi da tre o più facce, vertici *bow-tie*, giunzioni a T e lamine di spessore nullo.

⁶La *Structure from Motion* (SfM) è una tecnica di visione artificiale che ricostruisce la geometria tridimensionale di una scena e le traiettorie della camera a partire da immagini bidimensionali multiple, sfruttando la parallasse tra viste. Per un approfondimento si rimanda alla sezione 1.4.

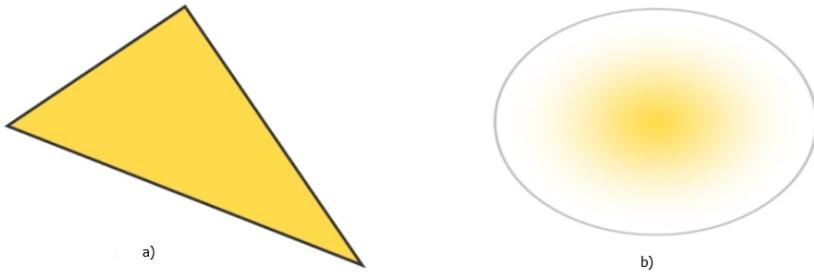


Figura 1.9: Due diverse primitive: a) triangolo b) gaussiana

necessario che le immagini abbiano sovrapposizione adeguata e coprano la scena da angolazioni diverse. Questa fase genera una point cloud sparsa e stima le pose delle camere, fornendo la struttura geometrica iniziale della scena.

2. **Inizializzazione delle gaussiane:** Ogni punto della point cloud sparsa viene successivamente convertito in una gaussiana 3D. Ogni gaussiana viene inizializzata con la posizione del punto corrispondente, il colore derivato dalle osservazioni delle camere, e parametri geometrici di default (covarianza isotropa e opacità iniziale).
3. **Ottimizzazione differenziabile:** Il cuore del training consiste nell'ottimizzazione dei parametri gaussiani attraverso **Stochastic Gradient Descent (SGD)**. SGD è un algoritmo di ottimizzazione che trova iterativamente i parametri del modello che minimizzano la differenza tra risultati predetti e target reali.

A ogni iterazione, il processo seleziona una singola immagine dal dataset di training e confronta l'immagine renderizzata dalla rappresentazione gaussiana corrente con questa immagine target, calcolando una loss function \mathcal{L} che combina:

- \mathcal{L}_1 (**L1 loss**): misura la differenza assoluta tra pixel corrispondenti delle immagini
- $\mathcal{L}_{\text{D-SSIM}}$ (**Structural Dissimilarity**): valuta la similarità strutturale e percettiva tra le immagini, catturando differenze che l'occhio umano noterebbe meglio della semplice differenza di colore

La funzione di loss combinata è definita come:

$$\mathcal{L} = (1 - \lambda)\mathcal{L}_1 + \lambda\mathcal{L}_{\text{D-SSIM}} \quad (1.3.5)$$

dove λ è un iperparametro (tipicamente 0.2) che bilancia l'importanza tra accuratezza del colore e similarità strutturale: \mathcal{L}_1 garantisce l'ac-

curatezza dei colori di base, che è più critica per la convergenza dell'ottimizzazione rispetto ai dettagli strutturali catturati da D-SSIM. L'algoritmo varia i parametri della gaussiane (posizione, colore, forma, dimensione e trasparenza) basandosi su questa funzione di loss.

4. **Controllo adattivo della densità:** Durante il training, il sistema implementa strategie di densificazione e pruning per ottimizzare il numero e la distribuzione delle gaussiane. Quando una regione è sottorepresentata, le gaussiane vengono duplicate (cloning) o suddivise (splitting) per catturare dettagli fini. Di contro, gaussiane con opacità molto bassa vengono rimosse per mantenere efficienza computazionale.

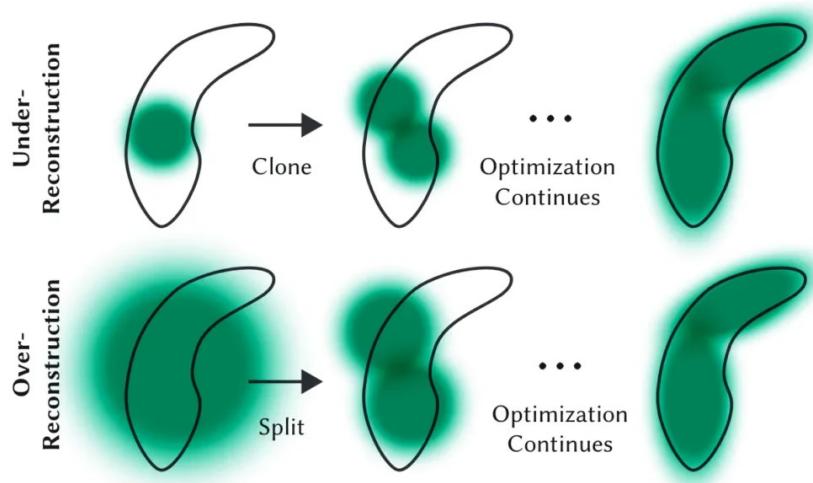


Figura 1.10: Densificazione della gaussiane cloning e splitting

Rasterizzazione differenziabile

Perché sia possibile l'ottimizzazione descritta sopra, è fondamentale che il processo di rendering sia differenziabile - ovvero che permetta di calcolare come una piccola modifica ai parametri delle gaussiane influenzi il risultato finale dell'immagine renderizzata. Questo collegamento è realizzato attraverso il processo di rasterizzazione differenziabile. Il rendering delle gaussiane 3D avviene attraverso un processo di rasterizzazione tile-based ottimizzato per hardware GPU, composto dai seguenti passaggi:

1. **Proiezione e ordinamento:** Le gaussiane 3D vengono proiettate sul piano immagine 2D dalla prospettiva della camera. Successivamente, il sistema suddivide lo schermo in tile di 16×16 pixel e raggruppa le gaussiane per tile in base alla loro posizione proiettata.

2. **Sorting per depth:** All'interno di ogni tile, le gaussiane vengono ordinate per profondità (distanza dal piano dell'immagine), permettendo un corretto alpha-blending front-to-back durante il rendering.
3. **Alpha-blending ottimizzato:** Per ogni pixel, il sistema esegue alpha-blending⁷ delle gaussiane ordinate, combinando i loro contributi di colore e opacità. Questo processo genera gradienti smooth e permette la rappresentazione di superfici continue e effetti di trasparenza complessi.

Il processo di rasterizzazione è completamente differenziabile, permettendo la propagazione dei gradienti attraverso l'intera pipeline di rendering per l'ottimizzazione end-to-end.

View-dependent rendering

Una caratteristica distintiva del Gaussian Splatting è la capacità di rappresentare effetti ottici che dipendono dalla direzione di osservazione. Questo viene ottenuto attraverso l'uso di **armoniche sferiche (Spherical Harmonics, SH)**.

Ogni gaussiana memorizza coefficienti di armoniche sferiche che modulano il colore in base alla direzione di vista. Questa rappresentazione permette di catturare fenomeni come:

- Riflessi speculari su superfici lucide
- Variazioni di colore dovute a materiali anisotropi
- Effetti di illuminazione complessi
- Proprietà ottiche direzionali dei materiali

1.3.3 Strutture dati per gaussiane 3D

Ogni gaussiana 3D richiede la memorizzazione di diversi parametri:

- **Posizione:** 3 coordinate float (12 bytes)
- **Rotazione:** 4 componenti quaternion (16 bytes)
- **Scala:** 3 valori per gli assi dell'ellissoide (12 bytes)
- **Opacità:** 1 valore alpha (4 bytes)

⁷L'*alpha-blending* è una tecnica di compositing che combina più colori in base alla loro opacità (canale alpha), producendo effetti di trasparenza graduata.

- **Coefficienti SH:** Variabili in base al grado delle armoniche sferiche (tipicamente 48-192 bytes)

Una singola gaussiana richiede quindi circa 92-236 bytes, e con rappresentazioni contenenti 1-5 milioni di gaussiane, i dataset possono facilmente raggiungere centinaia di MB o diversi GB.

1.3.3.1 Considerazioni per applicazioni web

Due fattori critici influenzano l'implementazione web:

- **Tempi di caricamento:** La dimensione dei dataset può variare significativamente. Scene semplici possono richiedere 50-100 MB, mentre scene complesse possono superare i 2-6 GB. Per applicazioni web, tempi di download prolungati compromettono l'esperienza utente.
- **Performance di rendering:** Le prestazioni sono misurate in FPS (frames per second), con l'obiettivo di mantenere 30 FPS per interazioni fluide. La variabilità dell'hardware utente (da dispositivi mobili economici a workstation grafiche) richiede strategie di ottimizzazione adattive.

1.3.3.2 Formati e ottimizzazioni

La rappresentazione GSplat, essendo concettualmente simile alle point cloud, utilizza formati di file che riflettono questa affinità. Il formato più comunemente adottato è il PLY (.ply), un formato standard ampiamente utilizzato per la memorizzazione di dati geometrici 3D. Questo formato si adatta naturalmente alla struttura dei dati gaussiani, permettendo di memorizzare tutti i parametri essenziali di ogni gaussiana: posizione, matrice di covarianza, opacità e coefficienti delle armoniche sferiche per il colore view-dependent. Parallelamente allo sviluppo della tecnologia GSplat, la comunità open source sta creando formati specializzati come .splat e .ksplat (versione compressa), ottimizzati specificamente per le caratteristiche uniche delle gaussiane 3D. Questo è un campo in rapida evoluzione, con nuovi formati che emergono continuamente per rispondere alle esigenze specifiche di diverse applicazioni. Tuttavia, per applicazioni web potrebbero essere necessarie una o più delle seguenti ottimizzazioni:

- **Compressione:** Tecniche di quantizzazione per ridurre la precisione floating-point mantenendo qualità visiva
- **Streaming progressivo:** Caricamento incrementale per visualizzazione immediata con raffinamento graduale
- **Level-of-Detail (LOD):** Rappresentazioni multiple a diverse risoluzioni per adattarsi alle capacità hardware

- **Culling spaziale:** Eliminazione di gaussiane non visibili per ridurre il carico computazionale

Queste considerazioni sono fondamentali per lo sviluppo di piattaforme accessibili che rendano il 3D Gaussian Splatting fruibile su una vasta gamma di dispositivi e connessioni di rete.

1.3.4 Posizionamento nel panorama delle tecniche attuali

Il 3D Gaussian Splatting si posiziona come soluzione ibrida che combina vantaggi di diverse famiglie di tecniche di rappresentazione 3D.

Rispetto alle mesh tradizionali: Mentre le mesh eccellono nella rappresentazione di superfici ben definite e sono ottimizzate per pipeline di rendering consolidate, il Gaussian Splatting offre maggiore flessibilità per fenomeni volumetrici e materiali trasparenti, pur mantenendo efficienza di rendering comparabile.

Rispetto ai Neural Radiance Fields: NeRF utilizza reti neurali per rappresentare implicitamente radiance field volumetrici, ottenendo alta qualità visiva ma richiedendo tempi di rendering significativamente più lunghi (secondi per frame). Il Gaussian Splatting raggiunge qualità visiva comparabile con velocità di rendering real-time (≥ 30 FPS), eliminando la necessità di query neurali durante il rendering.

Rispetto alle point cloud: Condividendo la natura di dati non strutturati, il Gaussian Splatting supera le limitazioni delle point cloud tradizionali attraverso primitive più espressive (ellissoidi vs punti) e capacità di rappresentare continuità spaziale e effetti ottici complessi.

Innovazioni architetturali

Il successo del Gaussian Splatting deriva da diverse innovazioni tecniche chiave:

- **Rasterizzazione tile-based:** L'approccio di suddivisione in tile riduce significativamente la complessità computazionale del sorting e dell'alpha-blending, rendendo possibile il rendering real-time di milioni di gaussiane.
- **Differenziabilità end-to-end:** L'intera pipeline, dalla rappresentazione 3D al rendering 2D, è completamente differenziabile, permettendo ottimizzazione diretta tramite gradient descent senza approssimazioni.

- **Controllo adattivo automatico:** Il sistema di densificazione e pruning automatico permette di bilanciare dinamicamente qualità visiva e efficienza computazionale durante il training.

Queste caratteristiche posizionano il 3D Gaussian Splatting come tecnica particolarmente adatta per applicazioni che richiedono il bilanciamento di alta qualità visiva, efficienza computazionale e tempi di training ragionevoli, rendendolo ideale per lo sviluppo di piattaforme accessibili come quella proposta in questo lavoro.

1.4 Structure from Motion e COLMAP: prerequisiti per 3D Gaussian Splatting

Prima di fornire una panoramica sugli algoritmi coinvolti nella realizzazione di questo specifico progetto, è fondamentale comprendere il processo di preprocessing che rende possibile l'applicazione di queste tecniche. Tutti gli algoritmi di 3DGS, indipendentemente dalle loro specifiche implementazioni, condividono un requisito comune: la necessità di una nuvola di punti 3D iniziale e delle pose precise delle camere per ogni immagine di input. Questi dati vengono tipicamente generati attraverso tecniche di Structure from Motion (SfM), con COLMAP che rappresenta lo standard de facto per questa fase di preprocessing.

1.4.1 Structure from Motion: ricostruzione 3D da immagini

Structure from Motion (SfM) è una famiglia di algoritmi di computer vision che permette di ricostruire la struttura tridimensionale di una scena e le posizioni delle camere a partire da una collezione di immagini bidimensionali non ordinate. Il principio fondamentale si basa sulla geometria epipolare: quando lo stesso punto fisico viene osservato da due o più posizioni differenti, è possibile triangolare la sua posizione nello spazio 3D utilizzando i principi della geometria proiettiva.

Il processo SfM può essere sintetizzato in tre fasi principali. Inizialmente, vengono estratti punti caratteristici (feature) da ogni immagine utilizzando algoritmi come SIFT, SURF o ORB. Successivamente, questi punti vengono messi in corrispondenza tra immagini diverse per identificare lo stesso punto fisico osservato da angolazioni multiple. Infine, utilizzando queste corrispondenze, viene ricostruita simultaneamente la struttura 3D della scena e le pose delle camere attraverso tecniche di ottimizzazione non lineare.

L'output di un algoritmo SfM consiste tipicamente in una nuvola di punti 3D sparsa che rappresenta i punti caratteristici della scena, le pose delle camere (posizione e orientamento) per ogni immagine, e i parametri intrinseci delle camere (lunghezza focale, centro ottico, distorsioni). Questa rappresentazione sparsa, pur non catturando completamente la geometria della scena,

fornisce una base solida per tecniche di ricostruzione più dense come il 3D Gaussian Splatting.

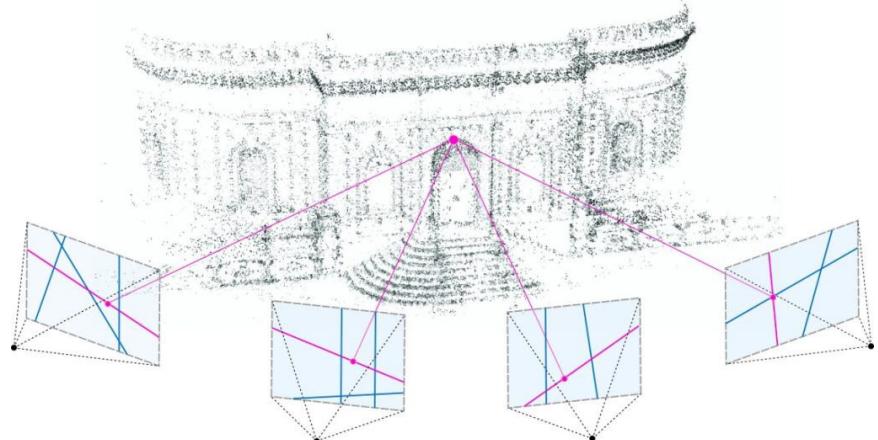


Figura 1.11: Rappresentazione del concetto di Structure From Motion
Fonte: [12]

1.4.2 COLMAP: implementazione di riferimento per SfM

COLMAP (COLlaborative Large-scale universal MApping and Positioning) è un framework open-source per Structure from Motion e Multi-View Stereo sviluppato presso l’ETH Zurich. Rappresenta una delle implementazioni più robuste e complete di algoritmi SfM, combinando efficienza computazionale con accuratezza geometrica. COLMAP è diventato lo standard de facto nella comunità di computer vision per la ricostruzione 3D da immagini, supportando una vasta gamma di scenari applicativi.

La pipeline COLMAP implementa un approccio incrementale alla ricostruzione, iniziando con una coppia di immagini ben condizionata e aggiungendo progressivamente nuove viste. Questo metodo garantisce robustezza anche in presenza di scene complesse o dati di input rumorosi. Il sistema integra automaticamente tecniche di bundle adjustment⁸ per l’ottimizzazione simultanea di tutti i parametri, gestione intelligente degli outlier⁹, e supporto per diversi modelli di camera e condizioni di acquisizione.

Una caratteristica distintiva di COLMAP è la sua capacità di gestire automaticamente molte delle complessità tipiche della ricostruzione SfM, come la

⁸Il *bundle adjustment* è una tecnica di ottimizzazione non lineare che raffina simultaneamente tutti i parametri della ricostruzione 3D (posizioni dei punti, pose delle camere e parametri intrinseci) minimizzando l’errore di riproiezione tra i punti ricostruiti e le loro osservazioni nelle immagini.

⁹Gli *outlier* sono dati errati o rumorosi che non seguono il modello geometrico della scena, come corrispondenze sbagliate tra feature o punti 3D mal triangolati, che devono essere identificati e rimossi per garantire l’accuratezza della ricostruzione.

selezione delle immagini di inizializzazione, la gestione di scene con componenti disconnesse, e l'identificazione di loop closure¹⁰ in sequenze di immagini. Queste funzionalità rendono COLMAP particolarmente adatto come preprocessing per applicazioni downstream come il 3D Gaussian Splatting.

1.4.3 Integrazione COLMAP-3DGS: dal preprocessing al rendering

La connessione tra COLMAP e 3D Gaussian Splatting è fondamentale e bidirezionale. COLMAP fornisce i dati di input essenziali per tutti gli algoritmi 3DGS, mentre la qualità della ricostruzione COLMAP determina direttamente le prestazioni achievable dai metodi di Gaussian Splatting.

L'output principale di COLMAP utilizzato da 3DGS è la nuvola di punti 3D sparsa, dove ogni punto viene utilizzato per inizializzare una gaussiana 3D. Le coordinate del punto diventano la posizione media della gaussiana, mentre parametri come scala, rotazione e opacità vengono inizializzati utilizzando euristiche. Inoltre, COLMAP fornisce le pose precise delle camere, indispensabili per il processo di rendering differenziabile che costituisce il cuore dell'ottimizzazione 3DGS.

La qualità della ricostruzione COLMAP influenza profondamente le prestazioni finali del sistema 3DGS. Una nuvola di punti densa e accurata fornisce una migliore inizializzazione, riducendo il tempo di convergenza e migliorando la qualità finale. Di contro, errori nella stima delle pose delle camere o nella triangolazione dei punti 3D si propagano attraverso l'intero processo di training, limitando la fedeltà del rendering finale.

È importante notare che COLMAP ricostruisce tipicamente solo i punti più caratteristici e facilmente triangolabili della scena, risultando in una rappresentazione sparsa. Aree uniformi, superfici riflettenti, o regioni con texture insufficiente potrebbero non essere rappresentate nella nuvola di punti iniziale. Questa limitazione è uno dei motivi per cui gli algoritmi 3DGS includono meccanismi di densificazione adattiva durante il training, per colmare le lacune nella rappresentazione iniziale.

1.4.4 Requisiti e limitazioni

Per ottenere risultati di qualità con la pipeline COLMAP-3DGS, è necessario rispettare alcuni requisiti fondamentali nella fase di acquisizione dati. Le immagini devono presentare un overlap sufficiente (tipicamente 60-80% tra immagini adiacenti) per garantire corrispondenze robuste. La scena deve contenere texture sufficienti per la feature detection, ed è preferibile mantenere condizioni di illuminazione consistenti durante l'acquisizione.

¹⁰La *loop closure* è il riconoscimento che la camera è tornata in una posizione già visitata, fondamentale per correggere la deriva accumulata durante la ricostruzione incrementale e migliorare la coerenza globale della scena ricostruita.

Le limitazioni principali riguardano scene con caratteristiche problematiche per gli algoritmi SfM: superfici uniformi o riflettenti che non generano feature distintive, pattern ripetitivi che possono causare corrispondenze errate, e variazioni di illuminazione drastiche che influenzano la consistenza delle feature. Inoltre, COLMAP, come tutti gli algoritmi SfM, richiede che la scena sia statica durante l'acquisizione.

Nonostante queste limitazioni, la combinazione COLMAP-3DGS rappresenta attualmente la soluzione più matura e affidabile per la generazione di radiance field di alta qualità a partire da immagini multi-vista. La robustezza di COLMAP nella fase di preprocessing, combinata con la capacità dei metodi 3DGS di densificare e raffinare la rappresentazione durante il training, permette di ottenere risultati di qualità eccellente su una vasta gamma di scene reali.

La comprensione di questa pipeline integrata è essenziale per l'utilizzo efficace delle tecniche di 3D Gaussian Splatting, poiché la qualità del preprocessing COLMAP costituisce spesso il fattore limitante per le prestazioni complessive del sistema.

1.5 Algoritmi di training utilizzati

1.5.1 Approccio standard (3DGS baseline)

L'implementazione originale di Kerbl et al. si basa su un approccio di ottimizzazione diretta mediante gradient descent, dove ogni gaussiana 3D viene parametrizzata attraverso posizione, covarianza (scala e rotazione), opacità e coefficienti delle armoniche sferiche per il colore. Il processo di training alterna fasi di ottimizzazione dei parametri gaussiani con operazioni di controllo della densità (densificazione e pruning) per adattare dinamicamente la rappresentazione alla complessità della scena. Questo approccio rappresenta il metodo standard e più consolidato, caratterizzato da semplicità implementativa e robustezza.

Vantaggi chiave

- Risultati stabili e riproducibili
- Requisiti hardware accessibili
- Buon bilanciamento qualità/prestazioni per scene standard

Svantaggi

- Tendenza a generare artefatti "needle-like" in alcune configurazioni
- Gestione non ottimale nella distribuzione delle Gaussiane

- Difficoltà in scene molto complesse o dense
- Controllo limitato sull'allocazione automatica delle risorse
- Tendenza a stabilizzarsi in configurazioni subottimali della scena, tipico dei metodi ottimizzati per gradient descent locale

1.5.2 Monte Carlo Markov Chain: miglioramento qualità tramite campionamento probabilistico

Il metodo MCMC (Monte Carlo Markov Chain), applicato al training delle Gaussiane 3D, rappresenta un'estensione del metodo originale proposto nel paper fondamentale di 3D Gaussian Splatting. Questo approccio avanzato utilizza una variante chiamata **Stochastic Gradient Langevin Dynamics (SGLD)** per campionare le configurazioni delle primitive da una distribuzione target, introducendo un rumore stocastico controllato che permette una migliore esplorazione dello spazio delle soluzioni e previene il collasso in minimi locali. Il framework MCMC eredita tutti i parametri in input già elencati in precedenza e ne introduce di aggiuntivi in modo da sfruttare strategie di ottimizzazione più sofisticate.

Inoltre, il metodo include un meccanismo di regolazione della complessità del modello, che consente di controllare dinamicamente il numero di Gaussiane attive durante il training, con l'obiettivo di bilanciare efficacia della rappresentazione e uso efficiente delle risorse computazionali. Questa estensione affronta specificamente alcune limitazioni del metodo originale, come la distribuzione spaziale non ottimale delle Gaussiane e la presenza di Gaussiane "morte" (cioè primitive che cessano di contribuire attivamente al rendering durante il processo di training).

Vantaggi chiave

- Relocazione automatica ed efficace delle Gaussiane non attive
- Prevenzione della stagnazione in minimi locali subottimali
- Riduzione degli artefatti visivi (needle-like)
- Distribuzione spaziale più regolare e adattiva
- Migliore esplorazione dello spazio delle soluzioni tramite rumore controllato

Svantaggi

- Aumento della complessità implementativa e concettuale
- Parametri aggiuntivi da ottimizzare con attenzione

- Overhead computazionale dovuto al sampling stocastico
- Possibile rallentamento della convergenza se `noise_lr` è mal bilanciato
- Risultati leggermente meno deterministicici a causa dell'introduzione del rumore

1.5.3 Taming 3DGS: ottimizzazione per risorse computazionali limitate

Il metodo Taming 3DGS nasce come risposta alle limitazioni computazionali tipiche del Gaussian Splatting su dispositivi non high-end. Il suo contributo principale è l'introduzione di un sistema incrementale e score-guided per la densificazione, in cui ogni nuova Gaussiana viene inserita solo se il suo apporto qualitativo alla scena giustifica il costo computazionale. L'intero processo è regolato da un sistema di punteggio che tiene conto della visibilità, dell'influenza sul gradiente della loss e del contributo alla qualità visiva.

Anche in questo caso, è previsto un sistema di controllo della complessità del modello, che consente di limitare o guidare la crescita del numero di Gaussiane in base a parametri specifici. Questo approccio si traduce in un controllo preciso e prevedibile delle risorse, con implicazioni dirette su tempo, memoria e qualità. L'architettura del metodo evita picchi di memoria durante il training, permettendo la generazione di scene complesse anche su hardware con risorse limitate. Grazie a parametri mirati, è possibile adattare dinamicamente la crescita del modello in funzione della scena e dei vincoli imposti (es. massimo numero di Gaussiane o soglie qualitative minime).

- Controllo preciso delle risorse computazionali
- Scalabilità su hardware limitato
- Processo costruttivo senza picchi di memoria
- Densificazione intelligente basata su score
- Adattabilità automatica alla complessità della scena

Svantaggi

- Possibile perdita di dettagli fini con budget limitati
- Complessità aggiuntiva nella configurazione dei parametri
- Overhead computazionale per il calcolo degli score
- Dipendenza dalla stima accurata dello score di importanza delle gaussiane
- Potenziale sottoottimizzazione per rispettare i vincoli di budget

1.5.4 Parametri di configurazione del training

I parametri di configurazione permettono di controllare tutti gli aspetti del training nei tre metodi presentati: durata dell'ottimizzazione, gestione delle risorse computazionali e qualità finale del modello. Ogni metodo estende il set di parametri base con opzioni specifiche per le proprie caratteristiche innovative.

I parametri base del 3D Gaussian Splatting (Tabella A.1) controllano aspetti fondamentali come numero di iterazioni, risoluzione e densificazione. Il metodo MCMC aggiunge parametri di regolarizzazione (Tabella A.2), mentre Taming 3DGS introduce controlli specifici per la gestione delle risorse (Tabella A.3). Le tabelle complete con tutti i parametri e i loro effetti sono riportate nell'Appendice A.

1.5.5 Confronto degli approcci

Criterio	3DGS	MCMC	Taming
Qualità risultato	Buona	Superiore	Adattabile al budget
Velocità training	Veloce	Moderata (overhead)	Veloce
Uso memoria	Variabile	Controllato	Predicibile
Complessità impl.	Bassa	Alta (catena MCMC)	Moderata
Requisiti HW	Moderati	Moderati–Alti	Flessibili
Stabilità risultati	Alta	Moderata	Alta
Controllo risorse	Limitato	Moderato	Eccellente

Tabella 1.1: Confronto tra metodi

Raccomandazioni d'uso

- **3DGS Standard:** ideale per applicazioni standard, prototipazione rapida e quando la semplicità implementativa è prioritaria
- **3DGS MCMC:** consigliato per scene complesse dove la qualità è critica e si dispone di risorse computazionali adeguate
- **Taming 3DGS:** ottimale per deployment su hardware limitato, applicazioni real-time e quando il controllo delle risorse è fondamentale

1.6 Metriche di Valutazione

La valutazione quantitativa della qualità dei modelli 3D generati tramite tecniche di rendering neurale rappresenta una sfida fondamentale nel campo della computer vision. A differenza delle tecniche tradizionali di modellazione 3D, dove la valutazione può concentrarsi su aspetti puramente geometrici, i metodi basati su Gaussian Splatting richiedono metriche che catturino sia la fedeltà visiva che la percezione umana del realismo.

