Alma Mater Studiorum - Università di Bologna

Dipartimento di Informatica - Scienza e Ingegneria (DISI)
Corso di Laurea Magistrale in Ingegneria Informatica

Digital Twin della Certosa di Bologna: dal framework al rendering AI con Gaussian Splatting

Tesi di Laurea Magistrale in
Fondamenti di Computer Graphics M

Candidata:
Noemi Messori

Relatore:
Chiar.ma Serena Morigi

Correlatori:
Ing. Antonella Guidazzoli

Daniele de Luca
Federico Andrucci

Anno Accademico 2025-2026

Introduzione

Negli ultimi anni il concetto di Digital Twin ha assunto un ruolo sempre più centrale nell'ambito dell'ingegneria e della rappresentazione di sistemi complessi. Nato in ambito industriale come modello digitale capace di replicare un sistema fisico integrandone stato, dati e comportamento, il Digital Twin è stato progressivamente esteso ad altri domini applicativi, tra cui l'urbanistica e, più recentemente, il patrimonio culturale. In questo contesto non rappresenta soltanto una ricostruzione tridimensionale, ma un ambiente informativo interattivo in grado di supportare esplorazione, analisi e fruizione avanzata. Esso ha aperto nuove prospettive nella documentazione, conservazione e fruizione dei monumenti storici, introducendo tuttavia problematiche specifiche.

La riproduzione digitale di un monumento storico richiede infatti un equilibrio tra fedeltà geometrica, qualità visiva, ottimizzazione delle prestazioni e progettazione dell'esperienza utente. A tali aspetti si affiancano questioni legate all'architettura software, alla gestione dei dati e alla fruizione tramite web, che impongono vincoli tecnologici non trascurabili.

L'evoluzione delle tecniche di modellazione tridimensionale e di rendering, unitamente all'avanzamento degli strumenti web per la visualizzazione interattiva, ha reso possibile la creazione di ambienti digitali complessi accessibili direttamente tramite browser.

In questo scenario si inserisce il presente lavoro di tesi, sviluppato nell'ambito di uno stage curricolare presso il Visual Information Technology Lab di Antonella Guidazzoli, all'interno del reparto HPC del Cineca, centro di rilevanza nazionale ed europea nel settore del supercalcolo. Il progetto propone la realizzazione di un Digital Twin del Monumento Bottoni presso la Certosa

di Bologna, denominato nel seguito **DT-Certosa**, come proof of concept di sistema interattivo per la valorizzazione del patrimonio culturale.

DT-Certosa estende e ottimizza un lavoro precedentemente avviato: la ricostruzione tridimensionale del monumento viene integrata con il concetto di *Monumento Parlante*, in cui l'ambiente virtuale non si limita alla visualizzazione statica ma diventa strumento di fruizione guidata, arricchendo l'esperienza dell'utente attraverso interazioni attive e percorsi strutturati, mira alla creazione di un ambiente digitale navigabile, estensibile e multi-utente, in cui la componente grafica, l'infrastruttura software e la logica di interazione sono strettamente integrate.

L'obiettivo dell'elaborato è dunque la progettazione e lo sviluppo di un progetto che unisca accuratezza geometrica, ottimizzazione delle prestazioni e funzionalità applicative avanzate in un'unica web application.
La webapp è destinata alla fruizione pubblica: al termine del progetto, verrà ospitata su un'istanza di ATON installata su ADA Cloud, l'infrastruttura cloud di Cineca basata sul supercomputer omonimo. Questo garantisce l'accessibilità del Digital Twin via browser da qualunque dispositivo connesso, senza necessità di installazioni locali.

Il lavoro si articola in una prima parte teorica e metodologica, seguita da una parte progettuale e implementativa relativa allo sviluppo del sistema.

Nel primo capitolo viene introdotto il concetto di Digital Twin, analizzandone definizioni, caratteristiche principali ed evoluzione nei diversi ambiti applicativi. Viene inoltre approfondito il concetto di *Monumento Parlante*, inquadrandolo come declinazione specifica del Digital Twin nel settore dei beni culturali, con particolare attenzione agli aspetti di interazione e mediazione digitale.

Segue un'analisi dell'evoluzione delle tecniche di rendering tridimensionale. Prima viene presentata una panoramica sulle tecniche tradizionali basate su modellazione poligonale e pipeline raster, successivamente vengono presentati approcci più recenti e innovativi basati su Intelligenza Artificiale, quali le Neural Radiance Fields (NeRF) e il Gaussian Splatting. Queste tecniche introducono modalità innovative di rappresentazione della scena, superando in

parte i limiti delle pipeline tradizionali. In particolare, viene motivata la scelta del Gaussian Splatting per la modellazione della vegetazione, avendo come obiettivo quello di raggiungere un compromesso tra qualità visiva e gestione della complessità in ambiente web.

Un ulteriore capitolo è dedicato ai framework web per la gestione di ambienti tridimensionali interattivi. Dopo una breve analisi comparativa di soluzioni diffuse come Three.js e Babylon.js, viene presentato il framework ATON, scelto come base architetturale della web application. La scelta è motivata in funzione della sua struttura modulare, della gestione integrata della scena 3D e delle possibilità di estensione offerte a livello applicativo.

La seconda parte dell'elaborato è dedicata allo sviluppo del progetto.

In primo luogo viene descritta la modellazione tridimensionale del Monumento Bottoni e delle aree circostanti. Vengono illustrate le fasi di rilievo, la ricostruzione del Monumento, la gestione dei testi incisi sulle tombe e l'adozione di strategie di *Level of Detail* (LOD) per ottimizzare le prestazioni della webapp. Le statue sono state ricostruite tramite tecniche di fotogrammetria, utilizzando il software Meshroom, mentre per gli elementi esterni sono state adottate metodologie differenziate di modellazione e texturing. La vegetazione è stata realizzata attraverso una pipeline basata su Gaussian Splatting, comprendente le fasi di ricostruzione degli splat e la gestione dinamica dei livelli di dettaglio in funzione della posizione della camera.

Successivamente viene descritta l'architettura applicativa del sistema. A partire dalla struttura di ATON, viene illustrata l'integrazione con un database realizzato tramite PocketBase, utilizzato per la gestione degli utenti e delle sessioni. Viene quindi approfondita l'implementazione delle funzionalità di navigazione all'interno della webapp, inclusa la definizione di percorsi e punti di vista, nonché l'adozione dell'algoritmo A* per il calcolo del percorso più breve verso una destinazione selezionata. L'integrazione del sistema di pathfinding con il framework ha richiesto interventi a livello di codice sorgente e l'introduzione di elementi visivi di supporto alla navigazione.

Infine, viene presentata l'implementazione della modalità multi-utente, che

consente la creazione di sessioni collaborative con distinzione dei ruoli e sincronizzazione dei percorsi guidati. Tale funzionalità rappresenta un elemento centrale del concetto di *Monumento Parlante*, in quanto abilita una fruizione condivisa e guidata dell'ambiente digitale.

A fine dell'implementazione, viene illustrato il funzionamento concreto del sistema attraverso due scenari d'uso rappresentativi: la navigazione esplorativa autonoma da parte di un singolo utente e la fruizione di una sessione collaborativa multi-utente condotta da un presenter. Tale analisi permette di evidenziare l'integrazione delle diverse componenti architetturali e grafiche sviluppate all'interno di un'unica web application.

L'elaborato si conclude con una valutazione complessiva del sistema sviluppato, evidenziandone i contributi tecnici, i limiti attuali e le possibili evoluzioni future. Il lavoro si colloca quindi all'intersezione tra grafica computazionale, ingegneria del software e tecnologie web, proponendo un'integrazione tra tecniche avanzate di rappresentazione tridimensionale e progettazione di sistemi interattivi per la valorizzazione del patrimonio monumentale.

Indice

Elenco delle figure

Capitolo 1

Digital Twin e Monumenti Parlanti

1.1 Digital Twin: definizione e stato dell'arte

Il Digital Twin, o gemello digitale, è un modello dinamico che replica con fedeltà lo stato, le proprietà e il comportamento di un sistema fisico reale, mantenendo con esso una connessione bidirezionale e continua. Considerarlo un semplice duplicato digitale sarebbe riduttivo: la sua caratteristica distintiva risiede nella capacità di creare un ponte tra il mondo fisico e quello virtuale, offrendo un ambiente simulato in cui testare ipotesi, simulare scenari, prevedere esiti e guidare le decisioni senza interferire direttamente con l'oggetto fisico.

Il concetto fu introdotto da Michael Grieves nei primi anni 2000, nel contesto della gestione del ciclo di vita dei prodotti manifatturieri, e formalizzato in un white paper in cui il Digital Twin viene descritto come sistema composto da tre elementi: l'entità fisica nello spazio reale, il suo corrispettivo virtuale nello spazio digitale e la connessione bidirezionale che li lega [17]. Tale connessione non è un semplice trasferimento di dati: consente al modello virtuale di acquisire lo stato corrente dell'entità fisica, eseguire analisi predittive e, in determinati contesti, retroagire sul processo reale attraverso comandi di controllo. Tao [31] ha ulteriormente sistematizzato il modello, riconoscendo che l'informazione non è incorporata staticamente nella geometria ma costituisce uno strato autonomo che alimenta e aggiorna il modello nel tempo.

Con la crescita delle tecnologie legate all'Internet of Things (IoT), all'analisi di big data e al cloud computing, la capacità di sviluppare e impiegare

Digital Twin è cresciuta significativamente, trasformando questo strumento in una componente rilevante per l'innovazione in numerosi settori. Da semplici repliche di oggetti fisici, i Digital Twin sono diventati sistemi capaci di simulare intere catene produttive, infrastrutture urbane e reti di servizi, con applicazioni concrete nella manutenzione predittiva, nella pianificazione urbana e nella gestione ottimale delle risorse.

Rispetto ai tradizionali modelli di simulazione, il Digital Twin si distingue per alcune proprietà qualificanti. La sincronizzazione continua con l'entità fisica è la più caratterizzante: non è un modello costruito una volta per tutte, ma un sistema che si aggiorna al variare dello stato reale. La bidirezionalità del flusso di dati costituisce un ulteriore elemento distintivo: il gemello non solo riceve informazioni dal dominio fisico, ma può fornire output che rientrano nel processo reale, come istruzioni operative, allarmi e aggiornamenti di configurazione. La capacità predittiva, infine, differenzia il Digital Twin da uno specchio passivo: attraverso l'analisi dei dati storici e la simulazione sul modello virtuale, il sistema è in grado di anticipare stati futuri e supportare la presa di decisioni prima che vengano effettuate nel dominio fisico.

1.2 Evoluzione del concetto e ambiti applicativi

Le origini del gemello digitale precedono la denominazione formale introdotta da Grieves. La NASA già dagli anni Sessanta sviluppava modelli computazionali aggiornati in tempo reale sulla base delle telemetrie dei veicoli spaziali, con l'obiettivo di testare interventi correttivi sul modello prima di applicarli al sistema reale. Con l'espansione della capacità computazionale e la diffusione dei sensori connessi in rete, il concetto si è esteso progressivamente ad altri ambiti.

Nel settore manifatturiero, nell'ambito dell'Industry 4.0, il Digital Twin si è affermato come strumento per la manutenzione predittiva, l'ottimizzazione dei processi e la riduzione dei fermi macchina, integrandosi nel ciclo di vita del prodotto dalla progettazione alla dismissione. A scala urbana, le iniziative di *smart city* hanno esteso il modello a sistemi di sistemi: infrastrutture, flussi di persone, reti energetiche e idriche, con applicazioni concrete nella gestione del traffico e nella pianificazione energetica.

Il patrimonio culturale rappresenta uno dei domini di applicazione più recenti. L'estensione del Digital Twin dai contesti produttivi ai beni storici ri-

chiede un adattamento degli obiettivi: non si tratta di monitorare un processo in esercizio, ma di documentare, conservare e valorizzare un artefatto la cui entità fisica non viene modificata attivamente dal sistema digitale.

1.3 Digital Twin applicato ai beni culturali

L'applicazione del Digital Twin ai beni culturali risponde a esigenze profondamente diverse rispetto a quelle che ne hanno motivato l'origine industriale. Un monumento storico non genera flussi di dati operativi in tempo reale, non deve essere ottimizzato produttivamente né controllato a distanza. Ciò che giustifica la costruzione di un suo gemello digitale è un insieme differente di obiettivi: la documentazione accurata dello stato attuale, la conservazione della memoria dell'artefatto in forme accessibili a lungo termine, la fruizione da parte di pubblici eterogenei, e il supporto all'attività di ricerca storica, artistica e architettonica.

Il modello virtuale viene tipicamente costruito attraverso campagne periodiche di rilievo e successivamente interrogato da utenti con esigenze differenti. La componente di servizio si traduce in strumenti di navigazione, accesso a database documentali e supporto alla mediazione educativa, piuttosto che in funzioni di controllo o manutenzione predittiva.

Sul piano tassonomico, Luther [24] classifica i virtual museum come Digital Twin in tre orientamenti: *content-centric*, focalizzato sulla riproduzione fedele degli artefatti; *communication-centric*, orientato al trasferimento di conoscenza; *collaboration-centric*, che include piattaforme condivise e approcci partecipativi. I sistemi più completi tendono a integrarli. Le evoluzioni più recenti beneficiano dell'incremento delle capacità di rendering disponibili nei browser web, delle nuove tecniche di acquisizione fotogrammetrica e di metodi basati su intelligenza artificiale come le Neural Radiance Fields (NeRF) e il Gaussian Splatting, trattati nel Capitolo 2.

1.4 Il concetto di "Monumento Parlante"

Nel dominio del patrimonio monumentale, la declinazione del Digital Twin adottata in questo lavoro va oltre la fedeltà geometrica. Il *Monumento Parlante* è un Digital Twin del patrimonio culturale che integra la dimensione spaziale con quella informativa e narrativa: il modello virtuale diventa un

ambiente capace di comunicare la storia dell'artefatto, le persone che vi sono legate e le vicende che ha attraversato. Ogni elemento della scena può essere portatore di contenuto: una statua rimanda alla biografia del commemorato, un'iscrizione funebre al contesto storico, un elemento architettonico alla cronologia costruttiva dell'insieme.

Questa concezione trova un antecedente diretto nelle sperimentazioni condotte sulla Certosa di Bologna, oggetto del presente lavoro. Calori [8] descrisse già nel 2003 un ambiente virtuale multi-livello per il museo della Certosa in cui il modello tridimensionale era concepito come punto di accesso a informazioni storiche, artistiche e catastali. Borgatti [6] estese l'approccio a più siti della città, sistematizzando la mediazione visuale come principio progettuale: il modello tridimensionale come punto di accesso attraverso risorse informative eterogenee.

La differenza rispetto al sistema contemporaneo risiede nell'accessibilità: quei sistemi richiedevano installazioni hardware dedicate; il *Monumento Parlante* contemporaneo è fruibile via browser su qualunque dispositivo connesso. Ci sono poi ulteriori elementi implementabili che rendono questo Digital Twin potenzialmente ancora più articolato: ne sono un esempio la navigazione assistita attraverso punti di interesse selezionati oppure la dimensione collaborativa. Quest'ultima può può essere formalizzata come workflow di fruizione guidata dove un operatore conduce un gruppo di visitatori sincronizzando prospettive e ricreando una versione digitale di quella che sarebbe una visita guidata tradizionale.

Il *Monumento Parlante* diventa così ambiente unico che unisce il visitatore e l'artefatto, con un sistema digitale che fa da tramite e interviene attivamente nella costruzione dell'esperienza. La scelta del web come piattaforma, inoltre, ha lo scopo di eliminare la barriera di installazione di software specializzati e di rendere il Digital Twin accessibile da qualunque dispositivo connesso. Una conseguenza di questa scelta sono sicuramente però nuovi vincoli tecnici non trascurabili: limitazioni computazionali del browser, latenza di rete e anche gestione della complessità del modello. La definizione dei livelli di dettaglio, la gestione asincrona del caricamento degli asset e la scelta delle tecniche di rendering non sono decisioni puramente grafiche, ma scelte architetturali che dipendono dal vincolo di accessibilità web. Spesso risulta necessaria un'architettura software capace di separare la logica di rendering, la logica applicativa

e la gestione dei dati. Un framework dedicato agli ambienti culturali interattivi, come ATON presentato nel Capitolo 3, assolve questa funzione fornendo astrazioni sulla scena tridimensionale, gestione nativa dei punti di vista e dei percorsi guidati, e un'architettura modulare che consente l'integrazione di componenti applicativi senza alterare il nucleo del sistema di rendering.

Il Monumento Bottoni alla Certosa di Bologna concentra tutte queste tensioni in un unico oggetto: geometria articolata, vegetazione, contenuti informativi stratificati e sessioni collaborative, il tutto fruibile via browser. Tradurre queste esigenze in architettura è il filo conduttore dei capitoli che seguono.

Capitolo 2

Evoluzione delle Tecniche di Rendering

Il rendering è il processo che trasforma una scena 3D in un'immagine bidimensionale: prende in ingresso geometria, materiali, sorgenti luminose e una telecamera virtuale, e produce in uscita una griglia di pixel colorati. Negli ultimi decenni sono emersi tecniche e approcci molto diversi per affrontare questo problema, ciascuno con implementazioni e compromessi differenti. Comprendere questa evoluzione è il presupposto per motivare le scelte tecniche del presente progetto, in particolar modo l'adozione del 3D Gaussian Splatting per la rappresentazione della vegetazione nell'ambiente virtuale della Certosa.

Il capitolo ripercorre le principali due tecniche del rendering tradizionale: la rasterizzazione e il Ray Tracing, ne confronta le caratteristiche e i limiti e introduce poi metodi più recenti: i Neural Radiance Fields e il 3D Gaussian Splatting. Per questi ultimi vengono analizzati i dettagli e viene proposto anche un confronto, affrontando alla fine la scelta dell'uso all'interno del progetto.

2.1 Rendering tradizionale

Entrambi i metodi tradizionali operano su una rappresentazione geometrica esplicita della scena: superfici approssimate da mesh di triangoli. I triangoli sono la primitiva scelta per la modellazione 3D perché sono sempre planari, convessi e gestibili con operazioni matriciali semplici. Superfici curve o modellate con spline vengono convertite in mesh tramite *tessellazione*, con un

numero di triangoli regolabile in base al compromesso tra fedeltà geometrica e costo computazionale. Su questa base comune si costruiscono due strategie opposte per determinare il colore di ciascun pixel.

2.1.1 Pipeline di rasterizzazione

La rasterizzazione percorre la scena *dalle primitive verso i pixel*. Come si può vedere dalla Figura 2.1 ogni triangolo attraversa una catena di stadi in sequenza, la cosiddetta *rendering pipeline*, dove vengono effettuate una serie di trasformazioni: la prima passa dall'oggetto allo spazio mondo, poi allo spazio telecamera, proiezione nelle coordinate normalizzate del dispositivo, e infine mappatura sullo schermo.
La fase di rasterizzazione vera e propria converte ciascun triangolo proiettato in un insieme di frammenti, interpolandone colore, normali e coordinate di texture tramite coordinate baricentriche. La visibilità è gestita con lo *z-buffer*: per ogni frammento si mantiene la profondità e solo il frammento più vicino alla telecamera scrive nel framebuffer.

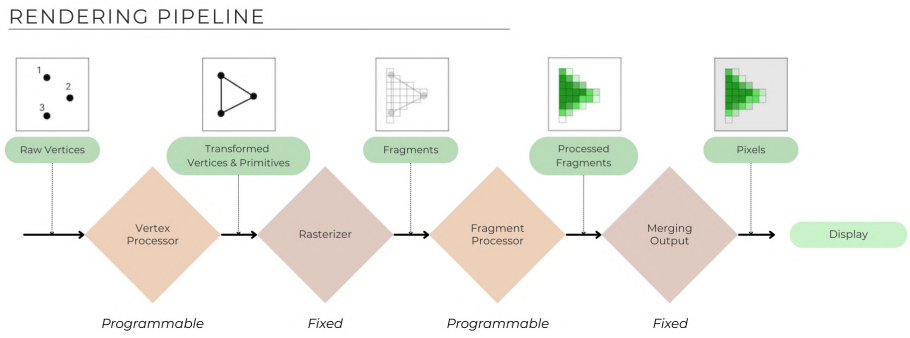

Figura 2.1: Pipeline di rendering tradizionale basata su rasterizzazione.

L'illuminazione in questo contesto è *locale*: il colore di ogni punto dipende unicamente dalle sorgenti luminose dirette, senza considerare i contributi di luce indiretta. Sono quindi ignorate le varie riflessioni e rifrazioni tra i vari oggetti che richiedono un'analisi differente dedicata.

L'intera pipeline, sia Geometry Stage che parte di Rasterization, è progettata per essere eseguita in parallelo su GPU: migliaia di core elaborano vertici e frammenti indipendentemente.

2.1.2 Ray Tracing

Il Ray Tracing parte *dai pixel e risale verso la scena*, invertendo il meccanismo descritto finora. Per ciascun pixel viene lanciato un *primary ray* dall'osservatore attraverso il piano immagine e alla prima intersezione con la geometria vengono generati raggi secondari: verso le sorgenti luminose (*shadow rays*), nella direzione di riflessione speculare e, se il materiale è traslucente, nella direzione di rifrazione. Questo è visibile nella Figura 2.2. Il processo è ricorsivo: ogni raggio secondario può generare ulteriori rimbalzi, costruendo ad albero il contributo luminoso complessivo del pixel.

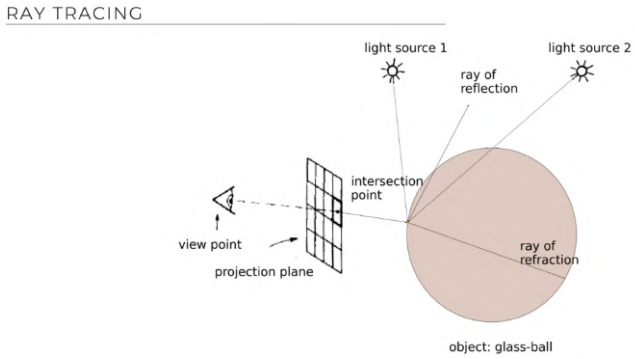

Figura 2.2: Propagazione dei raggi nel ray tracing.

Questo meccanismo implementa un modello di *illuminazione globale*: la luce si propaga correttamente attraverso la scena, producendo riflessioni su superfici specchio, rifrazioni in materiali trasparenti e ombre con bordi geometricamente esatti. Una variante probabilistica, il *Path Tracing*, campiona casualmente le direzioni di rimbalzo e risolve l'equazione del rendering in forma integrale con un metodo Monte Carlo, ottenendo illuminazione indiretta diffusa e caustiche.

2.1.3 Confronto e limiti del rendering tradizionale

In sintesi, la rasterizzazione è un processo orientato alla proiezione della geometria sul piano immagine: parte dagli oggetti, dai 3D triangles e li trasforma nello spazio della telecamera determinando quali pixel coprono. L'illuminazione viene calcolata in modo locale e approssimato, privilegiando efficienza e parallelizzazione rispetto alla fedeltà fisica.
Il Ray Tracing al contrario adotta un punto di vista opposto: parte dai pixel e

simula il percorso della luce nella scena. L'immagine non è più ottenuta proiettando la geometria, ma ricostruendo il contributo luminoso che raggiunge l'osservatore. Questo approccio è concettualmente più vicino al comportamento fisico della luce e consente una resa più realistica, al prezzo di un costo computazionale significativamente maggiore.

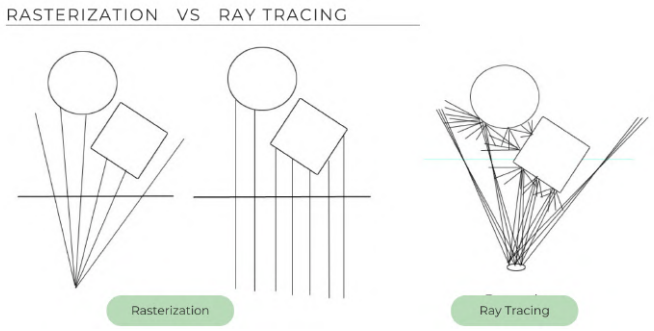

Figura 2.3: Confronto schematico tra rasterizzazione e ray tracing.

I due paradigmi, come si può vedere dalla Figura 2.3, rappresentano quindi due strategie complementari: la rasterizzazione privilegia l'efficienza e il controllo deterministico del processo di rendering, mentre il Ray Tracing privilegia la correttezza fisica e la coerenza globale dell'illuminazione.

2.2 Rendering tramite tecniche di Intelligenza Artificiale

Le limitazioni delle tecniche tradizionali hanno motivato, a partire dagli anni 2010, la ricerca di rappresentazioni alternative che non richiedano una mesh esplicita come punto di partenza. L'approccio emergente consiste nell'apprendere direttamente la funzione di rendering da un insieme di fotografie: dato un numero sufficiente di immagini della scena acquisite da angolazioni note, una rete neurale o una struttura dati ottimizzabile sintetizza nuove viste con un livello di fotorealismo difficilmente ottenibile con i metodi classici su geometria fotogrammetrica. La praticabilità di queste tecniche è strettamente legata alla disponibilità di hardware adeguato: l'aumento progressivo della potenza di calcolo delle GPU, lo sviluppo di architetture neurali più efficienti e la crescente diffusione di schede video dedicate anche nell'hardware consumer hanno

reso questi metodi accessibili anche al di fuori dei laboratori di ricerca. Le due tecniche più rilevanti in questo filone, per qualità dei risultati e adozione nella pratica, sono i Neural Radiance Fields (NeRF) e il 3D Gaussian Splatting (3DGS).

2.2.1 Neural Radiance Fields (NeRF)

I Neural Radiance Fields (NeRF), introdotti da Mildenhall nel 2020 [25], codificano una scena 3D interamente nei pesi di una rete neurale. L'idea che c'è alla base è la seguente: un Multi-Layer Perceptron (MLP) con parametri Θ impara ad associare a ogni punto dello spazio (x, y, z) e a ogni direzione di osservazione (θ, ϕ) due valori in uscita: il colore $\mathbf{c}$ emesso in quella direzione e la densità volumetrica σ, che indica quanto quel punto è opaco. La rete viene ottimizzata su un insieme di fotografie della scena senza alcuna supervisione esplicita su geometria o profondità, e i pesi convergono verso una rappresentazione implicita e continua dell'intera scena.

Affinché questa ottimizzazione possa procedere, ogni immagine del dataset deve essere accompagnata dalla propria posa di telecamera, ovvero dalla matrice di rotazione e traslazione che descrive posizione e orientamento della camera al momento dello scatto. Queste pose vengono stimate automaticamente tramite Structure-from-Motion (SfM). Uno strumento come COLMAP [29] analizza le corrispondenze di punti caratteristici tra le immagini, ricostruisce una nuvola di punti sparsa della scena e ne determina la posizione e l'orientamento di ciascun fotogramma nello spazio. In pratica, il dataset di addestramento si ottiene girando un video attorno alla scena ed estraendone i frame, oppure scattando fotografie manuali che coprano l'intera superficie d'interesse da angolazioni diverse. La copertura deve essere sufficientemente ampia e uniforme, perché le zone non coperte generano ambiguità nell'ottimizzazione e degrado della qualità ricostruita.

Il rendering di una nuova vista avviene tramite volume rendering differenziabile. Per ogni pixel si lancia un raggio $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ nella scena e si campionano punti lungo di esso; il colore atteso per quel raggio è:

$$\hat{C}(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\,\sigma(\mathbf{r}(t))\,\mathbf{c}(\mathbf{r}(t), \mathbf{d})\,dt, \qquad T(t) = \exp\left(-\int_{t_n}^{t} \sigma(\mathbf{r}(s))\,ds\right) \tag{2.1}$$

dove $T(t)$ è la trasmittanza accumulata lungo il raggio fino alla distanza t. Poiché la funzione di rendering è differenziabile rispetto ai parametri della rete, l'ottimizzazione minimizza la somma degli errori quadratici tra le immagini sintetizzate e quelle di addestramento, senza richiedere supervione su geometria o profondità.

Due accorgimenti tecnici migliorano significativamente i risultati. Il *positional encoding* trasforma le coordinate 5D tramite funzioni sinusoidali a frequenze crescenti, permettendo all'MLP di apprendere dettagli ad alta frequenza che altrimenti andrebbero persi. La *campionatura gerarchica* suddivide il processo in due passaggi: un primo campionamento grossolano stima la distribuzione della densità lungo il raggio e un secondo concentra i campioni nelle zone di maggiore contenuto geometrico. La Figura 2.4 mostra la struttura complessiva della pipeline: l'MLP produce densità e colore emesso per ciascun punto campionato lungo i raggi; questi valori vengono poi accumulati tramite volume rendering differenziabile per ottenere il colore finale del pixel.

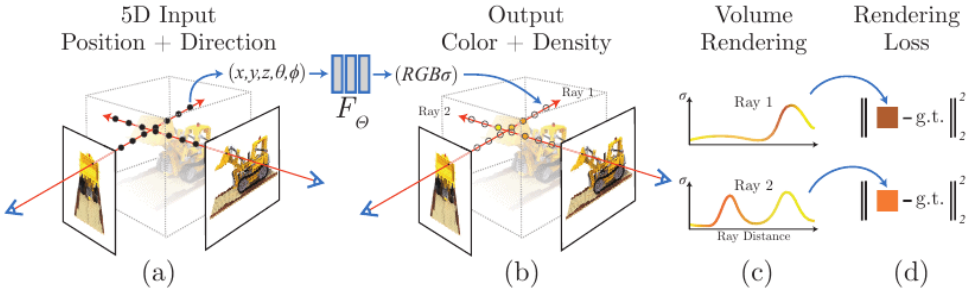

Figura 2.4: Pipeline di NeRF [23].

Al momento della pubblicazione, i risultati di NeRF erano nettamente superiori a quelli dei metodi precedenti per la sintesi di nuove viste, in particolare su scene con materiali speculari, trasparenze e geometria complessa. La rappresentazione volumetrica implicita gestisce naturalmente questi casi senza richiedere una mesh esplicita. I limiti emergono però sul piano applicativo: l'ottimizzazione della rete su una singola scena richiede ore o giorni di training; il rendering di ogni frame necessita di migliaia di query all'MLP per raggio, producendo latenze incompatibili con la navigazione interattiva; la qualità decade nelle zone coperte da poche immagini di addestramento. Su scene di larga scala, la capacità di un singolo MLP è insufficiente su scene di larga scala e

sono necessarie architetture specializzate che aumentano ulteriormente il costo.

2.2.2 3D Gaussian Splatting

Il 3D Gaussian Splatting (3DGS) si basa su un principio proposto già nei primi anni Novanta da Westover [33]: rappresentare una scena come somma di primitive gaussiane proiettate sul piano immagine tramite un'operazione di *splatting*. Il lavoro di Kerbl nel 2023 [20] ha trasformato questa idea in una pipeline ottimizzabile end-to-end a partire da fotografie, abbinandola a un rasterizzatore differenziabile su GPU sufficientemente efficiente da abilitare il rendering in tempo reale. Il risultato è una rappresentazione esplicita della scena tramite un insieme di primitive gaussiane tridimensionali ottimizzabili, senza alcuna rete neurale coinvolta durante il rendering. Come illustrato nella Figura 2.5, le immagini di input con pose stimate da SfM inizializzano un insieme di Gaussiane 3D, che vengono ottimizzate iterativamente tramite rasterizzazione differenziabile e controllo adattivo della densità, fino a ottenere un sistema capace di sintetizzare nuove viste in tempo reale.

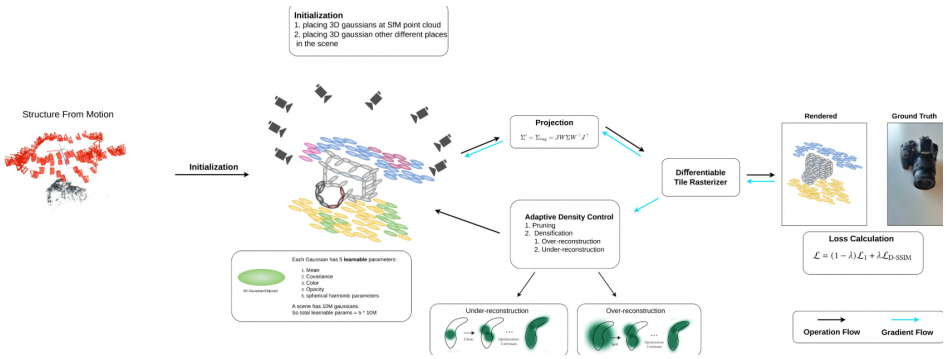

Figura 2.5: Pipeline del 3D Gaussian Splatting [23].

Rappresentazione. Ogni Gaussiana G_i è descritta da quattro gruppi di parametri:

- **Posizione** $\boldsymbol{\mu}_i \in \mathbb{R}^3$, corrispondente alla media della distribuzione;
- **Matrice di covarianza** Σ_i, parametrizzata come $\Sigma = RSS^\mathsf{T}R^\mathsf{T}$ con R matrice di rotazione e S matrice di scala diagonale, il che garantisce che Σ sia sempre semidefinita positiva durante l'ottimizzazione, ovvero che descriva sempre un ellissoide geometricamente valido;

- **Opacità** $\alpha_i \in [0, 1]$;
- **Colore view-dependent**, codificato tramite coefficienti di armonici sferici (Spherical Harmonics, SH), che modellano effetti di riflessione dipendenti dalla direzione di osservazione.

Ogni Gaussiana descrive graficamente un ellissoide 3D: gli autovalori di Σ ne determinano le dimensioni lungo i tre assi principali, mentre gli autovettori ne determinano l'orientamento nello spazio. L'insieme di tutte le Gaussiane forma una rappresentazione volumetrica discreta e sparsa dell'intera scena.

Inizializzazione e ottimizzazione. Il processo di training parte dalla stessa nuvola di punti sparsa prodotta da SfM tramite COLMAP [29] descritta per NeRF: le posizioni e gli orientamenti delle telecamere già calcolati vengono riutilizzati direttamente e ogni punto della nuvola inizializza una Gaussiana con posizione $\boldsymbol{\mu}_i$ corrispondente. L'ottimizzazione minimizza una funzione di perdita che combina un termine di errore assoluto $\mathcal{L}_1$ con un termine di similarità strutturale differenziabile $\mathcal{L}_{\text{D-SSIM}}$:

$$\mathcal{L} = (1 - \lambda)\,\mathcal{L}_1 + \lambda\,\mathcal{L}_{\text{D-SSIM}} \tag{2.2}$$

dove $\lambda = 0.2$ nella configurazione originale [20]. Il gradiente viene propagato attraverso la rasterizzazione differenziabile fino ai parametri di ogni Gaussiana.

Un meccanismo di *controllo adattivo della densità* regola dinamicamente il numero di gaussiane nel corso dell'ottimizzazione: le primitive con opacità sotto soglia vengono eliminate (*pruning*), quelle in regioni sotto-rappresentate vengono duplicate nella stessa posizione (*cloning*), quelle troppo grandi in zone già dense vengono scisse in due primitive più piccole (*splitting*). In questo modo la rappresentazione si addensa selettivamente dove la scena ha maggiore complessità visiva, senza vincolare a priori la risoluzione.