1.6.1 Peak Signal-to-Noise Ratio (PSNR)

Il PSNR rappresenta la metrica più diffusa per la valutazione pixel-wise della qualità di immagini ricostruite. Definito come il rapporto logaritmico tra il valore massimo possibile del segnale e la potenza del rumore di distorsione, il PSNR fornisce una misura oggettiva della fedeltà della ricostruzione:

$$\text{PSNR} = 10 \cdot \log_{10} \left(\frac{\text{MAX}^2}{\text{MSE}} \right) \quad (1.6.6)$$

dove MAX rappresenta il valore massimo possibile dei pixel (tipicamente 255 per immagini a 8 bit) e MSE è l'errore quadratico medio tra l'immagine di riferimento e quella ricostruita.

Nel contesto del Gaussian Splatting, il PSNR viene calcolato confrontando le immagini renderizzate dal modello 3D con le immagini ground truth¹¹ del dataset di test. Valori più elevati di PSNR indicano una maggiore fedeltà della ricostruzione, con soglie tipiche che considerano acceptable ricostruzioni con PSNR superiore a 20-25 dB per scene complesse.

Nonostante la sua diffusione, il PSNR presenta limitazioni significative nella valutazione della qualità percettiva. La metrica non tiene conto della struttura locale delle immagini né delle caratteristiche del sistema visivo umano, risultando spesso inadeguata per catturare aspetti qualitativi importanti come la nitidezza dei dettagli o la coerenza delle texture.

1.6.2 Structural Similarity Index Measure (SSIM)

Lo SSIM, introdotto da Wang et al., rappresenta un approccio più sofisticato alla valutazione della qualità delle immagini, progettato per correlare meglio con la percezione visiva umana. La metrica opera attraverso la valutazione separata di tre componenti fondamentali del sistema visivo umano, che vengono successivamente combinate in un singolo indice di qualità:

$$\text{SSIM}(x, y) = [l(x, y)]^\alpha \times [c(x, y)]^\beta \times [s(x, y)]^\gamma \quad (1.6.7)$$

¹¹Il termine *ground truth* indica i dati di riferimento considerati corretti e accurati, utilizzati come standard per valutare le prestazioni di un algoritmo o modello. Nel contesto del Gaussian Splatting, si riferisce alle immagini originali del dataset di acquisizione.

dove $l(x, y)$, $c(x, y)$ e $s(x, y)$ rappresentano rispettivamente la componente di luminanza, contrasto e struttura, con esponenti α , β , γ per controllare l'importanza relativa di ciascuna componente.

I tre componenti sono definiti come:

Luminanza:

$$l(x, y) = \frac{2\mu_x\mu_y + c_1}{\mu_x^2 + \mu_y^2 + c_1} \quad (1.6.8)$$

Contrasto:

$$c(x, y) = \frac{2\sigma_x\sigma_y + c_2}{\sigma_x^2 + \sigma_y^2 + c_2} \quad (1.6.9)$$

Struttura:

$$s(x, y) = \frac{\sigma_{xy} + c_3}{\sigma_x\sigma_y + c_3} \quad (1.6.10)$$

dove μ rappresenta la media locale, σ la deviazione standard locale, σ_{xy} la covarianza locale, e c_1 , c_2 , c_3 sono costanti di stabilizzazione.

Nella pratica, si utilizza comunemente la versione con $\alpha = \beta = \gamma = 1$ e $c_3 = c_2/2$, che produce la formula compatta:

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \quad (1.6.11)$$

I tre componenti valutano aspetti complementari:

- **Luminanza:** Confronto delle intensità medie locali - misura quanto le due immagini sono simili in termini di brillantezza complessiva
- **Contrasto:** Confronto della varianza delle intensità - valuta se le due immagini hanno simili livelli di dettaglio e texture
- **Struttura:** Confronto dei pattern di correlazione spaziale - cattura la similarità nei pattern geometrici e nelle relazioni spaziali tra pixel

Lo SSIM produce valori compresi tra -1 e 1, dove 1 indica identità perfetta. Nel rendering neurale, valori SSIM superiori a 0.8-0.9 sono generalmente considerati indicativi di alta qualità visiva.

La separazione in tre componenti consente di diagnosticare specifici aspetti della qualità dell'immagine. Una ricostruzione può avere buona luminanza e contrasto ma struttura degradata, indicando problemi nella preservazione dei dettagli geometrici tipici nelle scene renderizzate con Gaussian Splatting.

1.6.3 Learned Perceptual Image Patch Similarity (LPIPS)

La metrika LPIPS, sviluppata da Zhang et al., rappresenta l'evoluzione più recente nella valutazione percettiva delle immagini, utilizzando reti neurali pre-addestrate per catturare similarità percettive complesse. A differenza

delle metriche tradizionali basate su operazioni matematiche dirette sui pixel, LPIPS sfrutta le rappresentazioni apprese da deep network addestrate su task di computer vision.

La metrica opera estraendo feature da diversi layer di reti convoluzionali (tipicamente VGG, AlexNet, o SqueezeNet) e calcolando la distanza pesata tra le rappresentazioni delle immagini da confrontare:

$$\text{LPIPS} = \sum_l \frac{1}{H_l W_l} \sum_{h,w} \|\mathbf{w}_l \odot (\mathbf{y}_{hw}^l - \mathbf{y}_{hw}^{l*})\|_2^2 \quad (1.6.12)$$

dove l indica i layer della rete, \mathbf{w}_l sono pesi appresi, e \mathbf{y}^l rappresenta le feature normalizzate.

Caratteristiche distintive di LPIPS

- **Sensibilità percettiva:** Cattura distorsioni rilevanti per la percezione umana che sfuggono a metriche tradizionali
- **Robustezza alle trasformazioni:** Meno sensibile a spostamenti pixel-wise che non alterano la percezione
- **Correlazione con giudizi umani:** Elevata correlazione con valutazioni soggettive di qualità

Nel contesto del Gaussian Splatting, LPIPS si rivela particolarmente efficace nel distinguere tra ricostruzioni che, pur avendo PSNR/SSIM simili, presentano differenze qualitative significative in termini di realismo e fedeltà percettiva.

1.6.4 Applicazione nel Gaussian Splatting

1.6.4.1 Complementarità delle metriche

L'utilizzo combinato di PSNR, SSIM e LPIPS fornisce una valutazione multidimensionale della qualità dei modelli generati:

- **PSNR:** Valutazione della fedeltà pixel-wise e del rumore complessivo
- **SSIM:** Analisi della preservazione delle strutture locali e della coerenza spaziale
- **LPIPS:** Valutazione della qualità percettiva e del realismo visivo

1.6.4.2 Sfide specifiche

La valutazione dei modelli Gaussian Splatting presenta sfide peculiari, ovvero:

- **View-dependence:** A differenza delle immagini statiche, i modelli 3D devono essere valutati da multiple viewpoint, richiedendo l'aggregazione di metriche su diversi punti di vista.
- **Temporal consistency:** Per sequenze video, è necessario valutare anche la coerenza temporale tra frame consecutivi.
- **Scene complexity:** Scene con elementi trasparenti, riflessi o illuminazione complessa possono presentare trade-off diversi tra le metriche.

1.6.4.3 Benchmark e soglie di Riferimento

Nel contesto dei dataset standard per la valutazione del Gaussian Splatting (NeRF Synthetic, Mip-NeRF 360, Tanks and Temples), sono emersi range tipici di valori considerati rappresentativi di alta qualità:

- **PSNR:** 25-35 dB per scene indoor, 20-30 dB per scene outdoor
- **SSIM:** 0.85-0.95 per ricostruzioni di alta qualità
- **LPIPS:** 0.05-0.15 (valori più bassi indicano maggiore similarità percentiva)

Questi benchmark forniscono riferimenti fondamentali per valutare l'efficacia di nuove varianti algoritmiche e ottimizzazioni implementative nel campo del Gaussian Splatting.

1.7 Applicazioni e limitazioni attuali

1.7.1 Campi di applicazione nella computer vision

Il 3D Gaussian Splatting trova applicazione in diversi domini:

- **Novel View Synthesis:** è una tecnica che ha l'obiettivo di generare nuove immagini di una scena da punti di vista non osservati direttamente, a partire da un insieme limitato di viste iniziali, sfruttando informazioni spaziali e fotometriche per garantire realismo e coerenza.
- **Digital Content Creation:** Creazione di asset 3D per cinema, gaming e applicazioni VR/AR, sfruttando la velocità di rendering real-time.

- **Simulazione e Digital Twins:** Rappresentazione di ambienti reali per applicazioni industriali, training di sistemi autonomi e visualizzazione architettonica.
- **Medical Imaging:** Ricostruzione di strutture anatomiche da imaging medico, supportando visualizzazione interattiva per diagnosi e pianificazione chirurgica.

1.7.2 Limiti all'adozione diffusa

Nonostante i risultati promettenti, l'adozione su larga scala del 3D Gaussian Splatting è ostacolata da numerose **barriere tecniche e pratiche**.

Dal punto di vista *hardware e computazionale*, la tecnologia richiede GPU moderne per ottenere prestazioni accettabili sia in fase di training che di rendering. Inoltre, i modelli generati possono occupare centinaia di mega-byte, specialmente per scene complesse, comportando un *notevole consumo di memoria RAM*, in particolare durante il training di scene su larga scala. La fase di addestramento risulta inoltre sensibile alla qualità dell'input e alla scelta dei parametri di inizializzazione, con un impatto significativo sulla stabilità del processo.

Sul piano *dell'usabilità*, la tecnologia resta ancora confinata prevalentemente all'ambito accademico e specialistico. La documentazione è spesso pensata per un pubblico con background avanzato in *computer graphics* e *machine learning*, rendendo difficile l'approccio per utenti non esperti. L'utilizzo degli strumenti disponibili richiede una *configurazione complessa dell'ambiente* (CUDA, librerie specifiche), una buona comprensione degli *hyperparameter* da ottimizzare manualmente, nonché la gestione di un *workflow articolato* che include preprocessing, training e post-processing. Anche il debugging è tutt'altro che immediato, richiedendo competenze tecniche per interpretare errori e problemi di convergenza.

A queste difficoltà si aggiunge la *frammentazione degli strumenti attualmente disponibili*. Il panorama attuale prevede software e script separati per ogni fase del processo:

- **Framework di training:** spesso legati a implementazioni distinte per ciascuna variante algoritmica;
- **Strumenti di visualizzazione:** viewer non compatibili tra loro o limitati a specifici formati;
- **Utility di preprocessing:** script individuali e poco documentati per la preparazione dei dati;
- **Incoerenza nei formati:** assenza di standard per esportazione e importazione dei modelli.

Questa frammentazione ostacola la creazione di workflow fluidi, aumenta la curva di apprendimento per i nuovi utenti e rende più difficile l'adozione industriale della tecnologia. Tali limiti motivano la necessità di piattaforme integrate, come quella proposta in questo lavoro.

Capitolo 2

Architettura del Sistema Proposto

2.1 Obiettivi e funzionalità del sistema

Il sistema proposto è stato progettato per affrontare direttamente le limitazioni per l'adozione diffusa del 3D Gaussian Splatting identificate nel capitolo precedente, trasformando una tecnologia prevalentemente accademica in una soluzione accessibile e user-friendly.

2.1.1 Obiettivi di progetto

Migliore accesso alla tecnologia

L'obiettivo primario consiste nel rendere il 3D Gaussian Splatting accessibile a utenti senza competenze tecniche specialistiche in computer graphics o machine learning. Il sistema deve eliminare le barriere di ingresso tipiche degli strumenti di ricerca, permettendo a professionisti di settori diversi - dal design alla produzione multimediale - di sfruttare le potenzialità della tecnologia attraverso un'interfaccia intuitiva.

Unificazione di strumenti frammentati

Il secondo obiettivo riguarda l'integrazione di funzionalità attualmente disperse in implementazioni separate. Il sistema deve fornire una piattaforma unificata che comprenda training, visualizzazione e gestione modelli, eliminando la necessità di utilizzare tool specifici incompatibili tra loro e riducendo significativamente la curva di apprendimento.

Automazione del workflow tecnico

Il terzo obiettivo è l'automazione dell'intero processo, dalla preparazione dei dati alla visualizzazione finale, riducendo al minimo l'intervento manuale e la possibilità di errori. Questo include la gestione automatica di preprocessing, configurazione parametri e troubleshooting di problemi comuni.

2.1.2 Funzionalità del sistema

Per raggiungere questi obiettivi, il sistema implementa le seguenti funzionalità core:

Interfaccia web guidata

Il sistema fornisce un'interfaccia web responsive che guida l'utente attraverso un processo step-by-step, dalla selezione del video di input alla visualizzazione del modello 3D finale. L'interfaccia integra:

- **Processo guidato:** Workflow sequenziale con istruzioni chiare per ogni fase
- **Validazione intelligente:** Controllo automatico della qualità e formato dei file di input
- **Feedback:** Indicatore della fase in cui si trovano le elaborazioni
- **Documentazione contestuale:** Spiegazioni integrate e suggerimenti per ottimizzare i risultati

Training multi-algoritmo integrato

La piattaforma integra tre diversi approcci di training del Gaussian Splatting in un'unica interfaccia:

- **Standard 3DGS:** Implementazione baseline per risultati di alta qualità
- **MCMC Gaussian Splatting:** Approccio probabilistico per miglioramento della qualità e riduzione artefatti
- **Taming 3DGS:** Ottimizzazione per risorse computazionali limitate

L'utente può selezionare l'algoritmo desiderato attraverso un'interfaccia semplificata con parametri preconfigurati per casi d'uso comuni.

Pipeline di processing automatizzata

Il sistema implementa una pipeline completamente automatizzata che gestisce:

1. **Preprocessing video:** Estrazione automatica di frame ottimali dal video di input
2. **Structure from Motion:** Generazione automatica della point cloud iniziale
3. **Training asincrono:** Elaborazione in background con notifiche di completamento
4. **Upload modello:** Upload su repository del modello 3D generato dal training
5. **Post-processing:** Generazione e salvataggio delle metriche di elaborazione

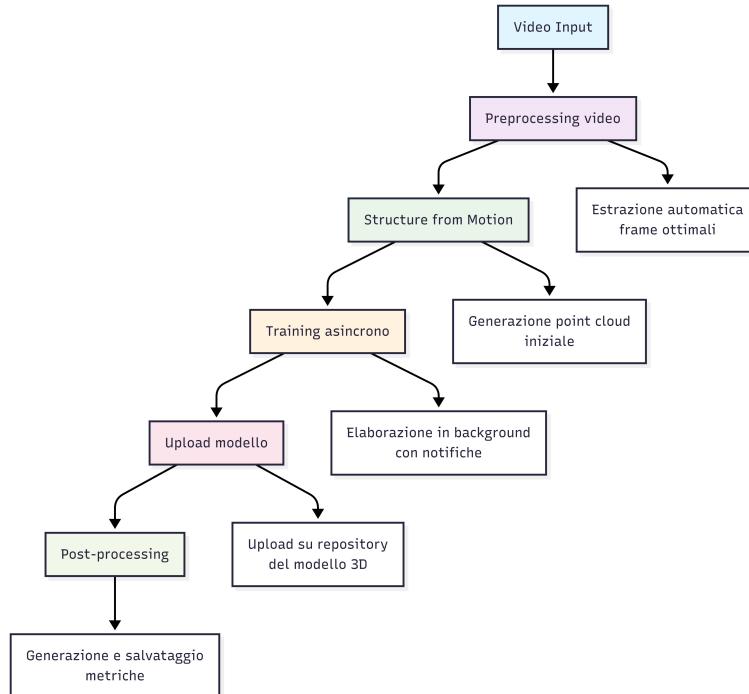


Figura 2.1: Pipeline di processing automatizzata

Visualizzazione 3D integrata

La piattaforma include un viewer 3D web-based che permette:

- **Rendering real-time:** Visualizzazione immediata dei modelli generati
- **Controlli interattivi:** Navigazione 3D intuitiva con mouse/touch
- **Valutazione:** Visualizzazione delle statistiche come FPS e numero di gaussiane generate

Gestione trasparente delle risorse

Il sistema nasconde la complessità tecnica attraverso:

- **Containerizzazione:** Ambiente di esecuzione isolato e riproducibile
- **Gestione automatica GPU:** Allocazione ottimale delle risorse computazionali
- **Scaling adattivo:** Gestione automatica del carico di lavoro
- **Monitoring integrato:** Raccolta automatica di metriche di performance e qualità

2.1.3 Benefici attesi

L'implementazione di queste funzionalità mira a produrre i seguenti benefici:

- **Riduzione time-to-market:** Dalla creazione del contenuto alla visualizzazione 3D in minuti invece che ore o giorni richiesti da workflow tradizionali.
- **Accessibilità ampliata:** Estensione dell'utilizzo del 3D Gaussian Splatting a settori non tecnici come marketing, e-commerce, architettura e produzione di contenuti.
- **Standardizzazione del processo:** Etabilimento di best practices e workflow standardizzati per la creazione di contenuti 3D da video.
- **Riduzione costi:** Eliminazione della necessità di competenze specificistiche dedicate e riduzione dei tempi di training del personale.

2.2 Organizzazione a Layer

Il sistema proposto implementa un'architettura distribuita a microservizi progettata per gestire l'intero workflow del 3D Gaussian Splatting, dall'upload del contenuto video alla visualizzazione dei modelli 3D generati. La Figura 2.2 presenta una vista complessiva dell'architettura, evidenziando i componenti principali e i loro pattern di interazione. L'architettura è strutturata in layer distinti, ciascuno con responsabilità specifiche e interfacce ben definite, seguendo i principi di separazione delle responsabilità e scalabilità indipendente. Questa organizzazione logica non implica una separazione fisica rigida, ma piuttosto una strutturazione delle responsabilità che può essere implementata all'interno di uno o più servizi containerizzati.

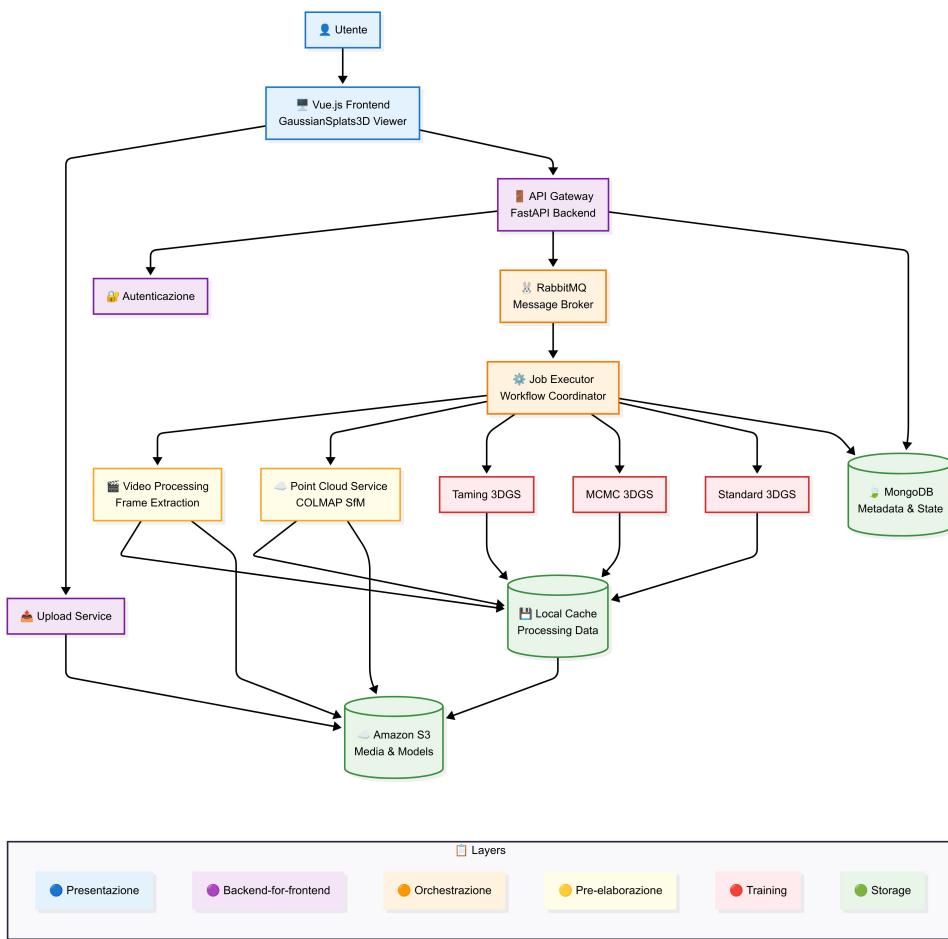


Figura 2.2: Architettura complessiva del sistema distribuito

2.2.1 Layer di Storage

Il **layer di storage** implementa una strategia ibrida che combina MongoDB per metadati e stato delle elaborazioni, Amazon S3 per contenuti multimediali e modelli 3D che si appoggia a sua volta ad una cache locale per dati di processing temporanei.

2.2.2 Layer di Orchestrazione

Il **layer di orchestrazione** rappresenta il cuore coordinativo del sistema, implementato attraverso RabbitMQ come message broker e il Job Executor come coordinatore del workflow. Questo layer gestisce la comunicazione asincrona tra i componenti e implementa un pattern Producer-Consumer ibrido che garantisce l'esecuzione sequenziale delle fasi di elaborazione. La scelta di RabbitMQ assicura persistenza dei messaggi, gestione delle code specializzate e resilienza in caso di fallimenti temporanei.

2.2.3 Layer di Pre-elaborazione

Il **layer di preprocessing** comprende i servizi specializzati responsabili dell'elaborazione preliminare dei dati: Video Processing per l'estrazione dei frame dal video di input e Point Cloud Service per la ricostruzione geometrica tramite COLMAP SfM.

2.2.4 Layer di Training

Il **layer di training** include i Training Services che eseguono gli algoritmi di Gaussian Splatting nelle diverse implementazioni (Taming 3DGS, MCMC 3DGS e Standard 3DGS), utilizzando le risorse computazionali GPU per l'addestramento dei modelli.

2.2.5 Layer Backend-for-Frontend

Il **layer Backend-for-Frontend** funge da punto di ingresso unificato per tutte le richieste del sistema, implementando pattern di routing, autenticazione e gestione delle richieste. Questo layer è univocamente rappresentato dall'applicazione (e container) API Gateway, è sviluppata in FastAPI ed espone endpoint REST standardizzati permettendo l'accesso ai servizi sottostanti.

2.2.6 Layer di Presentazione

Il **layer di presentazione** costituisce l'interfaccia utente del sistema e comprende l'applicazione web sviluppata in Vue.js integrata con il viewer GaussianSplats3D. Questo layer gestisce l'interazione utente, la visualizzazione dei modelli generati e la gestione degli stati dell'interfaccia. La scelta

di Vue.js consente un'architettura component-based leggera che facilita la manutenibilità e l'estensibilità dell'interfaccia.

2.3 Principi e pattern architetturali

2.3.1 Principi di design

L'architettura del sistema è stata progettata rispondendo alle sfide specifiche del 3D Gaussian Splatting come tecnologia emergente. A differenza delle applicazioni web tradizionali, il Gaussian Splatting presenta caratteristiche uniche che richiedono decisioni architetturali mirate: elaborazioni GPU-intensive con tempi variabili da minuti a ore, algoritmi in rapida evoluzione, gestione di contenuti multimediali di grandi dimensioni e necessità di isolamento per operazioni hardware-critical.

I principi di design adottati mirano a trasformare una tecnologia di ricerca complessa in una piattaforma accessibile e robusta. L'approccio privilegia la separazione delle preoccupazioni per isolare la complessità computazionale dalle interfacce utente, la scalabilità selettiva per ottimizzare l'uso di risorse costose, e la modularità per facilitare l'integrazione di nuovi algoritmi e l'evoluzione del sistema nel tempo.

Questi principi si concretizzano in scelte implementative specifiche che bilancino semplicità d'uso, efficienza operativa e flessibilità tecnica.

2.3.1.1 Separazione delle responsabilità per processing intensivo

Sebbene un'architettura monolitica possa gestire processing asincrono attraverso threading, l'approccio a microservizi offre vantaggi specifici per il Gaussian Splatting: isolamento degli errori hardware-intensive, scalabilità selettiva delle risorse GPU, e deployment indipendente di algoritmi in evoluzione. L'approccio a microservizi permette di separare logicamente:

- **Servizi di interfaccia:** Frontend e API che rimangono sempre responsivi
- **Servizi di processing:** Dedicati esclusivamente al training intensivo
- **Servizi di supporto:** Gestione dati, code messaggi e storage

Queste funzionalità si traducono in un'architettura tecnica specifica che verrà dettagliata nelle sezioni successive, dimostrando come gli obiettivi di alto livello si concretizzino in scelte implementative precise.

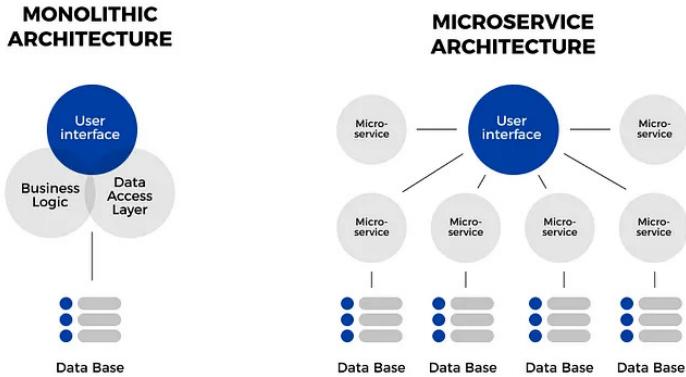


Figura 2.3: Architettura monolita vs architettura a microservizi

2.3.1.2 Scalabilità orizzontale specializzata

La natura del training di Gaussian Splatting, che richiede hardware dedicato (GPU), rende particolarmente vantaggiosa la possibilità di scalare orizzontalmente solo i componenti computazionalmente intensivi. I servizi di training, essendo containerizzati e stateless, possono essere deployati su multiple macchine dotate di GPU specializzate, mentre frontend e API rimangono centralizzati su hardware meno specializzato. Questa separazione consente di:

- **Ottimizzare i costi:** hardware GPU costoso utilizzato solo per training e il preprocessing
- **Gestire il carico:** aggiungere capacità computazionale senza modificare altri servizi
- **Isolamento degli errori:** crash o indisponibilità del training non compromette l'interfaccia utente

2.3.1.3 Deployment e manutenzione indipendente

L'architettura modulare facilita lo sviluppo e la manutenzione del sistema, permettendo di aggiornare, testare e deployare ogni componente indipendentemente. Questo è particolarmente vantaggioso per un progetto che integra algoritmi di training in evoluzione (Standard, MCMC, Taming), dove ogni implementazione può essere aggiornata senza impattare gli altri moduli.

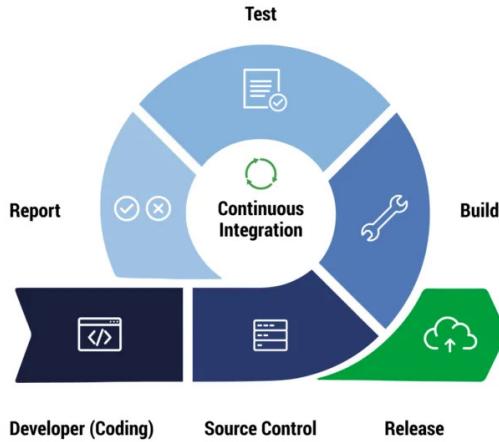


Figura 2.4: Schema della continuous integration

2.3.2 Pattern architetturali implementati

Per realizzare i principi di design descritti, il sistema adotta pattern architettonici consolidati, adattati alle specifiche esigenze del 3D Gaussian Splatting. Questi pattern forniscono soluzioni concrete per gestire la complessità del workflow, garantire la resilienza del sistema e ottimizzare l'uso delle risorse computazionali.

I pattern selezionati operano sinergicamente per creare un'architettura che supporta elaborazioni asincrone lunghe, gestione robusta degli errori e scalabilità delle operazioni intensive.

2.3.2.1 Architettura Event-driven

Il sistema implementa un'architettura guidata dagli eventi attraverso una coda di messaggi che coordina l'esecuzione delle diverse fasi di processing. Ogni completamento di fase genera un evento che triggerà la fase successiva, creando un workflow asincrono e resiliente.

Vantaggi per il Gaussian Splatting:

- **Gestione della coda:** Serializzazione automatica dei job per gestire la limitazione di GPU singola
- **Retry automatico:** Eventi possono essere riaccodati in caso di fallimento
- **Monitoring:** Stato del workflow tracciabile attraverso gli eventi

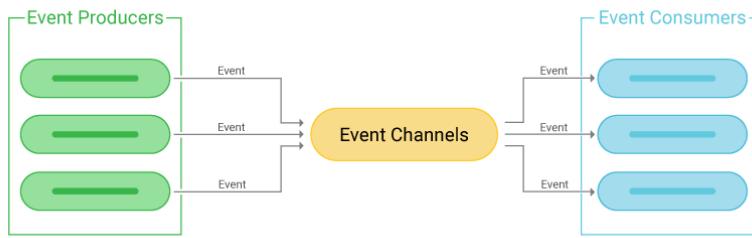


Figura 2.5: Architettura Event-driven

2.3.2.2 Pipeline Pattern

Il processing del Gaussian Splatting segue naturalmente un pattern pipeline con handoff esplicativi tra fasi:

- **Estrazione frame** → Output: frame video + metadati
- **Structure from Motion** → Output: point cloud + pose camere
- **Training** → Output: modello 3D ottimizzato

Ogni fase produce output standardizzati che vengono consumati dalla fase successiva, permettendo parallelizzazione e reprocessing selettivo.

2.3.2.3 Caching e Storage Pattern

Il sistema adotta una strategia di gestione dati multi-livello che combina diversi pattern architettonici per ottimizzare prestazioni, resilienza e ciclo di vita dei contenuti multimediali. L'approccio integra tre pattern complementari specificamente adattati alle caratteristiche del 3D Gaussian Splatting.

Cache-Aside Pattern per performance

Il sistema implementa una strategia di caching a due livelli per ridurre latenza e costi di accesso ai dati:

Questo approccio è particolarmente efficace per file di grandi dimensioni tipici del Gaussian Splatting, riducendo significativamente i costi e i tempi di accesso a S3 durante le operazioni intensive.

Staging Pattern per persistenza intermedia

Amazon S3 funge da storage intermedio tra le fasi del workflow, implementando un pattern staging che garantisce:

Algorithm 1: Cache-Aside Pattern per gestione risorse

```
Input: resource_identifier  
Output: resource_data  
if resource exists in local_cache then  
    | return load_from_local_cache(resource_identifier);  
else  
    | resource_data ←  
    | fetch_from_remote_storage(resource_identifier);  
    | save_to_local_cache(resource_identifier, resource_data);  
    | return resource_data;  
end
```

- **Persistenza:** Risultati intermedi sopravvivono a restart dei servizi
- **Reprocessing selettivo:** Possibilità di ripartire da fasi specifiche già completate
- **Auditability:** Tracciabilità completa dei dati prodotti durante l'elaborazione

Hybrid Storage per ciclo di vita dati

La strategia combina diversi livelli di storage ottimizzati per i pattern di accesso specifici del Gaussian Splatting:

- **Storage locale:** Cache temporanea per elaborazioni attive con accesso veloce, riduzione latenza I/O e cleanup automatico
- **Amazon S3 Standard:** Repository persistente per video originali e modelli finali con durabilità, scalabilità e versioning
- **Processing cache:** Dati intermedi (frame, point clouds) mantenuti localmente per supportare reprocessing parziale

L'architettura mantiene tutti i dati persistenti per massimizzare la flessibilità operativa, privilegiando la funzionalità di reprocessing e la semplicità gestionale. Questo approccio conservativo garantisce la stabilità del workflow, elemento centrale per l'esperienza utente del sistema.

2.4 Containerizzazione con Docker e Docker Compose

2.4.1 Isolamento delle dipendenze specializzate

Ogni algoritmo di training del Gaussian Splatting presenta dipendenze specifiche e spesso conflittuali:

- **Versioni CUDA:** Diverse implementazioni richiedono versioni specifiche di CUDA (driver NVIDIA)
- **Librerie Python:** Conflitti tra versioni di PyTorch, NumPy e librerie di computer vision
- **Configurazioni hardware:** Ottimizzazioni specifiche per GPU diverse

2.4.2 Orchestrazione con Docker Compose

Docker Compose facilita la gestione dell'intero stack applicativo, definendo in modo dichiarativo:

- **Networking:** Comunicazione sicura tra container
- **Volume mounting:** Condivisione dati tra servizi
- **Environment variables:** Configurazione centralizzata
- **Service dependencies:** Ordine di avvio dei servizi

Sebbene il deployment attuale avvenga su singola macchina per ragioni di sviluppo, l'architettura containerizzata prepara il sistema per un futuro deployment distribuito su cluster.

2.4.3 Gestione risorse GPU

La containerizzazione permette di gestire in modo granulare l'accesso alle risorse GPU, garantendo che il container di training abbia accesso esclusivo alle GPU disponibili, evitando conflitti di risorse e ottimizzando le performance.

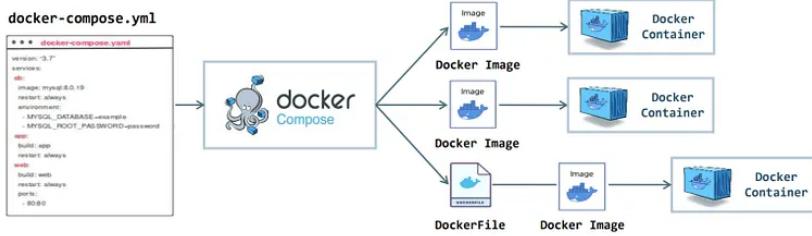


Figura 2.6: Architettura con Docker Compose

2.5 Gestione asincrona e code messaggi

Il sistema adotta un’architettura di messaggistica progettata per l’efficienza e la robustezza. La strategia centrale consiste nel separare completamente i dati dai messaggi di controllo: le code contengono esclusivamente identificativi dei modelli, mentre MongoDB centralizza tutti i parametri e lo stato di esecuzione.

2.5.1 Code leggere per workflow complessi

Ogni fase del pipeline (estrazione, SfM, training) utilizza una coda dedicata che trasporta solo gli ID dei modelli da processare. Questa scelta elimina la duplicazione di dati nelle code e mantiene i messaggi estremamente compatibili, migliorando le prestazioni del sistema di messaggistica anche con carichi intensivi.

2.5.2 Stato sempre sincronizzato

All’avvio di ogni fase, il servizio responsabile interroga MongoDB per recuperare configurazione, stato precedente e metadati aggiornati. Questo pattern garantisce che ogni elaborazione operi sempre sui dati più recenti, eliminando inconsistenze dovute a messaggi obsoleti o duplicati.

2.5.3 Resilienza e reprocessing selettivo

I fallimenti vengono gestiti con arresti controllati che preservano lo stato raggiunto, consentendo analisi dettagliate e retry mirati. Il sistema supporta riprocessing selettivo: un modello può essere rigenerato partendo da qualsiasi fase già completata, evitando elaborazioni ridondanti e ottimizzando l’utilizzo delle risorse GPU.

Questo design bilancia efficienza computazionale e flessibilità operativa, requisiti essenziali per un sistema di produzione dedicato al 3D Gaussian Splatting.

2.6 Flusso di elaborazione end-to-end

Il flusso di elaborazione del sistema segue un approccio sequenziale orchestrato attraverso code di messaggi specializzate. Ogni fase del workflow è gestita da job specifici che, una volta completati, innescano automaticamente la fase successiva della pipeline.

2.6.1 Fase di Upload e Inizializzazione

Il processo inizia con l'upload del video da parte dell'utente attraverso il frontend. Il sistema implementa un meccanismo di upload sicuro basato su presigned URL: il frontend richiede al backend un URL temporaneo per il caricamento diretto sull'object storage, evitando il transito del contenuto multimediale attraverso il backend API. Completato l'upload, il frontend effettua una richiesta POST di creazione del modello, specificando i parametri di training (algoritmo, livello di qualità, nome progetto).

2.6.2 Pipeline di Elaborazione

Il backend persiste i metadati del nuovo modello nel database e inserisce il primo job della catena nella coda di messaggi. Il flusso di elaborazione è strutturato in cinque fasi sequenziali:

1. **Frame Extraction:** estrazione dei frame dal video caricato
2. **Cloud Point Reconstruction:** generazione della nuvola di punti iniziale
3. **Training:** esecuzione dell'algoritmo di Gaussian Splatting selezionato
4. **Upload:** caricamento del modello 3D generato sull'object storage
5. **Metrics Generation:** calcolo delle metriche di qualità del modello

Coordinamento tra Job

Il sistema adotta un'architettura a code specializzate, dove ogni tipologia di job dispone di una coda dedicata. Al completamento di ciascun job, il job executor inserisce automaticamente il task successivo nella coda appropriata, garantendo la continuità del flusso di elaborazione. Questa architettura permette una gestione granulare delle risorse e una scalabilità indipendente per ogni fase del workflow.

Gestione dei Fallimenti

In caso di fallimento di un job, il flusso di elaborazione si interrompe e il modello viene marcato con stato di errore. Il sistema non implementa meccanismi automatici di retry, ma permette all'utente di riavviare manualmente l'elaborazione dal frontend, sfruttando la funzionalità di clonazione del modello per ripartire da eventuali step già completati con successo.

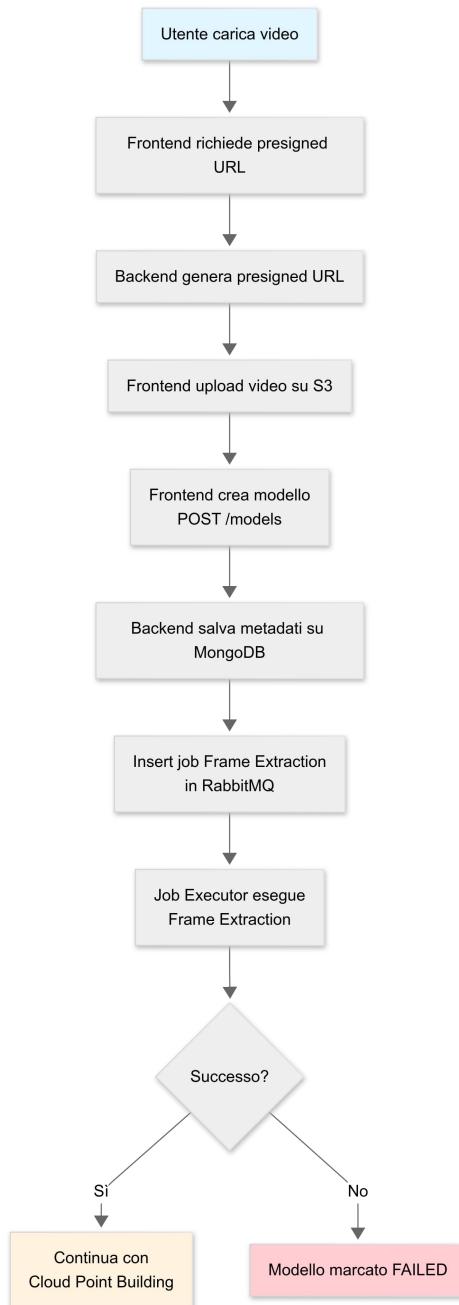


Figura 2.7: Architettura dei componenti del sistema (upload e preprocessing video)

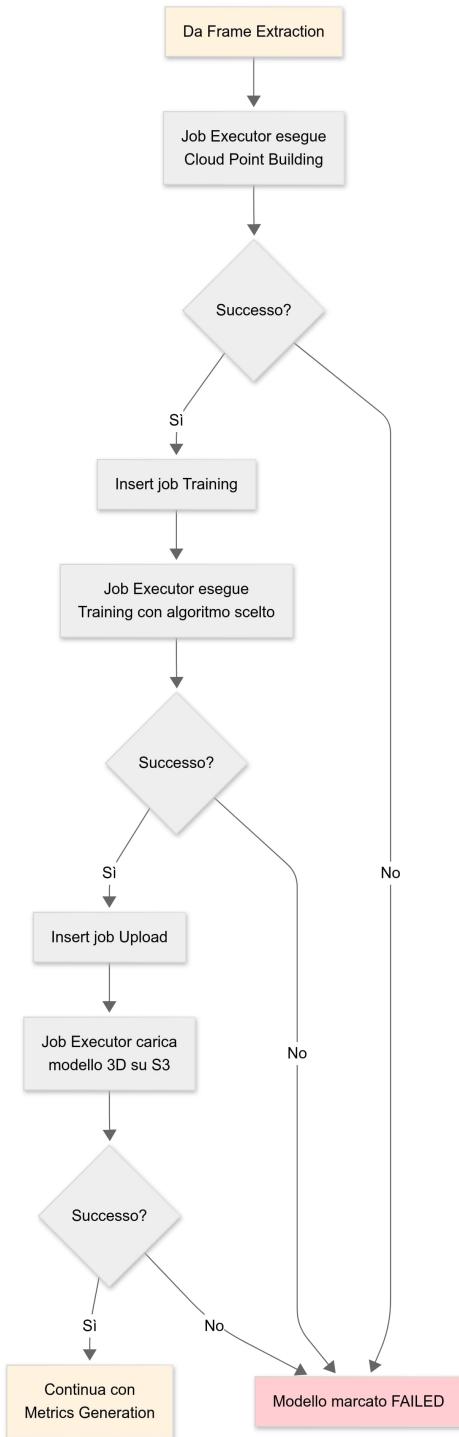


Figura 2.8: Architettura dei componenti del sistema (elaborazione e training)

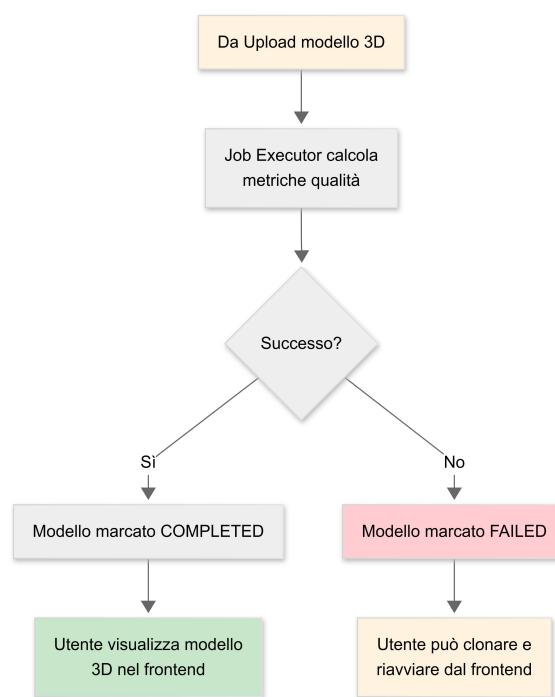


Figura 2.9: Architettura dei componenti del sistema (finalizzazione)

Capitolo 3

Implementazioni e tecnologie

In questo capitolo vengono descritte in dettaglio le scelte tecnologiche e le strategie implementative adottate per la realizzazione del sistema presentato nel Capitolo 2. L’obiettivo principale è offrire una panoramica completa sull’integrazione tra le diverse componenti software e hardware, evidenziando come la combinazione di tecnologie moderne e soluzioni open-source abbia permesso di ottenere un’infrastruttura scalabile, modulare e facilmente manutenibile.

A partire dalla selezione dello stack tecnologico — sia frontend che backend — vengono illustrate le motivazioni alla base delle scelte compiute, le metodologie di containerizzazione e orchestrazione dei servizi, nonché i pattern di persistenza dei dati strutturati e multimediali. Il capitolo approfondisce inoltre i sistemi di parametrizzazione adattiva per l’ottimizzazione delle performance in relazione all’hardware disponibile, le strategie di resilienza, fault tolerance e gestione degli errori implementate per garantire affidabilità operativa anche in contesti distribuiti.

Attraverso una descrizione puntuale dell’implementazione dei singoli layer funzionali — dalla pre-elaborazione del dato grezzo al training degli algoritmi, fino alla visualizzazione interattiva in ambiente web — il lettore potrà apprezzare l’articolazione della soluzione e comprenderne sia gli aspetti di innovazione tecnica, sia i criteri di progettazione orientati alla flessibilità e alla sperimentazione.

3.0.1 Stack Tecnologico Implementato

3.0.1.1 Tecnologie Frontend

Il frontend utilizza **Vue.js 3** come framework principale per lo sviluppo dell’interfaccia utente, scelto per il bilanciamento tra semplicità e funzionalità. La visualizzazione 3D è implementata attraverso **GaussianSplats3D**¹, una

¹<https://github.com/mkellogg/GaussianSplats3D>

libreria specializzata basata su **Three.js** che non supporta, nativamente, il rendering di primitive gaussiane.

3.0.1.2 Tecnologie di backend

Il backend è implementato in **Python** con **FastAPI** come framework per le API REST, scelto per le performance superiori e la generazione automatica di documentazione OpenAPI. La gestione asincrona utilizza le capacità native di Python con **asyncio**.

3.0.1.3 Infrastruttura e Persistenza

L'infrastruttura utilizza **Docker** e **Docker Compose** per la containerizzazione e l'orchestrazione locale, **RabbitMQ** come message broker per la comunicazione asincrona, **MongoDB** come database NoSQL per metadati, e **Amazon S3** per lo storage oggetti scalabile.

3.0.1.4 Pre-elaborazione specializzata

I servizi di processing integrano **COLMAP**² per Structure from Motion e per i training services, e **Sharp Frames**³ per l'estrazione video. Ogni servizio è ottimizzato per l'utilizzo specifico di risorse GPU attraverso configurazioni CUDA dedicate.

L'architettura complessiva garantisce scalabilità, manutenibilità e performance ottimali per l'elaborazione di contenuti 3D, fornendo una base solida per l'implementazione dei componenti specializzati descritti nei capitoli successivi.

3.0.1.5 Algoritmi di Training integrati

Il sistema integra progetti open-source specializzati per il training di rappresentazioni 3D:

- **3D Gaussian Splatting for Real-Time Radiance Field Rendering**⁴ per la generazione di primitive gaussiane da dataset fotografici
- **3D Gaussian Splatting as Markov Chain Monte Carlo** ⁵ con l'introduzione di rumore stocastico nel calcolo dei gradienti
- **Taming 3DGS: High-Quality Radiance Fields with Limited Resources** ⁶ per la generazione efficiente di scene 3D con vincoli computazionali ridotti

²<https://colmap.github.io>

³<https://github.com/Reflct/sharp-frames-python>

⁴<https://github.com/graphdeco-inria/gaussian-splatting>

⁵<https://github.com/ubc-vision/3dgs-mcmc>

⁶<https://github.com/ubc-vision/3dgs-mcmc>

Questi progetti sono stati integrati come servizi containerizzati nel workflow di elaborazione, mantenendo la compatibilità con le implementazioni originali e permettendo aggiornamenti indipendenti.

3.0.2 Servizi containerizzati

Per garantire modularità, scalabilità e portabilità del sistema, ciascun componente è stato containerizzato tramite **Docker**. L'intera infrastruttura si basa su microservizi che comunicano tra loro attraverso una rete virtuale comune, orchestrati da **Docker Compose**.

Ogni container svolge un ruolo specifico e indipendente nel flusso di elaborazione, riducendo le dipendenze tra i moduli. Di seguito si elencano i principali container e le loro funzioni:

- **Accesso e orchestrazione**
 - **api-gateway**: gestisce il routing delle richieste HTTP verso i microservizi interni.
 - **job-executor**: coordina i task tra i moduli e comprende alcune parti di pre e post processing.
- **Motori di training e rendering 3D**
 - **gaussian-splatting-api**: esegue il training del Gaussian Splatting 3D e si basa sul paper originale.
 - **3dgs-mcmc-api**: esegue il training del Gaussian Splatting 3D con la variante Monte Carlo as Markov Chain.
 - **taming-3dgs-api**: motore alternativo di training per scene 3D controllate.
- **Pre-elaborazione**
 - **colmap-converter**: wrapper per Colmap, utilizzato per la ricostruzione di point cloud a partire da immagini RGB.
- **Persistenza e messaggistica**
 - **mongo**: database NoSQL per la persistenza dei dati.
 - **rabbitmq**: message broker per la comunicazione asincrona tra i servizi.
- **Visualizzazione**
 - **web-viewer**: interfaccia frontend per l'interazione con il workflow di training e la visualizzazione dei risultati 3D generati.

3.0.2.1 Esempio di Dockerfile: Gaussian Splatting

Di seguito è riportato un esempio di Dockerfile utilizzato per il container `gaussian-splatting`, responsabile del rendering 3D basato su rappresentazioni gaussiane. L'immagine si basa su un container NVIDIA compatibile con CUDA e contiene tutte le dipendenze necessarie per eseguire il motore grafico:

Listing 3.1: Dockerfile per gaussian-splatting

```
FROM nvidia/cuda:12.2.2-cudnn8-devel-ubuntu22.04
ENV DEBIAN_FRONTEND=noninteractive

# Installazione dipendenze di sistema
RUN apt update && apt upgrade -y && apt install -y \
git cmake libxml2-dev libxi-dev libgl-dev libomp-dev \
python3-dev python3-venv python3-pip build-essential ninja \
-build wget \
libboost-program-options-dev libboost-filesystem-dev \
libboost-graph-dev \
libboost-system-dev libboost-test-dev libeigen3-dev \
libflann-dev \
libfreeimage-dev libmetis-dev libgoogle-glog-dev libgflags \
-dev \
libsdl2-dev libglew-dev qtbase5-dev libqt5opengl5-dev \
libcgal-dev \
gcc-10 g++-10 libatlas-base-dev libsuitesparse-dev

# Clonazione del repository
WORKDIR /workspace
RUN git clone https://github.com/graphdeco-inria/gaussian-splatting --recursive

# Installazione dipendenze Python
COPY requirements.txt /workspace/gaussian-splatting/
WORKDIR /workspace/gaussian-splatting
RUN pip3 install --upgrade pip && \
pip3 install torch torchvision --extra-index-url https://download.pytorch.org/whl/cu121 && \
pip3 install -r requirements.txt

# Compilazione moduli C++
RUN cd submodules/diff-gaussian-rasterization && pip3 \
install .
RUN cd submodules/simple-knn && pip3 install .
RUN cd submodules/fused-ssim && pip3 install .

# Supporto Depth Anything V2
WORKDIR /workspace
RUN git clone https://github.com/DepthAnything/Depth-Anything-V2.git
WORKDIR /workspace/Depth-Anything-V2
RUN pip3 install opencv-python transformers pillow numpy \
matplotlib
```

```

RUN mkdir -p checkpoints && \
wget -O checkpoints/depth_anything_v2_vitl.pth \
https://huggingface.co/depth-anything/Depth-Anything-V2-
Large/resolve/main/depth_anything_v2_vitl.pth

# Ritorno alla directory principale
WORKDIR /workspace/gaussian-splatting
CMD ["bash"]

```

In seguito, viene definita un'estensione del container per esporre un'interfaccia FastAPI, utile per l'interazione RESTful con il modulo computazionale:

Listing 3.2: Estensione API FastAPI per Gaussian Splatting

```

FROM gaussian-splatting

ARG API_PORT=8100
ENV API_PORT=$API_PORT
EXPOSE $API_PORT

WORKDIR /workspace/gaussian-splatting
COPY api.py /workspace/gaussian-splatting/api.py

RUN mkdir -p /tmp/runtime-root && chmod 0700 /tmp/runtime-
root
ENV QT_QPA_PLATFORM=offscreen
ENV PYTHONUNBUFFERED=1

RUN pip3 install fastapi uvicorn

CMD ["/bin/bash", "-c", "uvicorn api:app --host 0.0.0.0 --
port $API_PORT"]

```

Questa struttura modulare consente di disaccoppiare il motore grafico dall'interfaccia espositiva, migliorando la manutenibilità e favorendo l'integrazione in pipeline distribuite.

3.1 Sistema di parametrizzazione adattiva per livelli di qualità e auto-scaling hardware

Il workflow implementa un sistema di parametrizzazione multi-livello progettato per bilanciare qualità dell'output e vincoli hardware, provando a garantire sempre la generazione di un modello 3D indipendentemente dalle limitazioni del sistema. Tale parametrizzazione cerca di far coesistere i seguenti aspetti:

- **Livelli di qualità utente:** l'utente può scegliere tra tre preset (Fast, Balanced, Quality) che influenzano parametri critici come risoluzione target, numero di frame estratti, iterazioni di training e soglie di densificazione. Questa scelta rappresenta l'**intento** di qualità desiderato, ma non è vincolante per l'esecuzione.
- **Hardware adaptation e graceful degradation:** il sistema, tramite il TrainingParamsService, rileva la VRAM disponibile e applica un downscaling conservativo dei parametri critici (risoluzione, batch size, frame count, ecc.). Questo approccio di graceful degradation è implementato solo in fase di setup: il sistema preferisce ridurre preventivamente la qualità dell'output per prevenire errori di esaurimento risorse (OOM) o crash del container. Una volta avviato il workflow, non sono previsti adattamenti dinamici.

3.1.1 Struttura dei parametri

La configurazione di ogni algoritmo è organizzata in categorie distinte, ciascuna con uno scopo specifico nel workflow:

- **base_params:** definiscono i valori di partenza dell'algoritmo, tipicamente corrispondenti ai parametri di default della implementazione originale. Rappresentano la base dalla quale vengono costruiti tutti e tre i livelli di qualità attraverso l'applicazione dei moltiplicatori. Questi parametri influenzano direttamente il comportamento degli algoritmi di training (numero di iterazioni, soglie di densificazione, intervalli di ottimizzazione).
- **quality_multipliers:** moltiplicatori che vengono applicati esclusivamente ai **base_params** per generare i parametri finali di training per ciascun livello di qualità. Il livello "fast" riduce tipicamente le iterazioni e accelera il processo, mentre "quality" incrementa la precisione computazionale a scapito dei tempi di elaborazione.
- **preprocessing_params:** parametri che influenzano esclusivamente la fase di preprocessing (estrazione frame, ridimensionamento video)

che non vengono passati agli algoritmi di training. Include risoluzione target e numero di frame da estrarre, parametri che impattano sulla qualità del dataset di input ma rimangono esterni alla logica di training.

- **manual_overrides**: parametri opzionali che possono essere specificati manualmente dall’utente (o dal chiamante API) per forzare valori di training specifici. Gli override hanno priorità su qualsiasi altro parametro calcolato o derivato dai preset, adattamenti hardware e moltiplicatori di qualità.
- **hardware_config**: definisce le soglie VRAM e le formule di scaling automatico. I **resolution_thresholds** stabiliscono le risoluzioni massime supportate per ogni configurazione hardware, le **scaling_formulas** permettono un adattamento dinamico dei parametri di training in base alla memoria disponibile.