Rendering tramite rasterizzazione tile-based. Il rendering utilizza un rasterizzatore differenziabile ad hoc, progettato per massimizzare il parallelismo su GPU. Ciascuna Gaussiana viene proiettata sul piano immagine come un'ellisse 2D tramite un'approssimazione affine al primo ordine della trasformazione prospettica. L'immagine è suddivisa in tile di 16×16 pixel: le Gaussiane vengono ordinate per profondità, assegnate ai tile che le relative proiezioni

ricoprono e sovrapposte in sequenza tramite alpha-blending front-to-back su GPU. Il colore finale del pixel p è:

$$C(p) = \sum_{i \in \mathcal{N}} \mathbf{c}_i \, \alpha_i \prod_{j<i} (1 - \alpha_j) \tag{2.3}$$

dove $\mathcal{N}$ è l'insieme ordinato delle Gaussiane che coprono p, $\mathbf{c}_i$ è il colore valutato dagli armonici sferici nella direzione di osservazione corrente, e α_i è il prodotto tra l'opacità memorizzata della Gaussiana e la densità della relativa proiezione 2D gaussiana nella posizione del pixel. L'intera operazione è eseguita interamente in hardware, senza alcuna query a reti neurali, abilitando rendering in tempo reale a frequenze superiori a 30 fps anche su scene di complessità media.

2.2.3 Confronto tra NeRF e 3D Gaussian Splatting

Entrambe le tecniche partono dallo stesso input: un insieme di fotografie con pose note, da cui viene appresa una rappresentazione capace di sintetizzare viste non acquisite [23, 13]. Il modo in cui questa rappresentazione è costruita e utilizzata durante il rendering è invece radicalmente diverso.

NeRF codifica la scena nei pesi di un Multi-Layer Perceptron, una struttura implicita e non ispezionabile direttamente. Il 3DGS usa invece un insieme esplicito di Gaussiane: ciascuna ha parametri geometrici e di colore leggibili, modificabili, esportabili. Tra questi parametri figura anche l'opacità α, ottimizzata direttamente per ogni Gaussiana durante il training; NeRF al contrario produce densità volumetrica σ e la converte in opacità per integrazione lungo il raggio, con un costo computazionale aggiuntivo e una dipendenza dal campionamento che il 3DGS evita [13]. Questa struttura esplicita rende il 3DGS più adatto a operazioni di post-editing (rimozione di artefatti, riposizionamento di elementi), operazioni che con NeRF richiederebbero una nuova ottimizzazione completa della rete.

La differenza più rilevante sul piano applicativo riguarda la velocità di rendering. NeRF campiona centinaia di punti per raggio e interroga l'MLP per ciascuno: su hardware consumer senza ottimizzazioni dedicate, il rendering di un singolo frame richiede diversi secondi o minuti. Il 3DGS non esegue alcuna inferenza neurale durante il rendering; proietta e composita le Gaussiane direttamente su GPU, produce frame in pochi millisecondi e supera i 30 fps

su scene di complessità media [20]. Questa proprietà lo rende compatibile con ambienti web-based, dove NeRF non è praticabile senza infrastrutture server dedicate.

Sul fronte qualità, NeRF tende a produrre superfici più uniformi, con meno artefatti nelle zone a bassa copertura del dataset. Il 3DGS può presentare strutture spurie dove la densità delle Gaussiane è insufficiente, ma la qualità complessiva risulta paragonabile o superiore su scene ben acquisite, anche grazie all'assenza dell'alias introdotto dal campionamento discreto lungo i raggi.

Tabella 2.1: Confronto tra NeRF e 3D Gaussian Splatting sulle principali dimensioni di valutazione [23, 13].

Dimensione	**NeRF**	**3D Gaussian Splatting**
Rappresentazione della scena	Implicita (pesi MLP)	Esplicita (insieme di Gaussiane)
Rendering a runtime	Lento (query MLP per ogni punto del raggio)	Veloce (rasterizzazione tile-based su GPU)
Editing post-training	Non praticabile senza riottimizzazione	Diretto sui parametri delle Gaussiane
Qualità visiva	Alta; superfici uniformi	Alta; dipende dalla densità delle Gaussiane
Tempi di training	Ore – giorni	Ore (mediamente più rapido)
Compatibilità web/real-time	Limitata; richiede hardware dedicato	Compatibile con hardware consumer

2.2.4 Motivazione della scelta del 3DGS per la vegetazione

La vegetazione dell'area esterna del Monumento Bottoni, cespugli, siepi e alberature, è uno degli elementi visivamente più caratterizzanti dell'ambiente ma anche uno dei più difficili da riprodurre con le tecniche tradizionali.

La fotogrammetria applicata a elementi come foglie e rami, dove le superfici si sovrappongono e si occludono a vicenda in modo caotico, tende a produrre mesh rumorose, con lacune e artefatti che richiedono una fase di post-elaborazione estesa. Anche ottenendo una mesh pulita, il numero di poligoni necessario per preservare il dettaglio delle strutture fogliari è incompatibile con i vincoli di un'applicazione web-based, dove banda disponibile e potenza

computazionale del client non sono controllabili. Soluzioni approssimate come billboard o piani alpha-clipped simulano la siluetta della vegetazione ma perdono coerenza volumetrica al variare dell'angolo di osservazione.

NeRF, come discusso nella sezione 2.2.1, non soddisfa i requisiti di rendering interattivo. I tempi di inferenza per frame lo rendono inutilizzabile per la navigazione nel browser senza hardware dedicato lato server.

Il 3DGS risponde a entrambi questi vincoli. Le Gaussiane approssimano naturalmente strutture volumetriche diffuse come fogliame e vegetazione, senza richiedere una topologia manifold e gestendo implicitamente la traslucenza parziale delle foglie tramite il parametro α di ciascuna primitiva. Il renderer tile-based garantisce frequenze di aggiornamento compatibili con la navigazione interattiva anche in contesti web.

L'intera pipeline di acquisizione e produzione degli splat, dalle riprese video all'integrazione nel framework, includendo le scelte operative e i risultati ottenuti, sono descritti in dettaglio nel Capitolo 4.

Capitolo 3

Framework Web per Ambienti 3D Interattivi

La realizzazione di un ambiente tridimensionale navigabile via browser richiede la scelta di un'infrastruttura software in grado di gestire simultaneamente il rendering in tempo reale, la struttura della scena, l'interazione utente e, nel caso di applicazioni collaborative, la comunicazione tra client remoti. Questa scelta condiziona in modo profondo l'intera architettura del sistema: non riguarda solo le prestazioni grafiche, ma anche la modularità del codice, la compatibilità con i formati di contenuto, la possibilità di estensione e i vincoli imposti dal deployment su infrastrutture web standard.

Le soluzioni disponibili per la grafica 3D sul web si collocano lungo uno spettro che va da librerie di basso livello, assimilabili a wrapper di WebGL, fino a veri e propri engine completi con simulazione fisica, GUI e tooling integrato. È molto importante scegliere un framework che si adatti alle esigenze specifiche di progetto: alcuni più leggeri offrono maggiore controllo e flessibilità, ma richiedono più lavoro di integrazione; altri più completi accelerano lo sviluppo ma possono imporre vincoli architetturali che non si adattano a tutte le esigenze.

Le sezioni seguenti presentano le due librerie più diffuse nell'ecosistema Web3D attuale, Three.js e Babylon.js, e il framework ATON, sviluppato specificatamente per applicazioni nel campo dei beni culturali.

3.1 Panoramica dei framework web per grafica 3D

3.1.1 Three.js

Three.js [32] nasce nel 2010 da un progetto personale di Ricardo Cabello, con l'obiettivo di esporre un'interfaccia di alto livello sopra le API WebGL, la cui complessità rendeva molto difficile anche il rendering di geometrie semplici. La libreria racchiude il ciclo di rendering, la gestione del contesto WebGL e le operazioni comuni come proiezioni, trasformazioni e shading in un'API che si presta bene all'uso diretto da codice JavaScript o TypeScript senza configurazioni preliminari elaborate.

L'adozione è cresciuta rapidamente: Three.js è diventato uno dei framework più popolari per lo sviluppo di applicazioni 3D sul web ed è integrato in un'ampia varietà di progetti, dalla visualizzazione scientifica ai giochi browser-based, fino a configuratori di prodotto per l'e-commerce. Nel tempo si è accumulata una documentazione estesa, migliaia di esempi pubblici e un insieme di *add-on* ufficiali che estendono le funzionalità di base senza appesantire il bundle principale.

La libreria offre un sistema di scene-graph gerarchico tramite il quale oggetti, luci e telecamere vengono organizzati come nodi trasformabili. Il modello di illuminazione include sorgenti puntuali, direzionali, spot e ambientali, con il supporto al rendering *Physically Based* (PBR) attraverso i materiali `MeshStandardMaterial` e `MeshPhysicalMaterial`. La gestione dei formati 3D è affidata a un sistema di loader, con supporto nativo allo standard glTF, formato internazionale raccomandato per la distribuzione di contenuti 3D sul web. Three.js implementa anche il supporto alla specifica WebXR per sessioni di realtà virtuale e aumentata via browser.

L'architettura è deliberatamente minimalista: Three.js è un *motore di rendering*, non un engine completo. Ci sono infatti diverse funzionalità che richiedono l'intergrazione di librerie terze, tra queste: rilevamento delle collisioni, fisica dei corpi rigidi, sistemi GUI nel canvas o sincronizzazione multi-utente. Questa scelta di design lo rende massimamente flessibile ma aumenta il carico architetturale per applicazioni con requisiti complessi.

3.1.2 Babylon.js

Babylon.js [3] nasce nel 2013 come progetto di David Catuhe all'interno di Microsoft con una filosofia diversa da Three.js: non è infatti una libreria di rendering specializzata, ma un engine 3D completo distribuito direttamente per il web. La differenza principale rispetto al precedente si può notare nel set di funzionalità disponibili senza dipendenze esterne.

Il sistema di fisica è integrato tramite plugin per Cannon.js, Ammo.js, Oimo.js e nelle versioni più recenti Havok. La gestione delle collisioni e la simulazione di corpi rigidi sono configurabili con poche righe di codice. Il sistema di animazione copre animazioni scheletrali, deformazioni di mesh tra forme predefinite (morph target, usati ad esempio per espressioni facciali o deformazioni di oggetti) e l'animazione di qualsiasi proprietà numerica degli oggetti di scena, con supporto al blending tra clip e un editor delle sequenze accessibile dal pannello di debug integrato nel browser. La GUI è renderizzata direttamente nel canvas WebGL tramite il modulo `BabylonGUI`, eliminando la dipendenza da overlay HTML. WebXR è trattato come funzionalità di prima classe, con scene helper dedicate che semplificano la configurazione di esperienze immersive.

Dal punto di vista del rendering, il supporto PBR è disponibile nativamente attraverso il `PBRMaterial`, con un modello fisicamente accurato che include metalness, roughness, *Image-Based Lighting* (IBL) e ombre. Le versioni recenti hanno introdotto ombre IBL, area light e aggiornamenti al supporto Gaussian Splat nei formati `.spz` e PLY compresso.

Babylon.js mette a disposizione anche strumenti di sviluppo accessibili direttamente nel browser: il Playground per scrivere e condividere codice in tempo reale, la Sandbox per caricare e ispezionare modelli 3D e il Node Material Editor per costruire shader tramite un grafo visuale senza scrivere codice GLSL. L'insieme di questi strumenti, unito all'architettura più strutturata, la rende una scelta efficace per prototipi rapidi e per team che privilegiano la velocità di sviluppo rispetto al controllo fine sull'implementazione. Il numero di download è inferiore a quello di Three.js (circa 11.000 contro 1,8 milioni) e possiede in generale una base utenti più orientata a esperienze strutturate e applicazioni di tipo game-like, più che alla visualizzazione o all'integrazione in stack web generici.

3.2 Il framework ATON

ATON [10] è un framework open-source sviluppato e coordinato da Bruno Fanini presso il DHILab del CNR ISPC (Istituto di Scienze del Patrimonio Culturale), progettato specificamente per la creazione di applicazioni Web3D e WebXR nel dominio dei beni culturali. Il framework si costruisce sopra Node.js lato server e Three.js lato client, esponendo un'API di alto livello [9] che astrae la complessità della pipeline di rendering e della comunicazione tra componenti [12].

La filosofia progettuale si riassume nell'approccio *develop once, deploy everywhere*: una singola applicazione si adatta automaticamente a dispositivi mobili, desktop e visori VR/AR, senza richiedere installazioni aggiuntive da parte degli utenti finali. Questo è possibile grazie a un modulo di *device profiling* che rileva all'avvio le capacità dell'hardware, il tipo di connessione e i sensori disponibili, adattando di conseguenza l'interfaccia, il modello di navigazione e il livello di dettaglio del rendering.

Architettura server e deployment. La componente server gira su Node.js con un'architettura a microservizi basata su Express.js, socket.io e Passport.js. Il *Deployment Node* (DN) eroga contenuti 3D, gestisce l'autenticazione, coordina le sessioni collaborative e serve le web-application ospitate. L'architettura è scalabile, adatta a deployment su hardware locale o su infrastrutture cloud. Una REST API documentata consente l'integrazione con servizi esterni e piattaforme di ricerca. Il back-end di amministrazione, denominato *Shu*, permette agli autori autenticati di pubblicare e organizzare scene 3D tramite interfaccia web, senza interventi sul codice.

Dati: collezioni e scene. Il framework separa esplicitamente il concetto di *collezione*, intesa come insieme di asset web-compatibili (modelli glTF, panoramiche, audio, video), da quello di *scena*, che definisce la disposizione degli asset, la gerarchia della scena-graph, i punti di vista, le annotazioni semantiche e l'ambiente sonoro. Le scene sono descritte tramite file JSON compatti, generalmente compresi tra pochi byte e 100 kB, che possono essere modificati in modo incrementale tramite JSON patch. La separazione consente il riuso degli stessi asset in scene diverse, il caching efficiente lato browser e la duplicazione rapida di configurazioni alternative.

Sistema di rendering. Il rendering si basa su Three.js con estensioni specifiche per i beni culturali. ATON implementa un sistema di *light probe* multipli: ogni mesh viene assegnata alla probe più vicina, simulando l'illuminazione ambientale locale in modo più accurato rispetto ai sistemi con una singola probe globale adottati da molte piattaforme commerciali. Il formato glTF con compressione Draco è il principale formato di distribuzione per i modelli 3D, completato dal supporto agli *Cesium 3D Tiles* (standard OGC) per dataset massivi e multirisoluzioni, e ai formati Gaussian Splat `.spz`, `.splat` e `.sog`, recentemente inclusi. Il PBR è pienamente supportato, con materiali compatibili con le pipeline di esportazione da Blender, Maya e dai principali software di creazione di contenuti digitali 3D, nonché con i principali game engine, tra cui Unity e Unreal Engine.

Navigazione e query. Il modulo di navigazione si adatta automaticamente al dispositivo: orbita e *first-person* su desktop, multi-touch su mobile e modalità VR su visori. I punti di vista strutturati (POV) sono addressabili tramite ID univoco e consentono transizioni fluide con interpolazione del campo visivo. Il sistema di query 3D utilizza alberi BVH (*Bounding Volume Hierarchy*) per accelerare il ray-casting su geometrie complesse, mantenendo framerate interattivi anche su dispositivi mobili e in sessioni WebXR.

Sessioni collaborative. Il componente VRoadcast implementa sessioni collaborative sincrone via WebSocket, con trasmissione della posizione e orientamento degli utenti in tempo reale, audio peer-to-peer tramite WebRTC, chat testuale e broadcasting di modifiche alla scena. Un'API event-driven consente alle singole web-application di definire eventi di rete personalizzati senza interferire con le sessioni di altri applicativi ospitati sullo stesso nodo.

Estensibilità. ATON adotta un'architettura plug-and-play per le web-application: ogni applicazione risiede nella propria cartella sul *Deployment Node* e accede direttamente ai componenti client del framework. I *flare* sono moduli JavaScript caricabili dinamicamente in fase di avvio, progettati per estendere o personalizzare il comportamento dell'applicazione senza alterare il codice sorgente del framework. Ogni flare espone i propri handler e diventa disponibile all'applicazione dopo l'evento `AllFlaresReady`, secondo un ciclo di vita gestito da ATON.

Il front-end ufficiale *Hathor* è conforme al modello PWA (*Progressive Web Application*), uno standard web che consente l'installazione come applicazione nativa sul dispositivo e il funzionamento anche in assenza di connessione. Hathor offre navigazione tra punti di vista, gestione di livelli, annotazioni, misurazioni, ambienti illuminati e opzioni di condivisione QR code senza richiedere competenze di sviluppo per i contenuti di base. Per esigenze più avanzate, la struttura modulare permette sia l'estensione di Hathor che lo sviluppo di applicazioni completamente personalizzate.

3.3 Motivazione della scelta di ATON

La scelta di ATON non è stata presa a priori, ma è emersa dal confronto diretto tra i requisiti funzionali del sistema da realizzare e ciò che le soluzioni disponibili offrivano concretamente.

Il progetto DT-Certosa richiede un ambiente navigabile via browser, senza installazioni lato utente, in grado di presentare un modello tridimensionale complesso con materiali PBR, gestire percorsi guidati tra punti di vista specifici, associare a singole tombe dati biografici e multimediali, e supportare sessioni collaborative con distinzione dei ruoli tra presentatore e visitatore. Nessuno di questi requisiti, preso singolarmente, è impossibile da soddisfare con Three.js o Babylon.js. Il problema è l'integrazione: costruire su Three.js puro avrebbe richiesto l'implementazione da zero di gestione dei POV, sessioni multi-utente, annotazioni semantiche e sistema di autenticazione, aumentando significativamente il rischio architetturale. Babylon.js al contrario avrebbe offerto un engine più completo ma orientato a giochi e applicazioni game-like, senza le funzionalità specifiche per i beni culturali, come la gestione delle collezioni e le REST API per l'editoria di scene, già disponibili e documentate in ATON.

ATON è costruito su Three.js, il che significa che il layer di rendering è quello già analizzato nelle sezioni precedenti. Le estensioni introdotte da ATON sono state costruite su questa base senza sostituirla: l'API client rimane accessibile e questo ha permesso di reimplementare comportamenti del sistema di navigazione e di aggiungere logica personalizzata attraverso il meccanismo dei flare, senza toccare il codice sorgente del framework in modo estensivo. Ci sono, inoltre, tante altre funzionalità che si sono rivelate utili per DT-Certosa: il sistema di POV ha avuto un ruolo chiave nell'implementazione efficiente della

navigazione; il meccanismo di sessione collaborativa ha fornito la base per la modalità multi-utente; il supporto al formato glTF e alla pipeline di esportazione da Blender ha semplificato il passaggio dalla modellazione al deployment web. Infine, il supporto ai Gaussian Splat: questo formato è stato integrato solo recentemente all'interno del framework, ma nonostante si tratti di una funzionalità ancora in fase di consolidamento, ha rappresentato un elemento chiave per la visualizzazione nella vegetazione nel progetto DT-Certosa. Tutte queste integrazioni sono descritte nel Capitolo 4 e nel Capitolo 5.

A questo si aggiunge un aspetto che ha pesato quanto quelli tecnici: la natura open-source del framework (licenza GPL v3) e la sua specifica vocazione per i beni culturali. ATON è sviluppato e mantenuto da ricercatori del CNR ISPC, con un percorso di sviluppo orientato ai casi d'uso reali dell'heritage digitale (musei, siti archeologici, archivi) e con un'adozione documentata in progetti nazionali e internazionali. Questo garantisce che le funzionalità implementate rispondano a esigenze concrete del dominio, non a requisiti astratti, e che l'architettura sia stata validata in condizioni operative simili a quelle del progetto DT-Certosa.

Capitolo 4

DT-Certosa: Modellazione 3D

La seconda parte dell'elaborato si concentra sulla parte implementativa, nello specifico analizza le scelte progettuali e le soluzioni tecniche adottate per la realizzazione concreta del sistema.

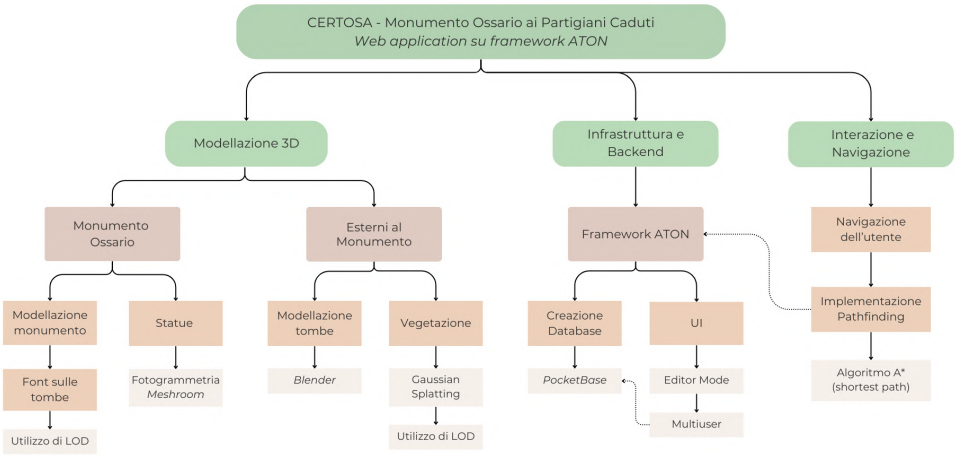

Figura 4.1: Mappa concettuale del progetto, articolata nei tre rami principali: modellazione tridimensionale, infrastruttura software e logica di interazione.

Per chiarire la struttura del lavoro, si fa riferimento alla mappa concettuale in Figura 4.1, che sintetizza l'intero progetto secondo tre rami principali: modellazione tridimensionale, infrastruttura software e logica di interazione. Tale suddivisione non introduce una gerarchia rigida, ma evidenzia le dipendenze

tra componenti che, pur trattate separatamente nei capitoli successivi, sono state progettate in modo strettamente integrato.

Il presente capitolo affronta il primo di questi ambiti, relativo allo sviluppo grafico 3D del progetto. Vengono descritte le attività di rilievo, la modellazione del Monumento Bottoni e delle aree esterne, nonchè le strategie adottate per gestire la complessità geometrica e garantire prestazioni compatibili con l'esecuzione in ambiente web. In particolare, la presenza di un numero elevato di elementi ripetitivi, come le iscrizioni sulle tombe, ha richiesto soluzioni differenziate in termini di rappresentazione e livelli di dettaglio, con un bilanciamento costante tra qualità visiva e costo computazionale. All'interno dello stesso capitolo si colloca la ricostruzione delle statue tramite fotogrammetria e l'adozione di una pipeline basata su Gaussian Splatting per la vegetazione. Anche in questo caso le scelte non sono state esclusivamente orientate alla resa visiva, ma condizionate dai vincoli imposti dall'integrazione nella webapp, dal peso dei dati trasferiti al client e dal comportamento dinamico della camera.

I restanti ambiti della mappa concettuale vengono approfonditi nel capitolo successivo. In particolare, la componente infrastrutturale analizza l'estensione del framework ATON, la progettazione del database tramite PocketBase e l'adattamento dell'interfaccia per supportare funzionalità applicative non previste nella configurazione di base. Tali interventi hanno richiesto modifiche mirate alla struttura del progetto e il coordinamento tra logica client e persistenza dei dati.

Infine, la logica di interazione viene trattata a partire dalla navigazione nello spazio virtuale, includendo la definizione dei percorsi, l'integrazione dell'algoritmo A* per il calcolo del cammino minimo e l'implementazione della modalità multi-utente con distinzione dei ruoli e sincronizzazione delle sessioni. Questa componente collega direttamente la struttura spaziale definita nel presente capitolo con la dimensione applicativa, rendendo il modello tridimensionale uno spazio effettivamente navigabile e condivisibile.

4.1 Monumento Bottoni: il punto di partenza

Il lavoro descritto in questo capitolo non è partito da zero. Esisteva già una ricostruzione tridimensionale del Monumento Ossario ai Partigiani di Bottoni,

sviluppata dal CINECA nel 2004 in collaborazione con il Comune di Bologna nell'ambito di un progetto più ampio dedicato alla Certosa di Bologna. In quel-

Figura 4.2: Progetto sviluppato dal CINECA nel 2004 in collaborazione con il Comune di Bologna.

la realizzazione, come si può notare dalle figure 4.2, era stato modellato il livello esterno del monumento con l'area circostante, mentre l'interno raccoglieva le sepolture dei partigiani bolognesi, ciascuna collegata a una scheda biografica. Dal punto di vista grafico erano state impiegate texture fotografiche, tecniche di Image Based Rendering (IBR) e livelli di dettaglio per mantenere prestazioni accettabili nell'ambiente interattivo, con strumenti diversi da Blender [4].

Questa versione, tuttavia, non costituisce il punto di partenza diretto della presente tesi. La geometria rifletteva strettamente i severi vincoli tecnici legati ai software dei primi anni 2000: agli albori della navigazione 3D sul web, né gli hardware né i framework di rendering erano in grado di gestire elevati carichi poligonali. Ne conseguiva la necessità assoluta di applicare pesanti ottimizzazioni sulla geometria, riducendo drasticamente il numero di poligoni e compensando il dettaglio tramite texture a risoluzione fortemente compressa, unico approccio tecnicamente praticabile in quel momento storico. Inoltre, a distanza di oltre vent'anni, l'area ha subito inevitabilmente lievi alterazioni dovute a restauri e ricognizioni, rendendo il modello storicamente significativo ma non più corrispondente allo stato di fatto attuale. Per queste ragioni, in una fase intermedia precedente a questo lavoro, il solo monumento centrale era stato rimodellato in Blender [4], con l'obiettivo di ottenere proporzioni più corrette e una geometria più fedele alla struttura reale.

Il punto di partenza concreto per questa tesi è stato quindi la combinazione di due elementi preesistenti: il modello più ampio derivante dal primissimo progetto CINECA, ottimizzato per i vincoli di rendering del periodo, utile

come riferimento planimetrico e per la disposizione degli elementi nell'area, e il modello del solo monumento centrale, geometricamente più accurato ma privo di texture e senza l'area circostante, come mostrato in Figura 4.3.

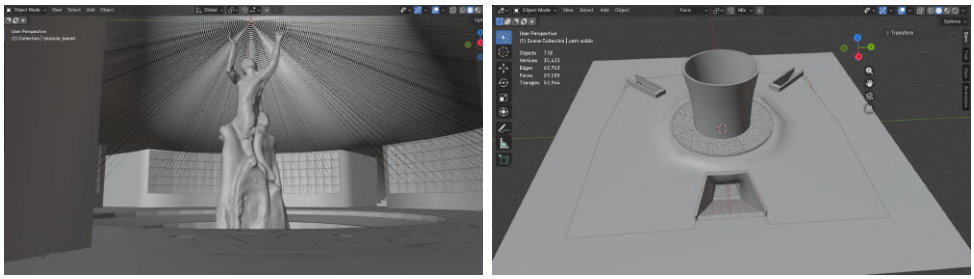

Figura 4.3: Rimodellazione più recente del solo monumento centrale in Blender

A partire da questa base, l'attività descritta nelle sezioni seguenti ha comportato un rilievo sistematico dell'area, la rimodellazione di porzioni significative della struttura, l'acquisizione e la lavorazione delle texture, la ricostruzione delle statue tramite fotogrammetria e la generazione della vegetazione tramite Gaussian Splatting (GS). L'obiettivo non era replicare il sistema preesistente con strumenti aggiornati, ma costruire un ambiente tridimensionale navigabile via web, ottimizzato per l'esecuzione in-browser e integrato con funzionalità applicative che nessuna delle versioni precedenti contemplava.

4.2 Rilevazioni del Monumento Bottoni

La realizzazione del modello tridimensionale dell'area circostante il Monumento Bottoni ha richiesto una fase preliminare di acquisizione dati articolata e progressivamente raffinata nel corso di più sopralluoghi presso la Certosa di Bologna. L'obiettivo non era limitato alla documentazione del solo monumento, ma comprendeva la ricostruzione coerente dell'intera area di interesse: le file di tombe davanti al monumento, le pavimentazioni, gli elementi architettonici accessori, le superfici marmoree, le componenti decorative, la vegetazione (cipressi, arbusti, piante ornamentali) e le statue collocate sia all'interno che in sommità della struttura.

4.2.1 Sopralluoghi e documentazione dell'area

Sono stati effettuati complessivamente quattro sopralluoghi. La prima campagna aveva carattere esplorativo: lo scopo era raccogliere materiale visivo ampio, senza ancora una delimitazione definitiva dell'area da includere nella web app. Man mano che la modellazione avanzava, è stato possibile definire con maggiore precisione quali porzioni includere, quali elementi richiedessero dettaglio accurato e quali informazioni fotografiche fossero davvero indispensabili. Il processo si è quindi definito in itinere, adattandosi alle esigenze che emergevano durante il lavoro.

I primi materiali acquisiti consistono in video generali dell'area. Non erano destinati alla generazione diretta di modelli o texture, ma servivano a capire come erano disposte le tombe, le distanze reciproche e le relazioni spaziali tra gli elementi. Piattaforme come Google Maps, Google Earth e le ortofoto del portale Open Data del Comune di Bologna [11] non restituivano un livello di dettaglio sufficiente per questo scopo: in particolare, Google Earth, come si può notare dalla figura 4.4, disponendo di visualizzazione tridimensionale, non raggiunge la risoluzione necessaria per verificare proporzioni e posizionamento delle singole strutture. I video fatti direttamente in loco hanno permesso di capire quali tombe fossero adiacenti, ricostruire l'andamento delle file e comprendere la configurazione complessiva dell'area.

L'area intorno al monumento ha una forma approssimabile a un triangolo isoscele, con una base anteriore, due lati obliqui retrostanti e una zona posteriore particolarmente densa di tombe. Nella fase iniziale sono state documentate anche le file posteriori e le porzioni laterali più estese; successivamente, tenendo conto del carico di lavoro per la modellazione manuale delle singole tombe e dei vincoli tecnici della web app, si è scelto di limitare l'inclusione alle file frontali rispetto alla base del triangolo. Parte del materiale acquisito non è stato quindi utilizzato nella versione finale, ma la raccolta più ampia è stata comunque utile per definire i confini effettivi del progetto.

4.2.2 Condizioni di ripresa

Un aspetto da gestire con attenzione è stato la scelta delle condizioni meteorologiche per le sessioni fotografiche. Le tombe sono prevalentemente realizzate in marmo o materiali lapidei con componenti semi-lucide o riflettenti: in presenza

Figura 4.4: Visione dall'alto dell'area su Google Earth sopra, ortofoto del portale Open Data del Comune di Bologna sotto.

di umidità o luce solare diretta, queste superfici generano riflessi speculari che compromettono la qualità delle texture, alterano il tono cromatico e rendono il materiale fotografico difficile da riutilizzare. Era quindi necessario evitare superfici bagnate, forti raggi di sole e contrasti marcati con ombre nette. La condizione migliore risultava una giornata nuvolosa con luce diffusa, che garantisce illuminazione uniforme senza highlight speculari. Questo vincolo ha condizionato la pianificazione dei sopralluoghi ed è rimasto valido sia per le foto delle tombe sia per le riprese video della vegetazione, utilizzate poi per il Gaussian Splatting (GS) [21].

4.2.3 Documentazione fotografica per la modellazione

Le Figure 4.5 mostrano esempi di foto scattate per la documentazione delle tombe. Per ciascuna tomba inclusa nel progetto sono state scattate tre tipologie di foto: una con angolazione laterale, che includesse la base, il corpo centrale ed eventuali elementi superiori con linee prospettiche ben leggibili, utile per la ricostruzione volumetrica tramite fSpy [14]; una frontale ortogona-

le per la texture; foto di dettaglio per rilievi, incisioni e decorazioni. Le foto per le texture venivano scattate frontalmente, in ombra, evitando vegetazione sovrapposta, foglie, detriti o superfici bagnate. Nel caso delle tombe a terra, spesso poco manutenute o parzialmente coperte da vegetazione spontanea, è stato necessario trovare porzioni di superficie abbastanza ampie e pulite da poter essere rielaborate digitalmente.

Oltre alle tombe, sono stati fotografati anche gli elementi del monumento stesso, (riferimento Figure 4.6): le lastre marmoree dell'ingresso, il granigliato della superficie dela struttura stessa, le superfici lapidee interne e le porzioni che nel modello preesistente risultavano prive di texture. Anche in questo caso si sono applicate le stesse accortezze in termini di illuminazione e angolazione.

Figura 4.5: Esempio di foto scattate per la documentazione delle tombe: a sinistra, angolazione laterale con linee prospettiche; a destra, foto frontale per la texture.

Figura 4.6: Foto scattate del monumento Bottoni

4.2.4 Acquisizione della vegetazione

Per la vegetazione l'obiettivo era generare modelli tridimensionali tramite GS a partire da sequenze video. La Figura 4.7 mostra un esempio di video acqui-

Figura 4.7: Frame di un video acquisito per la ricostruzione della vegetazione.

sito: Per ogni elemento è stato girato un video circolare a 360°, cercando di mantenere un movimento lento e continuo, senza cambi di direzione bruschi, con inquadratura stabile e copertura completa. è stato usato uno smartphone (iPhone), la cui stabilizzazione integrata si è rivelata sufficiente. In alcuni casi sono stati fatti due giri completi attorno alla pianta, per aumentare la copertura e ridurre il rischio di lacune nella ricostruzione.

Durante le prime acquisizioni sono emerse alcune problematiche legate alla distribuzione del dettaglio. Nel Gaussian Splatting la qualità della ricostruzione dipende molto da quanta informazione visiva viene raccolta per ciascuna parte dell'oggetto. Nel caso dei cipressi, piante con sviluppo molto verticale, inquadrare dal basso portava a concentrare la maggior parte delle informazioni sul tronco, con la chioma finale che risultava povera o quasi assente nella ricostruzione. Per i cespugli bassi il problema era opposto: riprendere dall'alto lasciava la parte inferiore con pochi dettagli. è stato quindi necessario bilanciare l'altezza di ripresa rispetto alla forma di ogni pianta, cercando di coprire uniformemente tutto il volume.

Sono stati acquisiti video di cipressi, cespugli compatti e più allungati, fiori e piante ornamentali. Questo aveva due scopi: testare come il Gaussian Splatting si comportasse su forme diverse e disporre di una libreria di elementi tra cui scegliere in base alla resa visiva e al peso computazionale. Non era chiaro fin dall'inizio quanti modelli sarebbero stati effettivamente necessari, quanti splat ATON avrebbe potuto gestire e se convenisse replicare pochi modelli o variarne molti. Alcuni video sono stati ripetuti in campagne successive perchè le prime riprese presentavano eccessiva esposizione luminosa, instabilità o inquadrature non bilanciate.

Un caso a parte ha riguardato due grandi cespugli che si sviluppano longitudinalmente davanti alla parte frontale del monumento. Riprenderli intero in un unico video avrebbe reso difficile mantenere una distanza costante e avrebbe prodotto un asset non replicabile in scena. Si è quindi deciso di riprendere uno o più cespugli più piccoli e compatti, generare i relativi splat e assemblarli poi in un secondo momento per ricostruire i cespugli lunghi come composizione di più moduli ripetuti, scalati e ruotati. La scelta è emersa durante la modellazione, non era prevista in fase di acquisizione.

4.2.5 Tentativo di ricostruzione globale tramite drone

Durante uno dei sopralluoghi è stato usato un drone con due scopi: riprendere l'area dall'alto per capire meglio la disposizione spaziale e tentare di generare un GS globale dell'intera zona. L'idea era produrre una ricostruzione navigabile da usare come riferimento volumetrico durante la modellazione in Blender.

Il tentativo non ha dato risultati utilizzabili. I movimenti avanti-indietro del drone, le rotazioni durante la ripresa e i cambi di direzione bruschi hanno reso la traiettoria troppo irregolare sia per la pipeline fotogrammetrica di Meshroom che per il tool di Gaussian Splatting: in entrambi i casi i punti caratteristici non venivano riconosciuti in modo coerente lungo la sequenza, producendo strutture frammentate e inutilizzabili. Anche suddividendo il video in segmenti più brevi i risultati sono rimasti insoddisfacenti e l'idea della ricostruzione globale è stata abbandonata. Le riprese del drone sono comunque rimaste utili come riferimento visivo per la modellazione delle tombe e per capire le relazioni spaziali dall'alto.