3.1.2 Processo di Calcolo Parametri

Il `TrainingParamsService` implementa una pipeline di calcolo che trasforma le configurazioni di base in parametri finali pronti per l’esecuzione:

La separazione netta tra parametri di training e preprocessing consente al sistema di adattare indipendentemente la qualità del dataset di input (risoluzione, numero di frame) e la precisione computazionale del training, ottimizzando l’utilizzo delle risorse hardware disponibili.

3.1.3 Configurabilità e Sperimentazione

Un aspetto fondamentale dell’architettura è la **persistenza delle configurazioni in MongoDB**, che offre all’owner dell’applicazione la possibilità di modificare dinamicamente i parametri senza dover ricompilare o ridistribuire il software. Questa caratteristica è particolarmente vantaggiosa per:

- **Ottimizzazione empirica**: L’owner può testare diverse combinazioni di parametri, moltiplicatori di qualità e soglie hardware per trovare il bilanciamento ottimale tra qualità dell’output e performance del sistema. Le modifiche alle configurazioni diventano immediatamente operative per tutti i nuovi job.
- **Adattamento hardware-specifico**: Ogni deployment può essere fine-tuned per l’hardware specifico disponibile, le **resolution_thresholds** e le **scaling_formulas** vengono modificate in base alle caratteristiche delle GPU installate e ai risultati dei test empirici.
- **Evoluzione degli algoritmi**: L’aggiunta di nuovi algoritmi o l’aggiornamento di quelli esistenti richiede solamente l’inserimento di un

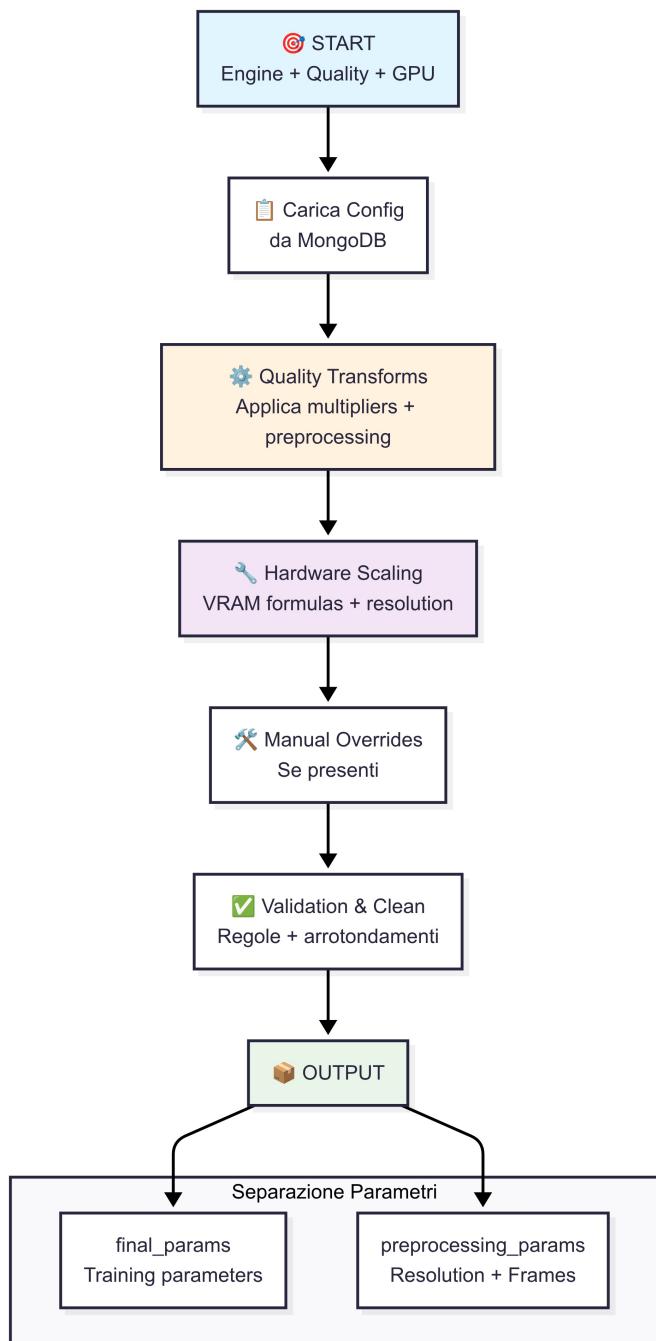


Figura 3.1: Workflow della generazione dei parametri adattivi

nuovo documento nella collection `training_params`, mantenendo la retrocompatibilità con il codice esistente.

Questa flessibilità consente un approccio iterativo al miglioramento delle performance, dove l'esperienza operativa può guidare l'evoluzione delle configurazioni verso setup sempre più ottimizzati per il contesto di deployment specifico.

3.2 Layer di Storage

Il layer di storage è implementato seguendo una strategia di persistenza ibrida che combina MongoDB per i metadati strutturati e Amazon S3 per lo storage di contenuti multimediali e modelli 3D. Questa architettura ottimizza l'accesso ai dati in base ai pattern di utilizzo specifici di ogni tipo di informazione.

3.2.1 MongoDB - Database dei metadati

MongoDB è configurato come replica set per garantire consistenza e supportare operazioni di change stream necessarie per le notifiche real-time. Il database è organizzato in tre collection principali, ciascuna ottimizzata per specifici pattern di accesso:

3.2.1.1 Collection "models"

La collection principale memorizza i metadati dei modelli 3D e il tracking delle fasi di elaborazione:

Listing 3.3: Struttura documento modello

```
{  
    "_id": "ObjectId",  
    "model_name": "Video Demo 1",  
    "created_at": "2024-07-24T10:30:00Z",  
    "video_s3_key": "videos/uuid-123/input_video.mp4",  
    "zip_model_suffix": "3d_model.zip",  
    "overall_status": "COMPLETED",  
    "current_phase": "metrics_evaluation",  
    "parent_model_id": null, // Per supportare la  
    fork  
  
    // Tracking dettagliato delle fasi  
    "phases": {  
        "frame_extraction": {  
            "status": "COMPLETED",  
            "started_at": "2024-07-24T10:30:00  
                Z",  
            "completed_at": "2024-07-24T10  
                :32:00Z",  
        }  
    }  
}
```

```

        "metadata": {
            "frame_count": 180,
            "processing_params": {"fps": 6, "width": 1920}
        }
    },
    "training": {
        "status": "COMPLETED",
        "metadata": {
            "training_duration_seconds": 1847.3,
            "training_parameters": {
                "engine": "INRIA",
                "quality_level": "balanced",
                "final_params": {"iterations": 30000, "lr": 0.0025}
            }
        }
    },
    "metrics_evaluation": {
        "status": "COMPLETED",
        "metadata": {
            "metrics": {
                "PSNR": 28.45,
                "SSIM": 0.89,
                "LPIPS": 0.12
            }
        }
    }
},
// Configurazione training
"training_config": {
    "engine": "INRIA",
    "quality_level": "balanced"
}
}

```

Motivazioni del design:

- **Struttura denormalizzata:** Le fasi sono embedded nel documento per ridurre le query multiple
- **Metadati flessibili:** Schema dinamico per adattarsi ai diversi tipi di informazioni per fase
- **Supporto fork:** Il campo `parent_model_id` abilita la creazione di varianti (fork)
- **Tracking temporale:** Timestamp dettagliati per analisi performance e debugging

3.2.1.2 Collection "training_params"

Memorizza le configurazioni degli algoritmi con parametrizzazione multi-livello:

Listing 3.4: Struttura configurazione algoritmo

```
{  
    "algorithm_name": "gaussian_splatting_original",  
    "display_name": "3D Gaussian Splatting (Fixed  
        Iterations)",  
    "version": "1.5",  
    "active": true,  
  
    // Parametri base (livello balanced)  
    "base_params": {  
        "iterations": 30000,  
        "densify_grad_threshold": 0.0002,  
        "densification_interval": 100,  
        "eval": true  
    },  
  
    // Moltiplicatori per livelli di qualità  
    "quality_multipliers": {  
        "fast": {"iterations": 0.8, "  
            densify_grad_threshold": 1.2},  
        "balanced": {"iterations": 1.0, "  
            densify_grad_threshold": 1.0},  
        "quality": {"iterations": 1.2, "  
            densify_grad_threshold": 0.8}  
    },  
  
    // Configurazione hardware-specific  
    "hardware_config": {  
        "baseline_vram_gb": 24,  
        "min_vram_gb": 8,  
        "resolution_thresholds": [  
            {"vram_threshold": 24, "target_width":  
                3840, "target_height": 2160},  
            {"vram_threshold": 16, "target_width":  
                1920, "target_height": 1080}  
        ],  
        "scaling_formulas": {  
            "densify_grad_threshold": {  
                "formula": "max(1.0, 2.5 -  
                    (vram_factor * 1.5))"  
  
                ,  
                "min": 1.0, "max": 4.0  
            }  
        }  
    }  
}
```

Vantaggi dell'approccio:

- **Configurabilità runtime:** Modifiche ai parametri senza redeploy
- **Sperimentazione facilitata:** Test di configurazioni diverse per ottimizzazioni
- **Adattamento hardware:** Scaling automatico basato su VRAM disponibile
- **Versionamento:** Tracking delle evoluzioni delle configurazioni

3.2.1.3 Collection "users"

Sistema di autenticazione con ruoli e gestione sessioni:

Listing 3.5: Struttura utente

```
{  
    "_id": "ObjectId",  
    "username": "admin",  
    "hashed_password": "$2b$12$...", // bcrypt hash  
    "role": "admin",  
    "created_at": "2024-07-24T10:00:00Z",  
    "last_login": "2024-07-24T15:30:00Z"  
}
```

3.2.2 Amazon S3 - Object Storage

3.2.2.1 Organizzazione Bucket e Prefissi

Il sistema utilizza una strategia di organizzazione gerarchica su S3 per ottimizzare l'accesso e la gestione del ciclo di vita dei contenuti:

```
s3://3dgs-bucket/  
├── videos/  
│   └── {model_id}/  
│       └── input_video.mp4  
├── delivery/  
│   └── {model_id}/  
│       └── 3d_model.zip  
└── staging/  
    └── {model_id}/  
        ├── phase_frame_extraction.zip  
        ├── point_cloud_building_phase.zip  
        └── training_phase.zip
```

3.2.2.2 Gestione contenuti per tipologia

Video sorgente (videos/)

I video originali caricati dagli utenti vengono memorizzati con chiavi UUID per evitare conflitti e garantire privacy:

Listing 3.6: Generazione chiave video

```
upload_id = str(uuid4())
s3_key = f"videos/{upload_id}/input_video.mp4"
presigned_url = repository_service.
    generate_presigned_url_upload(s3_key, content_type)
```

Caratteristiche:

- **Upload diretto:** Utilizzo di presigned URL per evitare transito attraverso backend
- **Retention policy:** Possibile archiviazione automatica dopo completamento processing
- **Versioning:** Supporto per multiple versioni dello stesso contenuto

Modelli finali (delivery/)

I modelli 3D ottimizzati per la visualizzazione web vengono memorizzati in formato compresso ed ogni archivio contiene la cloud point finale in formato .ksplat e la lista delle camere in formato .json prodotte dall'algoritmo di training:

Listing 3.7: Upload modello finale

```
zip_model_s3_key = f"delivery/{model_id}/3d_model.zip"
# Contiene: point_cloud.ksplat + cameras.json
repository_service.upload(zip_filename, zip_model_s3_key)
```

Ottimizzazioni:

- **Formato KSPLAT:** Conversione da PLY per ridurre dimensioni e ottimizzare rendering
- **CDN integration:** Possibile distribuzione attraverso CloudFront per performance globali
- **Caching headers:** Configurazione per ottimizzare caching browser

Dati Intermedi (staging/)

Le fasi intermedie vengono persistite per supportare fork, retry e debugging:

Listing 3.8: Pattern salvataggio fase intermedia

```
phase_zip_s3_key = f"staging/{parent_model_id}/{PHASE_NAME}  
                    }_phase.zip"  
phase_zip_helper.create_phase_zip_and_upload(model_id,  
                                              model_dir, PHASE_NAME, directories)
```

Benefici:

- **Recuperabilità:** Possibilità di ripartire da fasi intermedie in caso di errori
- **Sperimentazione:** Fork di modelli per testare algoritmi diversi senza reprocessing completo
- **Debugging:** Accesso ai dati intermedi per analisi e ottimizzazioni

3.2.2.3 Pattern di Accesso Ottimizzati

Workflow di elaborazione

Durante l'elaborazione, il sistema implementa un pattern cache-first per ottimizzare l'I/O:

Listing 3.9: Pattern cache-first per dati S3

```
# 1. Verifica cache locale  
if os.path.exists(local_cache_path) and is_valid_cache(  
    local_cache_path):  
    return local_cache_path  
  
# 2. Download da S3 con caching  
repository_service.download(s3_key, local_path)  
update_cache_metadata(local_path)  
return local_path
```

Frontend Access Pattern

Per l'accesso frontend, il sistema combina metadata queries su MongoDB con contenuti diretti da S3:

Listing 3.10: Pattern accesso ibrido frontend

```
# 1. Recupera metadati da MongoDB
model_metadata = model_service.get_model_by_id(model_id)

# 2. Genera URL diretto per contenuti S3
if model_metadata.zip_model_suffix:
    model_url = f"https://s3.amazonaws.com/bucket/delivery/{model_id}/{model_metadata.zip_model_suffix}"

    return {
        "metadata": model_metadata,
        "model_download_url": model_url
    }
```

Questa strategia ibrida garantisce performance ottimali combinando la flessibilità di MongoDB per query complesse sui metadati con la scalabilità e durabilità di S3 per contenuti di grandi dimensioni.

3.3 Layer di Orchestrazione

Il layer di orchestrazione coordina in modo asincrono l'esecuzione sequenziale delle cinque fasi di elaborazione del workflow. L'implementazione si basa su RabbitMQ come message broker e implementa un pattern Producer-Consumer specializzato per gestire workflow di training complessi con supporto per operazioni di retry e fork.

3.3.1 Code di messaggi e configurazione RabbitMQ

Il sistema implementa cinque code dedicate, ciascuna responsabile di una fase specifica del workflow:

Listing 3.11: Definizione code specializzate per workflow

```
queues = [
    'frame_extraction_queue',           # Fase 1: Estrazione frame
    'video',
    'point_cloud_queue',               # Fase 2: Generazione
    'nuvola_punti',
    'model_training_queue',            # Fase 3: Training
    'algoritmi_GS',
    'upload_queue',                   # Fase 4: Upload modello 3D
    'metrics_generation_queue'         # Fase 5: Calcolo metriche
    'qualità'
]
```

3.3.1.1 Configurazione di persistenza e acknowledgment

Ogni coda è configurata per garantire la massima affidabilità del sistema:

- **Persistenza delle code:** `durable=True` garantisce che le code sopravvivano ai riavvii del broker
- **Persistenza dei messaggi:** `delivery_mode=2` assicura che tutti i messaggi siano scritti su disco
- **Acknowledgment immediato:** I messaggi RabbitMQ vengono consumati e confermati immediatamente dopo la lettura e il parsing, prima dell'avvio del job vero e proprio. Questo garantisce che i messaggi non rimangano bloccati in coda in caso di problemi durante l'elaborazione e permette di interrompere definitivamente job problematici tramite il restart dei container, senza che questi vengano automaticamente riprocessati.

3.3.1.2 Monitoraggio RabbitMQ

Il sistema fornisce visibilità completa attraverso la RabbitMQ Management Interface, che offre monitoraggio real-time delle code, includendo:

- Numero di messaggi in attesa per ogni coda
- Throughput e metriche di performance
- Stato delle connessioni producer-consumer
- Gestione delle connessioni dedicate e isolate tra API Gateway (producer) e Job Executor (consumer)

Virtual host	Name	Type	Features	State	Messages			Message rates		
					Ready	Unacked	Total	Incoming	deliver / get	ack
/	frame_extraction_queue	classic	D	running	0	0	0	0.00/s	0.00/s	0.00/s
/	metrics_generation_queue	classic	D	running	0	0	0	0.00/s	0.00/s	0.00/s
/	model_training_queue	classic	D	running	0	0	0	0.00/s	0.00/s	0.00/s
/	point_cloud_queue	classic	D	running	0	0	0	0.00/s	0.00/s	0.00/s
/	upload_queue	classic	D	running	0	0	0	0.00/s	0.00/s	0.00/s

Figura 3.2: Interfaccia di gestione RabbitMQ - Vista code attive del sistema

3.3.2 Pattern Producer-Consumer ibrido

Il sistema implementa una variante specializzata del pattern Producer-Consumer tradizionale, dove il Job Executor assume alternativamente i ruoli di consumer e producer durante l'esecuzione del workflow sequenziale. Questa architettura elimina la necessità di un coordinatore esterno e garantisce un flusso continuo attraverso le cinque fasi di elaborazione.

3.3.2.1 Architettura del Pattern

Il pattern si basa su due componenti principali con responsabilità distinte:

- **API Gateway - Producer Esclusivo:** funge da punto di ingresso del sistema, responsabile esclusivamente dell'inserimento dei job iniziali nelle code appropriate. Utilizza un mapping standardizzato fase-coda per determinare il punto di partenza del workflow (fork e retry).
- **Job Executor - Consumer-Producer ibrido:** implementa il cuore del pattern attraverso un comportamento duale che si alterna dinamicamente durante l'elaborazione. Opera come consumer durante l'estrazione e il processing dei messaggi dalle code, quindi assume il ruolo di producer per l'invio del job alla fase successiva.

3.3.2.2 Meccanismo di Funzionamento

Il workflow segue un ciclo continuo strutturato in tre fasi principali:

1. **Fase Consumer:** Il Job Executor controlla sequenzialmente le code tramite polling attivo, utilizzando il metodo `process_queues_sequentially` che implementa un ciclo di controllo continuo su tutte le code del sistema.
2. **Fase Processing:** Una volta estratto un messaggio, il sistema utilizza un dispatcher centralizzato che effettua il routing verso l'handler specifico in base alla coda di origine (`handle_frame_extraction`, `handle_point_cloud_building`, etc.).
3. **Fase Producer:** Al completamento dell'elaborazione, `send_to_next_phase` implementa la transizione dal ruolo consumer al ruolo producer, inserendo automaticamente il job nella coda della fase successiva.

3.3.2.3 Vantaggi Architetturali

Questo approccio offre diversi benefici rispetto a implementazioni tradizionali:

- **Coordinamento automatico:** Ogni fase completata innesca automaticamente la successiva, eliminando la necessità di coordinamento esterno
- **Resilienza:** I messaggi persistenti garantiscono la continuità in caso di interruzioni temporanee
- **Semplicità:** Un singolo componente gestisce l'intero flusso, riducendo la complessità architetturale
- **Scalabilità:** Il pattern supporta facilmente l'aggiunta di nuove fasi senza modifiche strutturali

La Figura 3.3 illustra il flusso completo del pattern, evidenziando le transizioni tra i ruoli di consumer e producer e il coordinamento automatico tra le fasi di elaborazione.

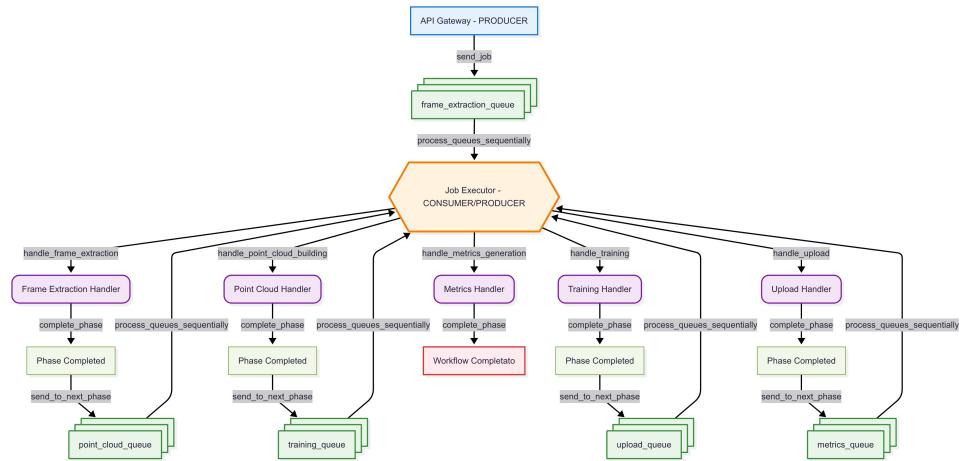


Figura 3.3: Schema del pattern Producer-Consumer ibrido implementato dal Job Executor

3.3.3 Persistenza stati intermedi

3.3.3.1 Pattern di gestione dati intermedi

Le prime tre fasi del workflow (Frame Extraction, Point Cloud Reconstruction, Training) implementano un pattern specializzato per la gestione dei dati intermedi che abilita funzionalità avanzate di fork e retry.

Ogni fase salva i propri risultati in archivi compressi standardizzati su Amazon S3, seguendo la convenzione di denominazione `{nome_fase}_phase.zip`. Questa struttura permette il recupero selettivo di dati intermedi per operazioni di reprocessing.

Listing 3.12: Pattern recupero e salvataggio dati intermedi

```
# Recupero dati fase precedente
success = phase_zip_helper.download_and_extract_phase_zip(
    phase_zip_s3_key, model_dir
)

# Salvataggio risultati fase corrente
is_zip_uploaded = phase_zip_helper.
    create_phase_zip_and_upload(
        model_id, model_dir, PHASE_ZIP_NAME, ['directory_output']
    )
```

3.3.3.2 Funzionalità di Fork e Retry

Il pattern dei dati intermedi abilita la **fork e il retry di modelli** a partire da fasi intermedie, permettendo comparazioni algoritmiche e sperimentazione senza dover ripetere l'intero workflow.

Ad esempio, un utente potrebbe biforcicare un modello completato almeno fino alla fase di Point Cloud Reconstruction per testare un algoritmo di training differente, riutilizzando la point cloud già generata in precedenza e salvata come dato intermedio su S3. Questo approccio riduce significativamente i tempi di sperimentazione e ottimizza l'utilizzo delle risorse computazionali.

3.3.3.3 Gestione fasi secondarie

Tra le fasi già citate in precedenza, ci sono due fasi che seguono un pattern semplificato per il quale non serve generare dati intermedi persistenti e sono considerate **secondarie** (seppur fondamentali). Si tratta di:

- **Upload modello finale:** Gestisce la preparazione e il caricamento del modello 3D per la visualizzazione, convertendo i risultati del training in formati ottimizzati per il web viewer.

- **Generazione metriche:** Calcola le metriche di qualità (PSNR, SSIM, LPIPS) confrontando rendering di test con immagini ground truth, salvando i risultati direttamente nel database.

Separando queste fasi permettiamo la visualizzazione immediata dei modelli 3D generati mentre le metriche vengono calcolate in background: questo migliora l'esperienza utente senza compromettere l'affidabilità del sistema.

3.3.4 Resilienza e Fault Tolerance

Il sistema implementa una strategia di gestione degli errori progettata per garantire continuità operativa attraverso meccanismi di isolamento e persistenza dei dati critici.

3.3.4.1 Strategie di Error Handling

Viene adottato un approccio stratificato per la gestione degli errori, implementando meccanismi di protezione a diversi livelli:

- **Isolamento degli errori:** Il fallimento di un singolo job non compromette l'elaborazione di altri job nel sistema. Ogni workflow opera in modo indipendente, permettendo al sistema di continuare l'elaborazione dei job rimanenti anche in presenza di errori specifici.
- **Gestione fallimenti controllata:** Il sistema evita loop infiniti di retry confermando anche i messaggi falliti, delegando la gestione dei retry all'interfaccia utente attraverso funzionalità di restart del modello.

3.3.4.2 Meccanismi di Persistenza degli Stati

La resilienza del sistema si basa su diversi meccanismi di persistenza che garantiscono la continuità operativa:

- **Checkpointing degli stati:** Il sistema implementa tracking degli stati di elaborazione attraverso aggiornamenti dei metadati in MongoDB. Il tracking avviene all'inizio e alla fine di ogni fase (`start_phase` e `complete_phase`), permettendo il monitoraggio del progresso e la gestione dei retry.
- **Dati intermedi persistenti:** Le fasi intermedie del workflow vengono salvate come archivi compressi su Amazon S3, abilitando funzionalità di fork e retry senza dover riprocessare l'intero pipeline.

3.3.4.3 Database Monitoring

MongoDB fornisce tracking dettagliato dello stato di ogni modello, includendo:

- Progressione attraverso le fasi del workflow
- Tempi di elaborazione per ogni fase
- Metadati completi per troubleshooting e analisi delle performance
- Gestione degli stati di errore e recovery

Questa architettura di resilienza assicura che il sistema possa operare stabilmente, fornendo meccanismi per il recovery attraverso l'interfaccia utente e mantenendo la visibilità completa sullo stato delle elaborazioni.

3.4 Layer di Pre-elaborazione

Il Layer di Pre-elaborazione rappresenta la fase iniziale critica del workflow, responsabile della trasformazione del contenuto video di input in una rappresentazione geometrica 3D utilizzabile per il training. Questo layer combina due processi sequenziali complementari: l'estrazione intelligente di frame dal video sorgente e la ricostruzione della nuvola di punti 3D attraverso tecniche di Structure from Motion.

3.4.1 Architettura del Layer

Il Layer di Pre-elaborazione è organizzato come pipeline sequenziale che coordina due servizi specializzati:

1. **Video Preprocessing Service:** Trasformazione video in dataset di frame ottimizzato
2. **Point Cloud Reconstruction Service:** Ricostruzione geometrica 3D tramite COLMAP

3.4.1.1 Strategia di Containerizzazione Differenziata

Il layer implementa strategie di deployment diverse per i due servizi, basate su considerazioni pragmatiche di complessità e risorse.

Il **Video Preprocessing Service** è mantenuto **all'interno del container del Job Executor**, a differenza degli altri servizi computazionali che sono containerizzati separatamente.

Questa scelta deriva da valutazioni pragmatiche:

- **Peso computazionale marginale:** Il preprocessing video ha durata contenuta e non impatta l'esecuzione sequenziale dei job
- **Dipendenze leggere:** Utilizzo di librerie esterne facilmente integrabili senza necessità di ambienti dedicati
- **Complessità operativa ridotta:** La separazione introdurrebbe complessità di deploy senza benefici concreti

Il **Point Cloud Reconstruction Service** è implementato come container specializzato basato su PyTorch con supporto CUDA, giustificato da:

- **Complessità computazionale:** Operazioni intensive che richiedono ottimizzazioni specifiche
- **Dipendenze pesanti:** Integrazione di COLMAP e librerie di computer vision specializzate
- **Isolamento computazionale:** Necessità di ambiente dedicato per riproducibilità e gestione risorse

3.4.2 Video Preprocessing Service

Il preprocessing video è orchestrato dall'handler `handle_frame_extraction` che funge da punto di connessione tra il sistema di code RabbitMQ e il servizio specializzato. L'handler coordina il flusso di elaborazione delegando le operazioni atomiche al `VideoFrameExtractionService`, garantendo isolamento delle responsabilità e gestione robusta degli errori. Il `VideoFrameExtractionService` implementa tutte le operazioni di preprocessing come servizio atomico, assicurando consistenza e tracciabilità completa del processo attraverso un'unica interfaccia pubblica `extract_frames`.

3.4.2.1 Pipeline di Preprocessing Video

1) Download e Validazione

La pipeline inizia con il recupero del video sorgente dal storage S3 utilizzando il sistema di cache intelligente del `RepositoryService`, seguito da controlli rigorosi implementati dal metodo `_validate_inputs` per garantire integrità e accessibilità del file video.