4.2.6 Acquisizione fotogrammetrica delle statue

Le statue sono state trattate con un approccio diverso rispetto alla vegetazione: invece del Gaussian Splatting si è usata la fotogrammetria, con Meshroom [2] del framework AliceVision [1] per la generazione delle mesh. La pipeline fotogrammetrica è descritta in dettaglio nella Sezione 4.4.1; qui si riportano le modalità di acquisizione e le difficoltà incontrate in loco.

Le statue presenti nell'area includono una statua già ricostruita nella fase precedente del progetto che non è stata poi modificata (mostrata gia precedentemente nel riferimento 4.3), una statua in cima al monumento, un gruppo scultoreo anche questo sulla sommità e due figure nella parte superiore interna: una appoggiata al muro e una in posizione sospesa.

I primi tentativi per le statue superiori sono stati fatti da terra con lo smartphone, ma la prospettiva troppo inclinata, la distanza eccessiva e l'impossibilità di girare a 360°attorno all'oggetto rendevano il materiale poco utilizzabile. Per la statua interna appoggiata al muro è stato invece possibile fare una ripresa semicircolare di circa 180°, sfruttando la base circolare interna del monumento e ruotando attorno al punto centrale: essendo la statua aderente al muro, non era necessaria la copertura posteriore. La dinamica della ripresa è mostrata nella Figura 4.8.

Figura 4.8: Frame della statua interna appoggiata al muro, ripresa con smartphone da terra.

Durante la ripresa della statua interna è emerso un problema legato all'uso dello zoom: alcune porzioni del video hanno dovuto essere scartate perché alteravano la coerenza dei parametri interni della camera, compromettendo la ricostruzione. La questione viene approfondita nella Sezione 4.4.1.

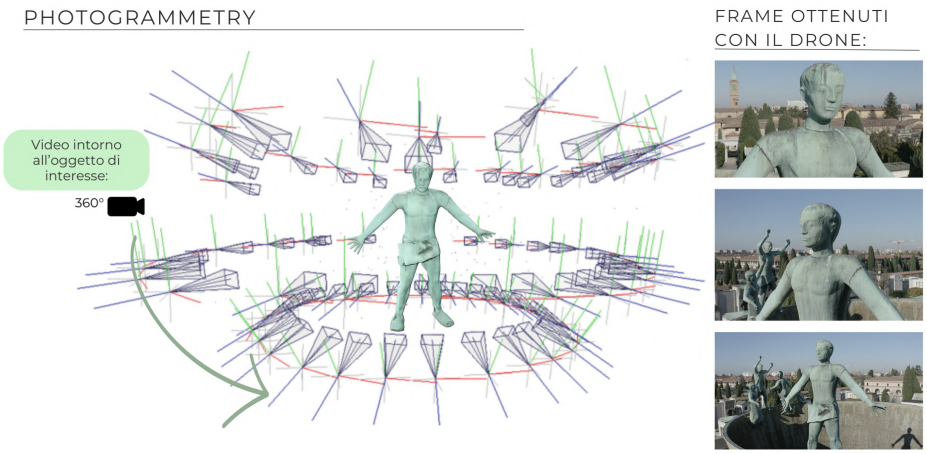

Figura 4.9: Immagini delle statue sulla sommità del monumento, riprese con drone.

Per le statue in sommità è stato necessario usare il drone, che ha permesso di operare all'altezza della statua con rotazioni complete a 360°, distanza costante e angolazione coerente lungo tutta la sequenza. Come mostrano le Figure 4.9, questi video erano ad una risoluzione più elevata rispetto a quelli fatti da terra, I video sono stati poi elaborati in Meshroom per la generazione della mesh.

Per la statua interna in posizione sospesa il drone non era praticabile: lo spazio ristretto, i sensori anticollisione, i limiti di quota imposti per sicurezza e il rischio concreto di urto con la struttura ne hanno impedito l'uso. La copertura di questa statua è rimasta quindi parziale rispetto alle altre.

4.3 Modellazione dell'Ossario

L'ossario costituisce il nucleo architettonico principale del Monumento Bottoni. La struttura interna ha una pianta circolare e raccoglie le sepolture dei partigiani bolognesi lungo la parete perimetrale interna, suddivisa in circa 500 nicchie sepolcrali. Ogni sepoltura è identificata da una lapide in materiale la-

pideo di dimensioni standardizzate (30 cm × 30 cm), denominata tombino nel contesto del modello tridimensionale.

Le lapidi sono disposte in colonne verticali e seguono uno schema di numerazione progressiva a serpentina: la sequenza parte dalla tomba nell'angolo superiore sinistro (tombino 1), percorre la prima colonna verso il basso, risale nella colonna successiva e prosegue così fino a coprire l'intera parete. Questa organizzazione numerica è rilevante perché la numerazione nel modello tridimensionale corrisponde direttamente all'indicizzazione nel database dei partigiani, rendendo possibile una corrispondenza diretta tra geometria e dati informativi.

Nel modello tridimensionale di partenza i tombini erano già presenti come piani geometrici posizionati correttamente secondo la disposizione radiale della struttura, ma privi di qualsiasi iscrizione: ogni oggetto aveva associato un identificativo numerico, tuttavia la superficie appariva completamente bianca. L'attività descritta nella sezione seguente si è basata su questo presupposto, si sono integrate le iscrizioni coi nomi dei partigiani tramite generazione automatica e si è ottimizzata la rappresentazione per il rendering in tempo reale in ambiente web.

4.3.1 Modellazione dei testi delle tombe

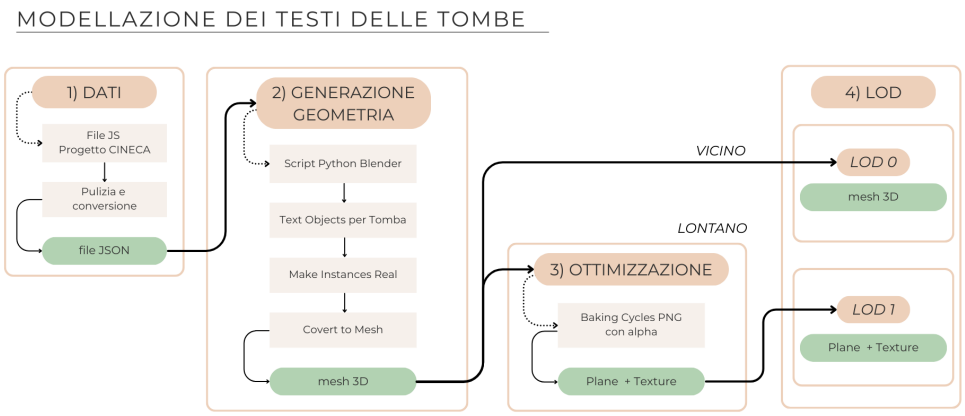

Figura 4.10: Mappa sulla modellazione delle iscrizioni.

La Figura 4.10 riassume la sequenza complessiva delle operazioni svolte, dalla preparazione dei dati alla generazione automatica delle iscrizioni fino al-

l'ottimizzazione per il rendering.

La realizzazione delle iscrizioni ha richiesto di affrontare un problema cercando di bilanciare due elementi principali. Il primo riguarda la qualità e la coerenza dei dati storici di partenza; il secondo riguarda la gestione efficiente di circa 500 oggetti testo in una scena destinata alla visualizzazione in-browser su hardware eterogeneo. Le due componenti si intrecciano costantemente lungo tutto il processo: le scelte sulla rappresentazione influenzano i requisiti sui dati e le limitazioni tecniche di Blender durante la generazione hanno condizionato le strategie di ottimizzazione adottate.

Trattandosi di circa 500 tombini, ciascuno con iscrizioni potenzialmente diverse per numero di nominativi, presenza della data e formattazione, la modellazione manuale era da escludere fin dall'inizio. Procedere tombino per tombino avrebbe richiesto un tempo incompatibile con i ritmi del progetto e avrebbe introdotto inevitabili incongruenze nella formattazione. La scelta è ricaduta su uno script Python eseguito direttamente all'interno del Text Editor di Blender [4], che consente di accedere all'API `bpy` e di operare sulla scena in modo programmatico. Questo approccio permette di applicare le stesse regole tipografiche in modo uniforme a tutti i tombini, gestire i casi particolari tramite logica condizionale e ripetere l'intera generazione in caso di correzioni o modifiche senza reinserire manualmente il contenuto. Affinché lo script potesse operare correttamente, era però necessario disporre dei dati in un formato leggibile da Python: il punto di partenza era un file JavaScript, non direttamente importabile, che ha richiesto una fase preliminare di pulizia e conversione.

Dati di partenza. Il sistema informativo preesistente era strutturato come un file JavaScript contenente due oggetti distinti: il primo mappava ogni numero di tombino ai nominativi delle persone corrispondenti, il secondo raccoglieva, per ciascuna persona, dati biografici completi tra cui nome, nome di battaglia, data di nascita, data di morte e link alla scheda biografica sul portale del Comune di Bologna.

Per la generazione delle iscrizioni erano rilevanti tre campi: `nome`, `tombino` e `data_morte`. Il file JavaScript originale non era direttamente importabile da Python in Blender a causa della sintassi JavaScript e di alcune irregolarità nella struttura. È stata quindi eseguita una fase di pulizia e conversione in formato JSON, estraendo i soli campi necessari, normalizzando le stringhe e

correggendo i casi che avrebbero causato errori di parsing. Il Listato che segue mostra un estratto del file risultante. Va notato che il numero di tombino non è univoco: più persone possono condividere la stessa lapide, quindi lo stesso valore di `tombino` compare in voci distinte.

```
{
  "ALBERTAZZI BRUNO Pippo": {
    "nome": "BRUNO ALBERTAZZI",
    "tombino": 476,
    "data_morte": "10 Aprile 1945"
  },
  "ARBIZZANI TRISTANO": {
    "nome": "TRISTANO ARBIZZANI",
    "tombino": 461,
    "data_morte": "24 Marzo 1945"
  },
  "ARIATI GILBERTO": {
    "nome": "GILBERTO ARIATI",
    "tombino": 461,
    "data_morte": "22 Settembre 1943"
  }
}
```

Listing 4.1: Estratto del file JSON delle sepolture (`db2_clean.json`). label

Analisi preliminare e correzione dei dati. Prima di avviare la generazione automatica, è stata condotta un'analisi delle iscrizioni reali tramite fotografie disponibili online e materiale acquisito durante i sopralluoghi. L'obiettivo era individuare le convenzioni tipografiche ricorrenti sulle lapidi e ricavare regole formali da implementare nello script.

Dall'analisi delle immagini è emersa una struttura abbastanza coerente: nel caso di una sola persona, l'iscrizione riporta il nome sulla prima riga e il cognome sulla seconda, spesso seguiti dalla data di morte; nei casi con più persone sulla stessa lapide, le iscrizioni reali utilizzano una forma abbreviata con l'iniziale del nome seguita dal cognome, ripetuta per ogni persona su righe successive.

Parallelamente al confronto tra fotografie e database, sono emerse delle discrepanze: alcune corrispondenze tra numero di tombino e nominativo nel file erano errate, con persone associate a tombini diversi da quelli reali.

Queste incongruenze sono state annotate manualmente e corrette prima della generazione.

Un ulteriore aspetto critico riguardava la presenza della data sulle lapidi reali rispetto a quanto riportava il database: in numerosi casi la data compariva sulla lapide anche quando il campo `data_morte` risultava vuoto nel database, e in altri casi accadeva il contrario. Non era quindi possibile usare la semplice presenza del campo nel database come criterio per decidere se includere o meno la data nell'iscrizione generata. La soluzione adottata è consistita nel rilevare manualmente, lapide per lapide, quali tombini recassero effettivamente la data, costruendo una lista esplicita di esclusione (`NO_DATE_LIST`) da incorporare nello script come dato di configurazione.

Regole di generazione delle iscrizioni. Le regole implementate nello script di generazione coprono l'insieme dei casi effettivamente presenti nel database e nelle lapidi reali:

- se il tombino non ha corrispondenza nel database, la lapide rimane vuota;
- nel caso generale (persona singola), il campo `nome` nel database riporta i token nell'ordine cognome-nome; vengono presi solo i primi due token e applicato uno swap, così da ottenere il formato nome-cognome su righe successive; eventuali contenuti tra parentesi tonde vengono ignorati (ad esempio `CRISKA (STALIN) PIETRO` produce `PIETRO CRISKA`);
- se il tombino è incluso nella `NO_DATE_LIST` (che comprende i tombini 33, 43–45, 50, 53–54, 57, 59, 65, 86, 88, 93–94, 109, 113, 118, 120–121, 125–127, 132, 134, 150, 154–156, 161, 165, 177, 179, 200, 203, 208, 211, 216, 222, 231, 242, 257–259, 265, 278, 284, 290, 305, 312–315, 344, 346–347, 350–351, 357, 362–363, 367–368, 375, 377, 380, 384, 388, 391, 398, 414 e il range 461–490), viene riportato solo il nominativo senza data;
- se il tombino non è nell'elenco di esclusione e il campo `data_morte` è presente, la data viene riformattata in notazione numerica (giorno.mese.anno) e aggiunta dopo il nominativo;
- nei casi in cui il nome contenga la dicitura "SCONOSCIUTO" (nelle varianti SCONOSCIUTO DI SESSO MASCHILE, SCONOSCIUTO

RUSSO, SCONOSCIUTO UOMO 2), l'iscrizione generata è "SCONOSCIUTO" sulla prima riga e "UOMO" sulla seconda, ignorando qualsiasi qualificazione aggiuntiva;

- se più persone condividono lo stesso tombino, i relativi nominativi vengono scritti consecutivamente (nome e cognome su righe separate per ciascuno), senza data indipendentemente dalla `NO_DATE_LIST`;

- il tombino 458 costituisce un caso distinto: corrisponde a resti recuperati nel campo di sterminio di Gusen e l'iscrizione ("RESTI RECUPERATI / NEL CAMPO DI / STERMINIO / DI GUSEN / 27.9.1960") è definita in modo statico nello script.

Font e template tipografico. Per la resa visiva dei testi sono stati usati i Text Object di Blender [4]: ogni carattere dell'alfabeto, le cifre numeriche e il punto separatore per le date sono stati modellati come mesh tridimensionali separate, raccolte in un'unica famiglia tipografica personalizzata (gli oggetti sono denominati `Lettera.A`, `Lettera.B`, ..., `Lettera.0`, `Lettera.1`, `Lettera./` e così via). Questo approccio produce caratteri con rilievo tridimensionale coerenti con lo stile inciso delle iscrizioni reali.

Il font è stato configurato su un oggetto di testo denominato `Text_example`, impostando dimensione, spaziatura tra i caratteri e materiale per replicare l'aspetto delle lapidi reali. Questo oggetto non è destinato alla scena finale ma funge da template di riferimento: tutti i testi generati automaticamente vengono ottenuti per duplicazione di `Text_example`, di cui ereditano le proprietà tipografiche senza dover riconfigurare ogni parametro da zero.

Generazione automatica tramite script Python. Con circa 500 tombini la modellazione manuale delle iscrizioni era impraticabile. Come spiegato sopra, lo script principale è stato sviluppato e gestito all'interno del Text Editor di Blender [4], dove è stato anche caricato il file JSON come Text Block interno, consentendo l'accesso ai dati senza gestire percorsi di file esterni.

Il flusso di esecuzione si articola come segue. Lo script carica il database dal Text Block tramite `bpy.data.texts`, costruisce una mappa che raggruppa le voci per numero di tombino, itera sugli oggetti della scena il cui nome corrisponde al pattern `tombino_N` e, per ciascuno, duplica il template `Text_example`,

imposta il campo `body` con la stringa generata secondo le regole sopra descritte, e chiama la funzione di posizionamento.

Le funzioni principali dello script, `build_text` per la costruzione del contenuto testuale e `place_text` per il posizionamento sulla lapide, sono riportate nel Listato A.1 in Appendice A.1.

Il posizionamento si basa sulla matrice di trasformazione mondo del tombino (`matrix_world`): la traslazione fornisce il centro della lapide e la normale locale (colonna Z della matrice) determina la direzione di offset. Il testo viene spostato di 5 mm in avanti rispetto alla superficie per evitare z-fighting. Il testo viene inoltre scalato automaticamente per rimanere all'interno di una bounding box di $0.28\,\mathrm{m} \times 0.28\,\mathrm{m}$, corrispondente all'area utile della lapide.

Difficoltà di rotazione. Una criticità sistematica ha riguardato l'orientamento dei testi generati. Alcune file di tombini nel modello preesistente presentavano rotazioni di 90° o 180° sull'asse locale, dovute alla struttura gerarchica del monumento a pianta radiale. I primi tentativi, basati sulla semplice copia della rotazione locale del tombino, producevano testi ruotati o capovolti su quelle file. Il problema non era nella logica di posizionamento in sé, ma nel fatto che le rotazioni locali dei tombini figli non coincidevano con la rotazione effettiva nello spazio mondo. Si è dunque arrivati alla seguente soluzione: invece di copiare la rotazione locale, si è ricavata la componente Z della rotazione direttamente dalla matrice mondo tramite `matrix_world.to_euler('XYZ').z`, che restituisce la rotazione effettiva indipendentemente dalla gerarchia di parentage. Applicando questa componente alla rotazione del testo, insieme alla rotazione fissa di 90° sull'asse X necessaria per orientare il piano del testo verso la camera, il posizionamento ha funzionato correttamente su tutti i tombini.

Conversione in geometria reale. I Text Object di Blender [4] generati dallo script non producono geometria poligonale diretta: ogni carattere è un'istanza della mesh corrispondente nella famiglia tipografica. Questa struttura è comoda in fase di editing, ma introduce due problemi concreti.

Il primo è di ordine prestazionale: con circa 500 tombini, ciascuno contenente in media più lettere, il numero totale di istanze attive nella scena raggiungeva approssimativamente 9500 elementi. Blender tendeva a rallentare significativamente o andare in crash durante le operazioni di modifica, rendendo necessario suddividere il lavoro in sessioni su sottoinsiemi della scena.

Il secondo problema è di ordine tecnico: il formato GLB utilizzato per il framework di visualizzazione non gestisce correttamente oggetti di tipo FONT con istanze tipografiche; è necessaria una mesh poligonale esplicita per l'esportazione.

Per entrambi i motivi è stata eseguita la conversione dei Text Object in mesh reali. Per ciascun oggetto la procedura segue tre passi sequenziali:

- *Make Instances Real*: sostituisce ogni istanza tipografica con un oggetto mesh indipendente;
- *Make Single User*: rende i dati di ciascun oggetto univoci e non condivisi;
- *Convert to Mesh*: converte i tipi FONT in triangolazioni poligonali esplicite.

Tutti gli oggetti relativi allo stesso tombino vengono infine uniti tramite *Join* in un'unica mesh denominata `font_tombino_N`.

A causa del costo computazionale, la procedura non è stata eseguita sull'intera scena in un'unica operazione: i tombini sono stati raggruppati in collection con range numerici (`1-26`, `27-60`, `311-331` e così via) e il processo è stato ripetuto sequenzialmente per ciascun gruppo. Il Listato A.2 in Appendice A.1 riporta lo script che gestisce questa conversione.

L'effetto finale ottenuto è quello mostrato nella Figura 4.11: ogni iscrizione è ora una mesh poligonale, con un numero di triangoli variabile a seconda della complessità del testo.

Figura 4.11: LOD0: Mesh font modellati poligonalmente.

Ottimizzazione tramite texture baking. Anche dopo la conversione, il numero complessivo di poligoni rimaneva elevato: ogni lettera è una geometria tridimensionale con curvature che richiedono una mesh densa. Con 500 tombini il costo di rendering complessivo risultava incompatibile con i requisiti della web app, dove poligoni e draw call devono restare entro limiti precisi.

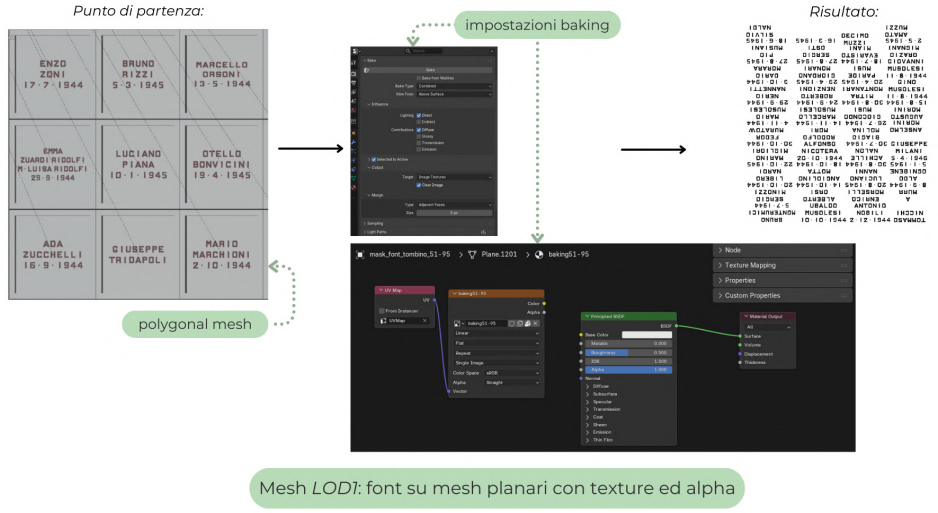

Figura 4.12: LOD1: font su mesh planari e relative impostazioni di baking su Blender.

La soluzione adottata consiste nel trasferire le informazioni visive delle iscrizioni all'interno di texture generate tramite *baking*. Per ciascun tombino viene creato un piano geometrico semplice posizionato esattamente sopra la mesh tridimensionale del testo, con la stessa posizione, rotazione e scala. Il contenuto visivo della mesh viene proiettato su questo piano tramite il renderer Cycles di Blender [4], producendo un'immagine raster che codifica le informazioni geometriche della scrittura come valori di colore su una superficie piana. Si possono vedere le configurazioni di baking nella Figura 4.12.

Le texture risultanti vengono salvate in formato PNG con canale alpha: l'alpha permette di rappresentare esclusivamente le zone visibili del testo, lasciando trasparenti le aree circostanti prive di contenuto grafico. Il piano geometrico sostituisce così la mesh delle lettere: le zone trasparenti nascondono la superficie nelle aree vuote, e la resa visiva risulta equivalente, ma con un costo computazionale nettamente inferiore.

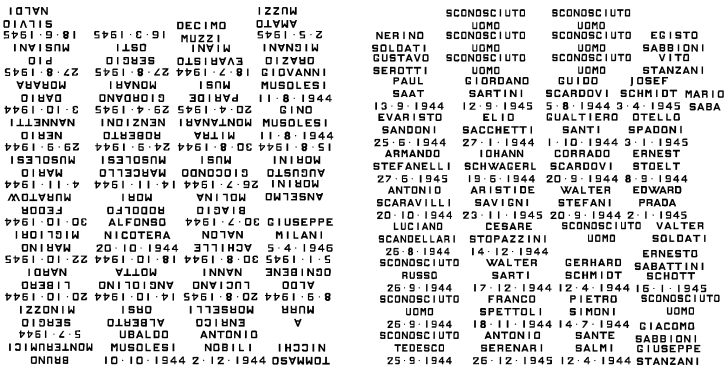

Figura 4.13: Baking di alcune texture.

Un aspetto rilevante riguarda l'organizzazione del baking. Generare una texture per ciascuno dei 500 tombini avrebbe moltiplicato le operazioni in produzione e i file da caricare a runtime. Per questo motivo i tombini sono stati raggruppati per collection di lavorazione: per ogni gruppo viene generata un'unica texture che raccoglie più iscrizioni, riducendo il numero totale di texture da alcune centinaia a qualche decina. Il risultato è visibile nelle Figure 4.13. Questa strategia riduce sia il costo della produzione sia il volume di memoria occupato dalle iscrizioni a runtime nel browser.

Sistema di Level of Detail. Per le iscrizioni delle tombe è stato adottato uno schema di *Level of Detail* (LOD) a due livelli discreti (Discrete LOD). Nel Discrete LOD la scena mantiene un insieme finito di versioni precompilate per ogni oggetto e seleziona quella attiva in base a una soglia di distanza fissa dall'osservatore. A differenza del Progressive LOD, che aggiorna la geometria in modo continuo riducendo progressivamente il numero di triangoli, il Discrete LOD prevede uno switch netto tra versioni distinte: il costo di transizione è inferiore, ma la variazione è potenzialmente percepibile al momento del cambio di soglia.

Nel caso delle iscrizioni la Figura 4.14 mostra bene la differenza tra questi due livell, i quali si distinguono come segue:

- **LOD 0**: la mesh tridimensionale ad alta risoluzione delle lettere, caricata quando l'osservatore si trova in prossimità della sepoltura, dove il dettaglio geometrico è visivamente percepibile;

- **LOD 1**: il piano con texture baked in formato PNG con canale alpha, attivato a distanza maggiore. A quella distanza la differenza visiva rispetto alla mesh tridimensionale non è percepibile, e il risparmio computazionale è sostanziale.

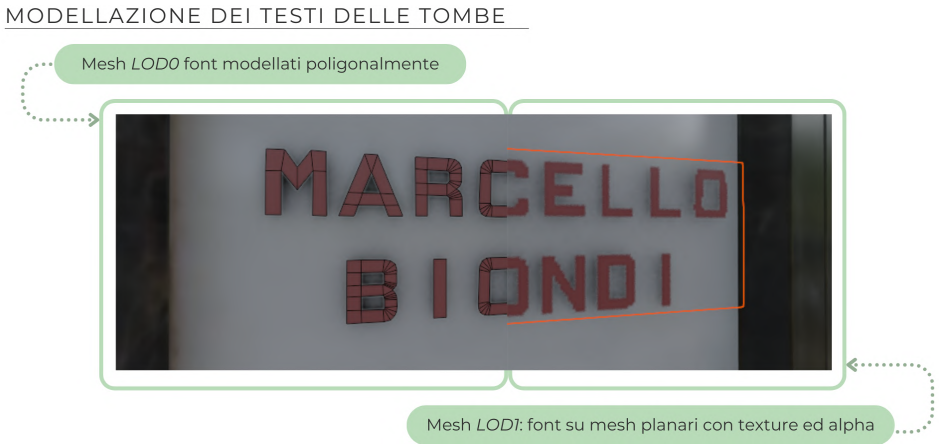

Figura 4.14: Confronto tra mesh tridimensionale e piano con texture baked.

La logica di selezione del livello attivo e la gestione dello swap vengono implementate nel framework ATON e vengono descritte nel Capitolo 5.

4.4 Modellazione delle Statue

Il Monumento Bottoni ospita un insieme di statue la cui complessità geometrica rende impraticabile una riproduzione accurata tramite modellazione poligonale manuale. Volti, mani e panneggi sono superfici che richiedono una densità poligonale elevata e una fedeltà geometrica difficile da raggiungere manualmente. Per questo si è scelto di usare la fotogrammetria, una tecnica che ricava la geometria tridimensionale direttamente da sequenze fotografiche acquisite attorno al soggetto.

La Figura 4.15 riassume la pipeline seguita per ciascuna statua o gruppo di statue: dall'acquisizione video in loco, attraverso la selezione dei keyframe e la ricostruzione in Meshroom, fino alle fasi di ottimizzazione manuale in Blender e all'esportazione dell'asset finale in formato GLB per l'integrazione nel framework ATON.

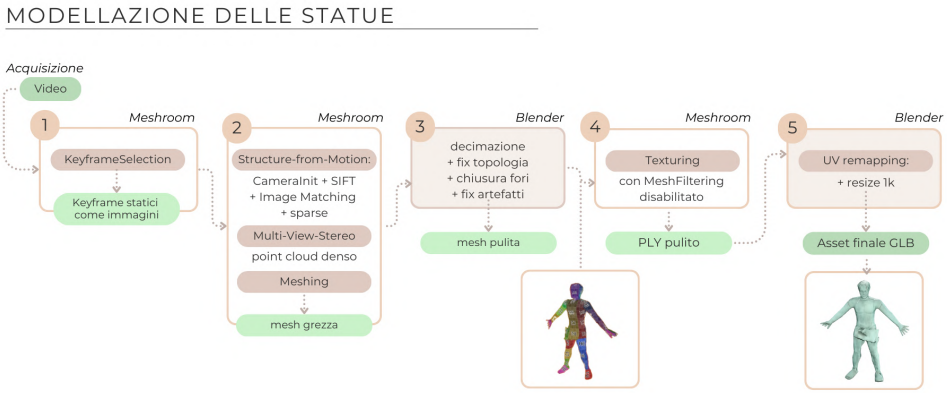

Figura 4.15: Sviluppo della modellazione delle statue.

4.4.1 Fotogrammetria digitale e selezione dei keyframe da sequenze video

La fotogrammetria digitale ricostruisce la geometria tridimensionale di un oggetto a partire da un insieme di immagini bidimensionali acquisite da punti di vista differenti, sfruttando la corrispondenza di aree visive comuni tra fotogrammi. La pipeline si articola in due fasi principali: Structure-from-Motion (SfM), che stima la posa delle camere e produce una nuvola di punti sparsa, e Multi-View Stereo (MVS), che calcola mappe di profondità per ciascuna vista e le fonde in una mesh densa. Nell'implementazione adottata, l'intero flusso gira su Meshroom [2], il software open source del framework AliceVision [1]. Meshroom organizza la pipeline come un grafo di nodi configurabili, ognuno responsabile di una fase specifica: dall'estrazione delle feature tramite SIFT (Scale-Invariant Feature Transform) fino al texturing finale tramite proiezione multi-vista.

L'acquisizione delle immagini per le statue del Monumento Bottoni avviene tramite sequenze video, scelta che garantisce copertura continua delle superfici e flessibilità operativa in un contesto architettonico con vincoli di accessibilità. Per le sculture collocate in posizione elevata si impiega, come affrontato precedentemente, un drone, che consente di ottenere angolazioni altrimenti non raggiungibili da terra.

L'utilizzo di video introduce però una criticità: alcune sequenze contenevano variazioni di zoom, adottate per catturare dettagli su aree specifiche come i volti. Lo zoom altera la lunghezza focale frame per frame e gli algoritmi di SfM

assumono parametri interni costanti lungo tutta la sequenza. Il risultato sono deformazioni geometriche e artefatti visibili nelle fasi di meshing e texturing. I primi test di ricostruzione con estrazione diretta dei frame hanno confermato il problema.

La soluzione consiste nell'introdurre a monte della pipeline il nodo KeyframeSelection di Meshroom [2], che analizza il video e seleziona automaticamente un sottoinsieme di frame sulla base di criteri visivi: diversità di contenuto tra frame consecutivi, indice di nitidezza e copertura regolare della sequenza. Rispetto a una semplice estrazione a intervalli temporali fissi, questa selezione elimina frame ridondanti, scarta quelli affetti da motion blur e riduce l'impatto delle variazioni di zoom scegliendo i frame più stabili. Dal punto di vista operativo, il nodo KeyframeSelection non si collega direttamente al nodo CameraInit perchè il numero di frame estratti non è noto a priori: i keyframe vengono salvati su disco e reimportati manualmente come immagini statiche, costituendo l'input effettivo della pipeline Structure-from-Motion + MVS.

Al termine dell'acquisizione e della selezione dei frame, la pipeline di Meshroom [2] procede attraverso i nodi Structure-from-Motion e Multi-View Stereo fino al nodo *Meshing*, che genera una superficie triangolare a partire dalla nuvola di punti densa. I nodi successivi nella configurazione standard sono *Mesh Filtering*, che applica operazioni automatiche di pulizia geometrica, e *Texturing*, che proietta il colore delle immagini originali sulla superficie ricostruita. Questa struttura a grafo di nodi, caratteristica di Meshroom, permette di accedere ai risultati intermedi di ogni fase tramite la directory `MeshroomCache`, e di sostituire manualmente l'output di un nodo specifico senza dover ripetere l'intera ricostruzione.

Criticità della mesh grezza e intervento manuale. La mesh generata automaticamente dal nodo Meshing presentava sistematicamente alcune criticità che ne rendevano necessaria la lavorazione. Il numero di vertici e triangoli risultava molto elevato per via della densità di ricostruzione, con conseguente peso computazionale eccessivo per un contesto real-time. Erano presenti irregolarità locali, artefatti superficiali e micro-frammentazioni. Le porzioni più complesse come mani, piedi e dettagli anatomici piccoli risultavano spesso imprecise o mal definite, nelle stesse zone dove la copertura fotografica era

più irregolare. Erano presenti inoltre discontinuità superficiali e piccoli fori, in corrispondenza di zone occluse o con transizioni di materiale che l'algoritmo non riusciva a ricostruire correttamente.

Per queste ragioni si è scelto di interrompere la pipeline dopo il nodo Meshing, prima della texturizzazione, ed esportare la mesh grezza per un intervento manuale in Blender [4]. Le operazioni eseguite sono state:

- decimazione controllata della superficie, per ridurre la complessità poligonale mantenendo la coerenza morfologica;
- chiusura dei fori e correzione delle discontinuità topologiche;
- rifinitura manuale delle porzioni più critiche, con attenzione alle aree di minore qualità;
- rimozione di artefatti e geometrie spurie;
- ricostruzione della mappa UV con *Unwrap* standard in Edit Mode, definendo manualmente i *seam* per ottenere isole più stabili e contenere le distorsioni sulle facce.

Gestione dell'UV mapping e ottimizzazione della texture. Un aspetto critico riguardava l'UV mapping automatico prodotto dal nodo Texturing. Meshroom genera per default una texture ad alta risoluzione (4K, 4096×4096 px), ma l'analisi del layout UV ha evidenziato che le isole UV occupavano solo una porzione limitata dell'area disponibile, con una distribuzione non ottimizzata in termini di densità texel. Ridurre la texture da 4K a 1K (1024×1024 px) comportava perciò una perdita qualitativa significativa: l'informazione utile era concentrata in una frazione ridotta dell'immagine, e la risoluzione effettiva per unità di superficie risultava molto inferiore a quanto il formato 1K avrebbe teoricamente consentito.

Per ovviare al problema si è proceduto in Blender a un remapping completo delle coordinate UV, ridistribuendo le isole sull'intera superficie disponibile e massimizzando l'occupazione dell'area texture.

Una volta ottenuta la mesh low-poly con UV ottimizzato, il file è stato esportato e reinserito nella pipeline di Meshroom, sostituendo il file prodotto automaticamente dal nodo Meshing con la stessa struttura di percorso e

lo stesso nome, in modo che i nodi successivi lo ricevessero come input. Il nodo Mesh Filtering è stato disattivato: lasciandolo attivo avrebbe rielaborato la mesh secondo criteri automatici, annullando le correzioni già applicate manualmente. Il nodo Meshing è stato quindi collegato direttamente al nodo Texturing, saltando la fase di filtraggio, come mostrato in Figura 4.16 e 4.17.

Figura 4.16: Pipeline finale utilizzata per la ricostruzione 3D con Meshroom.

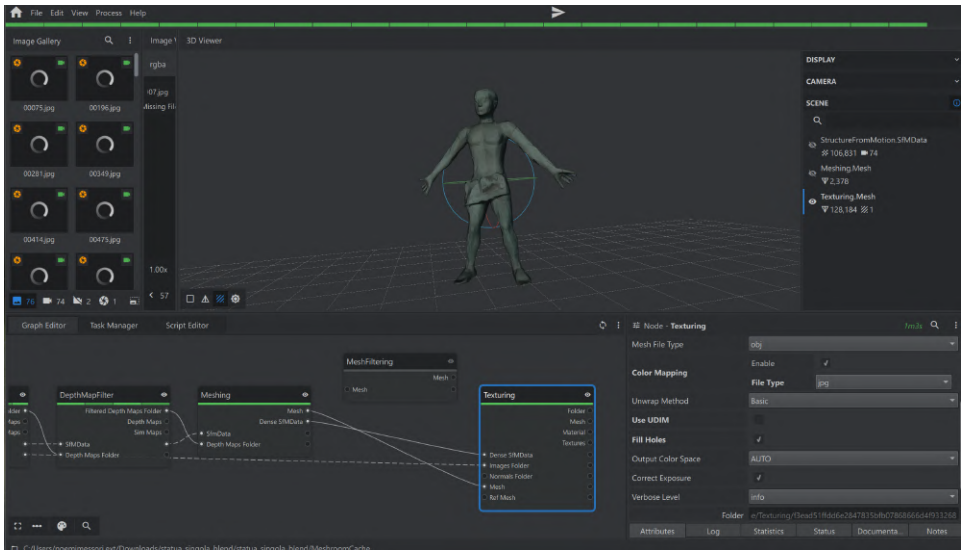

Figura 4.17: Pipeline di ricostruzione 3D con Meshroom, con focus sulla fase finale di Texturing.

Una volta rigenerata in Meshroom a partire dal layout UV ottimizzato, la texture è stata ridimensionata a 1K. Con le coordinate ridistribuite sull'intera area disponibile, la riduzione di risoluzione non ha comportato perdite qualitative rilevanti: la densità texel risultava omogenea e nessuna zona dell'immagine rimaneva inutilizzata. Si può vedere il confronto tra il layout UV originale e quello ottimizzato nella Figura 4.18.

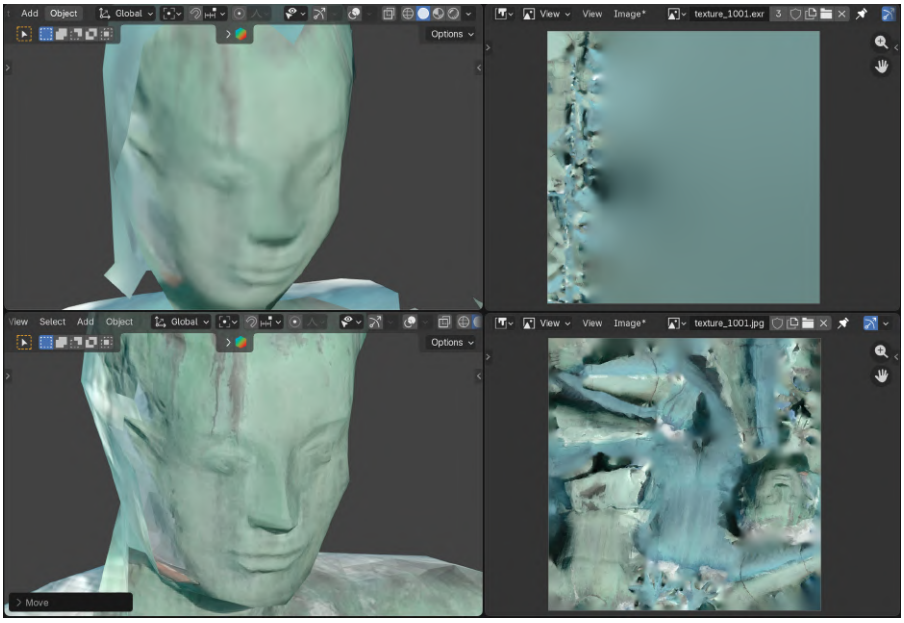

Figura 4.18: Confronto della resa tra il layout UV originale prodotto da Meshroom e quello ottimizzato manualmente in Blender.

Si noti come nella versione originale le isole UV siano concentrate in una porzione limitata dell'area, lasciando ampie zone vuote, mentre nella versione ottimizzata le isole sono distribuite in modo più uniforme, sfruttando al meglio lo spazio disponibile e garantendo una qualità visiva superiore anche a risoluzioni più basse. La Figura 4.19 mostra il risultato finale del gruppo scultoreo in Blender dopo il remapping delle coordinate UV.

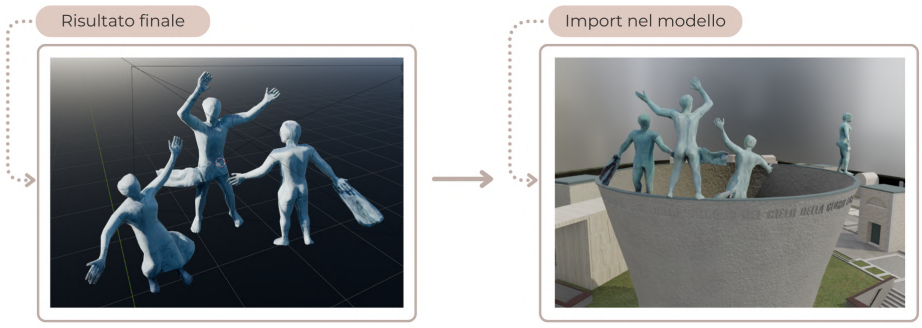

Figura 4.19: Resa finale del gruppo di statue in Blender dopo l'ottimizzazione del layout UV.

Problema del nodo Texturing e soluzione finale. Durante la rigenerazione della texture è emersa un'ulteriore criticità. Nonostante il nodo Texturing ricevesse in ingresso la mesh low-poly ottimizzata manualmente, l'output del nodo conteneva una versione ritessellata con un numero di triangoli superiore. La texture veniva calcolata correttamente a partire dal layout UV ottimizzato, ma la mesh associata non corrispondeva alla versione semplificata: Meshroom applicava internamente una suddivisione della superficie che vanificava la riduzione poligonale.

La soluzione consiste nel disaccoppiare la mesh dalla texture. Il risultato del nodo Texturing è stato importato in Blender con il solo scopo di estrarne la texture generata; la mesh low-poly, quella ottimizzata manualmente con il layout UV corretto, è stata mantenuta separata. La texture è stata quindi riapplicata su quest'ultima. In questo modo il modello finale usa la geometria semplificata prodotta in Blender e la texture calcolata da Meshroom, ottenendo un asset con topologia controllata e qualità visiva adeguata per il rendering real-time.

La Figura 4.20 riporta il risultato complessivo per la statua interna: a sinistra la mesh in Blender con la texture applicata, al centro la texture e la normal map generate da Meshroom, a destra la mesh low-poly finale ottimizzata.

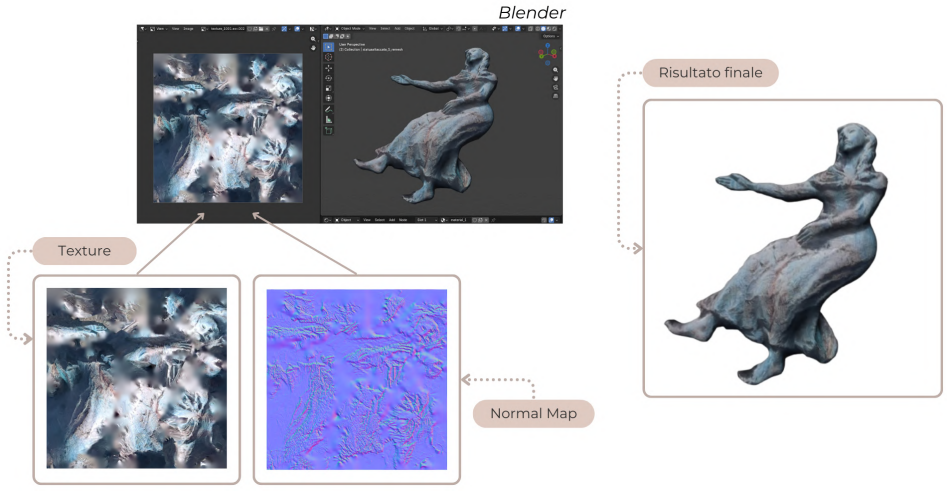

Figura 4.20: Risultato del processo di ottimizzazione per la statua interna: mesh texturizzata in Blender, texture e normal map generate, mesh low-poly finale.

Nella Figura 4.21 si può vedere la resa finale del gruppo di statue all'interno del framework ATON.

Figura 4.21: Resa finale del gruppo di statue su ATON da dentro il monumento.

4.5 Modellazione degli Esterni al Monumento

L'area esterna al Monumento Bottoni comprende i viali lastricati, le tombe monumentali circostanti e la vegetazione che caratterizza l'ambientazione della Certosa. Ciascuno di questi elementi ha richiesto strategie di modellazione specifiche, condizionate dalla diversa natura geometrica e dalle esigenze di rendering in un contesto di fruizione web con vincoli di prestazione lato client.

4.5.1 Modellazione delle Tombe

Come anticipato in precedenza, il punto di partenza di DT-Certosa per la modellazione dell'area esterna partiva da due modelli preesistenti. Il primo, realizzato in una fase precedente del progetto, includeva il solo Monumento Bottoni; il secondo, risalente al 2004, comprendeva anche le tombe e gli edifici circostanti. Quest'ultimo, in particolare, presentava una geometria necessariamente semplificata e a bassissima conta poligonale, riflettendo lo stato dell'arte e i severi limiti hardware dei software di navigazione 3D dei primi anni 2000. A quell'epoca, per garantire un'esperienza interattiva via web, era inevitabile ricorrere a forti ottimizzazioni strutturali. Inoltre, considerati i piccoli adeguamenti e i naturali restauri avvenuti nell'arco di vent'anni, tale riferimento non offriva più un grado di aderenza al mondo reale in linea con gli standard attuali, rendendo necessaria una ricostruzione sistematica e fedele delle forme e dei materiali.

Creazione delle mesh

Per la creazione delle mesh delle tombe dunque si è fatto uso del materiale fotografico e video acquisito durante le campagne di rilievo. Questo ha permesso di controllare forma, proporzioni e relazioni spaziali tra gli elementi in modo da ottenere una rappresentazione più fedele possibile alla realtà. Per avere invece un riferimento planimetrico, le piattaforme online come Google Maps, Google Earth e il portale Open Data del Comune di Bologna (ortofoto) hanno offerto una visione d'insieme utile, pur però rivelandosi insufficienti per la verifica di dettagli architettonici e proporzioni specifiche delle singole tombe.

Per gli elementi ritenuti recuperabili si è proceduto tramite l'operazione di *Append* tra scene di Blender [4], che consente di importare oggetti da un file

esterno nel progetto corrente. Le mesh importate presentavano diverse criticità geometriche: oggetti raggruppati sotto un unico Parent, vertici duplicati, triangolazione diffusa, normali non coerenti e ombreggiatura non ottimizzata. La pulizia ha seguito una sequenza precisa. Prima si è rimosso il parenting tramite *Clear Parent* con l'opzione *Keep Transform*, in modo da rendere gli oggetti indipendenti senza alterarne posizione, rotazione e scala. Dopo, in Edit Mode, si è applicato *Merge by Distance* per eliminare i vertici duplicati e la funzione *Tris to Quads* per convertire le triangolazioni in quad dove la topologia lo permetteva, migliorando la leggibilità e la modificabilità della mesh. Infine si è proceduto alla rimozione dei dati di normali personalizzati dalla sezione *Geometry Data*, in modo da poterle ricalcolare correttamente e all'applicazione di *Shade Auto Smooth* per ottenere un'ombreggiatura superficie uniforme e coerente.

Per il posizionamento nella nuova scena è stata inserita un'immagine di riferimento al livello Z=0 in modalità ortografica. Sono state utilizzate due fonti: uno screenshot da Google Maps e un'ortofoto ad alta risoluzione proveniente dal portale Open Data del Comune di Bologna. Queste immagini hanno permesso di collocare le mesh recuperate correttamente in pianta, di verificare allineamenti e distanze reciproche e di procedere per duplicazione e adattamento delle geometrie esistenti. Questo utilizzo si può vedere nella Figura 4.22 che segue.

Figura 4.22: Vista di Blender con le tombe posizionate su immagini di riferimento planimetriche (Google Maps e ortofoto del portale Open Data del Comune di Bologna).

Le tombe non fedeli alla realtà necessitavano di essere rimodellate dall'inizio. La resa finale a confronto è mostrata nelle Figure 4.23 sottostanti, che evidenziano la differenza tra la geometria preesistente e quella ricostruita a partire dal materiale fotografico. A sinistra si può vedere che la forma delle tombe,

soprattutto delle più monumentali, era approssimativa e non corrispondeva alla realtà, mentre a destra si vede la nuova geometria più fedele alle proporzioni reali.

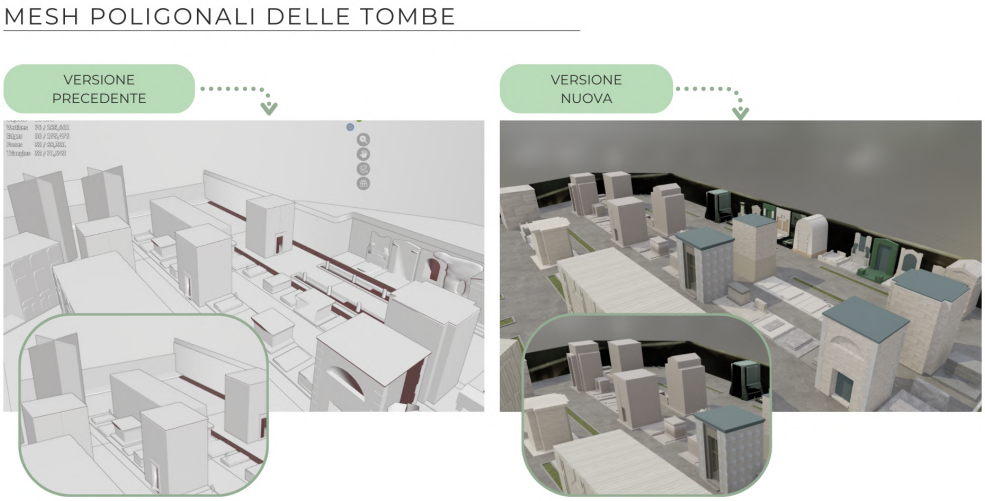

Figura 4.23: Confronto tra la geometria preesistente (a sinistra) e quella ricostruita (a destra) per alcune tombe monumentali.

Il materiale fotografico è stato organizzato per aree e file di tombe, facilitando la comprensione delle relazioni spaziali nel contesto e, per avere un riferimento riguardo raggruppamenti e prossimità, i video acquisiti in loco si sono rivelati particolarmente utili. Le fotografie singole infatti non sempre restituiscono questi dettagli in modo efficace.

Per ciascuna tomba da ricostruire si è ricorso al software open source fSpy [14] per la calibrazione prospettica della camera, tecnica nota come *Camera Matching*. Il principio su cui si basa fSpy è illustrato in Figura 4.24: il software individua i punti di fuga (*vanishing points*), ovvero i punti dell'immagine in cui convergono le proiezioni di rette parallele nello spazio reale.

A partire da una fotografia prospettica della tomba si definiscono manualmente segmenti di linea corrispondenti a direzioni parallele nella realtà; dall'intersezione di tali segmenti il software calcola i punti di fuga e, assumendo la perpendicolarità tra le direzioni identificate, stima la lunghezza focale, l'orientamento e la posizione della camera. Con almeno due direzioni ortogonali è possibile ricostruire orientamento e focale; una terza direzione permette di

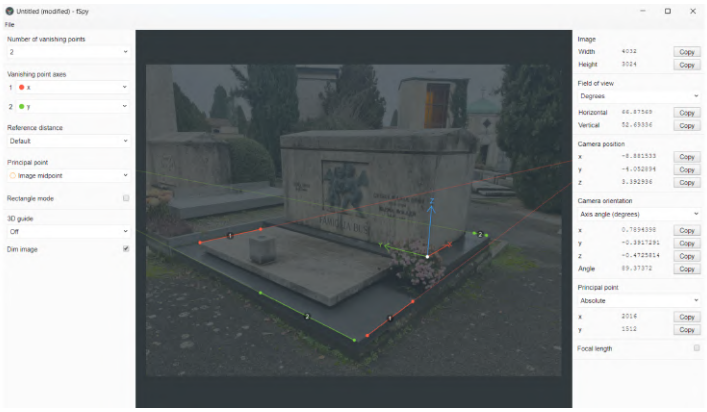

Figura 4.24: Esempio di funzionamento di fSpy.

stimare anche il principal point. fSpy consente inoltre di definire una distanza di riferimento lungo un asse noto, migliorando la coerenza dimensionale dell'intera operazione.

L'integrazione con Blender avviene tramite l'add-on ufficiale fSpy-Blender [15], che importa il file di progetto fSpy e genera automaticamente una camera con i parametri corrispondenti e l'immagine di sfondo associata. All'interno della vista camera di Blender è cosi' possibile modellare la tomba direttamente sopra l'immagine calibrata, rispettando correttamente la prospettiva reale.

La calibrazione prospettica garantisce proporzioni relative fedeli, ma non una scala assoluta coerente con l'intera scena. Al termine della modellazione ogni mesh è stata pertanto ridimensionata in base al riferimento planimetrico, sfruttando le immagini importate fatte dall'alto, ruotata sull'asse Z per allinearsi all'orientamento corretto e posizionata nella collocazione definitiva. Questo ciclo, ripetuto per circa ottanta tombe, ha assicurato coerenza geometrica, proporzionale e spaziale nell'insieme della scena.

La fase di integrazione finale ha richiesto un continuo passaggio tra scala locale e scala globale: ogni tomba richiedeva attenzione ai propri dettagli geometrici, ma il posizionamento andava sempre verificato rispetto all'intero contesto. Distanze reciproche, allineamenti lungo le file e connessione con gli edifici circostanti sono stati controllati attraverso il confronto sistematico tra immagini fatte dall'alto di Google Maps e delle ortofoto del portale Open Data del Comune di Bologna, materiale fotografico acquisito in loco e viste prospettiche ottenute tramite la calibrazione con fSpy. Questo si può vedere nella Figure 4.25.

L'obiettivo non era ottenere singoli oggetti corretti in modo isolato, ma costruire un insieme coerente in cui ogni elemento risultasse integrato nel contesto complessivo.

Figura 4.25: Modellazione delle tombe finali.

Il risultato è una ricostruzione tridimensionale in cui le tombe non svolgono solo una funzione riempitiva, ma assumono anche un ruolo strutturale e documentale puntando a creare un modello fedele e coerente del Digital Twin.

Texturing: generazione e lavorazione

Completata la fase geometrica, il lavoro è proseguito con la definizione dell'aspetto visivo delle tombe tramite il texturing. Lasciare le mesh prive di materiali avrebbe prodotto una resa visiva non coerente con gli obiettivi del progetto; al tempo stesso, l'adozione acritica di texture ad alta risoluzione avrebbe compromesso le prestazioni dell'applicazione web.

Il peso complessivo delle risorse grafiche trasferite al client influisce direttamente sulla fluidità di navigazione, in particolare in un contesto web dove la banda disponibile non è controllabile. Per questo le scelte di texturing hanno privilegiato il riuso: laddove le superfici presentavano materiali simili si è impiegata la stessa texture su più elementi, evitando la proliferazione di immagini sostanzialmente identiche. Le texture sono state esportate in formato JPG dove l'assenza di trasparenze lo consentiva, ottenendo file di dimensioni sensibilmente inferiori rispetto al PNG; la risoluzione è stata contenuta a 1K (1024×1024 pixel) come compromesso tra qualità visiva e peso delle risorse.

Le texture utilizzate derivano da tre sorgenti principali: librerie royalty-free, fotografie acquisite durante le campagne di rilievo e generazione tramite strumenti basati su intelligenza artificiale.

Texture royalty-free. Per alcune superfici uniformi non adeguatamente documentate dalle fotografie si sono utilizzate texture disponibili tramite BlenderKit [5], l'addon integrato in Blender che fornisce accesso a una libreria di materiali e asset tridimensionali. I materiali scaricati da queste librerie includono in genere base color, normal map, roughness map, metallic map e ulteriori canali; tuttavia il framework ATON, che utilizza Three.js come livello di rendering, supporta solo un sottoinsieme dei parametri del nodo *Principled BSDF* di Blender: base color, normal map, metallic e roughness vengono esportati e interpretati correttamente, mentre gli altri canali vengono ignorati in fase di conversione. Per questo sono state impiegate soltanto le componenti compatibili, in particolare base color e, dove significativa, normal map. Questo approccio è stato limitato a un numero ristretto di casi: per la maggior parte delle tombe si è preferito derivare i materiali direttamente dal materiale fotografico reale.

Creazione delle texture con GIMP. La strategia principale ha fatto uso delle fotografie acquisite in loco. Per la loro elaborazione si è utilizzato GIMP [16] (GNU Image Manipulation Program), un software libero e open source per la manipolazione di immagini raster che offre strumenti per la correzione prospettica, il bilanciamento cromatico e la gestione dei livelli.

Il processo prevedeva innanzitutto una fase di correzione geometrica. Poiché le fotografie venivano scattate con angolazioni non sempre ortogonali alla superficie, si rendevano necessarie trasformazioni locali per raddrizzare la prospettiva e uniformare la geometria dell'immagine. A questo scopo sono stati utilizzati gli strumenti *Unified Transform*, *Cage Transform* e *Warp Transform*, che operano sulla distribuzione dei pixel in modo locale senza agire sull'intera immagine. Successivamente veniva applicato il bilanciamento cromatico e la correzione dell'illuminazione. Le condizioni di ripresa, pur preferendo sempre giornate nuvolose con luce diffusa, non garantivano uniformità perfetta: alcune aree apparivano più scure o più sature rispetto ad altre. La correzione si avvaleva della regolazione dei livelli, degli strumenti schiarisci e brucia e di ritocchi manuali con pennello o clone.

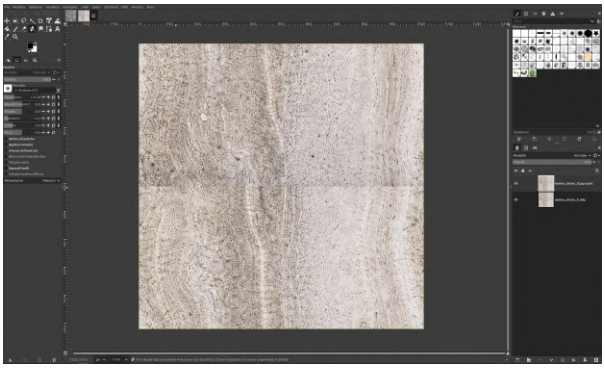

Figura 4.26: Verifica della ripetibilità in GIMP tramite offset di metà larghezza e metà altezza: le giunzioni originarie vengono portate al centro dell'immagine, rendendo immediatamente visibili le eventuali discontinuità da correggere.

Un requisito fondamentale per le texture destinate a superfici estese era la ripetibilità: una texture tileable è progettata per essere affiancata più volte senza che le giunzioni risultino visibili. Per verificare questa proprietà si applicava un offset pari alla metà delle dimensioni dell'immagine: in questo modo le giunzioni originarie venivano portate al centro, rendendo immediatamente visibili eventuali discontinuità. Le correzioni necessarie venivano effettuate con clone tool e healing tool. Un problema ricorrente riguardava la presenza di elementi caratteristici concentrati in zone limitate della texture, come macchie di muffa, depositi o segni particolari. Questi dettagli, accettabili in un'immagine usata una volta sola, diventano artefatti evidenti quando la texture viene ripetuta più volte sulla stessa superficie, comparendo in posizioni identiche e regolari. In diversi casi tali elementi sono stati attenuati o rimossi manualmente prima dell'esportazione.

Generazione di texture tramite intelligenza artificiale. In alcuni casi si è sperimentato l'utilizzo di modelli generativi di immagini per produrre texture sintetiche a partire da fotografie di riferimento e descrizioni testuali. Questo approccio si è rivelato utile soprattutto per superfici marmoree uniformi, dove la disponibilità di immagini fotografiche era limitata o la qualità non era sufficiente per un'elaborazione diretta. Le immagini generate erano comunque sottoposte a verifica e rifinitura manuale in GIMP, per correggere eventuali incongruenze o problemi di ripetibilità prima dell'integrazione nei materiali.

Figura 4.27: Esempi di texture per superfici marmoree generate tramite strumenti di intelligenza artificiale e successivamente rifinite in GIMP.

Generazione delle normal map. Oltre alle texture di colore, per numerose superfici sono state prodotte le normal map. Questi canali codificano nei valori RGB l'orientamento locale della normale, consentendo al motore di rendering di simulare rilievi, incisioni e irregolarità senza aggiungere geometria alla mesh. L'effetto è tuttavia condizionato dalla presenza di illuminazione dinamica nella scena: con luce puramente ambientale il contributo risulta trascurabile.

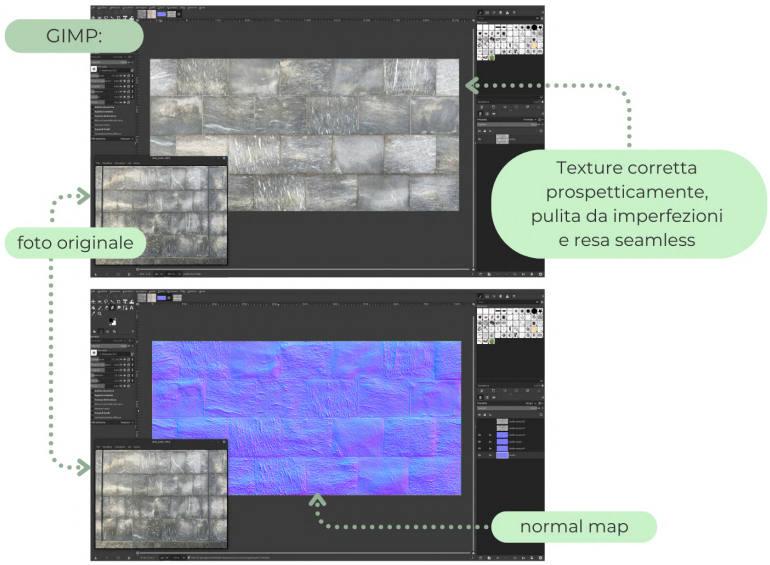

Figura 4.28: Lavorazione delle texture e delle normal map in GIMP.

Come mostra la Figura 4.28, la generazione è avvenuta in GIMP, che in-

terpreta le variazioni di luminosità della texture come variazioni di quota: le aree più chiare corrispondono a rilievi, quelle più scure a incavi. In alcuni casi si è sovrapposto più livelli a granularità diversa, combinati tramite la modalità Overlay e poi normalizzati per evitare artefatti in rendering.

La normal map è stata integrata in Blender collegandola al nodo *Normal Map*, connesso all'ingresso omonimo dello shader *Principled BSDF*.

UV Unwrapping e applicazione delle texture

Una volta prodotte le texture, è stato necessario definire come ciascuna immagine si distribuisce sulla superficie della mesh tramite il *UV mapping*: a ogni vertice vengono associati due valori scalari U e V che indicano la posizione corrispondente nella texture 2D. L'operazione di UV unwrapping proietta la superficie 3D su un piano, producendo regioni connesse chiamate *isole UV*; distorsioni o frammentazioni eccessive delle isole si traducono direttamente in artefatti visibili sulla superficie finale.

Struttura del materiale in Blender. Per ogni oggetto è stato definito un materiale tramite lo *Shader Editor* di Blender, con una configurazione semplice compatibile con il sottoinsieme di parametri supportato da ATON. Il setup prevede quattro nodi: *UV Map*, *Image Texture*, *Principled BSDF* e *Material Output*; la normal map, quando disponibile, viene collegata all'ingresso omonimo del BSDF tramite un nodo *Normal Map*. Il parametro di roughness è stato modulato in base alla lucidità specifica del materiale: si sono adottati valori bassi (tra 0.1 e 0.2) per i marmi lucidi, arrivando fino a 0.05 per le tombe a specchio, mentre per superfici più opache e ruvide, come i laterizi, si sono utilizzati valori compresi tra 0.5 e 0.7. L'impostazione diretta di questo parametro nello shader ha permesso di evitare l'uso di una texture dedicata, riducendo ulteriormente il peso delle risorse trasferite al client. Questo schema è visibile in Figura 4.29.

Procedura di unwrapping. L'unwrapping veniva eseguito in Edit Mode. Per le geometrie prevalentemente planari delle tombe si è usato come primo tentativo *Smart UV Project*, che introduce tagli automatici dove la curvatura supera una soglia configurabile. Il limite principale era la frammentazione eccessiva: sulle superfici con continuità cromatica uniforme le giunzioni tra isole

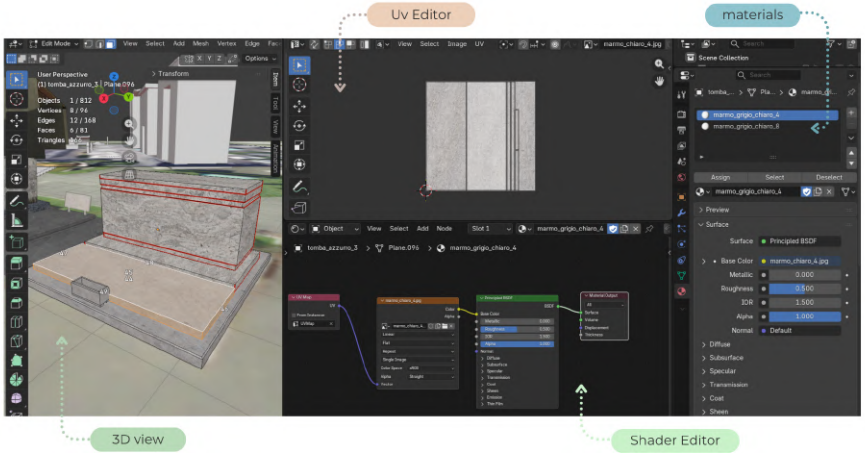

Figura 4.29: Interfaccia di Blender durante le fasi di UV unwrapping e texturing.

risultavano visibili. In questi casi si sono definiti manualmente i *seam*, bordi marcati in rosso nella viewport (visibili in Figura 4.29), e si è eseguito l'*Unwrap* con metodo *Angle Based*, concentrando i tagli nelle zone meno visibili e preservando la continuità sui fronti principali.
Nell'UV Editor le isole venivano poi ridimensionate per concentrare la risoluzione sulle facce più esposte; le superfici con curvatura più marcata, come i bordi smussati del basamento, hanno richiesto seam aggiuntivi per evitare allungamenti evidenti della texture.

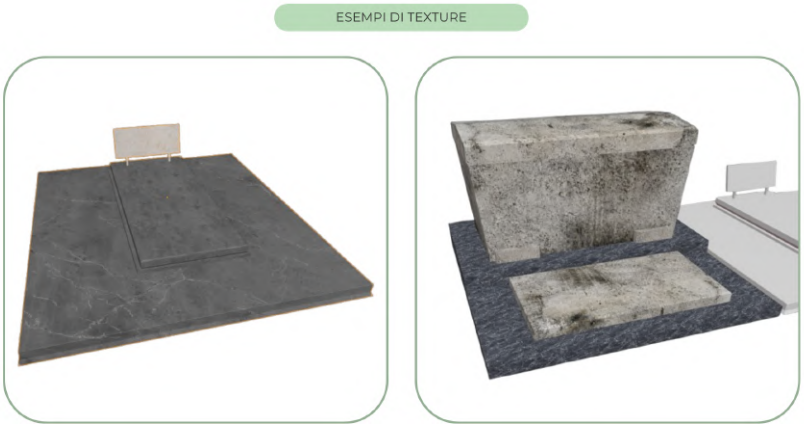

Figura 4.30: UV map di due tombe modellate con texture.

Il risultato complessivo di questo processo ha prodotto materiali con una

resa visiva coerente con il contesto reale, compatibili con i requisiti di esportazione verso ATON e con i vincoli di peso imposti dalla fruizione web. Se ne possono vedere due esempi nella Figura 4.30.

4.5.2 Modellazione della Vegetazione

La vegetazione costituisce uno degli elementi più difficili da gestire nel contesto di una scena tridimensionale destinata alla fruizione web. La modellazione poligonale tradizionale risulta inadeguata per alberi, siepi e cespugli: la complessità strutturale di rami, foglie e superfici intrecciate porta rapidamente a mesh di dimensioni proibitive, e qualsiasi tentativo di semplificazione produce artefatti visivi evidenti, in particolare sui profili e sulle silhouette.
Per questo motivo si è scelto di adottare il Gaussian Splatting (GS) [21], una tecnica di rendering che rappresenta la scena tramite una nuvola di primitive gaussiane tridimensionali ottimizzata a partire da sequenze fotografiche.
Come menzionato nel Capitolo 2, il GS non produce mesh nel senso tradizionale: la geometria dell'oggetto è il risultato di una distribuzione e di un orientamento delle gaussiane nello spazio, ciascuna caratterizzata da posizione, scala, rotazione, opacità e colore codificato tramite armoniche sferiche. Il risultato è una rappresentazione compatta e visivamente densa, adatta a catturare strutture organiche non facilmente riconducibili alla geometria poligonale.

Come base di partenza sono stati analizzati diversi progetti open source e riferimenti accademici. Il riferimento operativo principale è stato il repository di Kyle Johnson [19], che fornisce anche un addon open source per Blender denominato SkySplat [18]: questo integra direttamente nell'ambiente di modellazione l'intera pipeline, dall'estrazione dei frame alla stima delle pose con COLMAP [29] fino alla preparazione del dataset per il training. Sono stati esaminati anche il repository del gruppo INRIA [21], come riferimento teorico sulla struttura interna degli splat, e GaussianSplats3D come ulteriore esempio implementativo sulla gestione delle primitive in contesti interattivi.

L'adozione di questi strumenti non è stata lineare: ogni passaggio ha richiesto test iterativi, a causa sia della giovane età della tecnologia sia del supporto ancora parziale nei framework di rendering web.

Pipeline del Gaussian Splatting

Il workflow adottato si articola nelle fasi schematizzate in Figura 4.31: partiti dall'acquisizione video, c'è l'estrazione dei frame, stima delle pose con COLMAP, training del modello con Brush [7], pulizia e ottimizzazione in Blender e conversione finale con SuperSplat [26].

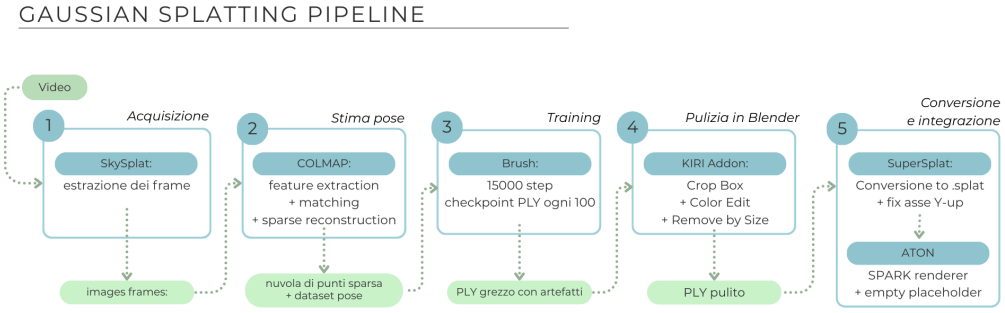

Figura 4.31: Schema a passi della pipeline adottata per la produzione degli splat: dall'acquisizione video all'integrazione in ATON.

Acquisizione video. Come descritto nella Sezione 4.2.4, la fase di acquisizione si è rivelata fondamentale. Il materiale di input condiziona direttamente la qualità finale degli splat: illuminazione incoerente, motion blur o traiettorie irregolari producono artefatti, lacune o gaussiane sfocate nella ricostruzione. I primi video, acquisiti a mano libera, presentavano problemi evidenti di stabilità e COLMAP non riusciva a stabilire corrispondenze affidabili tra i frame.