2) Analisi Metadati Video

Il sistema esegue un'analisi completa delle caratteristiche del video attraverso il metodo `_analyze_video`:

Listing 3.13: Analisi metadati video completa

```
def _analyze_video(self, video_path: str) -> Dict[str, Any]:
    cap = cv2.VideoCapture(video_path)
    original_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    original_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    video_fps = cap.get(cv2.CAP_PROP_FPS)
    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    duration = total_frames / video_fps if video_fps > 0 else
        0

    # Determinazione orientamento
    is_portrait = original_height > original_width

    return {
        'original_width': original_width,
        'original_height': original_height,
        'video_fps': video_fps,
        'duration_seconds': duration,
        'is_portrait': is_portrait,
        'orientation': 'Portrait' if is_portrait else
            'Landscape'
    }
```

3) Adattamento Orientamento

Una delle ottimizzazioni più importanti riguarda la gestione intelligente dell'orientamento video:

Listing 3.14: Logica di adattamento orientamento

```
# Per portrait, target_height diventa la width effettiva
final_width = target_height if is_portrait else
    target_width
```

I parametri di qualità sono configurati assumendo orientamento landscape standard. Per video portrait, il sistema utilizza `target_height` come dimensione principale per mantenere la qualità visiva desiderata.

4) Calibrazione FPS per Target Frame

Il cuore del preprocessing è la calibrazione intelligente del frame rate di estrazione:

Listing 3.15: Algoritmo di calibrazione FPS

```
if duration > 0:
```

```

        extraction_fps = target_frame_count / duration
        extraction_fps = min(extraction_fps, video_fps) # 
            Vincolo hardware
        extraction_fps = max(0.5, extraction_fps)          #
            Soglia qualità minima
    else:
        extraction_fps = 1.0  # Fallback sicuro

```

Sebbene il numero di frame target sia teoricamente parametrizzabile attraverso il sistema di parametrizzazione adattiva, la configurazione attuale fissa questo valore a 200 frame, identificato come ottimale per bilanciare qualità della ricostruzione e efficienza computazionale.

5) Estrazione Frame con Sharp-Frames

L'estrazione utilizza il tool specializzato Sharp-Frames con configurazione ottimizzata per la pipeline COLMAP:

Listing 3.16: Configurazione Sharp-Frames ottimizzata

```

cmd = [
    "sharp-frames",
    video_path,
    output_directory,
    "--selection-method", "best-n",
    "--min-buffer", "1",
    "--fps", str(optimized_params['extraction_fps'])
]

# Aggiunta parametro width se necessario
final_width = optimized_params.get('final_width')
if final_width is not None:
    cmd.extend(["--width", str(final_width)])

```

La scelta del metodo `best-n` prioritizza la qualità dei singoli frame, aspetto critico per il successo delle fasi successive di feature matching in COLMAP.

6) Finalizzazione e Persistenza

Il processo conclude con validazione output, generazione thumbnail per il layer di presentazione, creazione archivio fase per supportare fork/retry, e generazione metadati dettagliati per tracciabilità completa.

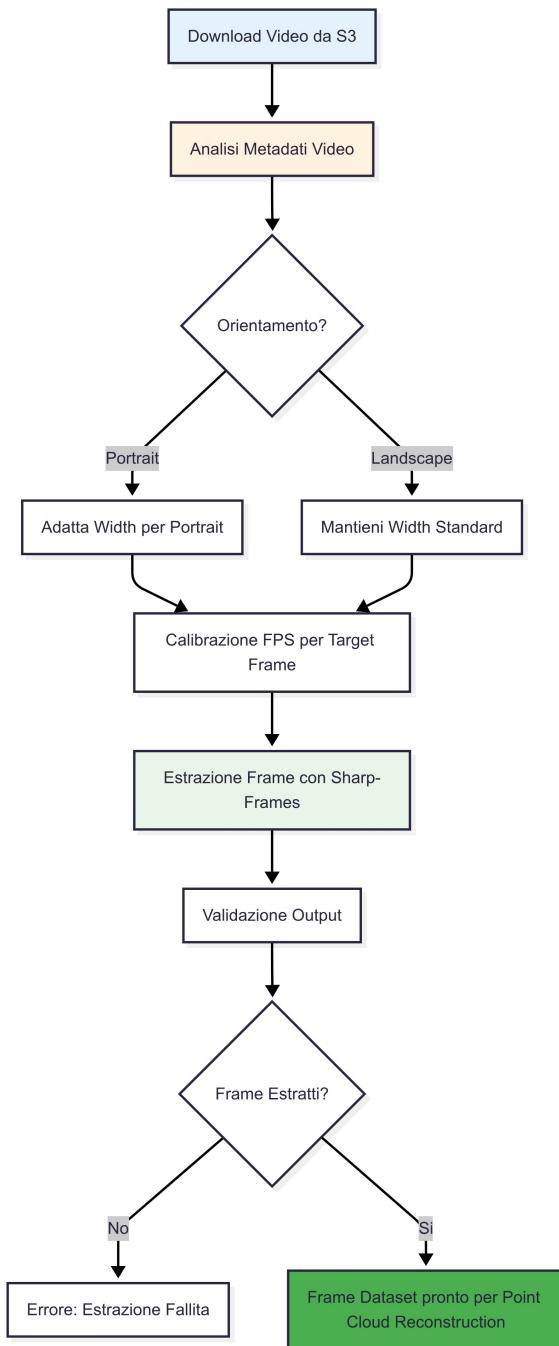


Figura 3.4: Schema del estrazione frames da video

3.4.3 Point Cloud Reconstruction Service

3.4.3.1 Handler Point Cloud Building

La Point Cloud Reconstruction è orchestrata dall'handler `handle_point_cloud_building` che coordina la ricostruzione della nuvola di punti 3D a partire dal dataset di frame ottimizzato.

Recupero Dataset e Validazione

Il sistema recupera il dataset di frame dalla fase precedente, gestendo automaticamente diverse fonti tramite il pattern di dati intermedi.

Generazione Point Cloud via COLMAP

La ricostruzione avviene tramite chiamata al servizio COLMAP containerizzato con analisi automatica della strategia ottimale:

Listing 3.17: Generazione point cloud con monitoraggio performance

```
sparse_dir = os.path.join(model_dir, 'sparse')

colmap_start_time = datetime.utcnow()

# COLMAP API call con analisi automatica strategia
convert_request = {"input_dir": model_dir}
response = requests.post(COLMAP_API_URL + "/convert", json
=convert_request)

colmap_duration_seconds = (datetime.utcnow() -
colmap_start_time).total_seconds()

# Verifica output e raccolta statistiche
generated_points = job_utils.
get_colmap_reconstruction_stats(model_dir)
```

3.4.3.2 Strategie di Feature Matching: Analisi Comparativa

- **Exhaustive Matching:** la strategia exhaustive confronta ogni immagine con tutte le altre nel dataset, con **complessità $O(n^2)$** . Garantisce qualità massima per dataset fino a diverse centinaia di immagini, ha tempi di elaborazione che variano da 2 a 5 minuti per scene rappresentate da circa 200 immagini a seconda della loro complessità.
- **Sequential Matching:** la strategia sequential sfrutta l'ordine temporale dei frame video, con **complessità $O(n \times overlap)$** . Offre efficienza temporale significativa ma produce point cloud meno dense che richiedono maggiore sforzo durante le fasi di pruning e densification del training Gaussian Splatting.

- **Vocabulary Tree Matching:** la strategia vocabulary tree utilizza un albero di vocabolario precompilato, con **complessità $O(n \log n)$** . È ottimale per grandi dataset (migliaia di immagini) e offre una qualità molto vicina a quella della strategia exhaustive quando la quantità di feature media è bassa.

3.4.3.3 Pipeline di Ricostruzione Ottimizzata

Il processo include feature extraction parametrizzata, bundle adjustment ottimizzato con tolleranze specifiche, e generazione di statistiche dettagliate per tracciabilità e debugging.

3.4.4 Integrazione e Coordinamento

3.4.4.1 Flusso di Dati tra Servizi

Il layer implementa un flusso di dati efficiente attraverso il pattern di archivi intermedi:

1. **Video Preprocessing:** Genera archivio `point_cloud_building_phase.zip` contenente frame estratti
2. **Point Cloud Reconstruction:** Consuma frame e genera archivio `training_phase.zip` con nuvola di punti
3. **Handoff al Training:** Fornisce dataset completo per gli algoritmi Gaussian Splatting

3.4.4.2 Gestione Metadati e Performance

Entrambi i servizi generano metadati dettagliati che documentano:

- **Parametri di elaborazione:** Configurazioni utilizzate per riproducibilità
- **Statistiche quantitative:** Conteggi frame, punti 3D generati, tempi di elaborazione
- **Performance data:** Metriche per analisi e ottimizzazioni future
- **Versioning:** Tracciabilità delle versioni software utilizzate

Il Layer di Pre-elaborazione rappresenta quindi un componente critico e sofisticato che, attraverso l'integrazione intelligente di algoritmi di preprocessing video e ricostruzione geometrica 3D, trasforma contenuti video grezzi in rappresentazioni geometriche ottimizzate per il training Gaussian Splatting, garantendo qualità, efficienza e tracciabilità completa del processo.

3.5 Layer di Training

Il motore di training rappresenta il cuore computazionale del sistema, responsabile dell'esecuzione degli algoritmi di Gaussian Splatting. L'implementazione adotta un approccio containerizzato che isola ogni algoritmo in un ambiente dedicato, esponendo le funzionalità attraverso API REST standardizzate.

3.5.1 Architettura containerizzata multi-algoritmo

Il sistema implementa tre container specializzati, ciascuno dedicato a un algoritmo specifico:

- **gaussian-splatting**: Implementazione di riferimento del paper originale
- **3dgs-mcmc**: Variante con campionamento MCMC per ottimizzazione stocastica
- **taming-3dgs**: Versione ottimizzata per scene complesse

Ogni container segue una strategia di build multi-stage per ottimizzare le dimensioni e i tempi di compilazione:

3.5.2 Handler Training

L'handler di training coordina l'esecuzione degli algoritmi di Gaussian Splatting, gestendo la configurazione dei parametri e le funzionalità specifiche come la depth regularization per il motore standard.

3.5.2.1 Recupero Dataset e Configurazione

Il sistema recupera la point cloud e le immagini processate dalla fase precedente, configurando dinamicamente i parametri in base all'algoritmo selezionato:

Listing 3.18: Inizializzazione e configurazione training

```
model_service.start_phase(model_id, "training")
model = model_service.get_model_by_id(model_id)

model_dir = os.path.join(WORKING_DIR, f"{model_id}")
image_dir = os.path.join(model_dir, "images")
sparse_dir = os.path.join(model_dir, "sparse")

# Recupera ZIP dalla fase point cloud building se necessario
if not (os.path.exists(image_dir) and os.path.exists(
    sparse_dir)):
```

```

    training_zip_s3_key = f"{S3_STAGING_PREFIX}/{model.
        parent_model_id}/{TRAINING_PHASE_ZIP_NAME}"
    success = phase_zip_helper.download_and_extract_phase_zip(
        training_zip_s3_key, model_dir
    )

    # Configurazione parametri tramite TrainingParamsService
    engine = model.training_config.get('engine')
    quality_level = model.training_config.get('quality_level')

    generated_params = training_params_service.generate_params(
        (
            Engine(engine), QualityLevel(quality_level)
        )
    )

```

3.5.2.2 Depth Regularization per Gaussian Splatting standard

Per il motore *Gaussian Splatting* standard, viene implementata una fase di **depth regularization**, come suggerito dalla documentazione ufficiale, al fine di migliorare la qualità della ricostruzione nelle scene real-world. Questa tecnica vincola le gaussiane a disporsi lungo superfici coerenti con le stime di profondità ottenute dalle immagini, riducendo la presenza di artefatti e gaussiane fluttuanti nello spazio 3D, tipiche delle scene acquisite dal mondo reale.

Nel caso delle varianti **MCMC** e **Taming**, invece, non viene impiegata depth regularization esplicita. Queste versioni adottano strategie di ottimizzazione alternative — rispettivamente, tramite sampling stocastico e controlli specifici sulla crescita e la fusione delle gaussiane — che mitigano la formazione di outlier e favoriscono la distribuzione regolare delle gaussiane nello spazio, rendendo la regularizzazione sulla profondità meno necessaria. È importante sottolineare che la presenza della depth regularization, pur migliorando la plausibilità geometrica della scena, **può in alcuni casi limitare le performance su metriche quantitative come il PSNR**, poiché impone vincoli aggiuntivi rispetto all'ottimizzazione puramente basata sulla ricostruzione delle immagini.

Listing 3.19: Endpoint depth regularization

```

@app.post("/depth_regularization")
async def make_depth_regularization(request:
    DepthRegularizationRequest):
    logger.info(f"Starting depth generation - Input directory:
        {request.input_dir}")

    images_dir = os.path.join(request.input_dir, 'images')
    depths_dir = os.path.join(request.input_dir, 'depths')

    # STEP 1: Genera depth maps con Depth Anything V2
    depth_command = (

```

```

f"cd /workspace/Depth-Anything-V2 && "
f"python3 run.py --encoder vitl --pred-only --grayscale "
f"--img-path {images_dir} --outdir {depths_dir}"
)

# Esecuzione depth generation con logging multi-thread
depth_process = subprocess.Popen(
    depth_command, shell=True, stdout=subprocess.PIPE,
    stderr=subprocess.PIPE, text=True, bufsize=1
)

# [Thread management per logging real-time]

# STEP 2: Genera depth_params.json con scaling automatico
params_command = (
    f"python3 /workspace/gaussian-splatting/utils/
        make_depth_scale.py "
    f"--base_dir {request.input_dir} --depths_dir {depths_dir}
        "
)
# [Esecuzione simile con thread management]

return {
    "message": "Depth generation and params completed
        successfully",
    "depth_maps_count": depth_count,
    "depth_maps_dir": depths_dir
}

```

3.5.2.3 Esecuzione Training

Il sistema seleziona dinamicamente l'API dell'engine containerizzato corrispondente e avvia il processo di training:

Listing 3.20: Esecuzione training con engine selezionato

```

# Selezione API engine dalla mappatura
api_url = engine_map.get(engine, {}).get('api-url')
if not api_url:
    self.fail(model_id, "training", f"Error: No api url found
        for engine {engine}")
return False

# Costruzione richiesta di training
train_request = {
    "input_dir": model_dir,
    "output_dir": train_output_folder,
    "params": generated_params.final_params, #
        Parametri ottimizzati
}

print(f" Starting training with engine: {engine}")

```

```

    training_start_time = datetime.utcnow()

    # Chiamata API training containerizzato
    response = requests.post(f"{api_url}/train", json=
        train_request)

    training_end_time = datetime.utcnow()
    training_duration = training_end_time -
        training_start_time
    training_duration_seconds = training_duration.
        total_seconds()

    if response.status_code != 200:
        self.fail(model_id, "training", f"Training failed: {response.text}")
    return False

```

3.5.2.4 Finalizzazione e Metadati

La fase si conclude con l'aggiornamento dei metadati, includendo statistiche di performance e informazioni sulla depth regularization:

Listing 3.21: Finalizzazione training con metadati estesi

```

# Aggiorna metadati con statistiche complete
phase_metadata = {
    "training_duration_seconds": round(
        training_duration_seconds, 2),
    "depth_regularization_seconds": round(
        depth_duration_seconds, 2),
    "training_start_time": training_start_time.
        isoformat(),
    "training_end_time": training_end_time.isoformat()
        ,
    "training_parameters": {
        "engine": engine,
        "quality_level": quality_level,
        "final_params": generated_params.
            final_params,
        "depth_regularization_enabled": engine ==
            Engine.INRIA.value
    }
}

model_service.complete_phase(model_id, "training",
    metadata=phase_metadata)

```

3.5.3 Interfaccia API REST unificata

3.5.3.1 Standardizzazione degli endpoint

Ogni container espone un'interfaccia REST standardizzata che astrae le specificità implementative degli algoritmi sottostanti:

Listing 3.22: Definizione modelli dati per API REST

```
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel, Field
import subprocess
import threading
import queue
import logging
from typing import Dict, Any

app = FastAPI()

class TrainRequest(BaseModel):
    input_dir: str
    output_dir: str
    params: Dict[str, Any] = Field(
        default_factory=dict,
        description="Parametri specifici dell'algoritmo"
    )

    class RenderRequest(BaseModel):
        output_dir: str

    class MetricsRequest(BaseModel):
        output_dir: str

    class DepthRegularizationRequest(BaseModel):
        input_dir: str
```

3.5.3.2 Gestione parametri dinamica

La gestione dei parametri di training è implementata attraverso un sistema dinamico che costruisce automaticamente gli argomenti della command line:

Listing 3.23: Costruzione dinamica comando di training

```
@app.post("/train")
async def run_train(request: TrainRequest):
    logger.info(f"Starting training - Input: {request.
        input_dir}")

    # Costruzione dinamica del comando
    command = ["python3", "/workspace/gaussian-splatting/train
        .py"]
    depths_dir = os.path.join(request.input_dir, 'depths')
```

```

# Parametri obbligatori con depth regularization
command.extend([
    "--resolution", "1",
    "-s", request.input_dir,
    "-m", request.output_dir,
    "-d", depths_dir # Directory depth maps per
              regularization
])

# Parametri dinamici dall'utente
boolean_flags = {"eval"}

for param_key, value in request.params.items():
    if param_key in boolean_flags:
        if value:
            command.append(f"--{param_key}")
    elif param_key != 'resolution':
        command.extend([f"--{param_key}", str(value)])

# Flag antialiasing automatico
command.append("--antialiasing")

# Esecuzione con logging real-time multi-thread
# [Implementazione gestione processo e thread]

```

3.5.3.3 Endpoint Specializzati

Oltre al training principale, ogni container espone endpoint per le fasi complementari del workflow:

- **/train:** Esecuzione dell'algoritmo di training principale
- **/render:** Generazione delle immagini di test per validazione
- **/metrics:** Calcolo delle metriche di qualità (PSNR, SSIM, LPIPS)
- **/depth_regularization:** Generazione depth maps (solo gaussian-splatting standard)

3.5.4 Handler Metrics Generation

L'handler di metrics generation rappresenta la fase finale del workflow, responsabile della valutazione quantitativa del modello addestrato attraverso metriche standard di qualità visiva. Come per l'upload, questa fase è stata isolata strategicamente per permettere al modello di essere disponibile per il rendering mentre le metriche vengono calcolate in background.

3.5.4.1 Inizializzazione e Validazione Output

Il sistema verifica la disponibilità dell'output di training e prepara l'ambiente per la generazione delle metriche di valutazione:

Listing 3.24: Inizializzazione e validazione output training

```
model_service.start_phase(model_id, "metrics_evaluation")
model = model_service.get_model_by_id(model_id)
model_dir = os.path.join(WORKING_DIR, f"{model_id}")
Verifica esistenza output del training
output_dir = os.path.join(model_dir, 'output')
if not os.path.exists(output_dir):
    self.fail(model_id, "metrics_evaluation", f"No folder
        output found")
return False
engine = model.training_config.get('engine', 'INRIA')
```

3.5.4.2 Generazione render e calcolo metriche

Il sistema utilizza lo stesso engine containerizzato del training per generare render di valutazione e calcolare le metriche standard di qualità visiva (SSIM, PSNR, LPIPS):

Listing 3.25: Generazione render e calcolo metriche

```
engine = model.training_config.get('engine', 'INRIA')
Generazione render per valutazione
render_request = {"output_dir": output_dir}
response = requests.post(engine_map.get(engine).get('api-
    url') + "/render", json=render_request)
response.raise_for_status()
Calcolo metriche di qualità
metrics_request = {"output_dir": output_dir}
response = requests.post(engine_map.get(engine).get('api-
    url') + "/metrics", json=metrics_request)
response.raise_for_status()
Verifica generazione del file risultati
results_json_path = os.path.join(output_dir, "results.json
    ")
if not os.path.exists(results_json_path):
    raise FileNotFoundError("Il file 'results.json' non è
        stato trovato.")
with open(results_json_path, 'r') as f:
    results_data = json.load(f)
```

3.5.4.3 Finalizzazione e Persistenza Metriche

La fase si conclude con la persistenza delle metriche calcolate nei metadati del modello, rendendo immediatamente disponibili per visualizzazione nel frontend:

Listing 3.26: Persistenza metriche e completamento workflow

```
Salvataggio metriche nei metadati del modello
model_service.complete_phase(
    model_id,
    "metrics_evaluation",
    overall_status="COMPLETED",
    metadata={"metrics": results}
)
print(f" Metriché generate e salvate per model {model_id}")
```

La gestione intelligente delle chiavi del file JSON è fondamentale perché il nome delle sezioni include il numero di iterazioni specificato nella parametrizzazione (es. `ours_30000` per 30.000 iterazioni). Questo collegamento diretto tra parametri di training e struttura delle metriche garantisce che la valutazione sia sempre coerente con la configurazione utilizzata per l'addestramento. L'isolamento di questa fase permette all'utente di visualizzare il modello 3D immediatamente dopo l'upload, mentre le metriche vengono calcolate in background e aggiornate progressivamente nell'interfaccia.

3.6 Layer Backend-for-Frontend

3.6.1 Architettura e Funzionalità Principali

Il layer Backend-for-Frontend (API Gateway) funge da punto di accesso unico e sicuro per tutte le operazioni sui modelli 3D, orchestrando la comunicazione tra frontend, database MongoDB, sistema di code RabbitMQ e storage S3 (Fig. 3.5).

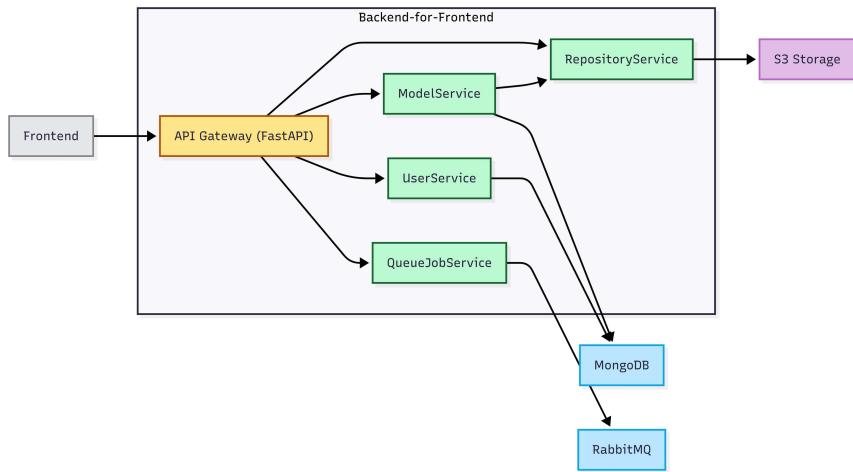


Figura 3.5: Architettura del layer Backend-for-Frontend

3.6.2 Riepilogo Endpoint REST e WebSocket

La Tabella 3.1 riepiloga i principali endpoint esposti dall'API Gateway.

Metodo	Endpoint	Descrizione	Autenticazione
POST	/register	Registrazione utente admin	Basic Auth
POST	/token	Login e rilascio JWT	Nessuna
POST	/models/	Creazione nuovo modello	Basic Auth
GET	/models/	Elenco modelli	Basic Auth
GET	/models/{id}	Dettaglio modello	Basic Auth
POST	/models/{id}/retry	Retry modello fallito	Basic Auth
DELETE	/models/{id}	Eliminazione modello	Basic Auth
POST	/s3/upload-url/	Generazione presigned URL S3	Basic Auth
GET	/health	Health check API	Nessuna
WS	/ws/notifications	Notifiche real-time	—

Tabella 3.1: Principali endpoint esposti dall'API Gateway

3.6.3 Logica e Flussi Principali degli Endpoint

3.6.3.1 Registrazione e Autenticazione

- Registrazione (POST /register):** Consente la creazione di nuovi utenti admin, protetta tramite HTTP Basic Authentication.
- Login (POST /token):** Permette agli utenti di autenticarsi tramite username e password e ricevere un JWT per l'accesso alle altre API.

3.6.3.2 Gestione Modelli 3D

- **Creazione modello** (POST /models/): Permette la creazione di un nuovo modello 3D (ex novo o fork). Il servizio:
 1. Valida la richiesta e crea il modello su MongoDB.
 2. Invia un job di elaborazione su RabbitMQ.
 3. Restituisce le informazioni del nuovo modello.
- **Elenco modelli** (GET /models/): Restituisce una lista di modelli 3D con supporto a paginazione, filtri (per nome e stato) e ordinamento.
- **Dettaglio modello** (GET /models/{id}): Recupera tutti i dettagli di un modello tramite il suo identificatore unico.
- **Retry modello** (POST /models/{id}/retry): Permette di riavviare l'elaborazione di un modello che ha subito un errore in una delle fasi.
- **Eliminazione modello** (DELETE /models/{id}): Consente di rimuovere un modello dal database (implementazione futura).

3.6.3.3 Gestione Upload Video

- **Generazione presigned URL** (POST /s3/upload-url/): Genera un URL presigned che consente l'upload sicuro di file video direttamente su S3, riducendo il carico sull'API Gateway.

3.6.3.4 Notifiche Real-time

- **WebSocket notifiche** (/ws/notifications): Permette ai client di ricevere aggiornamenti in tempo reale sullo stato dei modelli tramite WebSocket, grazie all'integrazione con i listener del database.

3.6.3.5 Endpoint di Servizio

- **Health check** (GET /health): Verifica lo stato di salute dell'API.

3.6.4 Autenticazione e Sicurezza

Il sistema implementa due modalità di autenticazione:

- **HTTP Basic Auth**: utilizzata per endpoint amministrativi e di registrazione.
- **JWT Bearer Token**: utilizzata per la maggior parte delle operazioni utente.

Le password degli utenti vengono gestite tramite hashing sicuro (**bcrypt**). I token JWT vengono firmati con una chiave segreta e prevedono una scadenza per garantire la sicurezza delle sessioni.

3.6.5 Creazione, fork e retry di un modello

Le fasi di creazione e rilancio di un modello sono illustrate nelle Figure 3.6 e 3.7, che mostrano le interazioni tra i principali componenti del sistema, dal gateway alle risorse di storage e messaggistica.

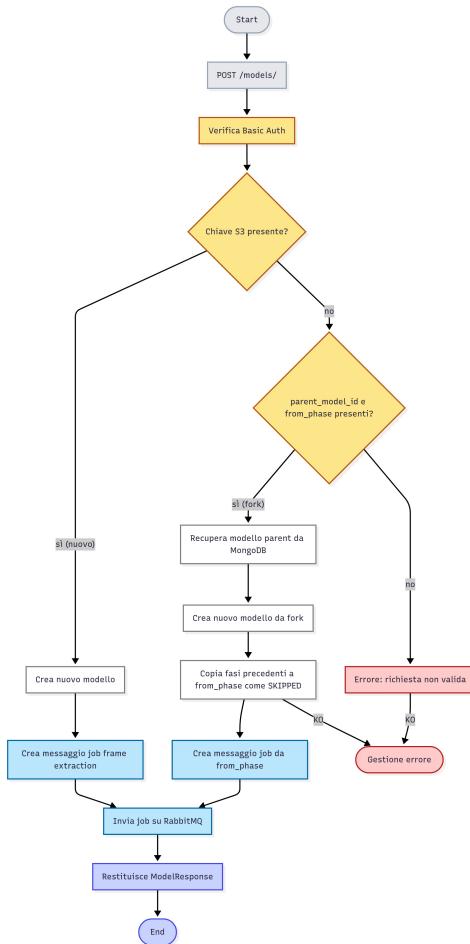
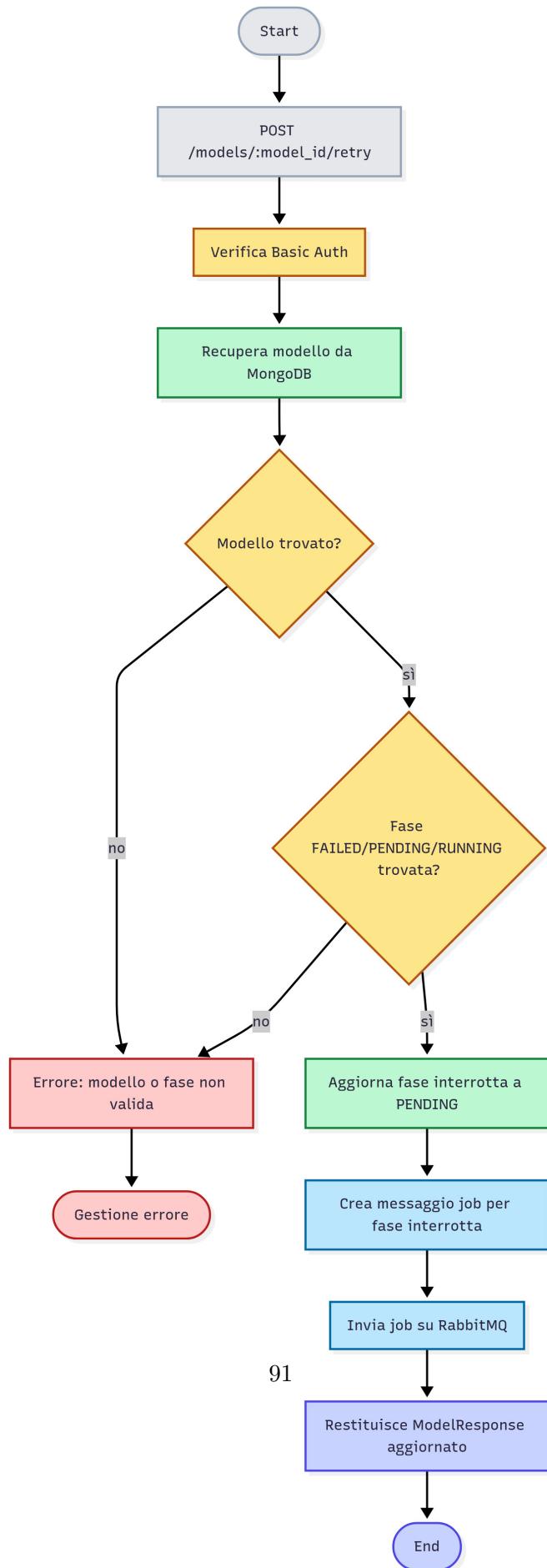


Figura 3.6: Diagramma per creazione e fork modello



3.6.6 Notifiche push in real-time

Ogni volta che avviene un cambiamento rilevante nello stato dei modelli all'interno del database, il sistema è in grado di propagare in tempo reale una notifica ai client connessi. Dopo l'apertura della connessione WebSocket, il frontend riceve aggiornamenti direttamente dall'API Gateway, garantendo all'utente informazioni sempre aggiornate sull'avanzamento delle elaborazioni, senza la necessità di refresh manuali o polling periodico.

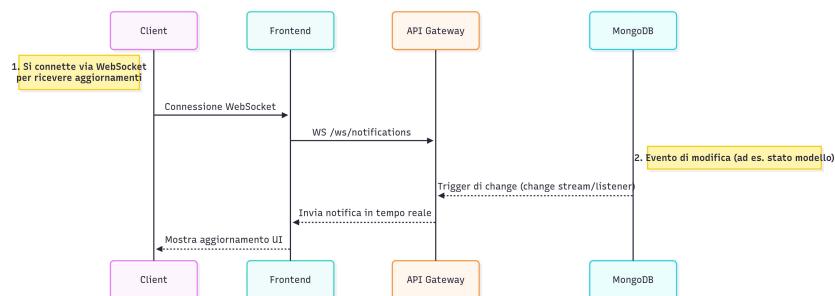


Figura 3.8: Flusso per notifiche push in real-time

3.6.7 Considerazioni e Best Practice

Il design di questo API Gateway consente di centralizzare la logica di business e la gestione della sicurezza, semplificando l'interazione tra frontend e servizi backend. La scelta di endpoint RESTful chiari, l'uso di WebSocket per le notifiche e l'integrazione di presigned URL per gli upload garantiscono **scalabilità, sicurezza e facilità d'integrazione**.

3.7 Layer di Presentazione

3.7.1 Architettura dell'interfaccia utente

Il frontend del sistema implementa un'interfaccia web moderna e responsiva progettata per rendere accessibile la tecnologia del 3D Gaussian Splatting a utenti non tecnici. L'applicazione, sviluppata con Vue.js 3 e Vuetify⁷, offre un'esperienza utente coerente e intuitiva attraverso componenti Material Design ottimizzati per la gestione di workflow complessi.

3.7.1.1 Dashboard Principale

La dashboard rappresenta il punto di ingresso principale del sistema, organizzata secondo principi di information architecture che privilegiano la

⁷<https://vuetifyjs.com/en/>

scansione rapida e l'accesso immediato alle funzionalità chiave. L'interfaccia adotta un layout a griglia responsivo che visualizza i modelli come card informative, ciascuna contenente:

- **Preview thumbnail** del modello renderizzato, generata automaticamente alla conclusione della prima fase del workflow (estrazione dei frame).
- **Badge informativi** che comunicano immediatamente engine utilizzato (INRIA, MCMC, TAMING) e livello di qualità (Fast, Balanced, Quality)
- **Status indicator** con codifica cromatica per identificare rapidamente lo stato di elaborazione
- **Timeline di processing** che visualizza le cinque fasi del workflow con tempi di esecuzione individuali
- **Metriche di qualità** (PSNR, SSIM, LPIPS) presentate in formato espandibile per non sovraccaricare l'interfaccia

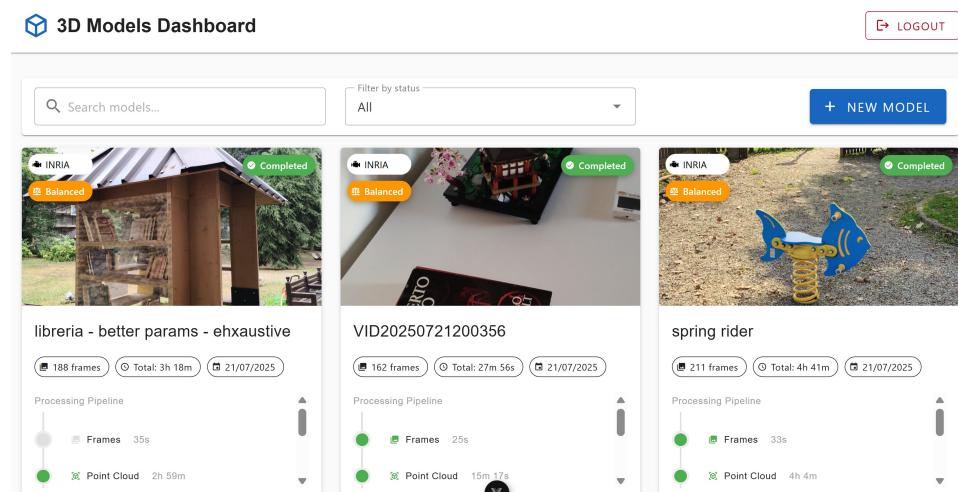


Figura 3.9: Dashboard principale con vista a griglia dei modelli, filtri di ricerca e indicatori di stato

Il sistema di **filtraggio e ricerca** permette agli utenti di navigare efficacemente anche in presenza di centinaia di modelli, con filtri per stato, ricerca testuale e paginazione server-side che garantisce performance costanti indipendentemente dal volume di dati.

3.7.1.2 Gestione del Workflow Utente

L’interfaccia implementa tre modalità principali di creazione modelli, ciascuna ottimizzata per specifici scenari d’uso:

Creazione di un nuovo modello

Il dialog di creazione nuovo modello guida l’utente attraverso un processo step-by-step che include:

- Upload diretto del video tramite drag-and-drop o selezione file
- Configurazione intuitiva dei parametri attraverso slider visuali e card informative
- Preview real-time della pipeline di elaborazione che verrà eseguita
- Sistema di rating visuale per comunicare l’impatto delle scelte su velocità, qualità e requisiti hardware

Fork di un modello preesistente

La funzionalità di fork permette di creare varianti di modelli esistenti riutilizzando fasi già completate. L’interfaccia visualizza chiaramente:

- Quali fasi verranno riutilizzate (indicate con icona di cache)
- Da quale punto ripartirà l’elaborazione
- Stima del risparmio temporale rispetto a una nuova elaborazione completa

Retry intelligente

Per modelli con elaborazione fallita, il sistema identifica automaticamente l’ultima fase completata con successo e propone di riprendere da quel punto, preservando il lavoro computazionale già svolto.

3.7.2 Sistema di notifiche real-time

Il frontend implementa un sofisticato sistema di notifiche basato su Web-Socket che mantiene gli utenti informati sull’avanzamento delle elaborazioni senza richiedere refresh manuali della pagina.

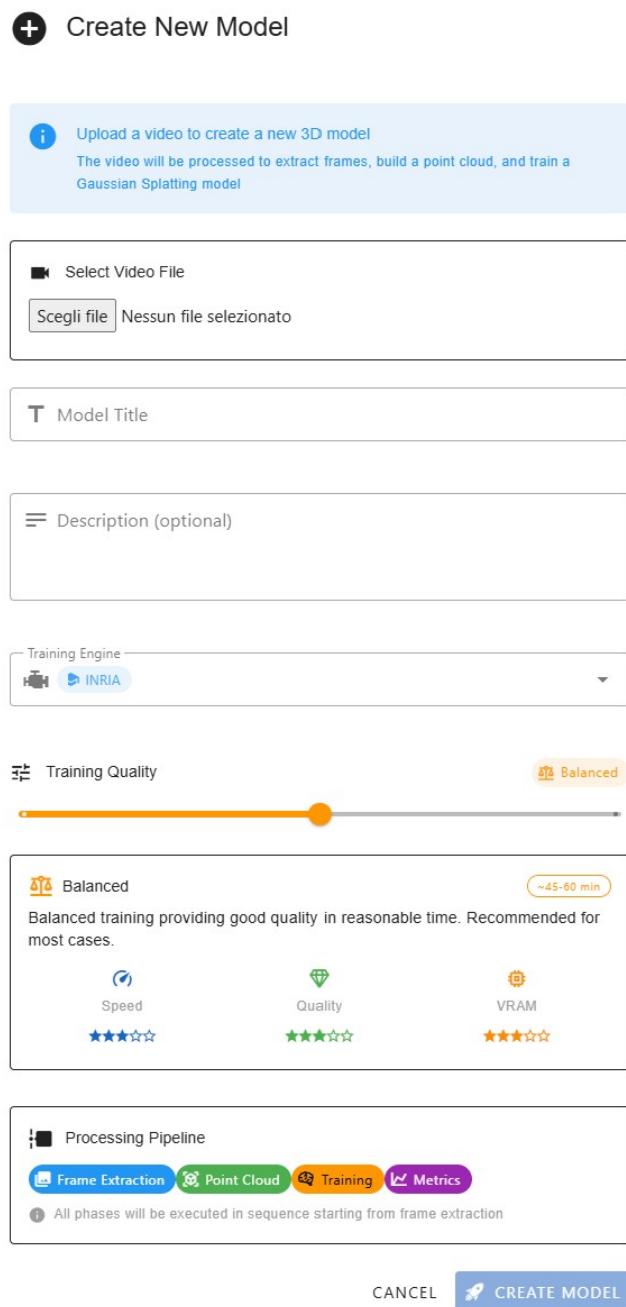


Figura 3.10: Dialog di creazione nuovo modello con selezione engine e configurazione qualità tramite slider interattivo

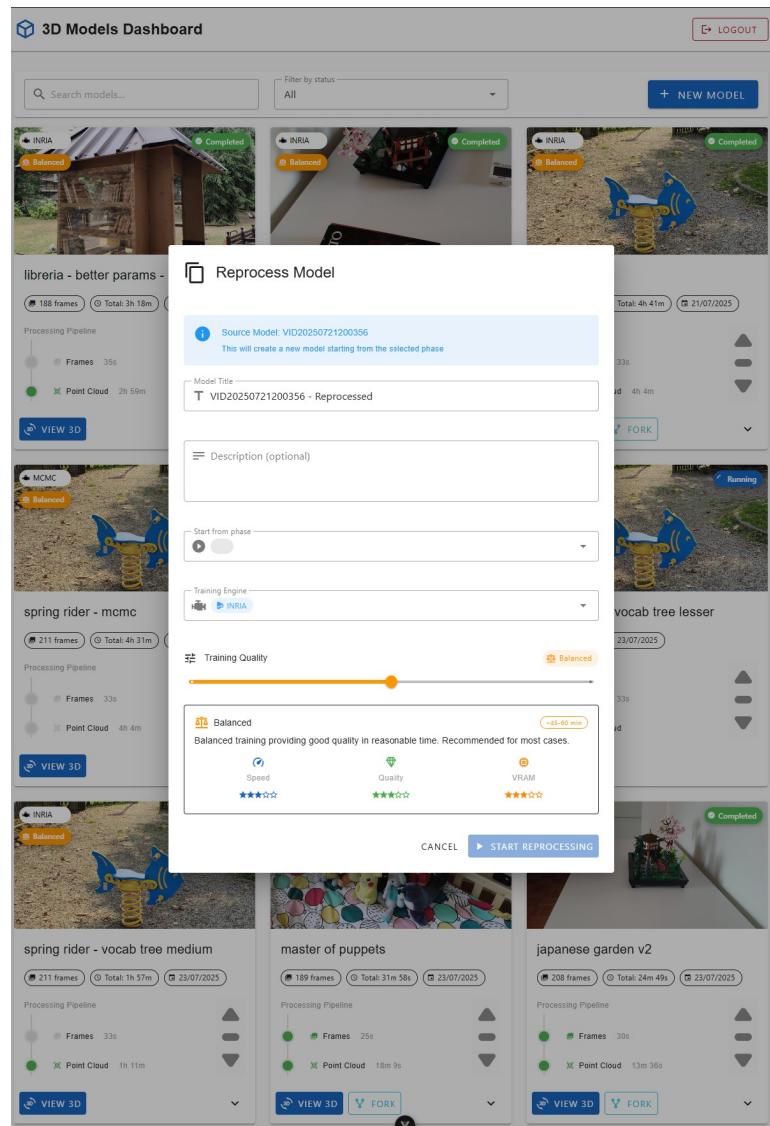


Figura 3.11: Dialog di reprocessing che mostra la selezione della fase di partenza e le fasi che verranno riutilizzate

3.7.2.1 Architettura delle Notifiche

Il sistema utilizza un approccio multi-livello per la gestione delle notifiche composto da:

- Toast Notifications: messaggi temporanei non invasivi che appaiono nell'angolo superiore destro per eventi chiave come completamento fasi o cambiamenti di stato. Ogni notifica è contestualizzata con icone e colori appropriati al tipo di evento.
- Notification Drawer: un pannello laterale accessibile tramite FAB (Floating Action Button) che mantiene uno storico delle notifiche recenti. Il drawer implementa:
 - Raggruppamento intelligente per modello
 - Indicatori di lettura/non lettura
 - Timestamp relativi ("2 ore fa") per immediata comprensione temporale
 - Azioni rapide per navigare al modello correlato

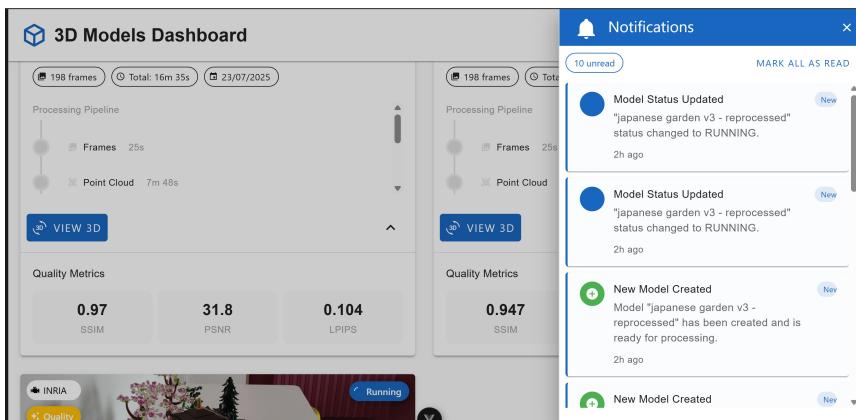


Figura 3.12: Sistema di notifiche con drawer laterale che mostra lo storico degli eventi e badge per notifiche non lette

- Live Updates La dashboard si aggiorna automaticamente quando arrivano notifiche relative a modelli visibili nella pagina corrente, eliminando la necessità di polling manuale e garantendo che le informazioni visualizzate siano sempre aggiornate.

3.7.2.2 Gestione stati di connessione

Il sistema monitora costantemente lo stato della connessione WebSocket, fornendo feedback visuale quando la connessione viene persa e implemen-

tando meccanismi di riconnessione automatica con backoff esponenziale per gestire interruzioni temporanee di rete.

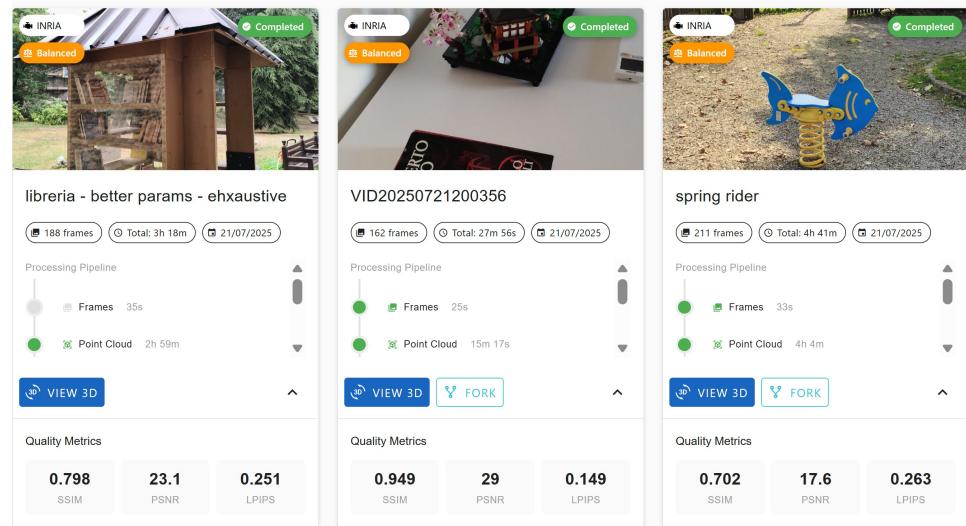


Figura 3.13: Vista espansa delle metriche di qualità (SSIM, PSNR, LPIPS) per confronto rapido tra modelli

3.7.3 Visualizzatore 3D Integrato

Il componente di visualizzazione 3D rappresenta il culmine dell’esperienza utente, permettendo l’esplorazione interattiva dei modelli generati direttamente nel browser.

3.7.3.1 Tecnologia di Rendering

Il viewer utilizza la libreria **GaussianSplats3D** basata su Three.js, specificamente ottimizzata per il rendering efficiente di primitive gaussiane. Questa scelta tecnologica garantisce:

- **Performance real-time**: Rendering a 60 FPS anche per modelli con centinaia di migliaia di splat
- **Compatibilità cross-platform**: Funzionamento su qualsiasi dispositivo con supporto WebGL
- **Interattività fluida**: Controlli camera intuitivi per rotazione, zoom e pan

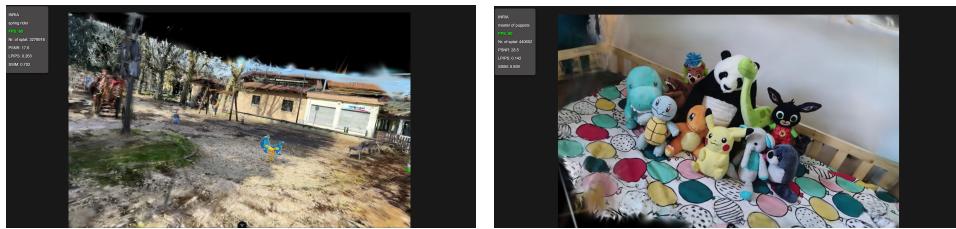
3.7.3.2 Interfaccia del Viewer

Il viewer 3D implementa un’interfaccia minimalista che massimizza l’area di visualizzazione mantenendo accessibili le informazioni essenziali:

Overlay Informativo

Un pannello semi-trasparente nell’angolo superiore sinistro visualizza:

- Statistiche di rendering (FPS, numero di splat)
- Metriche di qualità del modello
- Engine e parametri utilizzati per il training



(a) Spring rider - 327k splats

(b) Master of puppets - 440k splats

Figura 3.14: Visualizzatore 3D integrato con overlay informativo mostrante FPS, numero di splat e metriche di qualità

Controlli Camera

Il sistema recupera automaticamente la posizione camera ottimale dal dataset di training, garantendo che il modello sia presentato dalla prospettiva più favorevole al primo caricamento. Gli utenti possono poi esplorare liberamente utilizzando controlli standard di navigazione 3D.

Gestione formati

Il viewer gestisce trasparentemente la conversione tra formati, accettando sia file PLY che SPLAT ottimizzati e applicando le trasformazioni necessarie per la corretta visualizzazione, incluse conversioni di opacità specifiche per diversi algoritmi di training.

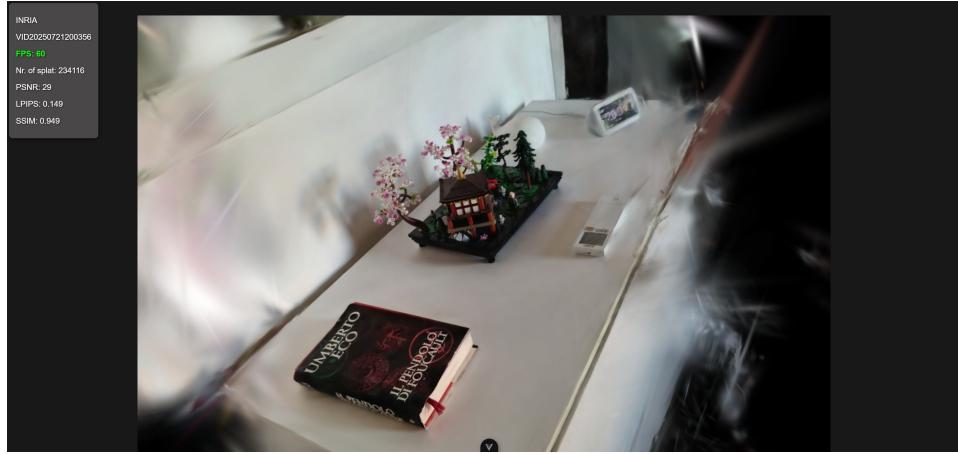


Figura 3.15: Visualizzazione di un modello complesso (Japanese garden) con rendering real-time a 60 FPS

3.7.4 Ottimizzazione performance e UX

Il frontend implementa diverse strategie per garantire un’esperienza utente fluida anche con dataset di grandi dimensioni:

Lazy Loading

Le thumbnail e i dati dei modelli vengono caricati progressivamente durante lo scroll, riducendo il tempo di caricamento iniziale della dashboard.

Caching intelligente

I modelli 3D visualizzati recentemente vengono mantenuti in cache del browser, permettendo switching istantaneo tra modelli già caricati.

Responsive Design

Layout adattivi garantiscono usabilità ottimale su dispositivi desktop, tablet e mobile, con breakpoint specificamente ottimizzati per i pattern di utilizzo tipici del sistema.

3.7.5 Integrazione con il Backend

Il frontend mantiene una separazione netta tra logica di presentazione e business logic attraverso l’utilizzo di **Pinia** per lo state management. Questo approccio centralizzato permette:

- **Sincronizzazione automatica** tra componenti che visualizzano gli stessi dati

- **Gestione consistente degli errori** con retry automatici per operazioni di rete fallite
- **Persistenza locale** di preferenze utente e stati di navigazione
- **Ottimizzazione delle chiamate API** attraverso batching e deduplicazione delle richieste

Il sistema di autenticazione JWT è integrato trasparentemente, con refresh automatico dei token e redirect intelligenti che preservano il contesto di navigazione dell'utente durante le operazioni di login/logout.

L'architettura complessiva del frontend riflette l'obiettivo primario del progetto: rendere la tecnologia del 3D Gaussian Splatting accessibile attraverso un'interfaccia che nasconde la complessità tecnica sottostante, guidando l'utente attraverso workflow ottimizzati che bilanciano automaticamente qualità dei risultati e risorse computazionali disponibili.

Capitolo 4

Analisi Sperimentale e Valutazione Prestazioni

4.1 Metodologia di Testing

4.1.1 Obiettivi della Valutazione Sperimentale

La valutazione sperimentale del sistema è stata strutturata per analizzare sia le prestazioni degli algoritmi di training implementati che l'efficacia dell'architettura distribuita proposta. Gli obiettivi principali dell'analisi includono:

- **Confronto qualitativo degli algoritmi:** Valutazione delle prestazioni relative tra Standard 3DGS, MCMC 3DGS e Taming 3DGS attraverso metriche standardizzate
- **Analisi dell'impatto dei parametri di qualità:** Studio dell'effetto dei livelli Fast, Balanced e Quality sulla qualità finale e sui tempi di elaborazione
- **Valutazione dell'architettura distribuita:** Analisi delle prestazioni del workflow end-to-end e identificazione di colli di bottiglia
- **Caratterizzazione utilizzo risorse:** Monitoring dettagliato di VRAM, GPU utilization e performance termiche durante l'elaborazione

4.1.2 Dataset Video e Caratteristiche di Input

Per garantire una valutazione comprensiva del sistema, è stato definito un set strutturato di scene reali che copre diversi livelli di complessità geometrica, ambientale e di ripresa. **Tutti i video sono stati acquisiti in risoluzione 4K (3840×2160) a 30 fps tramite la fotocamera di uno smartphone**

gamma medio-alta, cercando di garantire la massima qualità del dataset di partenza, prima delle successive fasi di preprocessing adattivo.

4.1.2.1 Categorizzazione delle Scene Video

S1 - Scene Geometriche Semplici

- **Spring Rider** (360°, Outdoor): Forme geometriche elementari con colori uniformi
 - Durata: 1 minuto e 10 secondi
 - Dimensione file: 345.569 KB
- **Lego Japanese Garden** (180°, Indoor): Oggetto con geometrie regolari ma dettagli architettonici
 - Durata: 1 minuto e 38 secondi
 - Dimensione file: 607.327 KB

S2 - Scene Multi-Oggetto

- **Master of Puppets** (180°, Indoor): Collezione di peluche con occlusioni multiple
 - Durata: 1 minuto e 34 secondi
 - Dimensione file: 579.644 KB
- **Multiple Objects** (180°, Indoor): Lego garden + oggetti accessori, occlusioni ridotte
 - Durata: 1 minuto e 21 secondi
 - Dimensione file: 498.843 KB
- **Wall Plants** (240°, Outdoor): Struttura + pianta, combinazione artificiale/naturale
 - Durata: 1 minuto e 8 secondi
 - Dimensione file: 421.296 KB

S3 - Scene ad Alta Complessità

- **Yellow Plant** (360°, Outdoor): Dettagli fini naturali, illuminazione variabile
 - Durata: 1 minuto e 40 secondi
 - Dimensione file: 566.772 KB

- **Lego Display Window** (360°, Indoor): Oggetti multipli in spazio confinato
 - Durata: 1 minuto e 18 secondi
 - Dimensione file: 479.009 KB
- **My Workstation** (180°, Indoor): Ambiente complesso (TV, PC, mobili)
 - Durata: 1 minuto e 14 secondi
 - Dimensione file: 457.335 KB

Tabella 4.1: Caratteristiche dei video del dataset utilizzato

Scena	Durata	Dimensione	Ambiente	Angolo	Tipologia
Spring Rider	00:01:10	345.569 KB	Outdoor	360°	Oggetto Singolo
Lego Japanese Garden	00:01:38	607.327 KB	Indoor	180°	Oggetto Singolo
Master of Puppets	00:01:34	579.644 KB	Indoor	180°	Multi-Oggetto
Multiple Objects	00:01:21	498.843 KB	Indoor	180°	Multi-Oggetto
Wall Plants	00:01:08	421.296 KB	Outdoor	240°	Misto
Yellow Plant	00:01:40	566.772 KB	Outdoor	360°	Naturale
Lego Display Window	00:01:18	479.009 KB	Indoor	360°	Multi-Oggetto
My Workstation	00:01:14	457.335 KB	Indoor	180°	Ambiente Complesso

4.1.3 Pipeline di Preprocessing e Adattamento Hardware

Il sistema implementa una **pipeline di preprocessing adattiva** che trasforma i video 4K di input in dataset ottimizzati per il training, applicando scaling dinamico basato sui vincoli hardware e sui livelli di qualità selezionati.