Il cambio di approccio ha previsto riprese circolari a 360° con iPhone, traettorie lente e uniformi, copertura completa del volume della pianta e, per gli elementi più alti come i cipressi, due giri completi per aumentare la ridondanza visiva. Le condizioni meteorologiche rimanevano quelle già descritte per le tombe: cielo coperto, luce diffusa, nessun riflesso speculare. L'altezza di ripresa è stata calibrata in base alla forma di ogni elemento: per i cipressi, che si sviluppano verticalmente, mantenere un'inquadratura bilanciata tra tronco e chioma è stato determinante per evitare lacune nelle zone superiori della ricostruzione.

Il funzionamento di questa parte è visibile nella Figura 4.32, che mostra un esempio di video acquisito per una pianta usata come test.

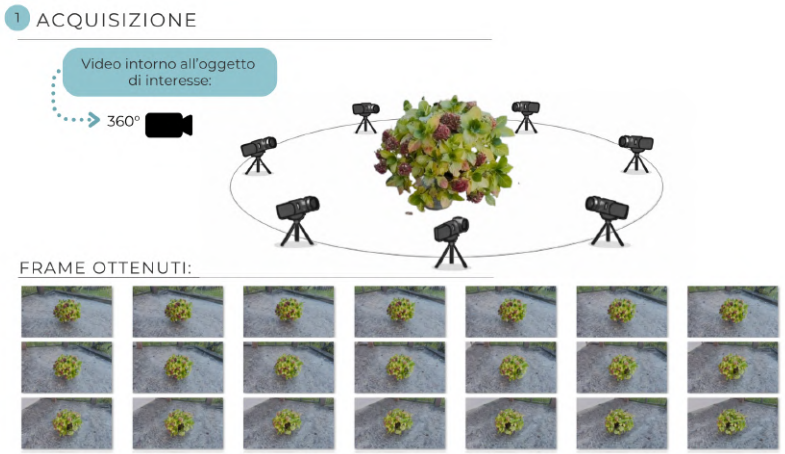

Figura 4.32: Immagine rappresentativa della fase di acquisizione video per la produzione degli splat.

Estrazione dei frame e stima delle pose con COLMAP. Il video acquisito viene caricato in Blender tramite il pannello SkySplat, che gestisce direttamente l'estrazione dei frame e l'invocazione di COLMAP. Il parametro *Frame Step* è stato impostato a 4, valore che bilancia la densità di campionamento temporale con i tempi di elaborazione: un valore inferiore produce più frame ma allunga significativamente la fase SfM (Structure-from-Motion) senza garantire un miglioramento proporzionale della qualità. COLMAP stima la posizione e l'orientamento di ciascuna camera e produce una nuvola di punti sparsa della scena.

Il workflow SfM all'interno di SkySplat prevede quattro passaggi sequenziali: estrazione delle feature, matching tra frame, ricostruzione sparse e undistorsione delle immagini. Al termine, il modello di punti viene caricato in Blender: in questa fase è possibile aggiustare l'orientamento della nuvola tramite un empty ausiliario, correggendo inclinazioni o traslazioni sull'asse Z prima di esportare il modello trasformato e preparare il dataset per il training.

Un numero troppo basso di frame portava a vuoti nella nuvola di punti e ricostruzioni incomplete; una densità eccessiva allungava i tempi di calcolo senza un miglioramento altrettanto netto. Qualsiasi imperfezione nell'acquisizione si propagava nella fase successiva: in diversi casi è stato necessario ripetere l'intera pipeline a causa di errori accumulati fin dall'inizio. La visualizzazione della nuvola di punti e delle pose stimate all'interno di Blender è illustrata nella

Figura 4.33. Sono messe in evidenza anche le camere frames di COLMAP.

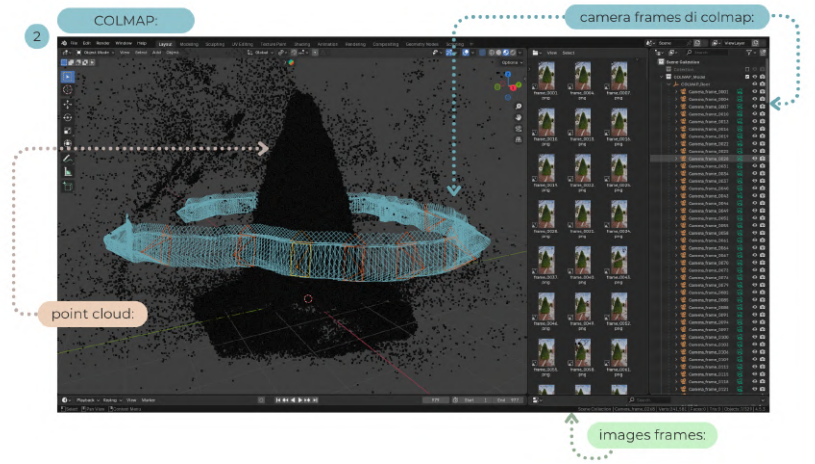

Figura 4.33: Estrazione dei frame e stima delle pose con COLMAP su Blender.

Training con Brush. Il dataset prodotto da SkySplat viene passato a Brush [7] per il training del modello. La configurazione adottata prevede 30000 step di ottimizzazione totali. La sua interfaccia è mostrata nella Figura 4.34, che evidenzia la visualizzazione in tempo reale del processo di training, con la nuvola di punti che si evolve progressivamente per approssimare la forma dell'oggetto fotografato.

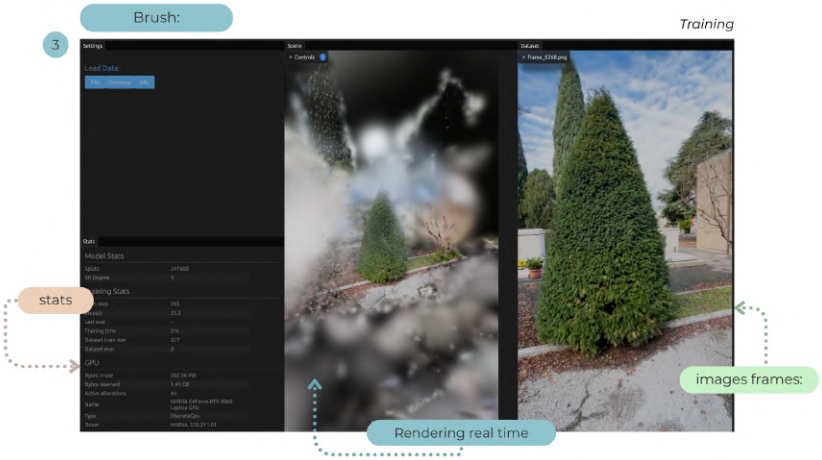

Figura 4.34: Training con Brush.

Il risultato finale è mostrato nella Figura 4.35, che evidenzia la distribuzione delle gaussiane e la loro colorazione.

Figura 4.35: Risultato dello Splat post training con Brush.

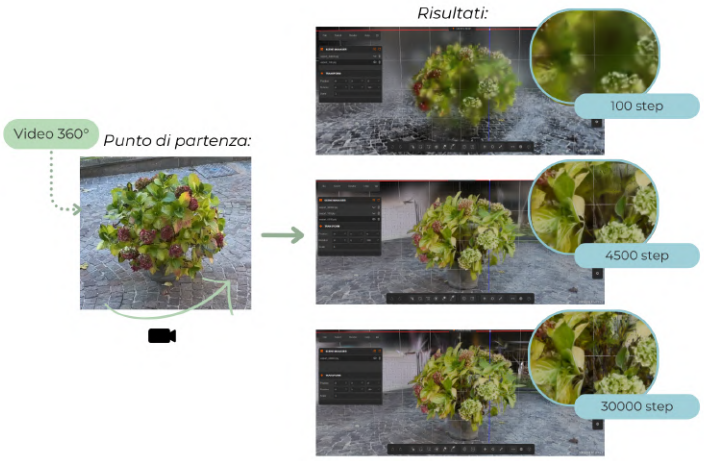

Figura 4.36: Confronto tra i risultati di training con Brush: 100, 4500, 30000 step.

I checkpoint intermedi permettono di monitorare l'evoluzione qualitativa senza attendere il completamento dell'intera sessione. La scelta del migliore checkpoint richiede una valutazione visiva, non è determinabile a priori dai soli parametri, per questo motivo si sfruttavano i risultati intermedi da confrontare. L'esempio a confronto è mostrato nella Figura 4.36.

Pulizia, Conversione e Integrazione

Importazione e visualizzazione in Blender. I file PLY prodotti includono sistematicamente porzioni indesiderate della scena (suolo, elementi di sfondo, rumore diffuso), rendendo indispensabile la fase di pulizia. Sono stati così importati in Blender [4] tramite il KIRI Engine 3DGS Blender Addon [22], utilizzando la modalità *Import as VERTS*.
Questa modalità carica le gaussiane come vertici di un oggetto proxy, rendendo possibile la selezione e la manipolazione della nuvola tramite gli strumenti standard di Edit Mode. Per visualizzare correttamente gli splat durante l'editing, è necessario attivare la modalità *Material Preview* al posto di *Solid*, e abilitare l'opzione *Enable Camera Updates* dell'addon oppure *Show as Point Cloud*: nella prima modalità il rendering degli splat si aggiorna seguendo la camera, offrendo una valutazione più fedele del risultato finale; la seconda è più leggera ma meno rappresentativa delle gaussiane reali. Le principali operazioni di puliza e ottimizzazione sono visibili nella Figura 4.37 che segue.

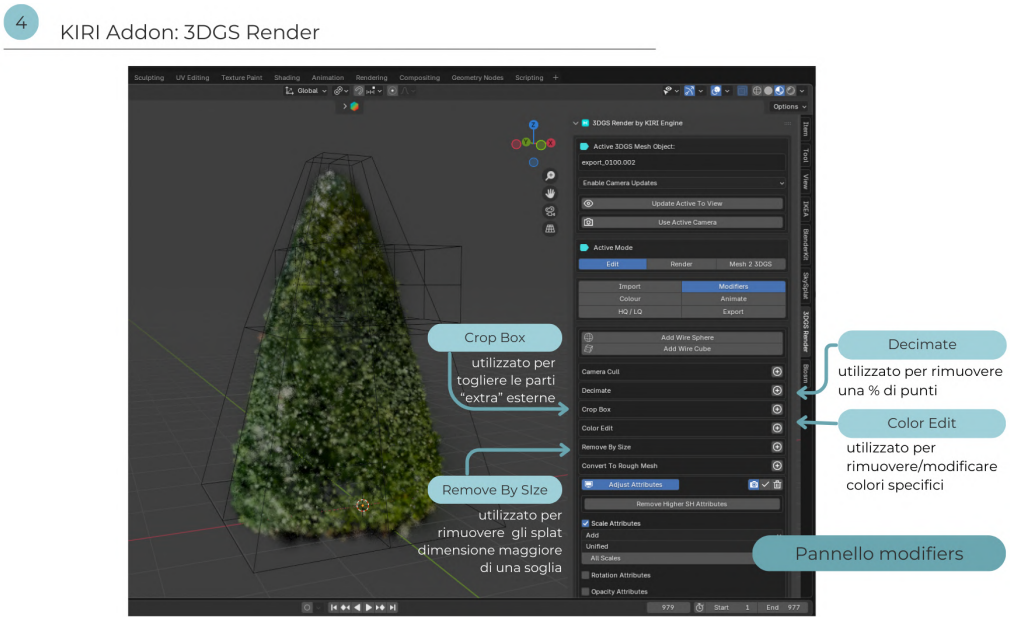

Figura 4.37: KIRI Addon: 3DGS Render.

Pulizia geometrica con Crop Box. La rimozione delle porzioni indesiderate si è basata principalmente sul modificatore *Crop Box* fornito dall'addon. La procedura prevede la creazione di un oggetto wireframe cubico che delimiti

il volume da conservare, seguito dall'applicazione del modificatore *Crop Box* sullo splat, con il cubo impostato come oggetto di riferimento e la modalità *Outside Crop Box* attiva per eliminare tutto ciò che cade al di fuori del volume definito. Una volta verificato il risultato, il modificatore viene applicato in modo permanente. Questa operazione si è dimostrata efficace per il taglio volumetrico grossolano, ma insufficiente in presenza di artefatti distribuiti all'interno del volume utile.

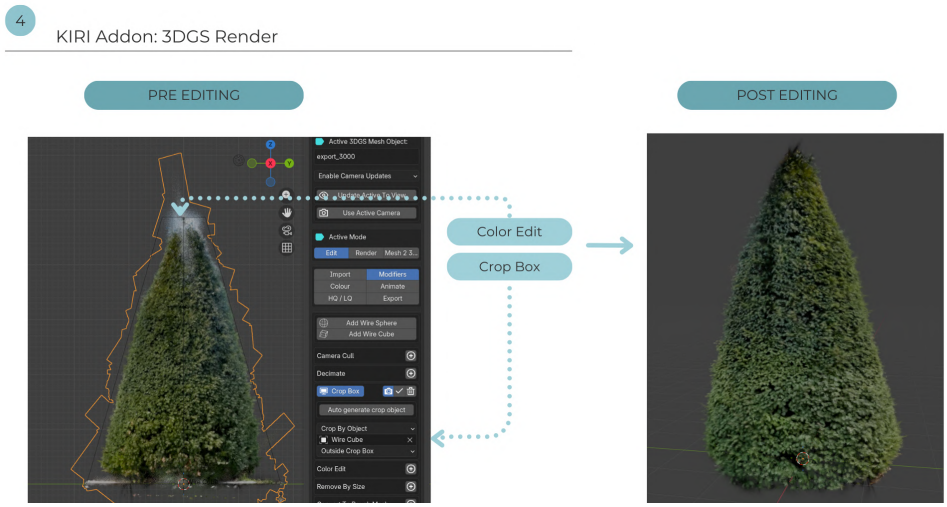

Figura 4.38: KIRI Addon: 3DGS Render: confronto tra pre e post editing utilizzando Color Edit e Crop Box.

Rimozione di artefatti cromatici e di scala. Per gli artefatti non eliminabili con il solo ritaglio volumetrico si sono resi necessari interventi più selettivi. Lo strumento *Color Edit* consente di rimuovere gaussiane in base al colore: la modalità *Equal to* elimina i vertici con colore esattamente corrispondente al campione selezionato, mentre *Brighter than* rimuove tutto ciò che supera una certa luminosità, utile per eliminare residui di cielo o riflessioni. In generale modifiche successive si accumulano in modo incrementale e ciascuna va applicata esplicitamente prima di procedere con la successiva.

Questi primi due passaggi sono quelli che determinavano la rimozione più significativa di artefatti: questa operazione è visibile nella Figura 4.39, che mostra un confronto tra una fase di pre editing e il risultato dell'applicazione post editing di Color Edit e Crop Box.

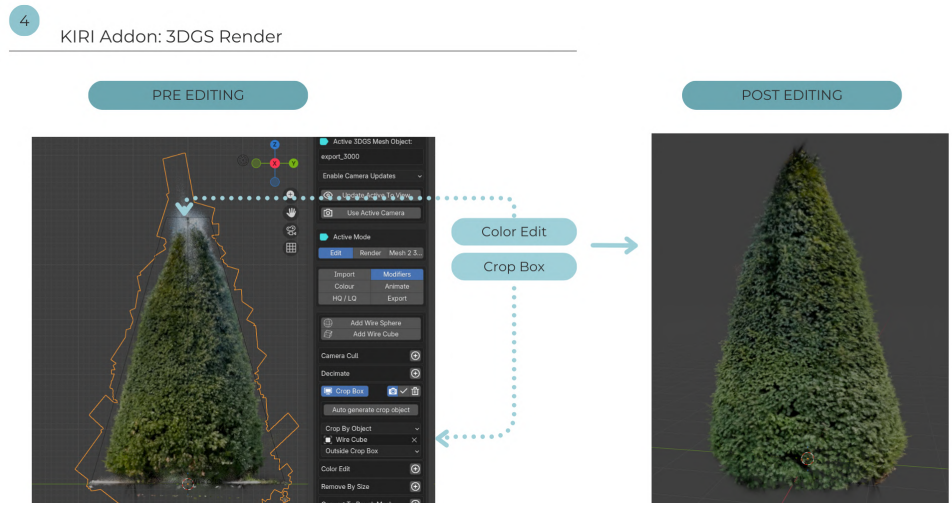

Figura 4.39: KIRI Addon: 3DGS Render: confronto tra pre e post editing utilizzando Color Edit e Crop Box.

Un problema ricorrente era la presenza di gaussiane di dimensioni anomale che generavano un effetto nuvola intorno all'oggetto, visibile soprattutto con *Enable Camera Updates*. Per risolverlo si è utilizzato *Remove by Size* con la modalità *Smaller Than Threshold*: questa opzione rimuove le gaussiane con scala superiore alla soglia indicata, solitamente impostata attorno a 0.3. Per gli splat che richiedevano una riduzione generalizzata della densità senza eliminare elementi specifici, lo strumento *Adjust Attributes* ha permesso di modificare scala e opacità di tutte le gaussiane in modo uniforme tramite operazioni di moltiplicazione. In alternativa, per file PLY particolarmente pesanti, si è valutato il modificatore *Decimate* con una percentuale intorno al 60%: questa strategia riduce il numero di primitive in modo non selettivo, con un impatto visivo che va verificato caso per caso.

Una ulteriore problematica emerge quando si applica la decimazione come misura di ottimizzazione: la riduzione della densità delle gaussiane genera aree visivamente incomplete, con un effetto di trasparenza che lascia intravedere lo sfondo attraverso la struttura dello splat. Il fenomeno è particolarmente evidente sugli elementi vegetali, dove la silhouette risulta discontinua e la profondità strutturale percepita diminuisce sensibilmente.

Si è dunque giunti a una serie di step mostrati nella Figura 4.40 da applicare in sequenza per compensare questo eccessivo diradamento della nuvola, riducendo le aree involontariamente trasparenti generate dalla decimazione.

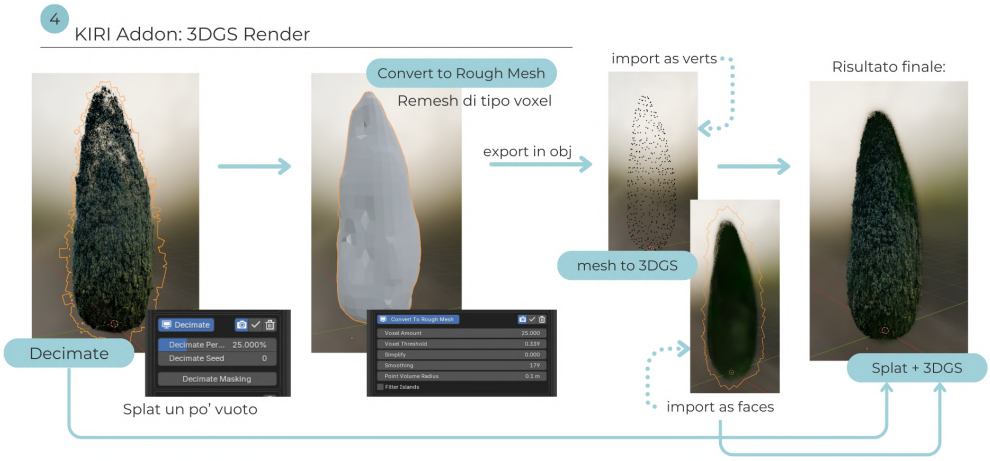

Figura 4.40: KIRI Addon: 3DGS Render: Ultimi step di modifica.

Per compensare infatti questa perdita è stata sfruttata un'ulteriore funzionalità dell'addon Kiri Engine, denominata *Convert to Rough Mesh*: questa converte lo splat in una mesh approssimata che ne riproduce la forma volumetrica generale sotto forma di un blob continuo. La mesh generata eredita comunque un numero rilevante di vertici, rendendo necessaria l'applicazione di modificatori di semplificazione topologica per ridurne il peso prima di procedere con la fase successiva.

Una volta ottimizzata, la mesh è stata esportata in formato OBJ con triangolazione abilitata, requisito necessario per la corretta reimportazione tramite l'addon, e successivamente riconvertita in splat utilizzando la modalità *Force as Faces*: questa genera le gaussiane a partire dalle facce della mesh anziché dai vertici, producendo una distribuzione spaziale più uniforme e un controllo cromatico più agevole. Il risultato è uno splat secondario che rappresenta la struttura dell'oggetto in forma geometricamente semplificata.

Questo splat secondario è stato leggermente ridimensionato e posizionato all'interno dello splat principale. La funzione non è geometrica ma visiva: le gaussiane dello splat derivato riempiono cromaticamente le zone in cui la decimazione ha ridotto eccessivamente la densità, creando un'illusione di con-

tinuità senza introdurre costo computazionale paragonabile allo splat originale. Poiché la conversione da mesh non preserva la colorazione originale, è stato necessario eseguire un vertex painting manuale in Blender, campionando i colori dallo splat principale per mantenere coerenza cromatica tra le due strutture sovrapposte. La verifica ha richiesto cicli iterativi di esportazione e confronto, condotti sia all'interno di Blender sia tramite SuperSplat, per valutare l'effetto complessivo della sovrapposizione e stabilire che le lacune visive risultassero adeguatamente mascherate.

Esportazione da Blender. Prima di esportare, è necessario applicare in sequenza tutti e quattro i modificatori `KIRI_3DGS_*` presenti nello stack dell'oggetto, poiché il file PLY finale deve incorporare tutte le trasformazioni in modo definitivo. L'esportazione avviene tramite *File* > *Export* > *PLY*, con l'opzione *Selection Only* attiva per evitare di includere altri oggetti presenti nella scena. Questo PLY esportato da Blender costituisce l'input per la fase successiva di conversione.

Conversione e validazione con SuperSplat. I file PLY esportati da Blender sono stati importati in SuperSplat [26], un editor web dedicato alla visualizzazione e alla conversione degli splat. La conversione è necessaria perché lo Spark renderer utilizzato in ATON non accetta il formato PLY come output diretto del training: i formati supportati sono `.splat` e `.ksplat`, dei quali solo il primo è esportabile da SuperSplat. Un requisito non documentato riguarda l'orientamento: l'asse Y deve puntare verso l'alto, altrimenti lo splat risulta ruotato nella scena ATON. Questo ha reso necessaria una correzione manuale dell'orientamento all'interno dell'editor prima dell'esportazione. SuperSplat ha svolto anche una funzione di validazione: visualizzando lo splat nel browser è stato possibile individuare problemi non percepibili in Blender, come regioni a densità anomala, gaussiane di orientamento incoerente con la forma dell'oggetto o artefatti cromatici residui che richiedevano un nuovo ciclo di pulizia. La possibilità di esportare gruppi di splat in un unico file consolidato è stata sfruttata per ridurre i problemi di stabilità riscontrati con il caricamento di file multipli in ATON.

Integrazione in ATON e posizionamento tramite empty. L'integrazione del GS in ATON è stata la fase più critica dell'intero workflow. ATON

supporta i Gaussian Splat tramite il renderer SPARK [30], a sua volta basato su Three.js [32], ma il supporto era ancora parziale al momento dello sviluppo. Il caricamento simultaneo di più file splat nella stessa scena produceva comportamenti instabili: alcuni splat non venivano renderizzati, altri causavano crash del renderer, e la coesistenza con mesh standard generava conflitti non documentati. Le soluzioni sono state identificate per via empirica, testando configurazioni diverse di caricamento, come mostrato in Figura 4.41 e monitorando la stabilità via console di sviluppo del browser.

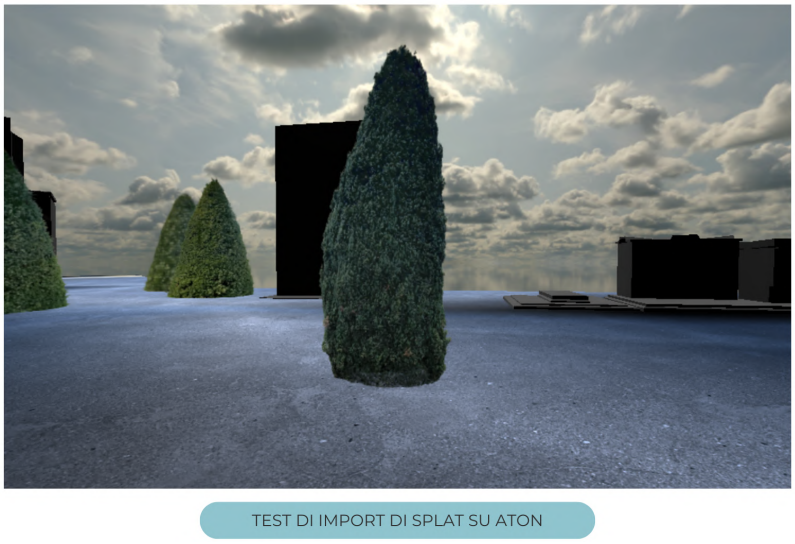

Figura 4.41: Test di caricamento di splat in ATON.

Il posizionamento spaziale degli splat ha richiesto una soluzione ad hoc. A differenza delle mesh, gli splat non ricevono trasformazioni spaziali attraverso il flusso di authoring standard del motore: il renderer non espone un meccanismo diretto per associare posizione, rotazione e scala a un file `.splat`. La strategia adottata sfrutta gli empty di Blender come placeholder spaziali. Per ogni elemento di vegetazione da inserire nell'ambiente virtuale è stato creato un empty nella posizione corrispondente, orientato e scalato in modo da rappresentare la trasformazione desiderata per lo splat associato. La scala non corrisponde a una misura assoluta, ma a un fattore moltiplicativo rispetto alle dimensioni dello splat generato in training: un empty con scala 1.2 produce uno splat il 20% più grande rispetto all'originale, consentendo di variare le

dimensioni della stessa pianta istanza per istanza senza ricostruzioni separate.

Per distinguere le tre tipologie a livello di codice, gli empty sono stati nominati seguendo una convenzione basato sul prefisso del nome: `T` per i cipressi ad alto sviluppo verticale, `C` per i cipressi dalla forma più compatta e `B` per i cespugli. Un suffisso numerico progressivo garantisce l'univocità di ciascun nodo. Gli empty sono stati esportati come file GLB separato dalla geometria principale della scena, contenente solo trasformazioni e senza alcuna mesh. In fase di caricamento in ATON, il sistema legge la gerarchia del file GLB, identifica gli empty dal prefisso del nome e istanzia dinamicamente lo splat corrispondente nella posizione corretta, applicando rotazione e scala estratte.

Gestione del Level of Detail (LOD)

Il costo computazionale del rendering delle gaussiane in ambiente web è sensibilmente più elevato rispetto alle mesh tradizionali. Ogni splat è una primitiva tridimensionale che richiede ordinamento per via della trasparenza, proiezione 2D e composizione per alpha-blending; con molte piante attive contemporaneamente, il numero totale di gaussiane può saturare rapidamente le risorse del client, soprattutto su dispositivi non dotati di GPU dedicata.

La gestione del LOD per la vegetazione GS si basa su uno schema a due livelli. A distanza ravvicinata dalla camera, nella zona di interesse visivo, viene caricato e renderizzato lo splat ad alta densità.
Al crescere della distanza, lo splat viene sostituito da una mesh poligonale semplificata ottenuta con Meshroom [2], con un numero di poligoni nettamente inferiore e un costo di rendering trascurabile rispetto alle gaussiane.
La transizione tra i due livelli è mediata da un overlay temporaneo: durante il caricamento asincrono dello splat, la mesh leggera rimane visibile, evitando che la zona appaia vuota nell'intervallo di tempo tra la richiesta del file e il completamento del rendering. Questo garantisce continuità dell'esperienza visiva indipendentemente dalla velocità di connessione del client.
Le soglie di attivazione sono state calibrate empiricamente su hardware di diversa potenza, con l'obiettivo di mantenere un framerate accettabile anche in condizioni di carico medio, senza che la qualità visiva nelle aree di interesse risultasse percettibilmente compromessa.

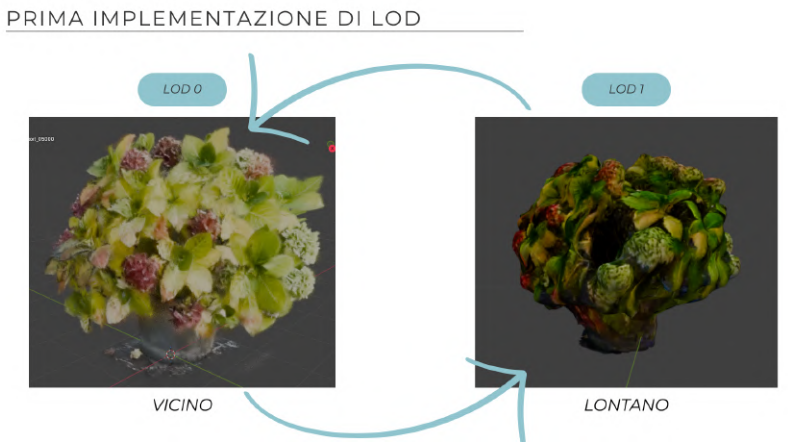

Figura 4.42: Prima implementazione dei Level of Detail (LOD).

La prima implementazione, visibile in Figura 4.42, ha confermato la validità dello schema concettuale, ma le prestazioni risultavano ancora insufficienti su hardware di fascia media quando il numero di elementi nella scena aumentava. Il problema principale non era la transizione tra splat e mesh come meccanismo di fallback distante, bensì il costo intrinseco degli splat ad alta densità già nelle fasi di avvicinamento. È stato quindi introdotto un livello di ottimizzazione aggiuntivo a livello degli splat stessi, basato su uno strumento di decimazione avanzata che aggrega le gaussiane per prossimità spaziale anziché ridurle in modo non selettivo.

Il funzionamento prevede la definizione di un raggio di aggregazione: tutte le gaussiane comprese entro tale distanza vengono fuse in un'unica primitiva, il cui colore è campionato da uno degli elementi originali. Regolando il raggio è possibile generare versioni dello stesso modello a differenti livelli di compressione: valori nell'ordine di 1–3 cm preservano il dettaglio visivo per la visualizzazione ravvicinata, mentre valori più elevati, fino a circa 12 cm, producono modelli significativamente più leggeri destinati alla visualizzazione a distanza.

La seconda implementazione, illustrata in Figura 4.43, integra questa stratifica di qualità all'interno dello schema LOD preesistente: lo splat ad alta densità viene utilizzato nelle vicinanze della camera, la versione aggregata a bassa risoluzione sostituisce quello ad alta densità oltre una certa soglia, e la mesh poligonale semplificata rimane come livello di fallback per le distanze maggiori e durante il caricamento asincrono.

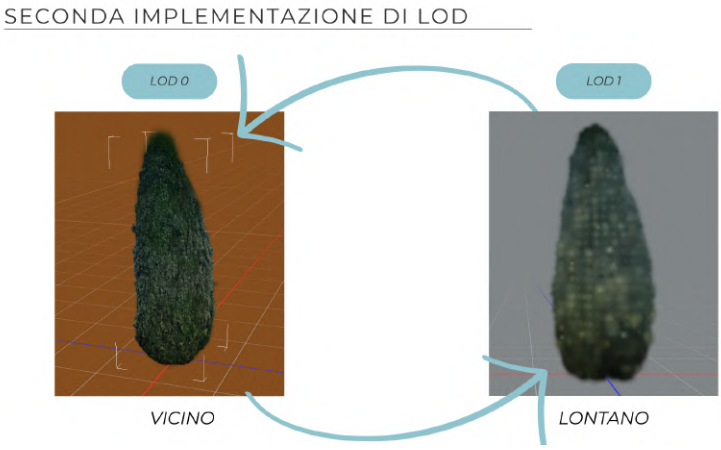

Figura 4.43: Seconda implementazione dei Level of Detail (LOD).

L'effetto visivo del modello a bassa risoluzione è caratteristico: le gaussiane fuse producono aree cromatiche più uniformi, percepite come macchie di colore piuttosto che come struttura vegetale dettagliata. A distanza ravvicinata questo risulterebbe inaccettabile, ma il livello low-resolution è progettato per entrare in scena solo quando la camera si trova sufficientemente lontana dall'elemento. In quella condizione, come mostrato in Figura 4.44, la differenza rispetto alla versione ad alta densità risulta difficilmente percepibile, e il guadagno in termini di framerate è misurabile.

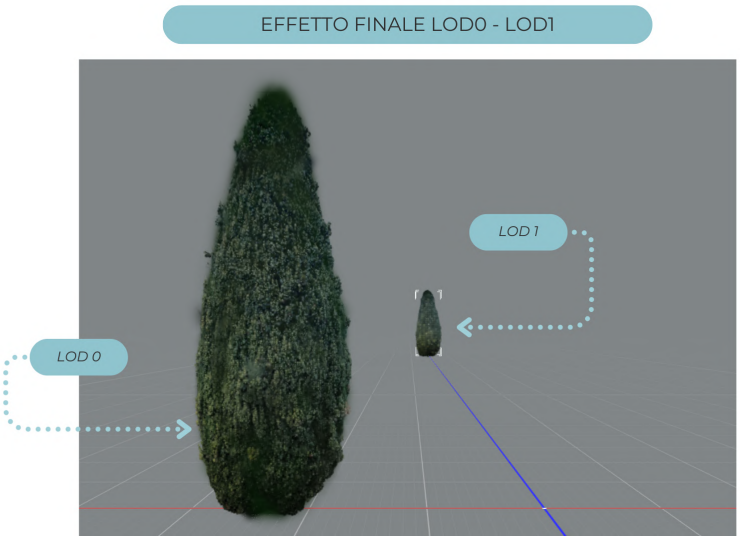

Figura 4.44: Resa finale dei Level of Detail (LOD).

Capitolo 5

DT-Certosa: Infrastruttura e Backend

Il capitolo precedente ha affrontato il primo dei tre rami della mappa concettuale in Figura 4.1, relativo alla modellazione tridimensionale del Monumento Bottoni e delle aree circostanti. Il presente capitolo completa il quadro trattando i due rami rimanenti: l'infrastruttura software e la logica di interazione. Le due componenti sono distinte, ma dipendono entrambe dalla geometria prodotta nel Capitolo 4. Senza un ambiente tridimensionale navigabile, né il sistema di persistenza né la navigazione guidata avrebbero avuto un contesto su cui operare.

La componente infrastrutturale riguarda l'estensione del framework ATON rispetto alla configurazione di base, con particolare attenzione alla gestione dei dati biografici associati alle sepolture. La soluzione adottata prevede l'introduzione di PocketBase come backend dedicato, con la conseguente migrazione da strutture dati statiche incorporate nel codice a un sistema di persistenza interrogabile tramite API REST. L'integrazione è stata progettata in modo da preservare la compatibilità con la logica applicativa preesistente, ricostruendo dinamicamente le strutture attese dal codice a partire dai dati caricati al momento dell'avvio.

La logica di interazione affronta invece il problema della navigazione guidata all'interno dell'ambiente virtuale. L'obiettivo era consentire all'utente di raggiungere una tomba specifica percorrendo un percorso fisicamente plausibile nello spazio della Certosa, indipendentemente dalla posizione corrente della camera. Questo ha richiesto la definizione esplicita di un grafo di navigazio-

ne, l'implementazione dell'algoritmo A* per il calcolo del cammino minimo e l'introduzione di un meccanismo di percorrenza sequenziale dei punti di vista che ATON non contemplava nella sua forma nativa. La stessa logica è stata estesa per supportare la modalità multi-utente, che consente a un presentatore di guidare uno o più visitatori lungo un percorso sincronizzato.