4.1.3.1 Depth Regularization per Algoritmo INRIA

Il sistema implementa automaticamente una fase di **depth regularization** esclusivamente per l'algoritmo INRIA (Standard 3DGS), utilizzando il modello Depth Anything V2 per generare depth maps di supporto al training.

Pipeline Depth Regularization

1. **Generazione Depth Maps:** Depth Anything V2 analizza ogni frame estratto generando mappe di profondità monocromatiche
2. **Scaling Parametri:** Generazione automatica di `depth_params.json` con scale factors specifici per la scena
3. **Training Depth-Aware:** Integrazione dei vincoli geometrici nel processo di ottimizzazione gaussiana

Motivazioni della Scelta Algoritmo-Specifica

- **INRIA necessita supporto geometrico:** L'algoritmo originale può beneficiare di prior geometrici per scene complesse
- **MCMC e Taming sono auto-sufficienti:** Questi algoritmi incorporano meccanismi interni di regolarizzazione spaziale
- **Bilanciamento computazionale:** Evitare overhead inutile per algoritmi già ottimizzati

Importante: Variabile Confondente nei Confronti

La depth regularization introduce una **variabile confondente** nei confronti diretti tra algoritmi. I risultati INRIA nei benchmark includeranno sempre depth regularization, rendendo i confronti non perfettamente equi. Questo riflette tuttavia l'utilizzo ottimale di ciascun algoritmo nelle condizioni reali di deployment.

4.1.3.2 Post-Processing: Conversione PLY→KSPLAT e Variante Taming

Il sistema implementa la **conversione PLY→KSPLAT** per tutti gli algoritmi di training. Tutti gli algoritmi generano output in formato PLY standard, che viene convertito nel formato KSPLAT per ottenere file significativamente più compatti e ottimizzati per la visualizzazione web.

Variante Specifica Taming: Durante la conversione standard PLY→KSPLAT, l'algoritmo Taming richiede un'ulteriore trasformazione dei valori di opacity. Taming genera valori di opacity in formato sigmoid che necessitano di conversione per compatibilità ottimale con i viewer 3D.

Trasformazione Implementata: Per l'output Taming, viene applicata automaticamente una trasformazione inverse sigmoid (logit) durante la fase di conversione:

Listing 4.1: Conversione opacity per Taming

```
logit_opacity = log(opacity / (1.0 - opacity))
```

Questa sotto-conversione ha impatto computazionale trascurabile ed è trasparente all'utente, garantendo formato uniforme e interoperabilità con tutti i viewer utilizzati.

4.1.4 Ambiente di Deployment e Configurazione Hardware

4.1.4.1 Setup Hardware di Test

I test sono stati eseguiti su configurazione hardware che riflette un ambiente di deployment medio piuttosto che una configurazione ideale da laboratorio:

Configurazione GPU

- **GPU:** NVIDIA GeForce RTX 4080
- **VRAM Disponibile:** 16 GB – **Significativamente inferiore** ai 24GB raccomandati nella documentazione del 3D Gaussian Splatting originale
- **Architettura:** CUDA Compute Capability 8.9 per Ada Lovelace

Sistema Host

- **OS:** Windows 11 Pro
- **Ambiente:** WSL2 (Windows Subsystem for Linux) con Ubuntu 22.04
- **Container Runtime:** Docker Desktop con Docker Compose
- **CPU:** AMD 7800X3D 4.2Ghz 96 Mo L3
- **RAM Sistema:** 32 GB DDR5 limitati a 20 GB per i container con memory swapping a 32 GB
- **Storage:** SSD NVMe per minimizzare latenze I/O

4.1.4.2 Sistema di Monitoring Hardware Real-Time

Per caratterizzare accuratamente il comportamento degli algoritmi sotto vincoli hardware, è stato implementato un **monitoring system** che raccoglie metriche GPU durante l'intero ciclo di training.

Metriche Hardware Monitorate

Listing 4.2: Esempio statistiche hardware raccolte

```
{  
    "peak_vram_mb": 14535,           // Picco utilizzo  
    "vram"                      // VRAM
```

```

    "avg_vram_mb": 12106.3,           // Utilizzo medio
    VRAM
    "min_vram_mb": 2242,             // Baseline VRAM (
        modello + overhead)
    "avg_gpu_utilization": 90,       // Utilizzo GPU
        medio (%)
    "max_temperature_c": 79,         // Temperatura
        massima raggiunta
    "monitoring_duration_s": 1362.4, // Durata totale
        training
    "total_samples": 449            // Campioni raccolti
        (ogni 3s)
}

```

Implementazione Tecnica

- **Sampling Rate:** 3 secondi per bilanciare granularità e overhead
- **Host-Level Monitoring:** Raccolta diretta da GPU driver, indipendente dai container
- **Persistenza Automatica:** Integrazione nei metadati del workflow per analisi retrospettive

4.1.5 Metodologia di Valutazione

4.1.5.1 Metriche Quantitative Automatizzate

Il sistema implementa automaticamente il calcolo delle tre metriche standard per la valutazione della qualità di rendering:

- **PSNR (Peak Signal-to-Noise Ratio)**
 - Range tipico: 20–40 dB
 - Valori superiori indicano minore rumore e maggiore fedeltà
 - *Osservazione:* valori >25 dB sono considerati buoni, ma soglie più basse (22–24 dB) possono comunque essere accettabili in contesti reali e con hardware limitato
- **SSIM (Structural Similarity Index)**
 - Range: 0–1
 - Valori elevati indicano maggiore similarità strutturale
 - Soglia di accettabilità: >0.8 (con valori tra 0.75 e 0.8 ancora ritenuti adeguati)
- **LPIPS (Learned Perceptual Image Patch Similarity)**

- Range: 0–1 (valori inferiori indicano maggiore similarità percentuale)
- Obiettivo qualitativo: <0.2 in condizioni ideali
- *Nota:* valori fino a 0.25 possono essere ritenuti accettabili per scene complesse o condizioni hardware non ottimali

4.2 Benchmark e Risultati Sperimentali

4.2.1 Benchmark 1: Confronto Algoritmi e Scalabilità Hardware

Il primo benchmark ha l’obiettivo di confrontare le tre varianti di 3D Gaussian Splatting considerate in questa tesi — INRIA (implementazione standard con *depth regularization*), MCMC 3DGS e Taming 3DGS — valutandone prestazioni, scalabilità e qualità visiva. Il test è stato condotto su tre scene di riferimento: **S1** *Spring Rider*, caratterizzata da geometria semplice e ripresa a 360° in ambiente outdoor; **S2** *Tomatoes*, con dettagli organici fini, ideale per analizzare l’impatto dell’aumento di risoluzione; e **S3** *My Workstation*, una scena indoor complessa con oggetti multipli.

L’analisi mira a:

- identificare punti di forza e debolezza di ciascun approccio;
- misurare la qualità di rendering mediante metriche quantitative (PSNR, SSIM, LPIPS);
- verificare la stabilità termica e l’utilizzo di VRAM in scenari reali;
- valutare la scalabilità con l’aumento della risoluzione fino al *breaking point*.

Le sottosezioni seguenti presentano:

1. una **sintesi comparativa** per avere a colpo d’occhio le differenze principali;
2. i **risultati quantitativi** sulle scene di test;
3. un **confronto qualitativo visuale** supportato da esempi;
4. **considerazioni operative** e raccomandazioni per l’uso.

Algoritmi:

INRIA (3DGS standard + depth regularization), MCMC 3DGS, Taming 3DGS

Scene di test:

- **S1** – Spring Rider: geometria semplice, outdoor, 360°
- **S2** – Tomatoes: dettagli organici fini, ideale per test di upscaling
- **S3** – My Workstation: indoor complesso, oggetti multipli

1. Sintesi comparativa

Tabella 4.2: Sintesi comparativa algoritmi

Criterio	INRIA	MCMC	Taming
Qualità media	●	✓	✓
Velocità	✓	✗	✓
Scalabilità risoluzione	✗	●	●
Stabilità termica	●	✓	✓
Predicibilità VRAM	✗	●	●
Artefatti	●	✗	✓

Legenda: **✓** = punto di forza, **●** = adeguato, **✗** = debolezza.

2. Risultati quantitativi

Metriche medie (**S1 + S3**, profilo *Balanced*)

Tabella 4.3: Metriche medie su scene rappresentative

Algoritmo	PSNR↑	SSIM↑	LPIPS↓	Tempo	Peak VRAM	Temp max
INRIA*	23.4	0.835	0.209	23m	12.7 GB	77.5°C
MCMC	27.6	0.909	0.151	42m	13.6 GB	70.5°C
Taming	28.3	0.922	0.132	20m	13.8 GB	69°C

Qualità alla massima risoluzione stabile (**S2**)

MCMC e Taming raggiungono 1080p nella maggior parte dei casi, ma possono manifestare degrado prestazionale o crash per spike di VRAM. INRIA si ferma stabilmente a 720p.

Tabella 4.4: Qualità alla massima risoluzione stabile

Algoritmo	Res Max	PSNR	SSIM	LPIPS
INRIA*	720p	21.5	0.724	0.249
MCMC	1080p	23.9	0.751	0.247
Taming	1080p	24.0	0.752	0.248

Andamento utilizzo VRAM per risoluzione

L’analisi dell’andamento temporale dell’utilizzo di memoria video evidenzia differenze significative tra algoritmi e risoluzioni. I punti contrassegnati come **OOM** indicano un arresto o degrado grave dovuto al superamento del limite di VRAM disponibile.

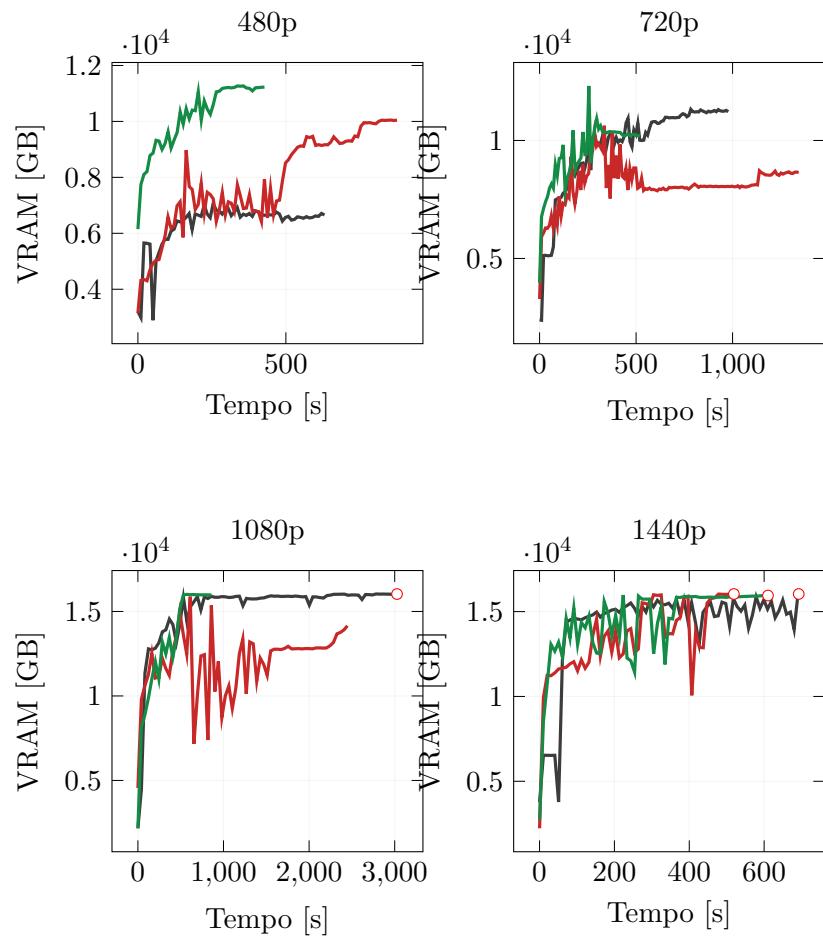


Figura 4.1: Profilo di utilizzo VRAM nel tempo su **S2 Tomatoes**, per risoluzioni crescenti e algoritmi (INRIA/MCMC/Taming). I marker rossi indicano spike di VRAM associati a degrado prestazionale o arresto del training (non sempre OOM “hard”).



Figura 4.2: Confronto visivo Spring Rider (crop centrale) – INRIA / MCMC / Taming



Figura 4.3: Confronto visivo My workstation (crop centrale) – INRIA / MCMC / Taming

3. Confronto qualitativo visuale

Tabella comparativa qualitativa

Tabella 4.5: Sintesi qualitativa visuale (S1 Spring Rider e S3 My Workstation)

Criterio	INRIA	MCMC	Taming
Nitidezza soggetto / oggetti vicini	●	✓	✓
Fedeltà cromatica	●	●	✓
Dettagli sfondo	●	✓	✓
Texture suolo / superfici	✗	●	✓
Gestione periferia / transizioni	✗	●	✓
Artefatti visibili	●	✗	✓

Legenda: ✓ = punto di forza, ● = adeguato, ✗ = debolezza.

4. Conclusioni operative

- **Taming 3DGS** – Miglior compromesso tra qualità, velocità e stabilità: indicato per deploy web e produzione.
- **MCMC 3DGS** – Potenziale qualitativo elevato, ma meno prevedibile in risorse e più lento: indicato per ricerca e scenari controllati.
- **INRIA** – Baseline veloce e leggera: utile per prototipazione rapida o scene semplici.

5. Appendice visiva

Artefatti MCMC



Figura 4.4: Trasparenze anomale nei rendering MCMC

Esempio upscaling – S2 Tomatoes



Figura 4.5: Upscaling Tomatoes: 480p → 720p → 1080p (MCMC)

4.2.2 Benchmark 2: Confronto tra livelli di qualità (Taming 3DGS)

Obiettivo

Valutare il trade-off tra qualità visiva, tempi di elaborazione e utilizzo delle risorse per i tre profili **Fast**, **Balanced** e **Quality** dell'algoritmo Taming

3DGS.

Metodologia

I test sono stati condotti esclusivamente a **720p** a causa dei vincoli di VRAM, variando soltanto due parametri:

- **Iterations:** 20 100 (Fast) → 25 500 (Balanced) → 30 000 (Quality)
- **Cams:** 10 → 20 → 30

Tutti i parametri di densificazione (`densify_grad_threshold`, `densification_interval`, `densify_until_iter`) sono rimasti invariati per garantire stabilità e riproducibilità.

Scene di test

- **Lego Japanese Garden** – geometrie regolari e dettagli architettonici.
- **Yellow Plant** – scena naturale con texture organiche complesse.

Nota metodologica

La scelta di una strategia conservativa (variazione di soli due parametri) deriva dall'osservazione che i parametri di densificazione possono avere effetti non lineari, scena-dipendenti e talvolta degradanti, aumentando il rischio di instabilità numerica.

Tabella 4.6: Lego Japanese Garden @ 720p – confronto livelli di qualità (Taming)

Metrica	Fast	Balanced	Quality
PSNR ↑	31.3	30.5	31.4
SSIM ↑	0.969	0.967	0.969
LPIPS ↓	0.102	0.103	0.103
Gaussiane	359K	370K	355K
Tempo	7m 19s	8m 57s	11m 22s
Avg VRAM (GB)	7.61	9.16	8.26
Peak VRAM (GB)	9.47	12.20	11.74
Avg GPU (%)	78.7	79.8	78.1
Max Temp (°C)	72	73	73

Tabella 4.7: Yellow Plant @ 720p – confronto livelli di qualità (Taming)

Metrica	Fast	Balanced	Quality
PSNR ↑	24.1	24.0	24.1
SSIM ↑	0.831	0.831	0.834
LPIPS ↓	0.146	0.143	0.142
Gaussiane	1.94M	1.95M	1.96M
Tempo	16m 48s	20m 24s	23m 50s
Avg VRAM (GB)	9.70	11.50	11.43
Peak VRAM (GB)	11.84	13.67	9.92
Avg GPU (%)	84.3	89.2	85.6
Max Temp (°C)	69	69	70

Lettura dei risultati

Le differenze qualitative tra i tre livelli sono contenute: in *Lego*, Fast e Quality raggiungono entrambi un PSNR di circa 31.3–31.4, mentre Balanced risulta leggermente inferiore. In *Yellow Plant*, LPIPS migliora marginalmente in Quality ma l’incremento è modesto. Il tempo di elaborazione cresce in modo significativo: +55% per Quality rispetto a Fast in *Lego* e +42% in *Yellow Plant*. La VRAM non mostra un andamento monotono e le temperature restano stabili (69–73 °C).

Raccomandazione operativa

- **Fast** – Profilo consigliato per deployment veloci: tempi ridotti del 35–50% rispetto a Quality, con differenze metriche trascurabili.
- **Balanced** – Poco utile con i settaggi attuali; può diventare rilevante solo se calibrato diversamente (es. densificazione intermedia).
- **Quality** – Da usare solo quando il tempo non è un vincolo e si desidera massimizzare ogni punto percentuale di qualità.



Figura 4.6: Confronto visivo dei livelli di qualità – scena *Lego Japanese Garden* (crop centrale).



(a) Fast

(b) Balanced

(c) Quality

Figura 4.7: Confronto visivo dei livelli di qualità – scena *Yellow Plant* (crop centrale).

4.2.3 Benchmark 3: Latency Budget End-to-End (Balanced, 30k iterazioni)

Setup

Tutti i test sono stati eseguiti con tutti gli algoritmi, preset **Balanced**, su una RTX 4080 16GB (WSL2). Il numero di iterazioni di training è fisso a **30 000** per tutte le scene. Le metriche PSNR/SSIM/LPIPS sono calcolate in un job asincrono e non influiscono sul *time-to-preview*.

Tabella 4.8: Riepilogo end-to-end per scena - INRIA

Scena	Tempo Totale (s)	#Frame	Punti COLMAP	Gauss. finali
S1 — <i>Master of Puppets</i>	917	189	61 014	324 662
S2 — <i>Lego Display Window</i>	866	234	30 304	470 058
S3 — <i>Yellow Plant</i>	1416	202	120 521	1 409 058

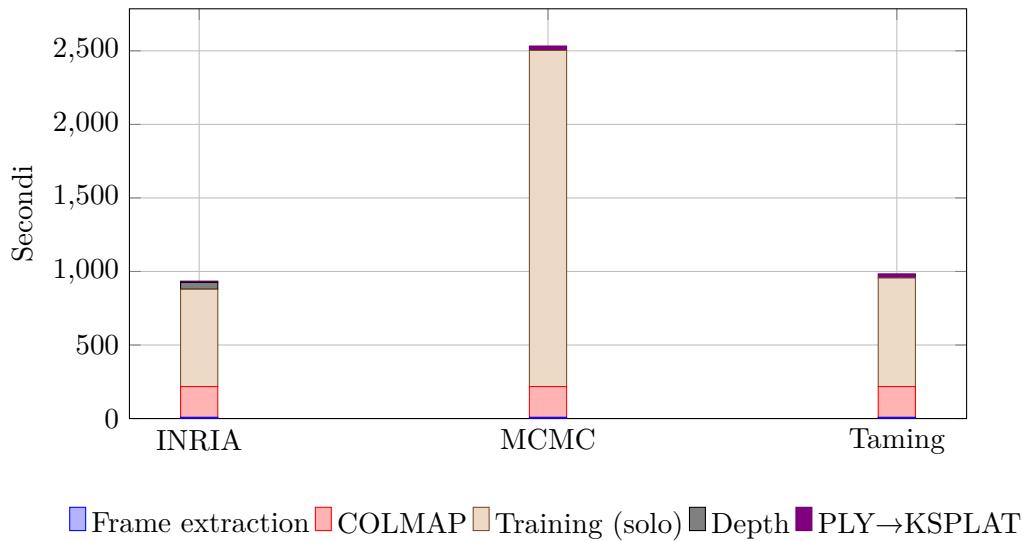


Figura 4.8: Latency budget per algoritmo — *S1: Master of Puppets* (Balanced, 30k).

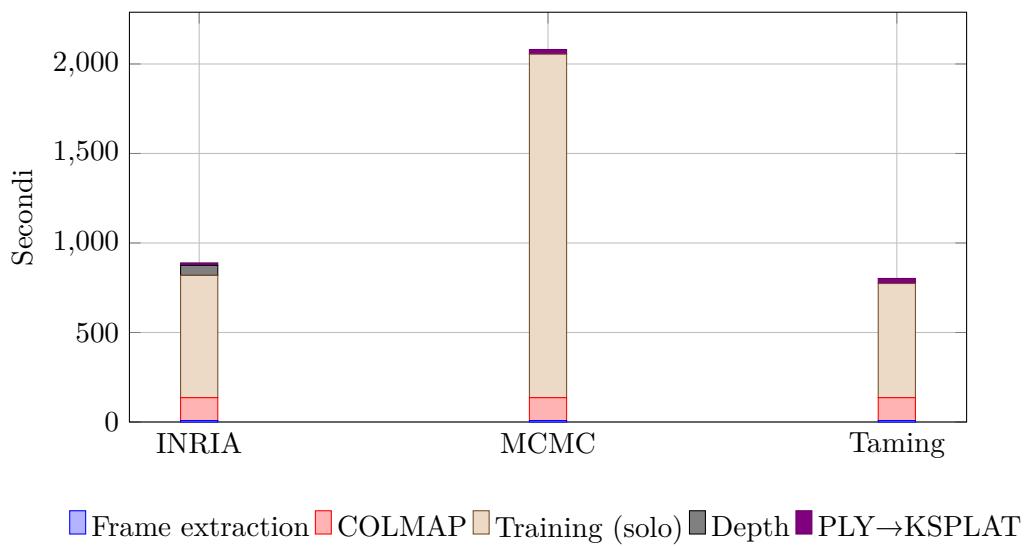


Figura 4.9: Latency budget per algoritmo — *S2: Lego Display Window* (Balanced, 30k).

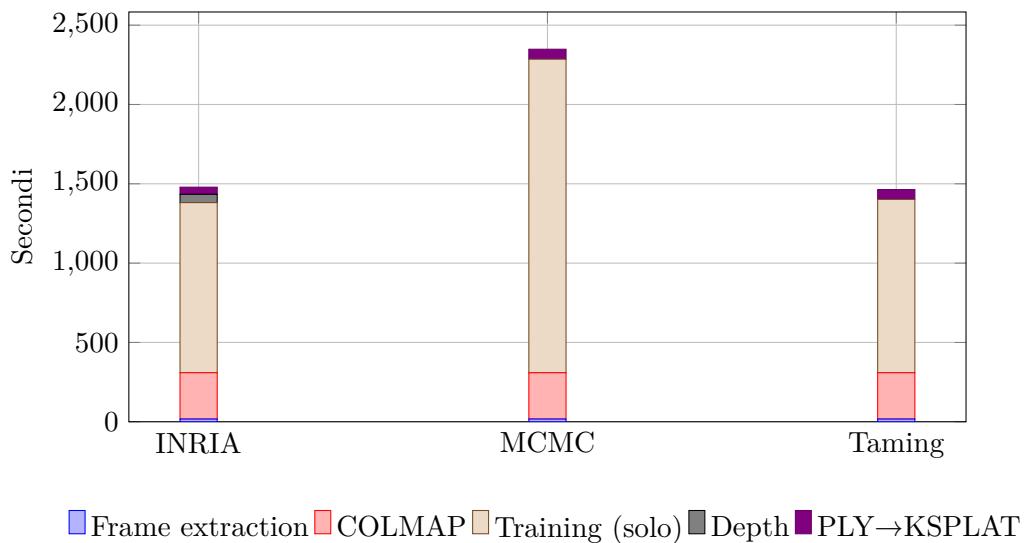


Figura 4.10: Latency budget per algoritmo — *S3: Yellow Plant* (Balanced, 30k).

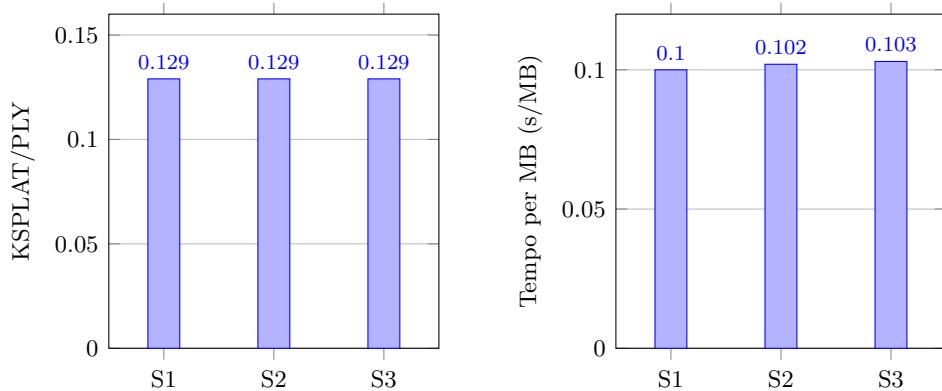


Figura 4.11: Efficienza conversione PLY→KSPLAT: rapporto di compressione e costo computazionale.

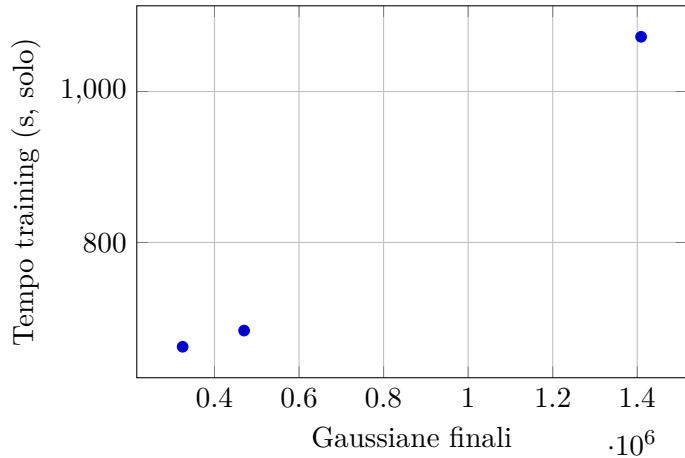


Figura 4.12: Durata del training (parte ottimizzazione) in funzione delle gaussiane finali.

Osservazioni

Dall’analisi del *latency budget* (Tab. ?? e Fig. ??) emerge come:

- La fase di **training** sia nettamente la più onerosa in termini di tempo, assorbendo dal 72% al 79% del *time-to-preview*, seguita dalla **ricostruzione COLMAP** (15–23%).
- La **depth regularization** incide in media per circa il 22–25% del tempo di training, con valori piuttosto stabili tra le scene.
- La **frame extraction** e la conversione **PLY→KSPLAT** hanno un peso marginale, inferiore al 4% complessivo, e costi computazionali per MB molto simili tra le scene (Fig. ??).
- Il rapporto di compressione PLY/KSPLAT è costante (0.129) grazie a una pipeline di conversione stabile e indipendente dalla complessità della scena.
- Come mostra Fig. 4.12, il tempo di training cresce con il numero di gaussiane finali, ma non in modo perfettamente lineare: scene con distribuzioni spaziali più complesse o ricche di dettagli (es. *Yellow Plant*) possono richiedere tempi superiori a parità di incremento percentuale di gaussiane.