Il capitolo tratta inoltre il posizionamento runtime degli splat di vegetazione, che si appoggia al file degli empty esportato da Blender descritto nel Capitolo 4: la logica di classificazione per tipo e di istanziazione on demand degli asset ad alta densità è parte integrante del layer applicativo.

Chiude il capitolo la gestione dello swap dei livelli di dettaglio (LOD) in funzione della posizione della camera, componente che collega direttamente la struttura del modello tridimensionale con il comportamento dinamico dell'applicazione durante la navigazione.

5.1 Creazione e import del database al framework

Nella fase iniziale del progetto, i dati relativi alle tombe e alle persone sepolte nel Monumento Bottoni erano gestiti tramite strutture dati statiche definite direttamente nel codice JavaScript dell'applicazione. Questa soluzione, benché funzionale per una prima prototipazione, presentava limiti evidenti in termini di manutenibilità, scalabilità e separazione tra logica applicativa e contenuto informativo.

Le due strutture principali erano denominate `db` e `db2`. La prima associava l'identificativo numerico di ciascuna tomba alla lista dei nomi delle persone in essa sepolte; la seconda conteneva per ogni persona un insieme di dati anagrafici e biografici, incluso il riferimento alla tomba di appartenenza tramite il campo `tombino`. Un'analisi di queste strutture ha evidenziato che `db2` conteneva già tutte le informazioni necessarie: la struttura `db` risultava ridondante, poiché le relazioni tra persone e tombe potevano essere ricostruite dinamicamente a partire dai dati di `db2` (si veda il Listato 5.1 per un estratto rappresentativo della struttura, riportato più avanti in questa sezione).

Per superare questi limiti si è scelto di introdurre **PocketBase** come sistema di backend dedicato alla gestione persistente e strutturata dei dati.

5.1.1 PocketBase

PocketBase [28] è un backend open-source leggero, distribuito come singolo binario eseguibile, progettato per applicazioni web che richiedono un'infrastruttura backend completa senza la complessità di soluzioni enterprise. Il sistema è scritto in Go e utilizza SQLite come motore di storage interno, garantendo performance adeguate per dataset di dimensioni moderate senza richiedere la configurazione di server di database separati.

Le principali funzionalità offerte dalla piattaforma comprendono:

- un database relazionale con supporto nativo a relazioni tra entità;
- un sistema di API REST generate automaticamente a partire dalla definizione delle collezioni;
- un pannello di amministrazione accessibile via browser per la gestione diretta dei dati;
- un sistema di autenticazione per utenti con supporto a più metodi di login;
- un meccanismo di notifiche in tempo reale tramite connessioni Server-Sent Events.

PocketBase si posiziona come alternativa self-hosted a servizi come Firebase. Tutta l'infrastruttura risiede su macchine controllate dal progetto, senza dipendenze da fornitori esterni. L'avvio del server avviene con un unico comando:

```
./pocketbase serve
```

Il server risulta accessibile di default all'indirizzo `http://127.0.0.1:8090`, mentre il pannello di amministrazione è raggiungibile alla rotta `/_/`.
Si può vedere un esempio di terminale nella Figura 5.1. Alla prima apertura viene richiesta la creazione di un utente amministratore. Nella stessa directory del binario viene creata automaticamente una cartella `pb_data/` contenente il database SQLite e i file di configurazione.

I dati in PocketBase sono organizzati in collezioni, analoghe a tabelle in un database relazionale, ciascuna definita da un insieme di campi tipizzati. Il

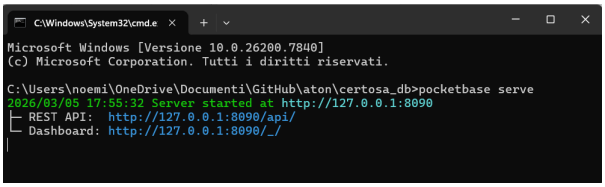

Figura 5.1: Esempio di terminale con PocketBase in esecuzione.

sistema supporta relazioni esplicite tra collezioni tramite campi di tipo *Relation*, che richiedono come valore l'identificativo interno del record correlato e non un valore arbitrario definito dall'utente.

5.1.2 Utilizzo di PocketBase nel framework

Progettazione delle collezioni

La modellazione del database ha seguito direttamente la struttura logica dei dati originari. Sono state definite due collezioni principali.

La collezione `tombe` (visibile sotto nella Figura 5.2) contiene un unico campo significativo: `numero`, di tipo numerico e con vincolo di unicità. Questo campo corrisponde all'identificativo numerico originariamente usato come chiave nella struttura `db`.

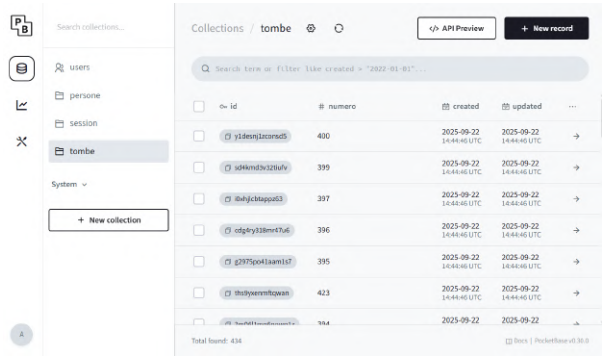

Figura 5.2: Configurazione della collezione `tombe` in PocketBase.

La collezione `persone` (nella Figura 5.3) replica i campi della struttura `db2`: `nome` e `nome_battaglia` di tipo testuale, `link` per l'URL alla fonte biografica, `biografia` come campo testo libero, e infine `data_nascita` e `data_morte` come campi testuali. Le date sono memorizzate come stringhe con i mesi in italiano, formato non compatibile con i tipi data nativi di PocketBase. Il campo

`tombino` è di tipo *Relation* con riferimento alla collezione `tombe`: la relazione è di tipo molti-a-uno, ovvero più persone possono essere associate alla stessa tomba mentre ogni persona appartiene a una sola tomba.

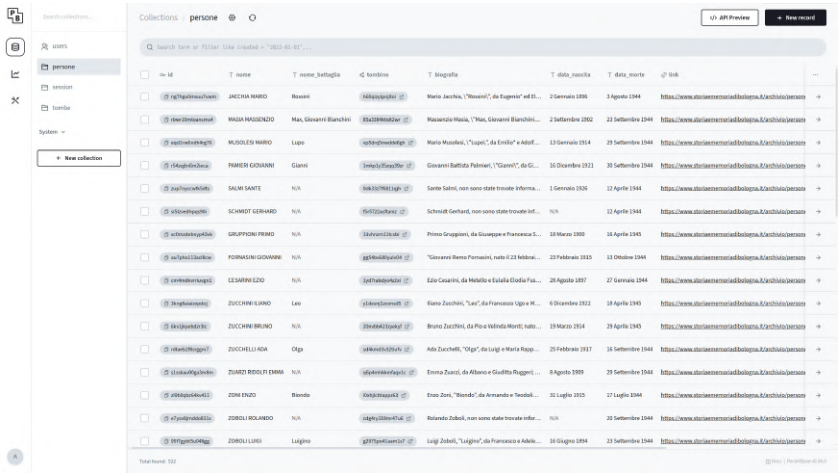

Figura 5.3: Configurazione della collezione `persone` in PocketBase.

Import dei dati

PocketBase non espone un meccanismo di importazione JSON diretto tramite interfaccia grafica: il caricamento dei dati avviene necessariamente attraverso le API REST.

Prima di descrivere il processo conviene richiamare la struttura effettiva di `db2`, ovvero la sorgente dati di partenza.
Ogni chiave è il nome completo del partigiano, comprensivo di nome di battaglia quando presente; il valore associato è un oggetto con i campi anagrafici e biografici, tra cui `tombino` come riferimento numerico alla tomba:

```
"ALBERTAZZI BRUNO  Pippo": {
    nome: "ALBERTAZZI BRUNO",
    nome_battaglia: "Pippo",
    link: "https://www.storiaememoriadibologna.it/...",
    tombino: 476,
    biografia: "Bruno Albertazzi, ...",
    data_nascita: "5 Giugno 1926",
    data_morte: "10 Aprile 1945"
},
```

```
"A. MURR": {
    nome: "A. MURR",
    nome_battaglia: "",
    link: "https://...",
    tombino: 247,
    biografia: "A. Murr, ...",
    data_nascita: "",
    data_morte: "8 Settembre 1944"
}
```

Listing 5.1: Estratto della struttura `db2` con due voci rappresentative.

Per automatizzare il caricamento è stato scritto uno script Node.js (`import.js`) che utilizza l'SDK JavaScript di PocketBase, installabile tramite:

```
npm install pocketbase
```

Lo script esegue le seguenti operazioni:

- si autentica come amministratore tramite le API PocketBase;
- itera tutte le voci di `db2`;
- per ogni persona, verifica se la tomba corrispondente esiste già nella collezione `tombe`; in caso contrario la crea;
- ottenuto l'identificativo interno della tomba, crea il record nella collezione `persone` impostando il campo relazionale `tombino`;
- mantiene una cache in memoria per evitare richieste ridondanti per tombe già elaborate.

Il Listato A.3 in Appendice A.2 riporta il nucleo dello script.

Nella prima esecuzione sono emersi due problemi distinti. Il campo `link` causava errori di validazione per URL privi del prefisso `http://`: il problema è stato risolto con una verifica preventiva che sostituisce il valore con una stringa vuota. Per i campi testuali lunghi come `biografia` è stata necessaria la conversione esplicita tramite `String()` per evitare errori di serializzazione con caratteri speciali o contenuto esteso. Il secondo problema riguardava il campo relazionale `tombino`: PocketBase richiede l'identificativo interno del record, non il numero della tomba. Una prima importazione aveva prodotto record con la relazione vuota; la correzione è stata uno script separato (`fixedRelations.js`)

che carica tutte le tombe, costruisce una mappa da numero a identificativo interno e aggiorna i record con la relazione mancante.

Un aspetto da configurare esplicitamente riguarda le regole di accesso: le collezioni PocketBase sono private per impostazione predefinita, e qualsiasi richiesta non autenticata viene rifiutata. Durante l'integrazione si sono verificati errori del tipo
`ClientResponseError 0: Only superusers can perform this action`.
Si sono dunque modificate le regole `listRule` e `viewRule` per consentire la lettura pubblica dei dati, lasciando inalterate le restrizioni di scrittura.

Integrazione con l'applicazione ATON

L'obiettivo dell'integrazione è stato preservare la compatibilità con la logica applicativa esistente, che si aspettava le strutture `db` e `db2` come variabili JavaScript. Anziché riscrivere la logica esistente, le strutture vengono ricostruite a partire dai dati letti da PocketBase all'avvio.

La funzione `loadData()` gestisce questo processo: recupera l'intera collezione `persone` tramite SDK JavaScript di PocketBase [27] usando il parametro `expand` per risolvere automaticamente la relazione verso `tombe`, poi ricostruisce in memoria le strutture `db` e `db2` nel formato atteso dal codice preesistente. Solo al termine del caricamento viene inizializzata la scena. La struttura completa della ricostruzione è riportata nel Listato A.4 in Appendice A.2.

In questo modo il backend PocketBase non conosce nulla della logica di visualizzazione, e il codice applicativo non dipende dalla struttura interna del database.

Gestione della navigazione tra tombe

Un problema specifico è emerso nella navigazione sequenziale tra tombe. L'implementazione originaria usava il numero della tomba come indice nell'array `APP.POVS`, presupponendo che il record con numero n si trovasse alla posizione n nell'array. Con l'introduzione di PocketBase, i record vengono restituiti in ordine non prevedibile e la numerazione delle tombe non è necessariamente contigua. La soluzione adottata è stata la costruzione di un array globale `tombiniOrdinati`, popolato una sola volta dopo il caricamento dei dati e ordinato numericamente. La navigazione avanti e indietro tra le tombe utilizza l'indice reale nell'array ordinato, non il numero della tomba.

5.2 Navigazione dell'utente nella webapp

La navigazione all'interno della webapp è uno degli aspetti tecnicamente più rilevanti dell'intero progetto. L'obiettivo era consentire a un utente di raggiungere una tomba specifica seguendo un percorso nell'ambiente virtuale, indipendentemente da dove si trovasse al momento della selezione. Questo requisito ha richiesto l'introduzione di un sistema di navigazione guidata assente nel framework ATON nella sua forma nativa.

5.2.1 Pipeline dell'implementazione della navigazione

Lo sviluppo è avanzato per fasi successive, con ciascuna fase che ha introdotto soluzioni a problemi concreti emersi nella precedente.

Il punto di partenza era una webapp che modellava l'interno dell'ossario e permetteva all'utente di muoversi liberamente tramite doppio tap. Con l'estensione del modello agli spazi esterni, la navigazione ha dovuto coprire anche il percorso tra l'area esterna e l'ingresso dell'ossario, richiedendo una gestione più strutturata degli spostamenti.

Il requisito centrale era il seguente: quando un utente seleziona un partigiano dalla lista presente nell'interfaccia, l'applicazione deve condurlo fino alla tomba corrispondente seguendo il percorso più breve disponibile. Questo indipendentemente dalla posizione corrente dell'utente, anche se molto distante dall'ossario.

Le fasi principali dello sviluppo sono state:

- definizione del grafo di navigazione;
- analisi dei limiti nativi di ATON nella gestione del percorso multipunto;
- implementazione dell'algoritmo di pathfinding;
- modifica del file sorgente del framework;
- realizzazione di un flare dedicato che incapsulasse il comportamento complessivo.

5.2.2 Path e creazione in ATON

Concetto della classe POV

All'interno di ATON, la posizione corrente dell'utente nello spazio tridimensionale è identificata dalla posizione della camera. Il framework mette a disposizione la classe `ATON.POV` [9] (Point of View), che rappresenta un punto di vista nello spazio e si compone di tre elementi:

- `position`: la posizione della camera nel mondo tridimensionale;
- `target`: il punto verso cui la camera è orientata;
- `fov`: il campo visivo.

I POV erano già utilizzati nella webapp per rappresentare le posizioni delle tombe: per ciascuna tomba era definito un `ATON.POV` che posizionava la camera in prossimità della lapide e orientava la visuale verso di essa, contenuti nell'array `APP.POVS`.

Namespace ATON.Nav

Il namespace `ATON.Nav` gestisce il sistema di navigazione del framework. Tra le sue funzioni principali vi è `requestPOV(pov, duration)`, che effettua una transizione animata dalla posizione corrente della camera al POV specificato nel tempo indicato. Questa funzione è progettata per spostamenti singoli: interpola posizione e orientamento della camera tra lo stato corrente e quello di destinazione producendo un effetto di transizione.

Creazione del grafo di navigazione

La definizione del percorso fisico all'interno della Certosa è avvenuta in Blender, utilizzando come riferimento la planimetria reale dell'area esterna all'ossario. Come mostrato in Figura 5.4, il percorso è stato modellato manualmente a partire da un singolo vertice, estrudendolo progressivamente fino a costruire una struttura composta da vertici e segmenti che descrive il cammino possibile nell'ambiente esterno.

Nella sua forma più semplice, questo oggetto Blender si riduce, al momento dell'esportazione in formato OBJ, a una lista di vertici (`v`) con relative coordinate tridimensionali e di linee (`l`) che specificano quali coppie di vertici sono

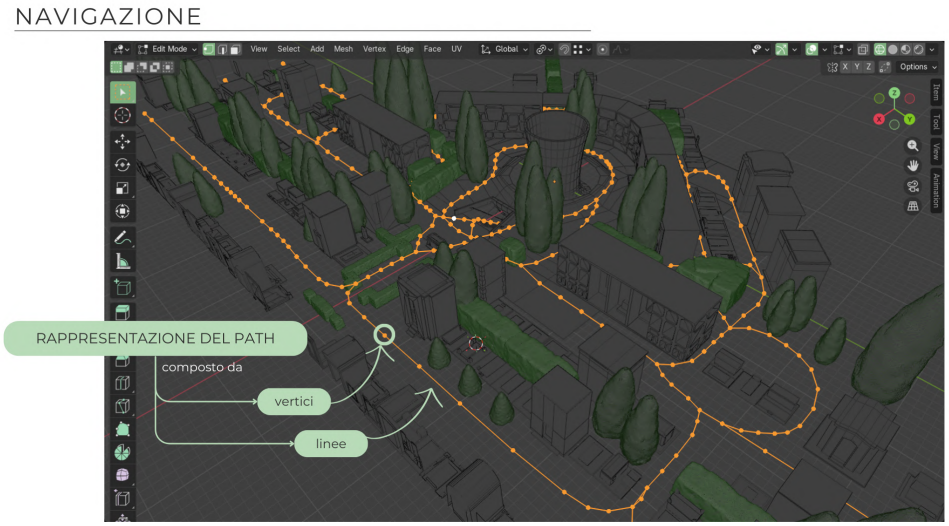

Figura 5.4: Vista esterna del path di navigazione in Blender.

collegate. Il file OBJ così prodotto costituisce la rappresentazione esplicita del grafo di navigazione.

```
v  -6.716263   1.367521   -4.952711
v  15.957236   1.367521   -4.090707
v  -13.727469  1.391067   15.377224
...
l  1  105
l  105  106
l  106  107
l  9  107
```

Listing 5.2: Estratto del file OBJ del grafo di navigazione: vertici (`v`) e connessioni (`l`).

All'avvio dell'applicazione, la funzione `importObjAsPOVs()` carica questo file, analizza le righe di tipo `v` e `l`, costruisce per ciascun vertice un oggetto `ATON.POV` con la posizione corrispondente, e popola gli array `APP.NAVIGATION_POV`, `APP.NAVIGATION_NODES` e `APP.NAVIGATION_EDGES`.

Il target iniziale di ciascun POV è impostato con un offset di default lungo l'asse `-Z`, in quanto la direzione di orientamento di ogni nodo non è determinabile staticamente ma dipende dal percorso effettivamente calcolato.

5.2.3 Concetto di pathfinding

Un grafo di navigazione descrive quali connessioni esistono tra i nodi, ma non determina quale percorso scegliere. Per questo è necessario un algoritmo esplicito.

Il termine *pathfinding* indica le tecniche algoritmiche per la ricerca del percorso minimo in un grafo: dato un nodo di partenza e uno di arrivo, si cerca la sequenza di nodi intermedi che minimizza un costo definito, in questo caso la distanza geometrica complessiva.

Un requisito di progetto era che l'implementazione fosse indipendente dalla topologia specifica del grafo della Certosa, così da risultare riutilizzabile in altri contesti senza modifiche.

5.2.4 Generazione del percorso più breve

Utilizzo di A*

L'algoritmo scelto per il calcolo del percorso minimo è A* (*A-star*). Rispetto a Dijkstra, che esplora il grafo espandendo uniformemente tutti i nodi in ordine di costo crescente, A* affianca al costo effettivo $g(n)$ una *funzione euristica* $h(n)$ che stima il costo residuo verso la destinazione. Il costo di valutazione di ogni nodo è quindi:

$$f(n) = g(n) + h(n)$$

La stima riduce il numero di nodi esplorati rispetto a una ricerca esaustiva, senza compromettere l'ottimalità del risultato, purché l'euristica sia *ammissibile*, ovvero non sovrastimi mai il costo reale. In questo caso si utilizza la distanza euclidea tra il nodo corrente e quello di arrivo, che è ammissibile perché i pesi degli archi corrispondono alle distanze geometriche effettive.

La funzione `shortestPath(nodes, edges, start, goal)`, che implementa A* su strutture dati native JavaScript, mantiene per ogni nodo un valore `gScore` e una priority queue ordinata per `fScore`. Al termine, il percorso viene ricostruito risalendo il vettore `prev` dal nodo di arrivo verso quello di partenza.

5.2.5 Modifica del file sorgente

Con l'algoritmo di pathfinding disponibile, il problema successivo era percorrere la sequenza di POV calcolata in modo fluido. La funzione nativa

`ATON.Nav.requestPOV()` è progettata per transizioni singole: interpola la camera dal punto corrente al punto di arrivo con un suo profilo di easing indipendente.

Concatenare più chiamate tramite `setTimeout` produce un percorso visivamente frammentato: ogni segmento accelera e decelera autonomamente, e la successione rapida di queste micro-transizioni risulta percettivamente scattosa. La limitazione è strutturale nella funzione, concepita per spostamenti singoli.

Dopo aver verificato che nessuna funzionalità equivalente fosse disponibile nel framework e dopo aver confermato con il creatore di ATON che questa feature non era prevista nell'immediato futuro, si è deciso di intervenire direttamente sul file sorgente `ATON.Nav.js`.

Il file sorgente può essere modificato e il progetto rebuilt tramite:

```
npm run build-aton
```

Le modifiche hanno introdotto nel namespace `Nav` la funzione `travelPath(povs, options)`, che accetta un array di `ATON.POV` e gestisce l'intero percorso come un'unica animazione continua. La funzione:

- calcola le distanze cumulative lungo il percorso, garantendo una velocità di avanzamento uniforme indipendentemente dalla distanza tra nodi consecutivi;
- applica una spline Catmull-Rom su posizione e orientamento, eliminando le discontinuità angolari nei cambi di direzione;
- usa una velocità adattiva che rallenta in corrispondenza delle curve più marcate.

Per la valutazione della curvatura si considera esclusivamente la componente nel piano orizzontale (assi `x` e `z`), poiché la variazione di quota lungo il percorso segue una logica propria e non deve influenzare la stima delle curve planari.

Il target della camera viene aggiornato guardando un punto più avanti lungo la curva rispetto alla posizione corrente, con un valore di *look-ahead* calibrato sperimentalmente, e filtrato con un ulteriore lerp per smorzare eventuali oscillazioni residue. Questa soluzione ha tuttavia un limite architetturale rilevante.

Modificare `ATON.Nav.js` è praticabile all'interno del repository privato della webapp, dove il codice del framework viene incluso direttamente, ma non è una strada sostenibile a lungo termine: il sorgente di ATON è mantenuto nel suo repository ufficiale, sul quale non è possibile effettuare commit. Qualsiasi aggiornamento del framework richiederebbe di reintrodurre manualmente le modifiche, con un rischio crescente di regressioni.

Per questa ragione la logica di percorrimento è stata spostata il più possibile al di fuori del sorgente, limitando le modifiche a `ATON.Nav.js` al minimo strettamente necessario per esporre i nuovi punti di ingresso. Il comportamento effettivo è incapsulato in un flare dedicato, distribuito in un repository separato e riutilizzabile indipendentemente dalla versione del framework. Il progetto mantiene quindi tre repository distinti: uno per la webapp specifica della Certosa, uno per il flare di pathfinding e uno per il backend PocketBase.

5.2.6 Implementazione di un flare

Cos'è un flare in ATON

ATON adotta il termine *flare* come denominazione interna per quello che in altri framework verrebbe chiamato plugin: moduli JavaScript esterni caricabili dinamicamente, progettati per estendere le funzionalità del framework senza modificarne il nucleo. I flare vengono dichiarati nell'applicazione tramite `APP.requireFlares([...])`, caricati in fase di inizializzazione e resi disponibili dopo l'evento `AllFlaresReady`. Questa architettura consente di distribuire funzionalità aggiuntive come componenti autonomi e riutilizzabili in webapp differenti.

La scelta di spostare il codice del path following all'interno di un flare dedicato ha risposto a due esigenze: mantenere le modifiche a `ATON.Nav.js` al minimo indispensabile e rendere la funzionalità di navigazione guidata disponibile come componente autonomo, riutilizzabile in webapp differenti senza dipendere dalla struttura specifica della Certosa.

Il flare `pathfinding` incapsula la logica di percorrimento e riceve dall'esterno i parametri necessari, tra cui la sequenza ordinata di POV da attraversare e una callback opzionale da invocare al termine del percorso.

L'interfaccia esposta dal flare è:

```
F.pathfinding.travelPath(povs, options, onTap, onComplete)
// povs: array di ATON.POV da percorrere in sequenza
// options: velocita', look-ahead, smoothing, accelerazione iniziale
// onTap: callback invocata a ogni tap dell'utente durante il percorso
// onComplete: callback invocata al termine del percorso
```

Listing 5.3: Firma della funzione principale del flare `pathfindingflare`.

Il flare espone inoltre le seguenti funzioni di controllo, collegate direttamente al namespace `ATON.Nav`:

- `stopPathTravel()`: interrompe definitivamente il percorso;
- `pausePathTravel()`: sospende temporaneamente l'avanzamento;
- `resumePathTravel()`: riprende dal punto in cui era stato sospeso.

Il Listato A.5 in Appendice A.3 riporta la struttura completa del flare.

5.2.7 Implementazione definitiva

Gestione del punto di partenza

Definito il comportamento del flare durante il percorso, resta da stabilire da quale posizione l'utente si trova al caricamento della webapp. Al primo accesso, la camera viene posizionata su un nodo fisso del grafo, scelto a priori e configurato tramite:

```
APP.START_NODE_INDEX = 472;
```

Il nodo selezionato è frontale al monumento, a distanza sufficiente da offrire una visione complessiva dell'ambiente alla prima apertura, come mostrato in Figura 5.5. Da questa posizione iniziale, o da qualsiasi altra in cui l'utente si trovi dopo aver navigato liberamente, l'avvio di un percorso guidato solleva un problema: la posizione corrente della camera non coincide necessariamente con un nodo del grafo. Avviare il flare direttamente dal nodo più vicino avrebbe richiesto un primo spostamento separato con `requestPOV()`, producendo due animazioni consecutive con profili di movimento distinti e un effetto percettivamente discontinuo.

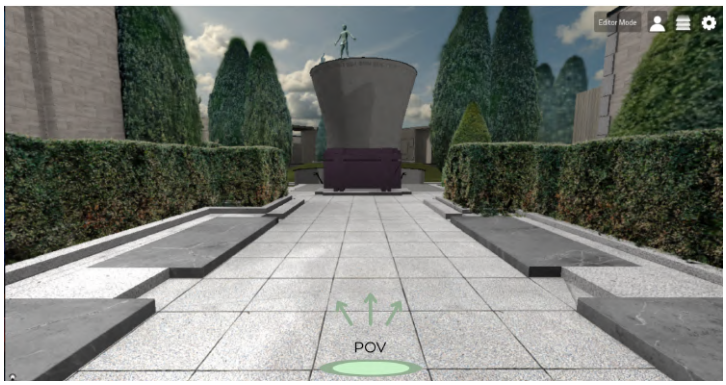

Figura 5.5: Posizione iniziale della camera all'avvio della webapp: nodo fisso frontale al Monumento Bottoni.

La soluzione adottata, mostrata in Figura 5.6, consiste nell'inserire in testa all'array del percorso la posizione corrente della camera al momento dell'attivazione, seguita dal nodo del grafo più vicino e dalla sequenza completa di POV calcolati da A*. Il flare tratta l'intera sequenza come un unico percorso: parte dalla posizione corrente con velocità zero, accelera gradualmente e prosegue senza interruzioni fino alla destinazione.

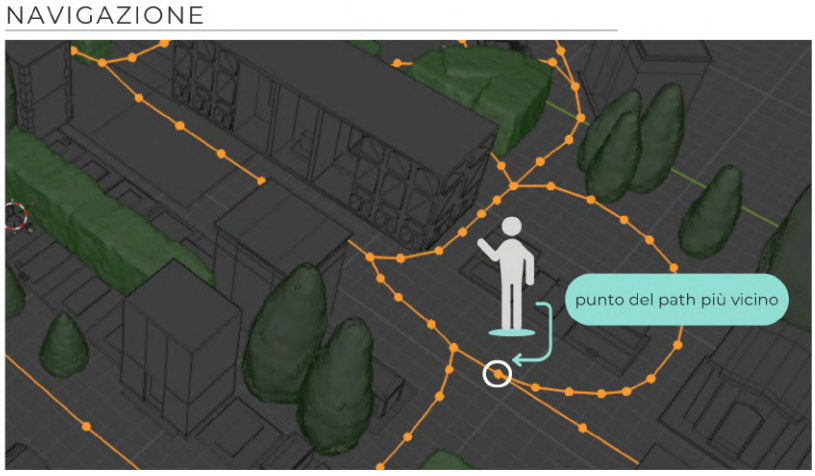

Figura 5.6: Gestione del punto di partenza: dalla posizione corrente della camera viene individuato il nodo del grafo più vicino; tale posizione viene inserita in testa all'array di POV calcolati, costruendo un percorso unico dalla posizione corrente fino alla destinazione.

Gestione del punto di destinazione del path

L'ossario del Monumento Bottoni ha una pianta circolare, con una statua al centro e le tombe distribuite lungo la parete perimetrale. Le sepolture sono organizzate in tre blocchi: sinistro, frontale e destro. Poiché l'ambiente è circolare, raggiungere il blocco frontale richiede di percorrere un semicerchio interno attorno alla statua.

Per avvicinare l'utente al blocco di tombe corretto, il sistema definisce tre nodi di destinazione distinti all'interno dello spazio dell'ossario. La funzione `selectNodeDestination(index)` riceve l'indice della tomba nell'array ordinato e restituisce il nodo appropriato:

- tombe con indice ≤ 190: nodo sul lato sinistro (`LEFT_NODE_INDEX`);
- tombe con indice compreso tra 191 e 310: nodo frontale (`FRONT_NODE_INDEX`);
- tombe con indice > 310: nodo sul lato destro (`RIGHT_NODE_INDEX`).

Ad esempio, selezionando una tomba con indice 170, il sistema imposta `LEFT_NODE_INDEX` come destinazione: il percorso esterno termina all'ingresso dell'ossario e il tratto interno conduce direttamente al lato sinistro, dove si trova il blocco di riferimento. I possibili percorsi sono illustrati in Figura 5.7.

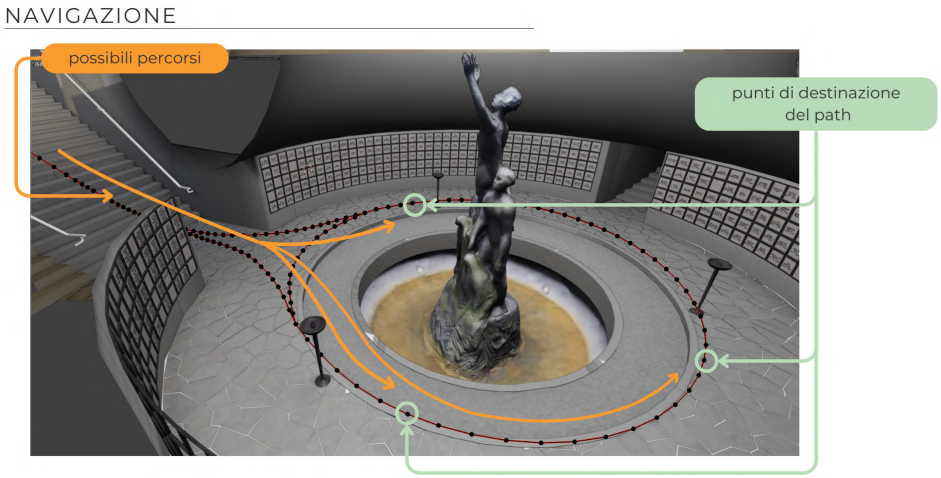

Figura 5.7: Schema dell'ossario con i tre nodi di destinazione del percorso guidato, selezionati dinamicamente in base al blocco di tombe da raggiungere.

Tutti e tre i nodi di destinazione si trovano a quota inferiore rispetto al piano esterno, poiché l'ossario è parzialmente interrato. Il flare gestisce esplici-

tamente la transizione verticale nell'ultimo segmento del percorso, anticipando la discesa con un orientamento del target verso il basso prima che il movimento inizi.

Invocazione del flare e struttura definitiva

Quando l'utente seleziona un partigiano dall'interfaccia, l'applicazione esegue la seguente sequenza:

1. calcola il percorso minimo tramite A* tra tutti i nodi del grafo, con destinazione al nodo selezionato da `selectNodeDestination`;

2. inserisce in testa all'array dei POV calcolati la posizione corrente della camera, costruendo un percorso unico dalla posizione corrente alla destinazione;

3. aggiorna il target di ciascun POV verso il nodo successivo nel path; questo passaggio è necessario perché tutti i POV del grafo vengono inizializzati con un offset fisso lungo l'asse `-Z` (si veda la Sezione 5.2.2), e occorre correggerli prima che il flare attivi il look-ahead adattivo;

4. invoca il flare passando la sequenza completa di POV e una callback da eseguire al termine del percorso.

A seguire si riportano alcune figure che illustrano le fasi principali del percorso guidato, dalla selezione del partigiano alla destinazione finale. La Figura 5.8 mostra il calcolo del percorso più breve a partire da tutti i percorsi del path possibili. Quello più breve è evidenziato in azzurro.

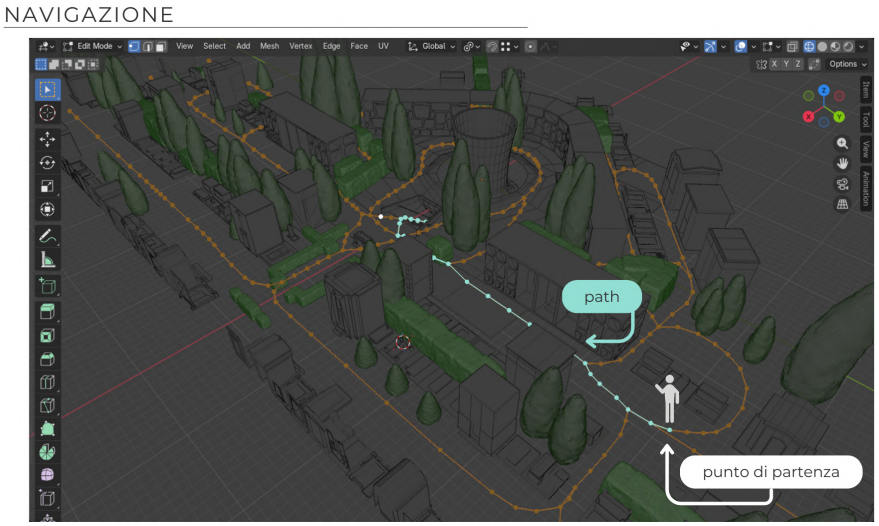

Figura 5.8: Schema del calcolo del percorso più breve con A*.

Segue la Figura 5.9 che mostra il punto di partenza del percorso, con la camera posizionata su un nodo del grafo e orientata verso il nodo successivo.

Figura 5.9: Punto di partenza del path.

La Figura 5.10 mostra la fase intermedia del percorso, con la camera che segue la curva adattiva tra i nodi.
La Figura 5.11 mostra l'approccio alla destinazione, con la camera che si avvicina all'ingresso dell'ossario, scende le scale e arriva vicino al nodo destinazione.

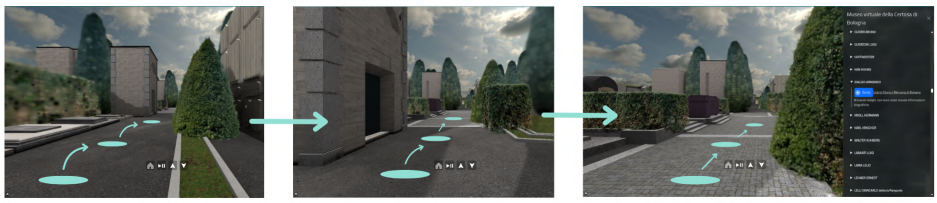

Figura 5.10: Percorso del path.