Questi dati, pur riferendosi al solo caso *INRIA Balanced* con 30 000 iterazioni, costituiscono una base per confronti futuri con altre configurazioni o algoritmi (MCMC, Taming) mantenendo costante la metodologia di misura.

Capitolo 5

Criticità, limitazioni e prospettive di miglioramento

5.1 Limitazioni e criticità

5.1.1 Acquisizione dei dati

5.1.1.1 Copertura e qualità delle riprese.

La fase di acquisizione video si è rivelata complessa, soprattutto nel garantire una copertura completa delle varie angolazioni di un oggetto o di una scena. In diversi casi è stato necessario ripetere le riprese a causa della mancanza di dettagli in certe aree, specialmente per oggetti di grandi dimensioni (es. tetto della libreria nel castello), con conseguente degrado della qualità visiva complessiva del modello. Tale criticità è rilevante anche in considerazione della durata delle fasi di training, che può raggiungere le 1–2 ore per singolo modello.

Possibili miglioramenti:

- Realizzazione di tutorial video e guide visive nella *presentation layer*, in grado di illustrare passo-passo come effettuare la ripresa ottimale.
- Implementazione di un sistema di supporto visivo per il controllo della copertura, perseguitibile secondo due approcci:
 1. **In tempo reale:** sviluppo di un'app di acquisizione dedicata, capace di fornire *overlay* o *heatmap* live che evidenzino le zone già riprese e quelle mancanti.
 2. **Post-upload:** analisi rapida del video caricato, con generazione di una mappa di copertura e suggerimenti per effettuare eventuali riprese integrative da specifiche angolazioni.

- Utilizzo di tecniche di intelligenza artificiale per stimare e generare viste mancanti, mediante *novel view synthesis* o *inpainting*¹, al fine di coprire angolazioni non acquisite.

5.1.1.2 Estrazione dei frame

Determinare una metodologia di estrazione dei frame ottimale si è rivelato non banale. L'approccio iniziale, basato su una soluzione custom, ha prodotto risultati incostanti, rendendo necessario l'impiego di una libreria preesistente. Quest'ultima ha fornito metodologie più robuste, ma richiedenti comunque una parametrizzazione accurata per ottenere un bilanciamento adeguato tra qualità visiva e numero di frame.

Possibili miglioramenti:

- Adozione di algoritmi di estrazione adattiva, che selezionino i frame in base alla variazione di contenuto (*content variation*) piuttosto che a intervalli temporali fissi, utilizzando metriche come l'indice di similarità strutturale (SSIM) o l'analisi del movimento.
- Implementazione di un sistema di pre-filtraggio automatico dei frame estratti, in grado di scartare quelli mossi, sfocati o sotto/sovraesposti, al fine di migliorare la qualità dei dati in ingresso al processo di training.
- Sperimentazione di tecniche di intelligenza artificiale per l'estrazione *content-aware*, capaci di identificare in modo automatico i frame più rappresentativi della scena in funzione della ricostruzione 3D, eventualmente addestrando modelli leggeri su dataset specifici del dominio.

5.1.2 Limitazioni hardware e scalabilità

5.1.2.1 Risorse limitate

L'hardware a disposizione non era ottimale: la GPU NVIDIA RTX 4080 con 16 GB di VRAM si è rivelata inferiore ai 24 GB consigliati per le configurazioni di riferimento. Inoltre, la disponibilità di una sola macchina ha impedito di verificare la scalabilità del sistema in configurazioni multi-GPU o multi-nodo.

¹Nel contesto di questa tesi, l'*inpainting* può essere applicato in due modi: (1) in 2D, completando immagini mancanti o parziali per generare viste non acquisite prima del training; (2) in 3D, ricostruendo porzioni mancanti di un modello tridimensionale (mesh o nuvola di punti) sulla base del contesto geometrico e cromatico circostante, approccio più complesso e considerato come possibile sviluppo futuro.

Possibili miglioramenti:

- Potenziamento diretto delle risorse, mediante l'acquisto di macchine dotate di GPU di fascia alta (es. RTX 6000 Ada, NVIDIA A100, H100) con maggiore VRAM, oppure configurazioni multi-GPU su singolo nodo.
- Utilizzo di servizi di calcolo su cloud (es. AWS EC2 P3/P4, Google Cloud, Azure NV series), che permettano di scalare verticalmente (GPU più potenti) e orizzontalmente (più nodi), con pagamento a consumo.
- Adozione di soluzioni ibride o ottimizzazioni software², ad esempio:
 - impiego di tecniche di riduzione del consumo di memoria;
 - distribuzione del carico di lavoro su più GPU meno potenti ma interconnesse;
 - configurazione di una piccola render farm interna.

5.1.2.2 Impatto sulla parametrizzazione

Le limitazioni hardware hanno influenzato direttamente la definizione dei livelli di qualità, imponendo range di valori ristretti. Questo ha ridotto la possibilità di sperimentare configurazioni più spinte, soprattutto in modalità “Qualità”, e ha reso necessario calibrare i parametri tenendo conto della capacità massima della GPU.

Possibili miglioramenti:

- Potenziamento delle risorse hardware, come descritto nella sottosezione precedente, per ampliare i margini di parametrizzazione.
- Ottimizzazione degli algoritmi di training per ridurre l'uso di memoria e migliorare l'efficienza computazionale, ad esempio mediante gestione dinamica dei batch e compressione temporanea delle strutture dati.
-
- Introduzione di una parametrizzazione adattiva, in cui il sistema regoli automaticamente i parametri di qualità (es. risoluzione, numero di gaussiane, ordine degli armonici sferici) in funzione delle risorse

²*Mixed precision training*: utilizzo di formati numerici a precisione ridotta (FP16 o BF16) per ridurre il consumo di memoria e accelerare i calcoli. *Gradient checkpointing*: tecnica che memorizza solo una parte delle attivazioni intermedie durante il training e ricalcola le altre quando servono, riducendo l'uso di VRAM. *Distributed training*: addestramento parallelo su più GPU o nodi collegati in rete veloce, ad esempio tramite NCCL o PyTorch Distributed Data Parallel (DDP).

disponibili, minimizzando il rischio di errori *Out-Of-Memory*. Nel sistema sviluppato è già stata implementata una forma di parametrizzazione adattiva, ma i risultati ottenuti non si sono rivelati pienamente soddisfacenti, indicando la necessità di ulteriori ottimizzazioni per migliorare l'efficacia e la stabilità di questo approccio.

5.1.3 Parametrizzazione e ottimizzazione

5.1.3.1 Complessità nella scelta dei parametri

L'individuazione della giusta parametrizzazione per processi come l'estrazione dei frame e il training della *point cloud* è risultata complessa, sia per la natura degli algoritmi coinvolti sia per l'assenza di valori ottimali universalmente validi. Ogni scena richiede una calibrazione specifica, aumentando i tempi complessivi di configurazione.

Una criticità particolare riguarda gli algoritmi *Taming* e *MCMC*, che richiedono la definizione di parametri come `cap_max` e `budget`, rispettivamente il numero di gaussiane da raggiungere e il limite massimo consentito³. Nel sistema sviluppato tali valori sono stati stimati proporzionalmente al numero di punti della nuvola iniziale passata in input, ma questa strategia non sempre si è dimostrata ottimale.

Ulteriore complessità è emersa nella definizione di livelli di qualità predefiniti, pensati per offrire all'utente un compromesso tra tempo di esecuzione e qualità del risultato. Se da un lato impostazioni più basse hanno garantito tempi di esecuzione ridotti, dall'altro non sempre si è osservato un incremento visivo o metrico significativo passando a livelli più alti. Questo fenomeno è legato non solo a limiti hardware, ma anche a fattori algoritmici: impostazioni troppo spinte possono condurre a fenomeni di *overfitting* o a una saturazione dei miglioramenti, senza un effettivo beneficio per la qualità percepita.

Possibili miglioramenti:

- Sviluppo di algoritmi di *scene analysis* in grado di determinare automaticamente le caratteristiche principali della scena (complessità geometrica, livello di dettaglio, dimensioni, presenza di superfici uniformi o ripetitive).
- Implementazione di una parametrizzazione *scene-aware*, che adatti automaticamente valori come la risoluzione, il numero di gaussiane ini-

³`cap_max` indica il numero di gaussiane target da ottenere a fine training, mentre `budget` definisce il massimo numero di gaussiane consentito durante l'esecuzione. Valori troppo bassi possono compromettere il dettaglio, mentre valori troppo alti possono portare a eccessivo consumo di memoria o overfitting.

ziali, l'ordine degli armonici sferici, nonché parametri specifici come `cap_max` e `budget`, in base ai risultati dell'analisi preliminare.

- Creazione di un sistema di raccomandazioni per l'utente, che suggerisca set di parametri o livelli di qualità preconfigurati in funzione della tipologia di scena, minimizzando il rischio di errori, overfitting e tempi di setup eccessivi.

5.1.4 Stabilità e maturità degli algoritmi

5.1.4.1 Algoritmi in fase sperimentale

Gli algoritmi di training utilizzati sono soggetti a frequenti aggiornamenti e presentano una documentazione limitata, spesso ristretta alle sole pagine GitHub dei progetti. Parametrizzazioni errate o incompatibili possono causare crash durante il training, richiedendo una costante attività di verifica.

Possibili miglioramenti:

- Contribuire attivamente alla documentazione, ad esempio includendo in questa tesi e nella piattaforma sperimentazioni empiriche e configurazioni validate.
- Partecipare allo sviluppo dei progetti open source, segnalando bug, proponendo migliorie o inviando *pull request* con ottimizzazioni e correzioni.
- Integrare nella piattaforma una documentazione interna e continuamente aggiornata, così che l'utente disponga di istruzioni e parametri testati anche in assenza di risorse ufficiali complete.
- Implementare un sistema di *workflow versioning*, registrando la versione dell'algoritmo e la configurazione usata per ogni training, per garantire la riproducibilità dei risultati e semplificare il ritorno a versioni stabili in caso di regressioni.

5.1.4.2 Instabilità con parametri elevati

All'aumentare dei parametri legati alla qualità (ad esempio la risoluzione), si sono verificati casi di degrado significativo del training, fino al blocco del processo o a picchi di utilizzo di memoria (*memory spike*) che hanno portato a errori *Out-Of-Memory (OOM)*.

Possibili miglioramenti:

- Implementazione di un sistema di rilevamento dei blocchi (*watchdog*), capace di monitorare lo stato del training e interromperlo automaticamente.

camente in assenza di progressi per un periodo di tempo definito, con eventuale riavvio in modalità a parametri ridotti.

- Introduzione di meccanismi di stima preventiva dell'uso di memoria in base alla configurazione scelta, per avvisare l'utente o bloccare l'esecuzione in caso di rischio elevato di errori *OOM*.
- Adozione di un approccio conservativo nella scelta di risoluzione e parametrizzazione quando si lavora su hardware con risorse limitate, come misura preventiva contro picchi di memoria.
- Registrazione e analisi dei log di utilizzo della memoria e dei tempi di esecuzione, così da identificare combinazioni di parametri particolarmente critiche.

5.1.5 Output e compatibilità

5.1.5.1 Dimensioni e formati dei modelli

I modelli 3D generati producono *point cloud* in formato .ply, arricchite con attributi specifici delle gaussiane (opacità, covarianza, armonici sferici fino al grado 3). Questi file, troppo pesanti per un utilizzo web diretto, vengono convertiti in formato binario compatto .ksplat, che consente di mantenere in modo parametrico un ordine SH variabile, dal solo termine costante (grado 0) fino al massimo generato (grado 3). In questo modo è possibile scegliere il compromesso ottimale tra qualità visiva e dimensioni finali del modello, riducendo il carico computazionale nei *viewer* web quando si utilizzano ordini inferiori.

L'algoritmo *Taming* produce valori di opacità non direttamente compatibili con i *viewer*, richiedendo una trasformazione (inverse sigmoid) in fase di conversione.

Nel contesto web, la dimensione dei file .ksplat ottenuti è risultata generalmente più che accettabile sia per le prestazioni di caricamento lato client, sia per i costi di archiviazione e trasferimento su servizi cloud come Amazon S3, anche in scenari di distribuzione su larga scala.

Possibili miglioramenti:

- Valutare l'adozione di tecniche di compressione per nuvole di punti, come *Draco*⁴, che però risulta compatibile nativamente solo con dati geometrici e cromatici (XYZ + colore). Per gli altri attributi caratteristici delle gaussiane (opacità, covarianza, armonici sferici) sarebbe necessaria un'estensione del formato o una codifica ibrida.

⁴<https://google.github.io/draco/>

- Implementare strategie di *progressive loading*, caricando inizialmente le gaussiane più rilevanti e aggiungendo i dettagli in un secondo momento.
- Ottimizzare la conversione in `.ksplat` con modalità *lossy* per l'uso web, eliminando gaussiane irrilevanti (es. con opacità molto bassa o dimensioni trascurabili).
- Gestire i file nel cloud con politiche di *tiered storage*, conservando in storage rapido solo i modelli di uso frequente e spostando gli altri su archiviazioni più economiche.
- Consentire la selezione e il download del modello originale in formato completo (es. `.ply` con SH fino al grado 3) per usi offline o visualizzazione in un *viewer* desktop, preservando la qualità massima senza vincoli delle ottimizzazioni web.

5.1.5.2 Limiti dei viewer web

Gli algoritmi permettono di generare gaussiane con armonici sferici (SH) fino all'ordine 3⁵, il che consente di rappresentare in modo più accurato le variazioni angolari ad alta frequenza e migliorare la resa visiva complessiva, soprattutto nei dettagli più fini e nei riflessi complessi.

La conversione in formato binario `.ksplat` consente di mantenere in modo parametrico un ordine SH variabile, fino al massimo generato (SH3), scegliendo così il compromesso tra qualità visiva e dimensioni finali del modello.

Il mantenimento di ordini SH più elevati comporta file di dimensioni maggiori e un carico computazionale più alto lato viewer, ma offre una resa più fedele. Questa flessibilità consente di adattare il formato di output alle esigenze specifiche: ad esempio, mantenere solo SH0 o SH1 per scenari mobile a bassa banda, o conservare SH3 per usi desktop e presentazioni ad alta fedeltà.

Per sfruttare appieno SH3 in contesto web, è comunque necessario che il *viewer* sia in grado di gestirli efficientemente; ciò può richiedere ottimizzazioni dedicate o l'uso di tecnologie più recenti come *WebGPU*, con eventuali compromessi in termini di compatibilità con hardware e browser meno performanti.

⁵L'ordine SH indica il grado massimo l degli armonici sferici utilizzati. Il numero di coefficienti per canale cresce con la formula $(l + 1)^2$: ordine 0 → 1 coefficiente, ordine 1 → 4 coefficienti, ordine 2 → 9 coefficienti, ordine 3 → 16 coefficienti. Con 3 canali colore (RGB), mantenere SH3 richiede 48 coefficienti per gaussiana, contro i soli 3 di SH0.

5.2 Sviluppi futuri

L'analisi delle limitazioni e criticità presentata nelle sezioni precedenti mette in evidenza come molti degli ostacoli individuati possano essere trasformati in opportunità di evoluzione del sistema. In alcuni casi, gli sviluppi futuri rappresentano un naturale proseguimento del lavoro svolto, con l'obiettivo di superare vincoli tecnici o migliorare l'esperienza dell'utente. In altri, si tratta di estensioni concettuali e applicative che aprono a nuovi scenari d'uso, anche al di fuori del contesto sperimentale di questa tesi. Di seguito si elencano alcune direzioni di sviluppo ritenute particolarmente promettenti:

1. **Estensione a Gaussian Splatting 4D** Evoluzione del modello attuale verso il *4D Gaussian Splatting*, in cui la quarta dimensione è il tempo. Questa tecnica consente di rappresentare scene dinamiche, in cui le gaussiane variano posizione, colore o opacità nel tempo, aprendo la strada ad applicazioni come video volumetrici, cattura di movimenti e ricostruzioni di eventi in tempo reale.
2. **Analisi video assistita da intelligenza artificiale** Sviluppo di un modulo in grado di analizzare il video acquisito, valutando copertura spaziale e temporale e segnalando eventuali zone non riprese. Tale sistema potrebbe suggerire riprese integrative all'utente, riducendo il rischio di ricostruzioni incomplete senza richiedere conoscenze tecniche specifiche.
3. **Parametrizzazione automatica *scene-aware*** Implementazione di algoritmi di analisi della scena (densità dei punti, complessità geometrica, livello di dettaglio) per impostare automaticamente parametri chiave come `cap_max`, `budget`, risoluzione iniziale e ordine SH. Questo approccio ridurrebbe la necessità di intervento manuale, migliorando la coerenza e l'efficienza del processo di training.
4. **Ottimizzazione per realtà aumentata (AR) e realtà virtuale (VR)** Adattamento del sistema per la fruizione dei modelli in ambienti AR e VR, con tecniche di caricamento progressivo e gestione ottimizzata della latenza. Questa evoluzione permetterebbe l'integrazione in applicazioni immersive, anche con modelli complessi.
5. **Supporto a formati e viewer avanzati** Estensione della compatibilità a formati 3D standard come USDZ o glTF (con eventuali estensioni per gaussiane), o sviluppo di un *viewer* basato su *WebGPU* con supporto nativo fino a SH3. Ciò migliorerebbe l'interoperabilità con altri software e piattaforme e permetterebbe una resa visiva più fedele.

5.3 Conclusioni

Il lavoro presentato in questa tesi ha dimostrato la fattibilità e l'efficacia di un sistema di ricostruzione tridimensionale basato su tecniche di *Gaussian Splatting*, integrato in una pipeline che copre l'intero flusso: dall'acquisizione video, al pre-processing, al training del modello, fino alla visualizzazione web.

Il progetto ha richiesto un bilanciamento costante tra vincoli teorici, limitazioni pratiche e obiettivi prestazionali. L'adozione e l'adattamento di algoritmi come *Taming* e *MCMC* hanno permesso di esplorare soluzioni avanzate di ottimizzazione, affrontando sfide legate alla parametrizzazione, alla stabilità e alla compatibilità con le piattaforme di fruizione. Particolare attenzione è stata dedicata alla fase di acquisizione e preparazione dei dati, dove la qualità delle riprese e la completezza della copertura si sono rivelate fattori determinanti per la resa finale del modello.

L'analisi delle criticità ha evidenziato come molti limiti non siano insormontabili, ma piuttosto punti di partenza per futuri sviluppi: dalla parametrizzazione automatica *scene-aware*, all'estensione verso rappresentazioni temporali (4D), fino alla realizzazione di viewer web più sofisticati in grado di supportare ordini SH più elevati. Questi obiettivi non solo mirano a migliorare le prestazioni tecniche, ma anche a rendere il sistema più accessibile e adattabile a scenari applicativi differenti, dal settore culturale a quello industriale.

Il contributo principale di questo lavoro risiede dunque nell'aver integrato e adattato tecniche di *Gaussian Splatting* a un contesto operativo reale, sviluppando una pipeline funzionale e flessibile che può fungere da base per ulteriori evoluzioni. La combinazione di analisi tecnica, sperimentazione empirica e riflessione sulle prospettive future fornisce una visione chiara delle potenzialità di questa tecnologia, suggerendo come essa possa consolidarsi nei prossimi anni come strumento di riferimento per la rappresentazione tridimensionale di alta qualità.

In sintesi, questa tesi non si limita a presentare una soluzione tecnica, ma propone un approccio metodologico che unisce rigore scientifico, attenzione all'usabilità e apertura all'innovazione, ponendo solide basi per ricerche e applicazioni future.

Appendice A

Parametri di configurazione del training

A.1 Parametri base (3D Gaussian Splatting)

A.2 Parametri aggiuntivi MCMC

A.3 Parametri aggiuntivi Taming 3DGS

Parametro	Default	Ruolo	Valori più alti	Valori più bassi
--iterations	30.000	Numero totale di iterazioni di training	Migliore qualità finale, maggior costo computazionale, rischio overfitting	Convergenza insufficiente, qualità limitata, training più veloce
--resolution	-1 (auto)	Fattore di downscaling risoluzione immagini di input	Migliore qualità visiva e dettagli, maggior tempo e memoria	Training più veloce e leggero, perdita di dettagli e precisione
--sh_degree	3	Ordine armoniche sfériche per proprietà angolari	Effetti di luce più realistici e complessi (max 3)	Illuminazione e riflettanza limitate, minore realismo
--densify_from_iter	500	Iterazione di inizio densificazione	Processo iniziale più stabile, miglioramento più lento	Copertura scena più rapida, possibile instabilità iniziale
--densify_until_iter	15.000	Iterazione di fine densificazione	Migliori dettagli finali, maggior costo computazionale, rischio overfitting	Minore costo computazionale, limitata cattura dettagli nelle fasi avanzate
--densify_grad_threshold	0,0002	Soglia gradiente 2D per densificazione	Processo conservativo, minor dettaglio, maggiore efficienza	Densificazione aggressiva, maggior dettaglio, più prioritiva e costo
--densification_interval	100	Frequenza densificazione (ogni N iterazioni)	Densificazione meno frequente, minor controllo dettagli	Densificazione più frequente, miglior controllo dettagli, maggior costo
--opacity_reset_interval	3.000	Frequenza reset opacità Gaussiane	Reset meno frequenti, possibili accumuli di opacità stagnante	Reset più frequenti, evita accumuli ma può destabilizzare
--percent_dense	0,01	Percentuale volume scena per densificazione forzata	Densificazione più selettiva, minor costo computazionale	Densificazione più estesa, più Gaussiane, maggior costo
--lambda_dssim	0,2	Peso componente DSSIM vs L1 Loss	Maggior dettaglio strutturale, possibile rallentamento ottimizzazione	Focus su loss pixel-wise, minor attenzione alla struttura

Tabella A.1: Parametri principali del training 3D Gaussian Splatting

Nota: Il metodo MCMC estende l'algoritmo base 3D Gaussian Splatting, pertanto tutti i parametri della tabella precedente rimangono disponibili e utilizzabili in combinazione con quelli qui descritti.

Parametro	Default	Ruolo	Valori più alti	Valori più bassi
--cap_max	1.000.000	Numero massimo di Gaussiane utilizzabili durante l'intero training	Maggiore capacità di rappresentare dettagli fini, aumento significativo di memoria e tempo di rendering	Limitazione nella rappresentazione di dettagli fini, minore consumo di risorse computazionali
--scale_reg	0,01	Penalizzazione per Gaussiane troppo piccole o troppo grandi, mantiene range fisicamente plausibile	Regolarizzazione più forte, scale più uniformi, possibile limitazione di dettagli molto fini o molto grossolani	Regolarizzazione più debole, maggiore libertà nelle scale, rischio di degenerazioni e instabilità nel rendering
--opacity_reg	0,01	Penalizzazione valori estremi di opacità (0 o 1), incarreggia valori intermedi	Opacità più moderate, blending omogeneo, minore dominanza locale, possibile appiattimento resa visiva	Opacità più estreme, rappresentazione più netta e contrastata, rischio gaussiane irrivervanti o artefatti visivi
--noise_lr	0,005	Learning rate del termine stocastico in SGLD per l'esplorazione dello spazio delle soluzioni	Maggiore capacità esplorativa, miglior evitamento minimi locali, rallentamento convergenza	Convergenza più rapida e stabile, ridotta capacità di evitare minimi locali subottimali

Tabella A.2: Parametri aggiuntivi del metodo MCMC per 3D Gaussian Splatting

Nota: Il metodo Taming 3DGS estende l'algoritmo base 3D Gaussian Splatting, pertanto tutti i parametri della prima tabella rimangono disponibili e utilizzabili in combinazione con quelli qui descritti.

Parametro	Default	Ruolo	Valori più alti	Valori più bassi
--cams	10	Numero di viste utilizzate per il calcolo dello score di rilevanza di ogni Gaussiana	Stima più robusta della rilevanza delle Gaussiane, maggior costo computazionale per il calcolo degli score	Valutazione più rapida ed efficiente, rischio di sottostimare l'importanza di alcune Gaussiane
--budget	Variabile	Numero finale di Gaussiane considerate (intero o float, dipende da --mode)	Maggiore dettaglio e qualità della ricostruzione, uso più intensivo di memoria e tempo di calcolo	Ricostruzione più leggera e veloce, rischio di perdita di dettaglio nella rappresentazione finale
--mode	-	Modalità di interpretazione del budget: final_count (numero esatto) o multiplier (fattore su point cloud iniziale)	Con multiplier: fattore maggiore aumenta proporzionalmente le Gaussiane rispetto ai punti iniziali	Con final_count: controllo assoluto ma meno adattivo alla complessità intrinseca della scena
--ho.iteration	15.000	Iterazione di attivazione delle Gaussiane ad alta opacità per stabilizzare il training iniziale	Training più conservativo e stabile, ritardo nell'introduzione di dettagli ad alta visibilità	Attivazione precoce delle Gaussiane, accelerazione del dettaglio visivo ma rischio instabilità
--sh_lower	OFF	Riduce frequenza aggiornamenti armonici sferrici (ogni 16 iter.) per ottimizzare il costo del lighting	Maggior risparmio computazionale, adatto per scene con illuminazione uniforme	Aggiornamenti più frequenti del lighting, maggior fedeltà negli effetti angolari

Tabella A.3: Parametri aggiuntivi del metodo Taming 3DGS

Bibliografia

- [1] pIXELsHAM. *Rasterization vs Ray tracing*. <https://www.pixelsham.com/2019/10/24/whats-the-difference-between-ray-tracing-and-rasterization/>. Consultato il 5 Agosto 2025. 2025.
- [2] LearnOpenGL. *The process of volume rendering*. <https://learnopengl.com/Guest-Articles/2021/Scene/Frustum-Culling>. Consultato il 2 luglio 2025.
- [3] Nvidia. *Perspective vs parallel projection*. <https://www.ahirlabs.com/difference/perspective-projections-parallel-projection/>. Consultato il 2 luglio 2025.
- [4] RadianceFields. *Rasterization*. <https://radiancefields.com/understanding-3d-gaussian-splatting-via-render-engines/>. Consultato il 2 luglio 2025.
- [5] ResearchGate. *The Stanford Bunny 3D model in different representations*. <https://www.researchgate.net/figure/The-Stanford-Bunny-38-model-in-different-three-dimensional-representations-a-P>. Consultato il 2 luglio 2025. 2025.
- [6] Milan Ikits et al. «Volume Rendering Techniques». In: *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*. Addison-Wesley, 2004. Cap. 39. URL: <https://developer.nvidia.com/gpugems/gpugems/part-vi-beyond-triangles/chapter-39-volume-rendering-techniques>.
- [7] Nvidia. *The view frustum*. <https://developer.nvidia.com/gpugems/gpugems/part-vi-beyond-triangles/chapter-39-volume-rendering-techniques>. Consultato il 2 luglio 2025.
- [8] Ben Mildenhall et al. «NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis». In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer, 2020, pp. 405–421. DOI: 10.1007/978-3-030-58452-8_24. URL: https://www.ecva.net/papers/eccv_2020/papers_ECCV/papers/123460392.pdf.

- [9] ECCV. *5D continuous representation optimization*. https://www.ecva.net/papers/eccv_2020/papers_ECCV/papers/123460392.pdf. Consultato il 2 luglio 2025. 2025.
- [10] RadianceFields. *Point cloud traditional*. <https://radiancfields.com/understanding-3d-gaussian-splatting-via-render-engines/>. Consultato il 2 luglio 2025.
- [11] RadianceFields. *Point cloud with splats*. <https://radiancfields.com/understanding-3d-gaussian-splatting-via-render-engines/>. Consultato il 2 luglio 2025.
- [12] RadianceFields. *SfM*. <https://radiancfields.com/understanding-3d-gaussian-splatting-via-render-engines/>. Consultato il 2 luglio 2025.