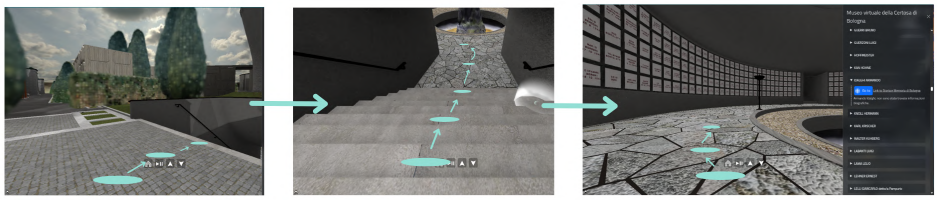

Figura 5.11: Continuazione del percorso del path.

Infine, la Figura 5.12 mostra la fase finale del percorso, con la camera che raggiunge il punto di destinazione all'interno dell'ossario e si orienta verso il tombino di interesse prima di eseguire l'avvicinamento finale alla lapide tramite close-up.

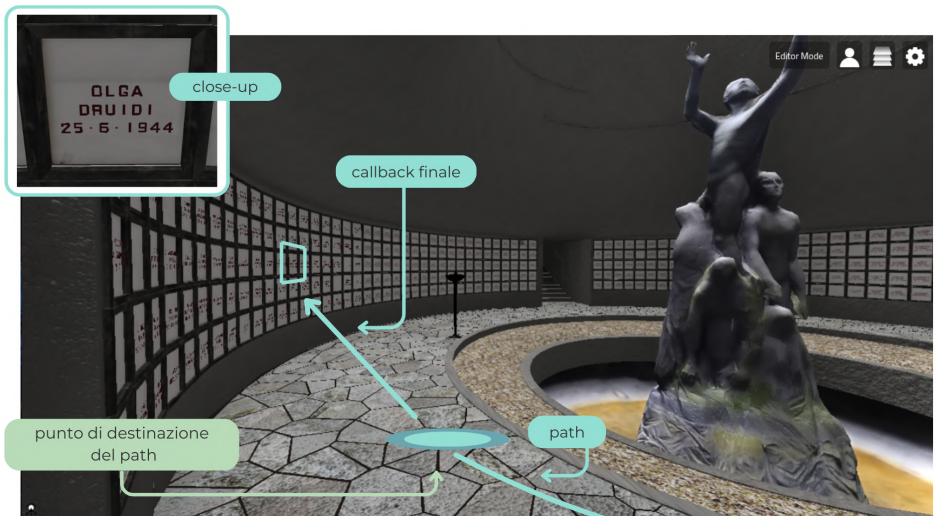

Figura 5.12: Schema dell'ultima fase del percorso guidato: a destinazione raggiunta, la callback esegue prima una rotazione in-place della camera verso il tombino di interesse, poi una `requestPOV()` di avvicinamento alla lapide.

Una volta raggiunto il punto di destinazione, il flare invoca la callback. La camera si trova a quel punto orientata nella direzione dell'ultimo segmento percorso, che in un ossario circolare non coincide necessariamente con la direzione delle tombe: ad esempio, arrivando dal lato frontale dopo aver percorso un semicerchio, la camera guarda ancora nella direzione del movimento, non verso la parete.

La callback gestisce questo con due operazioni in sequenza. Prima invoca `requestPOV()` con la stessa posizione corrente ma con il target aggiornato verso il tombino di interesse: la camera ruota su sé stessa, senza spostarsi, fino ad allinearsi con la tomba. Solo al termine di questa rotazione viene invocata la `requestPOV()` definitiva verso il POV ravvicinato della lapide.

Questo approccio in due fasi rende la transizione finale percettivamente naturale: prima ci si orienta, poi ci si avvicina. Una singola `requestPOV()` diretta al tombino manterrebbe l'orientamento corrente per la maggior parte dell'animazione, producendo uno spostamento laterale visivamente incoerente.

Il flare prevede la possibilità di passare una funzione ausiliaria, invocata ad ogni aggiornamento di frame durante il percorso. In questa webapp tale funzione corrisponde a `setupLodBehavior()`, una funzione generica che gestisce lo swap dei livelli di dettaglio sia per le iscrizioni dei tombini sia per la vegetazione in Gaussian Splatting. L'architettura è descritta nel dettaglio nella Sezione 5.5.

5.3 Implementazione della modalità multi-utente e Editor Mode

L'applicazione sviluppata per la Certosa, a partire dalle prime fasi di progettazione, non è stata concepita soltanto come ambiente tridimensionale esplorabile individualmente. L'obiettivo di fondo era quello di realizzare uno spazio narrativo capace di valorizzare il contenuto culturale attraverso l'interazione guidata, coerentemente con il concetto di monumento parlante discusso nei capitoli precedenti. In questa prospettiva, la sola modalità di fruizione individuale risultava insufficiente: si è resa necessaria l'introduzione di un sistema che permettesse a una figura guida di condurre altri utenti all'interno dello spazio virtuale, replicando la struttura di una visita museale.

Da questa esigenza derivano le due modalità operative implementate: **Viewer Mode**, ovvero la configurazione di fruizione passiva accessibile a qualsiasi utente, e **Editor Mode**, riservata al presenter che gestisce la sessione collaborativa.

5.3.1 Architettura multi-utente collaborativa in ATON

La comunicazione tra client distinti è gestita dal modulo **VRoadcast** di ATON, che implementa un sistema di eventi distribuibili in tempo reale tra istanze differenti dell'applicazione. Il meccanismo si basa su due primitive: `fireEvent`, per trasmettere un evento con payload arbitrario, e il relativo listener attivo sui client connessi alla stessa sessione.

Questo approccio introduce una separazione netta tra logica di sincronizzazione e logica applicativa: il layer di comunicazione non ha conoscenza del contenuto degli eventi, ma si limita a garantirne la distribuzione in tempo reale. Il codice applicativo può quindi definire liberamente quali dati trasmettere e come reagire alla loro ricezione, senza dipendere dalla struttura interna del modulo di rete.

Dal punto di vista dei requisiti tecnici, la collaborazione sincrona in un ambiente web 3D richiede che tutti i client mantengano uno stato coerente, che gli eventi di rete trasportino solo le informazioni strettamente necessarie all'aggiornamento dello stato e che il sistema scala in modo adeguato al crescere del numero di partecipanti. L'architettura basata su eventi di VRoadcast risponde a queste esigenze senza richiedere installazioni aggiuntive né configurazioni lato client, poiché tutto avviene nel browser tramite connessioni gestite dal framework.

5.3.2 Sistema di autenticazione e distinzione dei ruoli

Il punto di partenza dello sviluppo è stato un esempio di base fornito da ATON, che mostrava il solo funzionamento della sincronizzazione tra utenti. Questo esempio è stato completamente rielaborato e integrato con la struttura specifica della webapp della Certosa.

Nelle prime versioni dell'applicazione ogni utente accedeva alle stesse funzionalità senza alcuna distinzione. L'introduzione della modalità collaborativa ha reso necessaria una separazione esplicita dei livelli di accesso. Il sistema di autenticazione è stato costruito sopra PocketBase, già utilizzato come bac-

kend per i dati anagrafici, aggiungendo una collezione dedicata agli account presenter. Al momento è registrato un singolo utente con credenziali email e password, come mostrato in Figura 5.13, ma la struttura consente di aggiungere ulteriori presenter senza modifiche architetturali.

Figura 5.13: La collezione `users` in PocketBase, usata per l'autenticazione dei presenter.

L'autenticazione avviene tramite una finestra modale accessibile dall'interfaccia principale. L'utente inserisce le proprie credenziali; la password è mascherata graficamente.

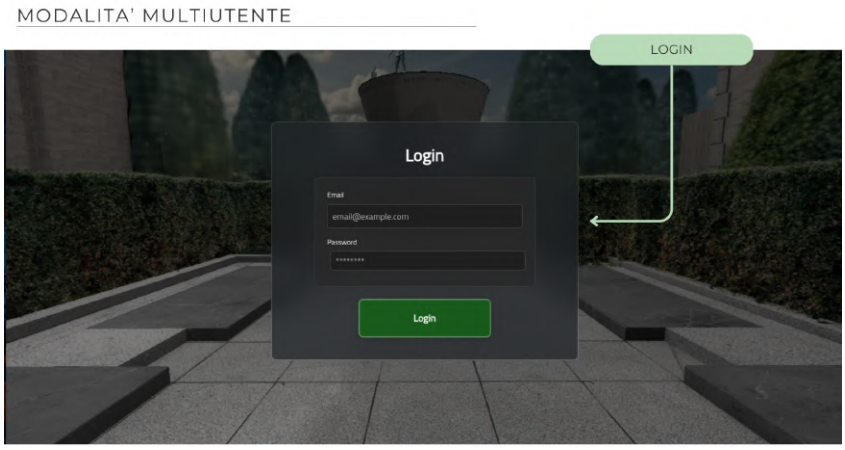

Figura 5.14: Finestra di login.

La verifica avviene tramite le API di PocketBase: in caso di successo lo stato applicativo viene aggiornato, in caso di errore viene mostrato un messaggio con la possibilità di riprovare.

Sul piano implementativo, la distinzione tra ruoli è gestita attraverso due variabili booleane globali:

```
APP.ViewerMode = true;
APP.EditorMode = false;
```

`ViewerMode` è attiva per impostazione predefinita su tutti gli utenti. Al completamento positivo dell'autenticazione i valori vengono invertiti, abilitando le componenti di interfaccia riservate al presenter. La transizione non altera la visualizzazione della scena tridimensionale, ma sblocca dinamicamente gli strumenti di gestione della sessione.

5.3.3 Creazione e gestione delle sessioni collaborative

Una volta autenticato, il presenter attiva la modalità editor tramite l'apposito controllo nell'interfaccia. L'attivazione determina un insieme di effetti immediati sulla scena.

La camera viene spostata automaticamente in una posizione sopraelevata con visione ortografica sull'intera area della Certosa. Da questa prospettiva il presenter può leggere la distribuzione dello spazio e pianificare il percorso guidato. Al contempo vengono applicate ottimizzazioni alle prestazioni: gli splat vengono forzati in bassa risoluzione, le ombre vengono semplificate e le texture caricate in versione ridotta. In questa fase l'obiettivo non è la qualità visiva ma la stabilità del sistema e la leggibilità della mappa.

La posizione originaria della camera, lo stato dei livelli di dettaglio e la configurazione degli asset caricati vengono salvati in memoria per consentire il ripristino coerente al termine della sessione.

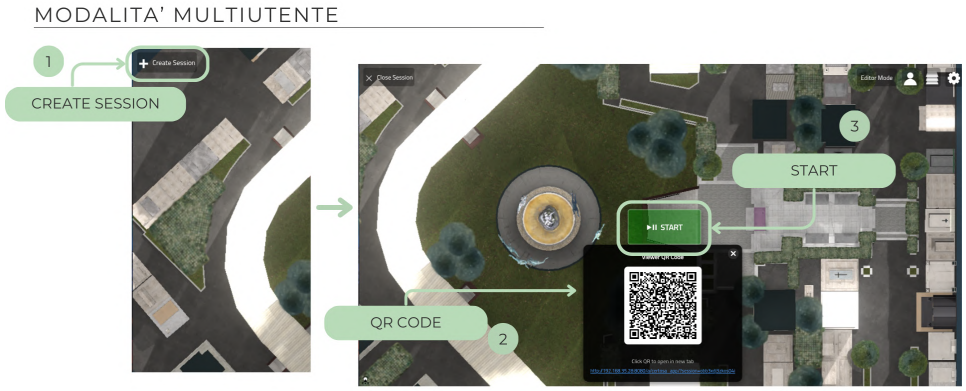

Figura 5.15: Creazione della sessione con generazione di QR code e link.

Come mostrato in Figura 5.15, il presenter può creare una nuova sessione collaborativa. La creazione genera un **identificativo univoco di sessione**

(session ID), salvato su PocketBase insieme ai dati associati. Contestualmente vengono prodotti un link e un QR code che puntano allo stesso identificativo. Il QR code viene generato una sola volta per sessione: generarne uno nuovo dopo che i viewer si sono già connessi causerebbe disallineamenti nello stato condiviso, rendendo il sistema instabile.

I viewer possono accedere alla sessione in tre modalità: apertura del link in un nuovo tab del browser, utilizzo di un dispositivo diverso sulla medesima rete oppure scansione del QR code con smartphone. In tutti i casi, l'utente che accede viene automaticamente configurato in Viewer Mode, con le funzionalità avanzate disabilitate. In particolare, sono escluse le funzioni GoTo del flare di pathfinding (descritto nella Sezione 5.2.6) e il menu laterale con la lista dei partigiani.

5.3.4 Creazione dinamica del percorso guidato

La funzionalità più complessa all'interno dell'Editor Mode è il sistema *Create Path*, che consente al presenter di costruire un percorso di visita personalizzato selezionando punti sulla mappa tramite doppio click. Ogni punto selezionato viene rappresentato visivamente da un indicatore circolare e inserito in un array temporaneo. La Figura 5.16 mostra come funziona la creazione del percorso guidato. Una volta cliccato il pulsante *Create Path*, il presenter può selezionare i punti sulla mappa, questi sono mostrati in verde.

Figura 5.16: Creazione del percorso guidato.

I punti selezionati manualmente non corrispondono necessariamente a nodi presenti nel grafo di navigazione, già caricato nell'array `APP.NAVIGATION_POV`. Per generare un percorso effettivamente percorribile il sistema esegue due fasi consecutive.

Nella prima fase, per ciascun punto selezionato viene individuato il nodo del grafo più vicino in termini di distanza euclidea. Questa proiezione produce un array di nodi validi che appartengono al grafo definito dal file OBJ.

Nella seconda fase, poiché i nodi ottenuti non sono necessariamente adiacenti nel grafo, tra ogni coppia consecutiva viene applicato l'algoritmo A* già implementato nella funzione `shortestPath`. Per ciascuna coppia (n_i, n_{i+1}) l'algoritmo restituisce la sequenza di nodi intermedi necessaria a connetterla lungo il grafo. Tutti i segmenti vengono concatenati in un unico array finale denominato *path*, che costituisce la sequenza completa e ordinata dei nodi da percorrere.

In sintesi, il flusso di costruzione del percorso procede come segue:

1. selezione manuale dei punti sulla mappa tramite doppio click;
2. proiezione di ciascun punto sul nodo reale più vicino nel grafo;
3. calcolo del percorso minimo tramite A* tra ogni coppia di nodi consecutivi;
4. espansione dei segmenti con i nodi intermedi;
5. generazione dell'array finale contenente il percorso continuo.

Questo passaggio è il più oneroso computazionalmente, con un costo proporzionale al numero di punti selezionati e alla dimensione del grafo. Nel caso della Certosa il grafo rimane comunque contenuto, quindi l'operazione è rapida in pratica.

5.3.5 Sincronizzazione del percorso tra Presenter e Viewer

Con il percorso costruito, il presenter può avviare la visita tramite il comando *Go To Path*, visibile nella Figura 5.17.

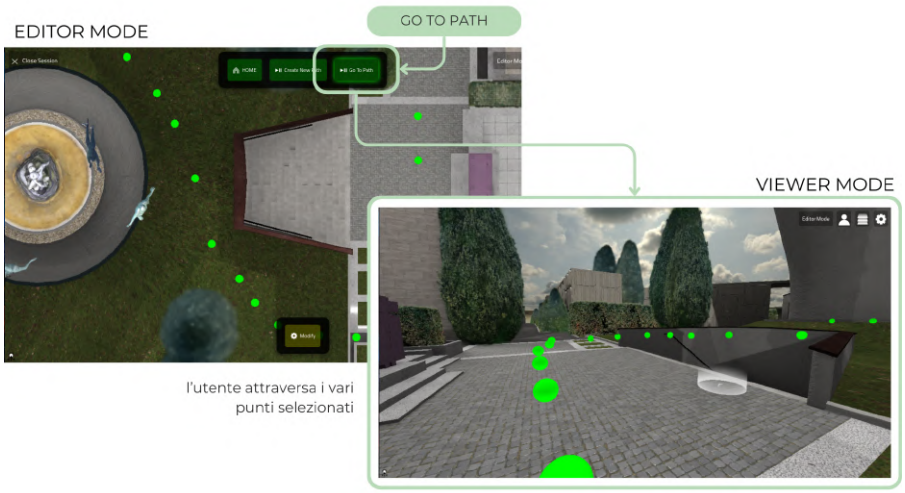

Figura 5.17: Avvio del percorso guidato dal punto di vista del presenter e dell'utente (viewer).

L'azione produce un `fireEvent` che trasmette l'array *path* completo a tutti i viewer connessi alla sessione. Ciascun client riceve la sequenza dei nodi e avvia autonomamente il percorrimento tramite il flare `pathfinding`, con la stessa logica già descritta per la navigazione individuale.

Durante il percorrimento viene invocata continuamente la funzione `setupLodBehavior`, responsabile dell'aggiornamento dinamico dei livelli di dettaglio. Il comportamento segue un principio di prossimità: gli elementi distanti rimangono in bassa risoluzione mentre splat e texture in alta qualità vengono caricati progressivamente man mano che il viewer si avvicina. Questo meccanismo era già operativo nel flare di navigazione individuale e viene riutilizzato direttamente nella modalità collaborativa, evitando duplicazioni logiche.

5.3.6 Gestione dello stato e chiusura della sessione

Il sistema include alcune funzioni di controllo per gestire situazioni operative comuni. Il presenter può in qualsiasi momento cancellare il percorso costruito e ricominciare, oppure interrompere la fase di creazione senza salvare. Sono presenti finestre di conferma per le azioni irreversibili, al fine di prevenire modifiche accidentali durante la sessione.

La sessione del presenter è mantenuta attiva attraverso i refresh della pagina, evitando la necessità di autenticarsi nuovamente ad ogni accesso. La chiu-

sura della sessione avviene tramite il comando *Close Session*, che richiede una conferma esplicita prima di eliminare il record corrispondente da PocketBase.

All'uscita dall'Editor Mode il sistema esegue un ripristino completo dello stato: la camera viene riportata alla posizione precedente all'attivazione, i livelli di dettaglio degli splat e delle texture vengono ripristinati alla configurazione ad alta risoluzione e tutti gli asset vengono ricaricati coerentemente con la configurazione originaria. Questo comportamento è reso possibile dallo snapshot dello stato salvato all'attivazione della modalità, che viene applicato in ordine inverso alla disattivazione.

5.4 Posizionamento degli splat di vegetazione

Il meccanismo di posizionamento degli splat si basa sul file GLB degli empty descritto nella Sezione 4.5.2 del Capitolo 4. All'avvio, il file viene caricato come nodo ATON invisibile: non deve comparire nella scena renderizzata, ma serve a rendere disponibili le trasformazioni degli empty come gerarchia di oggetti Three.js. Al completamento del caricamento, tramite l'evento `AllNodeRequestsCompleted`, la funzione `spawnSplatsForTrees()` itera i children del nodo, identifica il tipo di vegetazione dal prefisso del nome (`t` per i cipressi alti, `c` per i cipressi compatti, `b` per i cespugli) ed estrae posizione, rotazione e scala in coordinate world tramite `getWorldPosition()`, `getWorldQuaternion()` e `getWorldScale()`. L'uso delle coordinate world è necessario perché il GLB introduce un nodo padre intermedio con trasformazioni proprie: leggere la sola trasformazione locale produrebbe posizioni errate in scena.

Per ciascun empty classificato, la funzione crea immediatamente il nodo per lo splat a bassa densità e gli applica le trasformazioni estratte. Lo splat ad alta densità viene invece registrato come `null` e caricato on demand da `ensureHighForIndex(i)` solo quando la camera si avvicina: caricare tutto all'avvio appesantirebbe inutilmente il client, poiché il browser non espone controllo diretto sulla memoria GPU.

Il risultato architetturale è che modificare la disposizione della vegetazione richiede solo di aggiornare gli empty in Blender e riesportare il GLB, senza toccare il codice JavaScript. L'unica dipendenza implicita è la convenzione prefissale: un empty mal nominato non genera errori espliciti, ma viene

assegnato al tipo di default con l'asset sbagliato. La struttura completa di `spawnSplatsForTrees()` è riportata nel Listato A.6 in Appendice A.4.

5.5 Gestione dello swap dei livelli di dettaglio

Il Capitolo 4 ha definito, per due categorie di elementi distinte, uno schema di Level of Detail (LOD) a livelli discreti: le iscrizioni dei tombini (Sezione 4.3.1) e la vegetazione realizzata tramite Gaussian Splatting (Sezione 4.5.2). Per entrambe sono state preparate in fase di modellazione più versioni dello stesso elemento, con costi di rendering diversi. Come anticipato nei capitoli precedenti, la logica di selezione a runtime è trattata in questo capitolo: nello specifico, come l'applicazione decide quale versione mostrare, quando effettuare il cambio e come gestire la transizione senza introdurre discontinuità visive percepibili dall'utente.

Questo compito ricade interamente sul layer applicativo. Né ATON né Three.js offrono un meccanismo generico di LOD automatico adatto al progetto: la gestione dello swap è implementata nell'applicazione tramite la funzione `setupLodBehavior()`, invocata in modo continuativo durante la navigazione per mantenere la coerenza tra posizione della camera e qualità degli asset caricati. La funzione è progettata come componente generico: gestisce sia le iscrizioni dei tombini (font LOD) sia la vegetazione in Gaussian Splatting, accettando in ingresso gli array di asset ad alta e bassa risoluzione e le soglie di distanza. Per i font la preparazione degli array è affidata a `preparingUpdateViewLod()`, mentre per gli splat è `spawnSplatsForTrees()` a inizializzare le versioni a bassa densità e `ensureHighForIndex()` a caricare quelle ad alta densità on demand.

5.5.1 Criterio di attivazione basato sulla distanza

Il parametro di controllo del livello attivo è la distanza euclidea tra la posizione corrente della camera e ciascun elemento soggetto a LOD. Per le iscrizioni dei tombini la soglia è calibrata in modo che il passaggio alla geometria tridimensionale delle lettere (LOD 0) avvenga soltanto quando l'utente è abbastanza vicino da percepirne il rilievo. Al di sopra della soglia, il sistema mantiene attivo il piano con texture baked (LOD 1): visivamente equivalente alla distanza in questione, ma con un costo di rendering significativamente inferiore. Da vi-

cino la differenza tra i due livelli è ben visibile; a grande distanza dall'ossario, invece, i due livelli sono percettivamente indistinguibili. Caricare la geometria tridimensionale delle iscrizioni quando l'utente è lontano non avrebbe alcun effetto visivo, quindi il sistema mantiene attivo il piano con texture baked finché non è necessario.

Per gli splat della vegetazione il meccanismo è analogo, ma le soglie sono più alte. Il motivo è che il costo degli splat è additivo: ogni gaussiana attiva pesa sulla pipeline di rendering, e con molte piante in scena le risorse del client si esauriscono rapidamente. La vegetazione è suddivisa in tre tipologie distinte, ciascuna con asset separati ad alta e bassa densità: cipressi (`cypress`), alberi generici (`trees`) e cespugli (`bush`). Al di sotto della soglia viene attivato lo splat ad alta densità per la tipologia corrispondente; oltre la soglia rimane visibile la mesh semplificata ottenuta da Meshroom.

La transizione tra i due livelli non avviene con uno scambio diretto. Effettuando uno switch netto, comparivano istanti in cui lo splat ad alta densità non era ancora caricato ma quello a bassa risoluzione era già stato rimosso, con il risultato di una zona visibilmente vuota. Per evitare questo effetto è stato adottato un meccanismo di sovrapposizione temporanea a tre fasi, illustrato nella Figura 5.18.

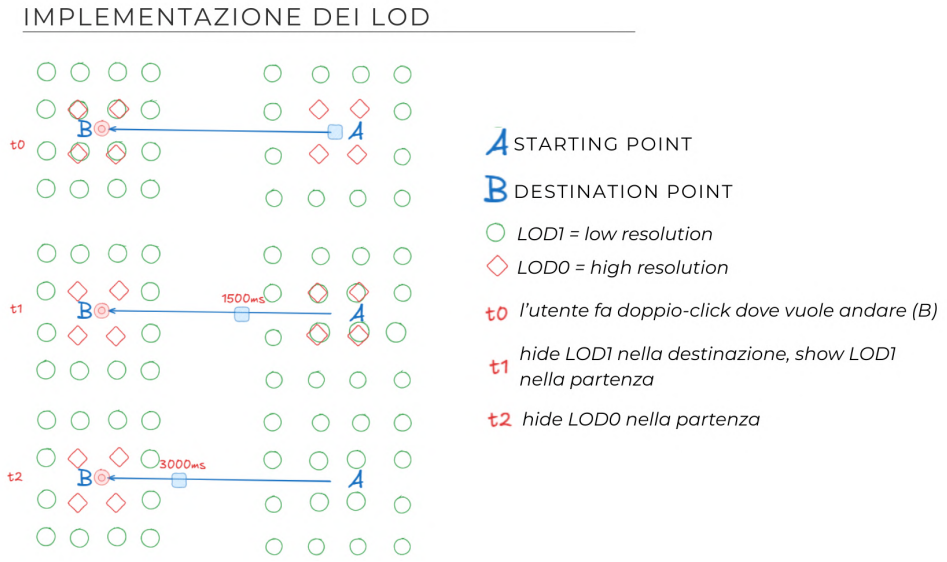

Figura 5.18: Schema del meccanismo di transizione LOD tra la posizione di partenza A e la destinazione B.

Nella prima fase (t_0), quando l'utente avvia lo spostamento verso B, gli splat ad alta densità intorno a B vengono caricati in anticipo, mantenendo comunque attivi quelli a bassa risoluzione. In questo modo non c'è nessun momento di vuoto: se l'high non è ancora pronto, il low è ancora visibile. Dopo circa 750 ms (t_1), i low in B vengono nascosti perché gli high corrispondenti sono disponibili, mentre in A vengono lasciati entrambi i livelli sovrapposti. Dopo altri 750 ms (t_2), gli high in A vengono disattivati: la camera si è ormai allontanata e la differenza non è più percepibile. La funzione `setupLodBehavior()` implementa questa sequenza con due timeout racchiusi in un controller interrompibile.

Le soglie di attivazione sono state calibrate empiricamente su hardware diverso. Un browser non ha accesso diretto alla GPU, e le differenze tra dispositivi sono significative: su un portatile senza scheda dedicata, lo stesso numero di splat che gira senza problemi su una macchina più potente può rallentare il rendering in modo evidente. Il compromesso adottato punta alla stabilità del framerate su hardware medio, rinunciando alla massima qualità visiva in tutte le condizioni.

5.5.2 Integrazione con la navigazione guidata

Durante il percorrimento automatico del path, il flare di pathfinding invoca `setupLodBehavior()` ad ogni aggiornamento di frame. Senza questa integrazione continua, i LOD potrebbero rimanere bloccati sulla configurazione attiva al momento dell'avvio del percorso, caricando asset ad alta qualità anche a grande distanza oppure presentando elementi a bassa risoluzione nelle zone di maggiore interesse visivo. Il fatto che `setupLodBehavior()` venga richiamata continuamente durante la navigazione fa sì che il livello di dettaglio si aggiorni in sincronia con il movimento della camera. Man mano che il viewer avanza lungo il percorso e si avvicina a un elemento, lo swap avviene nel momento giusto, prima che la rappresentazione semplificata diventi visivamente inadeguata.

Nella modalità multi-utente, ogni client viewer esegue autonomamente `setupLodBehavior()` durante il percorrimento del path sincronizzato, indipendentemente dallo stato degli altri client connessi alla sessione. La sincronizzazione gestita da VRoadcast riguarda la sequenza dei nodi del percorso, non la gestione interna delle risorse locali: ogni dispositivo calcola il proprio LOD in

base alla propria posizione e alle proprie capacità hardware. Questo è coerente con il principio di separazione tra logica di rete e logica di rendering su cui si basa l'architettura collaborativa del sistema.

5.5.3 Gestione del LOD nell'Editor Mode

All'attivazione dell'Editor Mode, `setupLodBehavior()` viene sostituita da una riduzione forzata e immediata di tutti i livelli di dettaglio, indipendentemente dalla posizione della camera. Gli splat vengono impostati in modalità a bassa densità, le texture vengono caricate in versione ridotta e le ombre vengono semplificate. La motivazione è di natura prestazionale: in Editor Mode la camera si trova in una posizione sopraelevata con visione ortografica sull'intera area, condizione in cui gli splat ad alta densità non apportano alcun contributo visivo percepibile ma continuano a pesare sulla pipeline di rendering. Il ridimensionamento forzato degli asset libera risorse per garantire la stabilità del sistema durante la fase di pianificazione del percorso.

Prima di applicare la riduzione, il sistema salva la configurazione LOD attiva in quel momento: la densità degli splat, le risoluzioni texture e lo stato delle ombre. Quando il presenter esce dall'Editor Mode, questa configurazione viene ripristinata in ordine inverso: prima gli splat, poi le texture, poi le ombre. L'ordine non è casuale: ripristinare tutto in parallelo rischierebbe di lasciare alcune categorie già ad alta qualità mentre altre sono ancora in versione ridotta, producendo un effetto visivo vistoso durante la transizione.

Capitolo 6

DT-Certosa: Scenari d'Uso

I capitoli precedenti hanno descritto in dettaglio le singole componenti del sistema: la modellazione tridimensionale del Monumento Bottoni, le scelte architetturali del framework e del backend, la logica di navigazione e la gestione delle sessioni collaborative. Questo capitolo presenta invece il sistema nella sua interezza, percorrendo due scenari d'uso rappresentativi che ne mettono in luce il funzionamento concreto.

Il primo scenario descrive la visita di un utente singolo non autenticato, che esplora liberamente l'ambiente esterno e utilizza il sistema di navigazione guidata per raggiungere una sepoltura specifica. Il secondo scenario illustra una sessione collaborativa multi-utente, in cui un presenter autenticato guida uno o più visitatori lungo un percorso predefinito all'interno del monumento.

I due scenari coprono le funzionalità principali del sistema e ne mostrano l'integrazione tra componenti eterogenee: mesh derivate da fotogrammetria, mesh modellate manualmente e vegetazione resa con Gaussian Splatting (GS) convergono nella stessa scena e si collegano ai servizi applicativi e backend. In questo quadro emergono anche la qualità visiva complessiva, la coerenza dei dati biografici forniti da PocketBase, la risposta dell'algoritmo A* ai comandi di navigazione e il meccanismo di sincronizzazione delle sessioni.

6.1 Navigazione libera: lo scenario utente singolo

Lo scenario descrive un utente che accede alla web application tramite browser senza credenziali di accesso. L'interazione avviene interamente lato client, sfruttando le funzionalità di navigazione libera e di ricerca delle sepolture messe a disposizione dal layer applicativo costruito sopra al framework ATON.

6.1.1 Punto di ingresso: la home page

Al caricamento della pagina, l'utente si trova in una posizione predefinita all'esterno del Monumento Bottoni, mostrato nella Figura 6.1. Il punto di vista iniziale è stato scelto per offrire una visuale d'insieme dell'area frontale: si percepiscono le file di tombe, la facciata del monumento e la vegetazione circostante, ricostruita tramite Gaussian Splatting (GS) come descritto nel Capitolo 4.

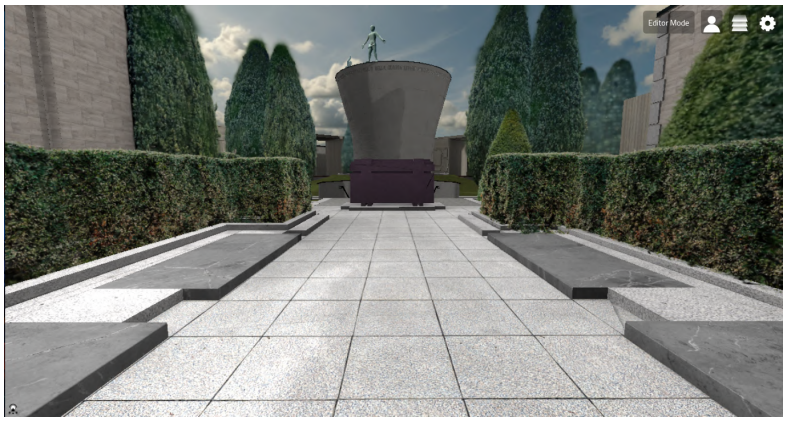

Figura 6.1: Punto di ingresso della web application: vista iniziale sull'area esterna del Monumento Bottoni al caricamento della pagina.

6.1.2 Esplorazione libera degli esterni

L'utente può muoversi nell'ambiente tramite doppio click su qualsiasi punto della scena: il sistema lo muove e la camera si sposta verso di essa seguendo una transizione fluida. Questo meccanismo consente di esplorare l'intera area esterna del monumento, incluse le tombe frontali, le statue ai lati della struttura e gli elementi vegetali distribuiti nell'area.

Figura 6.2: Vista ravvicinata del monumento e degli alberi intorno.

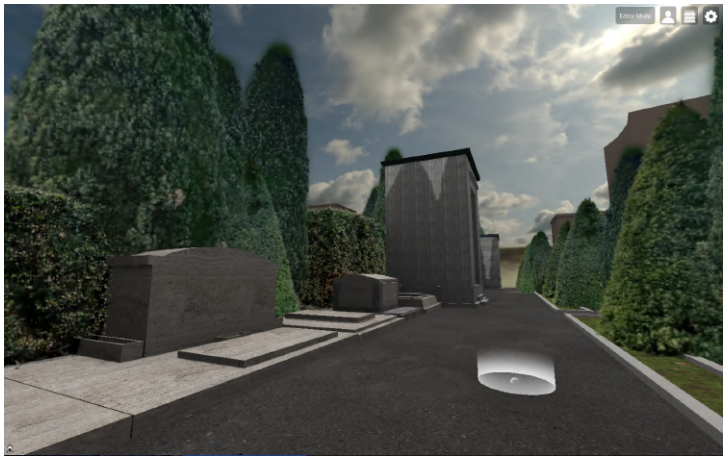

Figura 6.3: Vegetazione ricostruita tramite GS: cipresso e arbusti nella zona laterale dell'area.

La gestione dinamica del Level of Detail (LOD) agisce in modo trasparente durante la navigazione: il passaggio tra le versioni del modello avviene in funzione della distanza della camera dall'oggetto, senza interruzioni visibili. Le piante modellate tramite GS vengono istanziate in corrispondenza delle posizioni degli empty esportati da Blender, come descritto nel Capitolo 5.

6.1.3 Selezione di una sepoltura dal menu

Tramite il menu laterale dell'interfaccia, visibile nella Figura 6.4, l'utente accede all'elenco delle persone sepolte nel Monumento Bottoni. I dati vengono

caricati dinamicamente all'avvio dell'applicazione dalla collezione `persone` di PocketBase, ricostruendo le strutture attese dal codice applicativo.

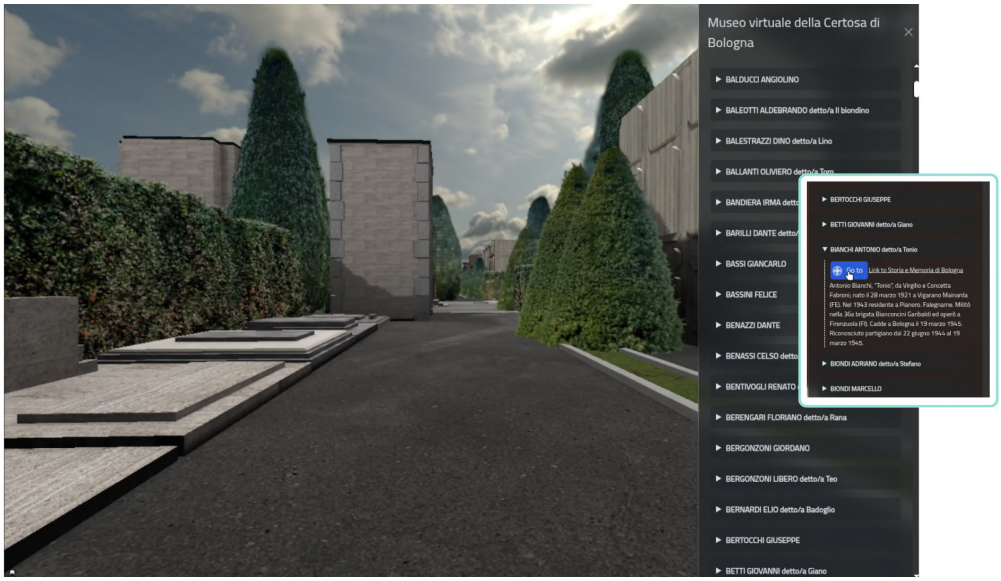

Figura 6.4: Menu laterale con l'elenco delle persone sepolte, caricato dinamicamente da PocketBase. Ogni voce riporta il nome e, dove disponibile, il nome di battaglia del partigiano.

L'utente può scorrere l'elenco e selezionare una persona specifica. La selezione avvia la sequenza di navigazione lungo il percorso più breve calcolato.

6.1.4 Navigazione lungo un percorso verso la destinazione

Alla selezione di un nominativo, il sistema calcola il percorso minimo dalla posizione corrente della camera al punto di vista associato alla tomba corrispondente. Il calcolo utilizza l'algoritmo A* applicato al grafo di navigazione definito manualmente sull'area del monumento, come descritto nel Capitolo 5.

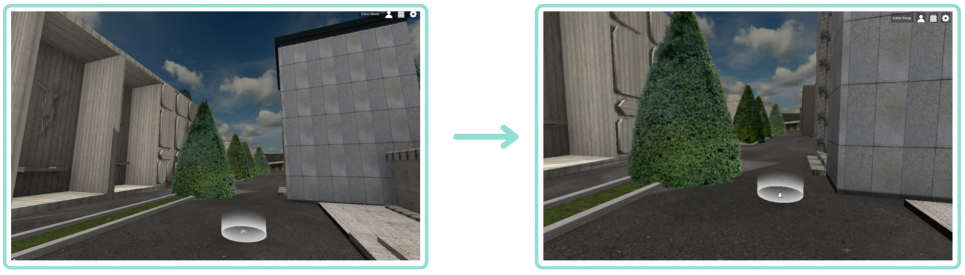

Figura 6.5: Inizio del percorso guidato.

Come si può vedere nelle Figure 6.5 e 6.6, la camera percorre sequenzialmente i nodi del percorso calcolato, con una transizione animata tra un punto di vista e il successivo. L'utente attraversa il percorso in modo passivo, ma può interrompere la navigazione in qualsiasi momento.

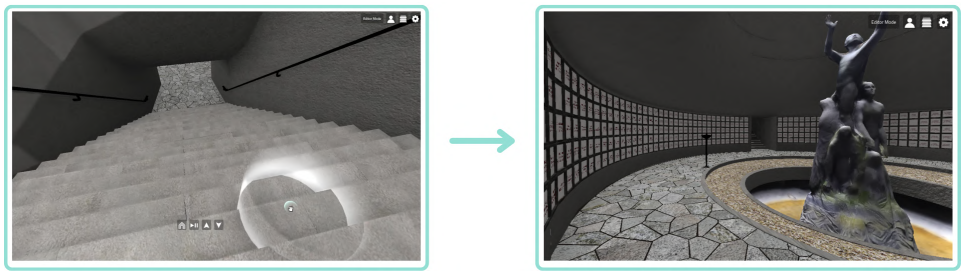

Figura 6.6: Parte del percorso in cui la camera entra nell'area del monumento.

Al termine del percorso, la camera si posiziona nel punto di vista associato alla tomba, con una vista ravvicinata sull'elemento selezionato. Contestualmente vengono visualizzati i dati biografici del partigiano corrispondente, letti direttamente da PocketBase.

6.1.5 Esplorazione interna del monumento

Una volta raggiunta la tomba, l'utente può uscire dalla vista ravvicinata e riprendere la navigazione libera all'interno del monumento, come mostrato nella Figura 6.7.

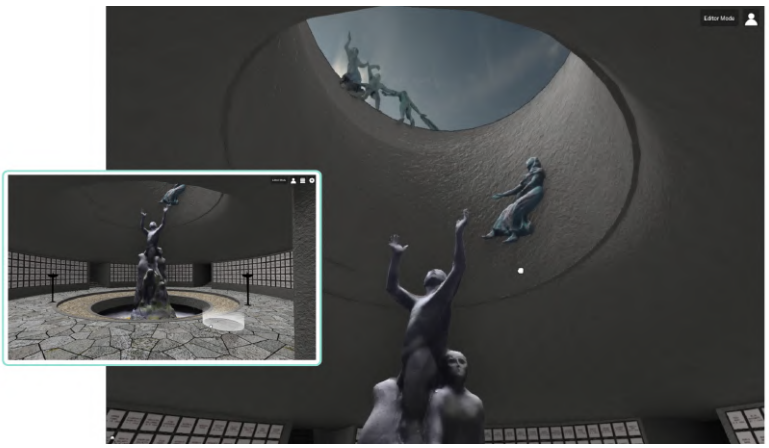

Figura 6.7: Interno del monumento con focus sulle statue.

6.2 Sessione collaborativa: lo scenario multi-utente

Il secondo scenario illustra le funzionalità multi-utente del sistema.
Sono previsti due ruoli: il presenter, che si autentica e gestisce la sessione, e il viewer, che accede tramite il codice o il QR code generato dal primo e segue la navigazione in modo sincronizzato. Entrambi i ruoli operano sulla stessa istanza dell'applicazione; la distinzione è determinata unicamente dallo stato di autenticazione e dal ruolo assegnato alla sessione attiva.

6.2.1 Autenticazione e creazione della sessione

Il presenter accede alla web application con le proprie credenziali tramite il pannello di login. Una volta autenticato, può creare una nuova sessione collaborativa. Il sistema, come mostra la Figura 6.8, genera automaticamente un codice sessione univoco e il corrispondente QR code che il presenter può condividere con i partecipanti per consentire l'accesso.

Figura 6.8: Interfaccia di creazione della sessione con QR code generato per la condivisione.

Il presenter può quindi predisporre un percorso guidato, selezionando i punti di vista da visitare in sequenza. La struttura del percorso riutilizza lo stesso grafo di navigazione impiegato dall'A* per la navigazione automatica.

6.2.2 Navigazione guidata sincronizzata

Il presenter avvia il percorso tramite il comando *Goto path*. A ogni transizione, la posizione della camera del presenter viene trasmessa ai viewer connessi tramite il meccanismo di sincronizzazione basato sugli eventi ATON.

Figura 6.9: Path creato dal presenter e inviato ai viewer connessi tramite il pulsante.

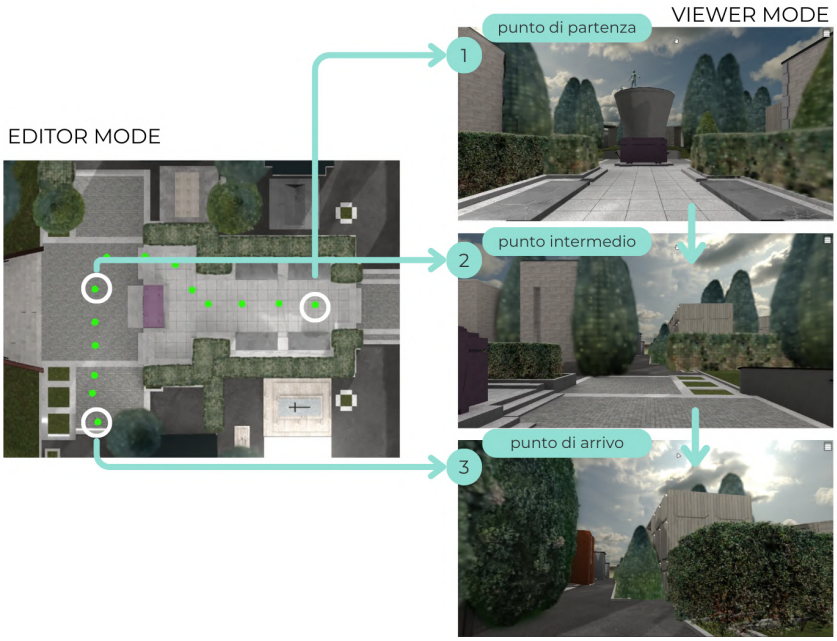

Figura 6.10: I viewer seguono il path creato dal presenter e sincronizzato in tempo reale.

I viewer vengono riposizionati automaticamente nel punto di vista corrente, senza possibilità di deviare dal percorso mentre la sessione è attiva. Come è rappresentato nella Figura 6.10 la parte del Viewer Mode segue tutti i punti selezionati dal Presenter. Sono messi in evidenza tre punti del percorso: il punto di partenza, il punto intermedio e il punto di arrivo del path.
La latenza di sincronizzazione dipende dalla connessione di rete, ma il meccanismo a eventi garantisce che ogni comando emesso dal presenter venga ricevuto e applicato pressochè istantaneamente.

6.2.3 Reindirizzamento alla home virtuale

Al termine del percorso guidato, il presenter può inviare un comando di reindirizzamento diretto ai viewer tramite la funzione *requestPov*. A differenza della navigazione passo-passo del percorso, questo comando trasferisce istantaneamente il punto di vista del viewer a una posizione fissa predefinita, senza percorrere i nodi intermedi. Nell'esempio illustrato in 6.11, il presenter preme il pulsante HOME relativo all'interno del monumento. I viewer vengono riposizionati direttamente al punto di ingresso interno, indipendentemente dalla loro posizione corrente.

Figura 6.11: Reindirizzamento tramite *requestPov*: il presenter invia il comando HOME e il viewer viene trasferito direttamente all'interno del monumento.

Questo meccanismo è utile in tutti i casi in cui il presenter intende riposizionare rapidamente i viewer senza percorrere un cammino intermedio, ad esempio per ricominciare una visita, isolare un punto di interesse specifico o gestire situazioni in cui i viewer si sono disconnessi e riconnessi in posizioni arbitrarie. La semplicità dell'operazione dal lato presenter, un singolo pulsante nell'interfaccia, contrasta con la complessità del coordinamento che avviene

a livello di backend, dove la sessione viene aggiornata e l'evento propagato a tutti i viewer registrati.

Conclusioni e Sviluppi Futuri

La Certosa di Bologna custodisce varie delle memorie storiche più significative della città: tra queste,

il Monumento Ossario ai Partigiani di Bottoni raccoglie le sepolture di oltre 500 persone che hanno perso la vita durante la Resistenza. Per anni quell'ambiente è stato accessibile anche attraverso una rappresentazione digitale, sviluppata da Cineca nei primi anni 2000, che permetteva di navigare gli spazi virtuali del monumento e consultare le schede biografiche delle persone sepolte. Quel lavoro di digitalizzazione pionieristico era stato realizzato nel forte rispetto dei limiti tecnici dei primissimi motori di rendering web: poiché il software del periodo non consentiva altrimenti, l'applicazione faceva affidamento su modelli a bassa conta poligonale e forti ottimizzazioni visive che garantivano la fluidità. Con il passare del tempo e dei progressivi restauri in loco, quella versione ha inevitabilmente perso aderenza con l'attuale conformazione e alla fine è stata dismessa, disperdendo temporaneamente il suo importante portato storico e documentale.

DT-Certosa nasce con l'obiettivo di restituire questa accessibilità, aggiornando gli strumenti e ampliando le funzionalità rispetto a quanto era stato realizzato in precedenza. Non si tratta di una semplice ricostruzione grafica: il sistema è progettato per essere uno strumento di fruizione reale, accessibile via browser senza alcuna installazione, da qualsiasi dispositivo. Un utente qualunque, senza competenze tecniche specifiche, può raggiungere una sepoltura precisa, percorrere il monumento guidato dall'algoritmo di navigazione e leggere i dati biografici del partigiano sepolto. La componente storica non è accessoria ma è al centro del progetto, distribuita su un backend interrogabile e collegata direttamente agli elementi della scena tridimensionale.

Il lavoro ha affrontato tre aree di intervento distinte ma strettamente con-

nesse.

La prima ha riguardato la costruzione di un ambiente tridimensionale fedele e compatibile con l'esecuzione in-browser: la struttura del Monumento Bottoni è stata rimodellata in Blender a partire da rilievi fotografici condotti in più sopralluoghi, con gestione differenziata delle iscrizioni, delle superfici marmoree e degli elementi architettonici di dettaglio. Le statue sono state ricostruite tramite fotogrammetria con Meshroom, mentre la vegetazione è stata generata attraverso una pipeline basata su Gaussian Splatting (GS), con livelli di dettaglio gestiti dinamicamente in funzione della posizione della camera.

La seconda ha riguardato l'integrazione del modello in un'infrastruttura applicativa coerente. Il framework ATON è stato esteso per supportare la gestione dei dati biografici tramite PocketBase, sostituendo strutture dati statiche con un sistema di persistenza interrogabile via API REST. La migrazione ha richiesto la progettazione delle collezioni, l'importazione automatizzata dei dati tramite uno script Node.js e la ricostruzione dinamica, all'avvio, delle strutture attese dalla logica applicativa preesistente.

La terza ha riguardato la navigazione e la collaborazione. L'implementazione dell'algoritmo A* applicato a un grafo di navigazione definito manualmente consente di calcolare percorsi fisicamente plausibili verso qualsiasi sepoltura selezionata dall'utente. La stessa infrastruttura è stata estesa per supportare sessioni multi-utente con distinzione dei ruoli, sincronizzazione dei punti di vista e meccanismi di reindirizzamento diretto tramite *requestPov*.

I contributi tecnici principali del lavoro si articolano in quattro componenti.

La pipeline GS per la vegetazione rappresenta un approccio non convenzionale alla gestione di elementi organici in ambiente web, con istanziazione on demand degli splat ad alta densità e Level of Detail (LOD) gestito dalla posizione della camera. L'integrazione di PocketBase in ATON ha richiesto una strategia di compatibilità precisa per preservare la logica applicativa preesistente senza riscriverla, richiedendo al backend la sola gestione dei dati e mantenendo la ricostruzione delle strutture lato client.

Per la navigazione guidata è stato implementato un componente generalizzabile, non legato al solo caso del Monumento Bottoni. La scelta nasce da un obiettivo architetturale preciso: riusare la stessa logica in progetti ATON diversi mantenendo separati contenuto della scena e comportamento applicativo. L'implementazione A* su grafo manuale, con percorrenza sequenziale

animata dei nodi, introduce infatti in ATON una funzionalità assente nella configurazione nativa. Questa logica è stata deliberatamente separata dalla webapp specifica: il comportamento di percorrimento è incapsulato in un flare dedicato, distribuito in un repository autonomo e riutilizzabile in qualsiasi webapp basata su ATON, indipendentemente dal contenuto specifico della scena. Questo dettaglio architetturale ha un peso che supera la singola funzionalità: il flare non appartiene alla Certosa, appartiene al framework. Chiunque voglia abilitare la navigazione guidata basata su grafo in un progetto ATON può caricare lo stesso componente, configurare i propri punti di vista e ottenere il comportamento descritto in questo elaborato. Il progetto non è costruito come soluzione ad hoc, ma come sistema ingegnerizzato, con una separazione esplicita tra la webapp specifica, il modulo di pathfinding e il backend di dati. Questa struttura garantisce che le singole parti possano evolversi, essere riutilizzate o sostituite senza impatto sulle altre.

Il sistema di sessioni collaborative ha abilitato la modalità *Monumento Parlante*, in cui un presenter autenticato può guidare uno o più visitatori remoti attraverso un percorso sincronizzato. Questa funzionalità non è accessoria: abilita scenari di fruizione che vanno oltre la visita individuale. Un mediatore culturale può condurre una visita guidata a distanza, seguita in simultanea da partecipanti distribuiti geograficamente. Un docente può usare lo stesso ambiente per una lezione contestualizzata, con la possibilità di spostarsi da un punto di interesse all'altro mantenendo il gruppo sincronizzato. Ricercatori possono usare la sessione come ambiente condiviso per analizzare e annotare elementi della replica digitale. La qualità della fruizione dipende dall'intento di chi guida e dal contenuto caricato nel sistema: la piattaforma fornisce l'infrastruttura, i casi d'uso dipendono da chi la usa.

Il sistema presenta limiti che derivano dalle scelte progettuali adottate o dai vincoli del contesto di sviluppo. Il grafo di navigazione è costruito manualmente: l'aggiunta di nuove aree richiede un intervento esplicito sulla struttura del grafo, senza meccanismi di generazione automatica a partire dalla geometria della scena. La pipeline GS per la vegetazione produce asset specifici per la configurazione attuale; una variazione significativa nella disposizione degli elementi richiederebbe una nuova fase di ricostruzione e ottimizzazione. PocketBase è dimensionato per dataset di media scala e carichi di accesso moderati; scenari con un numero molto elevato di utenti simultanei potrebbero richiede-

re una revisione dell'architettura di backend. Le prestazioni della componente GS mostrano una dipendenza dalla GPU disponibile lato client, con possibili degradazioni su dispositivi con configurazioni hardware limitate.

Le direzioni di sviluppo più immediate riguardano l'estensione dell'area coperta: la Certosa di Bologna comprende un insieme molto più ampio di monumenti e strutture, e l'architettura del sistema è progettata per supportare l'aggiunta di nuovi modelli senza modifiche strutturali sostanziali. La stessa combinazione di strumenti utilizzata per il Monumento Bottoni, modellazione manuale in Blender, fotogrammetria per le statue, GS per la vegetazione, PocketBase per i dati, costituisce una pipeline replicabile su qualsiasi altro monumento della Certosa o di contesti analoghi.

La generazione automatica del grafo di navigazione a partire dalla geometria della scena ridurrebbe significativamente l'intervento manuale necessario all'espansione delle aree navigabili. L'ottimizzazione della pipeline GS per dispositivi mobili aprirebbe l'accesso alla maggior parte degli utenti finali, superando il vincolo attuale sulla potenza hardware.

Sul fronte dei contenuti, l'integrazione di materiale multimediale aggiuntivo, registrazioni audio, documenti d'archivio, fotografie storiche, collegato alle singole sepolture tramite PocketBase, arricchirebbe la dimensione informativa senza richiedere modifiche alla geometria o all'architettura applicativa. La struttura del database è già predisposta per ospitare campi di questo tipo.

Al di là degli sviluppi specifici di DT-Certosa, il lavoro suggerisce un orizzonte più ampio. La combinazione di GS per ambienti organici complessi, flare ATON per funzionalità modulari e backend leggeri come PocketBase per la gestione dei contenuti rappresentano un pattern applicabile a qualsiasi progetto di valorizzazione digitale del patrimonio culturale che richieda navigabilità, contestualizzazione storica e fruizione condivisa. Il proof of concept realizzato mostra che questo tipo di sistema è oggi accessibile via browser a chiunque disponga di una connessione. La distanza tra un monumento fisico e la sua replica navigabile, arricchita e condivisibile, ora è molto ridotta.

Appendice A

Appendice

A.1 Script Blender per la generazione delle iscrizioni

I listati seguenti riportano il codice Python sviluppato all'interno del Text Editor di Blender per la generazione automatica e la conversione delle iscrizioni sulle lapidi del Monumento Bottoni.

```
def build_text(entries, tomb_num):
    lines = []
    for e in entries:
        name = e.get("nome", "").upper()
        data = e.get("data_morte", "").strip()
        name_parts = name.split(" ", 1)
        first, last = (name_parts + [""])[:2]
        lines.append(first)
        if last:
            lines.append(last)
        if tomb_num not in NO_DATE_SET and data:
            lines.append(format_date(data))
    return "\n".join(lines)

def place_text(text_obj, tombino_obj):
    text_obj.scale = Vector((1, 1, 1))
    text_obj.data.align_x = 'CENTER'
    text_obj.data.align_y = 'CENTER'
    # posiziona il testo al centro della superficie del tombino
    text_obj.location = tombino_obj.matrix_world.to_translation()
    normal_world = tombino_obj.matrix_world.to_3x3() @ Vector((0, 0, 1))
    text_obj.location += normal_world.normalized() * OFFSET_FORWARD
    # rotazione: x fisso a 90 gradi, z ereditato dal tombino padre
    eul_parent = tombino_obj.matrix_world.to_euler('XYZ')
    text_obj.rotation_euler = Euler(
        (math.radians(90), 0, eul_parent.z), 'XYZ'
    )
    bpy.context.view_layer.update()
    dims = text_obj.dimensions.copy()
    if dims.x > 0 and dims.y > 0:
```

```
        scale_x = min(1.0, MAX_BOX.x / dims.x)
        scale_y = min(1.0, MAX_BOX.y / dims.y)
        text_obj.scale = Vector((min(scale_x, scale_y),) * 3)
```

Listing A.1: Estratto dello script di generazione dei testi: costruzione del testo e posizionamento sul tombino.

```
for txt_obj in txt_objects:
    match = re.search(r"(\d+(\.\d+)?)$", txt_obj.name)
    numero = match.group(1) if match else "x"

    bpy.ops.object.select_all(action='DESELECT')
    bpy.context.view_layer.objects.active = txt_obj
    txt_obj.select_set(True)

    try:
        bpy.ops.object.duplicates_make_real()
    except RuntimeError:
        pass

    bpy.ops.object.make_single_user(
        type='SELECTED_OBJECTS', object=True, obdata=True
    )
    bpy.ops.object.convert(target='MESH')
    bpy.ops.object.mode_set(mode='OBJECT')

    objs_to_join = bpy.context.selected_objects
    if len(objs_to_join) > 1:
        bpy.context.view_layer.objects.active = objs_to_join[0]
        for o in objs_to_join:
            o.select_set(True)
        bpy.ops.object.join()

    joined_obj = bpy.context.view_layer.objects.active
    joined_obj.name = f"font_tombino_{numero}"
    bpy.data.objects.remove(txt_obj, do_unlink=True)
```

Listing A.2: Script di conversione dei Text Object in mesh e join per tombino.

A.2 Script JavaScript per l'integrazione con PocketBase

I listati seguenti riportano il codice JavaScript sviluppato per l'importazione dei dati biografici in PocketBase e per la ricostruzione dinamica delle strutture dati all'avvio dell'applicazione ATON.

```
const tombeCache = {}
for (const key in db2) {
    const persona = db2[key]
    const tombino = persona.tombino
    if (!tombeCache[tombino]) {
        try {
```

```
            const t = await pb.collection('tombe')
                .getFirstListItem(`numero=${tombino}`)
            tombeCache[tombino] = t.id
        } catch {
            const t = await pb.collection('tombe')
                .create({ numero: tombino })
            tombeCache[tombino] = t.id
        }
    }
    await pb.collection('persone').create({
        nome:           String(persona.nome || ""),
        nome_battaglia: String(persona.nome_battaglia || ""),
        link:           (persona.link && persona.link.startsWith("http"))
                            ? persona.link : "",
        biografia:      String(persona.biografia || ""),
        data_nascita:   String(persona.data_nascita || ""),
        data_morte:     String(persona.data_morte || ""),
        tombino:        tombeCache[tombino]
    })
}
```

Listing A.3: Script Node.js (`import.js`) per il caricamento dei dati da `db2` nelle collezioni PocketBase.

```
const persone = await pb.collection("persone")
    .getFullList({ expand: "tombino", requestKey: null });

// ricostruzione di db = { numero_tomba: [nomi] }
db = {};
for (const t of tombe) db[t.numero] = [];
for (const p of persone) {
    if (p.expand && p.expand.tombino) {
        const num = p.expand.tombino.numero;
        if (!db[num]) db[num] = [];
        db[num].push(p.nome);
    }
}

// ricostruzione di db2 = { nome: { dettagli } }
db2 = {};
for (const p of persone) {
    db2[p.nome] = {
        nome:           p.nome,
        nome_battaglia: p.nome_battaglia,
        link:           p.link,
        biografia:      p.biografia,
        data_nascita:   p.data_nascita,
        data_morte:     p.data_morte,
        tombino: p.expand && p.expand.tombino
                    ? p.expand.tombino.numero : null
    };
```

```
}
```

Listing A.4: Funzione `loadData()`: recupero da PocketBase e ricostruzione delle strutture `db` e `db2` attese dal codice preesistente.

A.3 Flare di pathfinding

Il listato seguente riporta la struttura del flare `pathfindingflare.js`, il modulo JavaScript che implementa la logica di percorrimento guidato all'interno della webapp ATON.

```
{
    let F = new ATON.Flare("pathfindingflare");

    F._stop   = false;
    F._pause  = false;
    F._resume = false;

    let _pathTravel = null;

    F.pathfinding = {
        stopPathTravel:  function() { ... },
        isPathTraveling: function() { return _pathTravel !== null; },

        travelPath: function(povs, options = {}, onTap, onComplete) {
            // opzioni: targetSpeed, easing, lookAheadPoints,
            //          lookSmoothness, minSpeed, maxSpeed,
            //          accelerationTime
            const curve = new THREE.CatmullRomCurve3(
                povs.map(p => p.pos.clone())
            );
            curve.curveType = 'catmullrom';
            curve.tension   = 0.4;
            // ... animazione frame-by-frame con velocita' variabile,
            //     look-ahead adattivo e gestione discesa
        },

        calculateLookAheadTarget: function(curve, t, n, pts) { ... },
        calculateAverageDistanceBetweenPoints: function(pts)  { ... }
    };

    F.setup = () => {
        ATON.Nav.travelPath       = F.pathfinding.travelPath;
        ATON.Nav.stopPathTravel   = F.pathfinding.stopPathTravel;
        ATON.Nav.pausePathTravel  = () => { F._pause  = true; };
        ATON.Nav.resumePathTravel = () => { F._resume = true; };
    };
}
```

Listing A.5: Struttura del flare `pathfindingflare`: inizializzazione e firma della funzione `travelPath`.

A.4 Funzione spawnSplatsForTrees

Il listato seguente riporta il nucleo della funzione `spawnSplatsForTrees()`, responsabile del caricamento e del posizionamento degli splat di vegetazione a partire dal file GLB degli empty. La funzione è definita in `systems.js` e viene invocata una sola volta dopo l'evento `AllNodeRequestsCompleted`.

```
spawnSplatsForTrees = async () => {
    if (APP._splatsReady) return { high: APP.HIGHSPLATS, low: APP.LOWSPLATS };
    if (APP._splatsLoading) return;
    APP._splatsLoading = true;

    // Attende che il nodo empty_trees sia pronto con i suoi children
    await waitFor(() => {
        const n = ATON.getSceneNode("empty_trees");
        return n && n.children && n.children[0] &&
               n.children[0].children && n.children[0].children.length > 0;
    }, 150, 8000);

    const root    = ATON.getSceneNode("empty_trees").children[0];
    const empties = root.children || [];

    const lowAssetMap = {
        trees:   APP.assetsPath + "mesh_low/3_cipresso_high_D_radius12cm.splat",
        cypress: APP.assetsPath + "mesh_low/cipresso_low_compressed_4k.splat",
        bush:    APP.assetsPath + "mesh_low/siepe_low_3k.splat"
    };

    for (let i = 0; i < empties.length; i++) {
        const empty = empties[i];
        if (!empty) { APP.HIGHSPLATS[i] = APP.LOWSPLATS[i] = null; continue; }

        // Classificazione per prefisso del nome
        let emptyType = 'trees';
        if (empty.name) {
            const name = empty.name.toLowerCase();
            if      (name.startsWith('c')) emptyType = 'cypress';
            else if (name.startsWith('b')) emptyType = 'bush';
            else if (name.startsWith('t')) emptyType = 'trees';
        }
        APP._emptyTypes[i] = emptyType;

        // Estrazione trasformazioni in coordinate world
        const pos   = new THREE.Vector3();
        const scale = new THREE.Vector3(1, 1, 1);
        const rot   = new THREE.Euler(0, 0, 0);
        const quat  = new THREE.Quaternion();

```

```
42          if (typeof empty.getWorldPosition === "function")
43              empty.getWorldPosition(pos);
44          if (typeof empty.getWorldScale === "function")
45              empty.getWorldScale(scale);
46          if (typeof empty.getWorldQuaternion === "function") {
47              empty.getWorldQuaternion(quat);
48              rot.setFromQuaternion(quat);
49          }
50
51          APP._highPlaceholders[i]         = pos.clone();
52          APP._highPlaceholdersScale[i]    = scale.clone();
53          APP._highPlaceholdersRotation[i] = rot.clone();
54
55          // Caricamento immediato dello splat a bassa densita'
56          const lowName = `splat_${emptyType}_low_${i}`;
57          const lowNode = ATON.createSceneNode(lowName);
58          lowNode.load(lowAssetMap[emptyType]);
59          lowNode.visible = false;
60          lowNode.attachToRoot();
61          lowNode.position.copy(pos);
62          lowNode.rotation.copy(rot);
63          lowNode.scale.copy(scale);
64          lowNode.visible = true;
65
66          APP.LOWSPLATS[i]  = lowNode;
67          // Lo splat ad alta densita' viene caricato on demand da
    ensureHighForIndex()
68          APP.HIGHSPLATS[i] = null;
69      }
70
71      APP._splatsReady   = true;
72      APP._splatsLoading = false;
73      return { high: APP.HIGHSPLATS, low: APP.LOWSPLATS };
74  };
```

Listing A.6: Nucleo di `spawnSplatsForTrees()`: classificazione degli empty, estrazione delle trasformazioni world e caricamento degli splat a bassa densità con posizionamento immediato.

Bibliografia

[1] AliceVision Consortium. *AliceVision – Photogrammetric Computer Vision Framework*. URL: https://alicevision.org/.

[2] AliceVision Consortium. *Meshroom – Free and Open-Source 3D Reconstruction Software*. URL: https://alicevision.org/#meshroom.

[3] Babylon.js Contributors. *Babylon.js – Powerful, Beautiful, Simple, Open – Web-Based 3D at Its Best*. URL: https://www.babylonjs.com/.

[4] Blender Foundation. *Blender – The Free and Open Source 3D Creation Software*. URL: https://www.blender.org/.

[5] BlenderKit s.r.o. *BlenderKit – 3D Assets*. URL: https://www.blenderkit.com/.

[6] Claudio Borgatti et al. «Databases and Virtual Environments: a Good Match for Communicating Complex Cultural Sites». In: *ACM SIGGRAPH 2004 Educators Program*. ACM, 2004. DOI: 10.1145/1186107.1186143.

[7] Arthur Brussee. *Brush – 3D Reconstruction for All*. URL: https://github.com/ArthurBrussee/brush.

[8] L. Calori et al. «Certosa virtual museum: a dynamic multilevel desktop VR application». In: *EUROGRAPHICS 2003 – Interactive Demos and Posters*. Eurographics Association, 2003. URL: https://www.researchgate.net/publication/230744908.

[9] CNR ISPC. *ATON Client API Documentation*. URL: https://aton.ispc.cnr.it/apidoc/client/.

[10] CNR ISPC. *ATON Framework*. URL: https://osiris.itabc.cnr.it/aton/.

[11] Comune di Bologna. *Ortofoto del portale Open Data*. URL: https://opendata.comune.bologna.it/map/+85917b4b836540de/edit/.

[12] Bruno Fanini. *ATON Framework – GitHub Repository*. URL: https://github.com/phoenixbf/aton.

[13] Ben Fei et al. «3D Gaussian Splatting as a New Era: A Survey». In: *IEEE Transactions on Visualization and Computer Graphics* 31.8 (2025), pp. 4429–4449. DOI: 10.1109/TVCG.2024.3397828.

[14] Per Gantelius. *fSpy – Open source still image camera matching*. URL: https://fspy.io/.

[15] Per Gantelius. *fSpy-Blender – Official fSpy importer add-on for Blender*. URL: https://github.com/stuffmatic/fSpy-Blender.

[16] GIMP Development Team. *GIMP – GNU Image Manipulation Program*. URL: https://www.gimp.org/.

[17] Michael Grieves. *Digital Twin: Manufacturing Excellence through Virtual Factory Replication*. White paper, Florida Institute of Technology. 2014. URL: https://www.researchgate.net/publication/275211047.

[18] Kyle Johnson. *SkySplat – 3D Gaussian Splatting for Blender*. URL: https://skysplat.org/.

[19] Kyle Johnson. *skysplat_blender – GitHub Repository*. URL: https://github.com/kyjohnso/skysplat_blender.

[20] Bernhard Kerbl et al. «3D Gaussian Splatting for Real-Time Radiance Field Rendering». In: *ACM Transactions on Graphics* 42.4 (2023). DOI: 10.1145/3592433.

[21] Bernhard Kerbl et al. *3D Gaussian Splatting for Real-Time Radiance Field Rendering – Project Page*. URL: https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/.

[22] KIRI Innovation. *3DGS Render by KIRI Engine – Blender Addon*. URL: https://www.kiriengine.app/blender-addon/3dgs-render.

[23] LearnOpenCV. *Introduction to 3D Gaussian Splatting*. 2023. URL: https://learnopencv.com/3d-gaussian-splatting/.

[24] Wolfram Luther et al. «Digital Twins and Enabling Technologies in Museums and Cultural Heritage: An Overview». In: *Sensors* 23.3 (2023), p. 1583. DOI: 10.3390/s23031583.

[25] Ben Mildenhall et al. «NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis». In: *Computer Vision – ECCV 2020*. Springer, 2020. DOI: 10.1007/978-3-030-58452-8_24. URL: https://arxiv.org/abs/2003.08934.

[26] PlayCanvas. *SuperSplat – 3D Gaussian Splat Editor*. URL: https://superspl.at/editor.

[27] PocketBase. *PocketBase – GitHub Repository*. URL: https://github.com/pocketbase/pocketbase.

[28] PocketBase. *PocketBase – Open Source Backend in 1 File*. URL: https://pocketbase.io/.

[29] Johannes Lutz Schönberger e Jan-Michael Frahm. *COLMAP – Structure-from-Motion and Multi-View Stereo*. URL: https://colmap.github.io/.

[30] SPARK Contributors. *SPARK – WebGL Gaussian Splatting Renderer*. URL: https://sparkjs.dev/.

[31] Fei Tao et al. «Digital Twin in Industry: State-of-the-Art». In: *IEEE Transactions on Industrial Informatics* 15.4 (2018), pp. 2405–2415. DOI: 10.1109/TII.2018.2873186.

[32] Three.js Authors. *Three.js – JavaScript 3D Library*. URL: https://threejs.org/.

[33] Lee Allen Westover. «Splatting: A Parallel, Feed-Forward Volume Rendering Algorithm». Tesi di dott. University of North Carolina at Chapel Hill, 1991. URL: https://articles.tomasparks.name/publications/Westover1991.pdf.

Ringraziamenti

Giungere alla fine di questa tesi mi fa comprendere che si chiude un capitolo importante della mia vita. Un percorso fatto di impegno, difficoltà e, soprattutto, crescita, che non avrei mai potuto affrontare senza il sostegno umano e professionale di tutte le persone che mi hanno accompagnato lungo il cammino.

In primo luogo, desidero esprimere la mia più profonda gratitudine alla Professoressa Serena Morigi, che attraverso i suoi insegnamenti mi ha permesso di appassionarmi a questa disciplina e mi ha concesso il privilegio di approfondire questo progetto di tesi.

Un ringraziamento speciale, inoltre, va al VisitLab Cineca per avermi accolto in un ambiente così stimolante e collaborativo.
In particolare, desidero ringraziare l'Ing. Antonella Guidazzoli per aver coordinato ogni fase di questo tirocinio con grande competenza e disponibilità.
Desidero inoltre esprimere un sentito ringraziamento a Daniele e Federico per l'attenzione riposta in me, il supporto costante e la pazienza dimostrata durante tutto il percorso di sviluppo del progetto.