Alma Mater Studiorum – Università di Bologna

DIPARTIMENTO DI INFORMATICA - SCIENZA E INGEGNERIA (DISI)
Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale
in
Fondamenti di Computer Graphics M

Bologna Digital Twin: Modellazione Urbana Data-Driven

Candidato:
Alex Gianelli

Relatore:
Chiar.ma Prof.ssa Serena Morigi

Correlatore:
Ing. Antonella Guidazzoli

Anno Accademico 2022-2023
Sessione VI

Introduzione

L'avanzamento incessante delle tecnologie informatiche sta rivoluzionando profondamente il nostro modo di interagire con l'ambiente urbano, apportando innovazioni nella progettazione, realizzazione, gestione e abitazione delle città. In tale scenario, emerge il concetto di Digital Twin, ossia una replica digitale dinamica di un sistema fisico, che si propone come una soluzione efficace per superare le sfide urbane complesse del nostro secolo.

Il lavoro di tesi qui presentato è frutto di uno stage curricolare presso il Cineca di Bologna, un centro di eccellenza europeo nel campo dell'high performance computing, nel quale il focus è stato posto sullo sviluppo di un prototipo di Digital Twin per la città di Bologna. Questo progetto, denominato Bologna Digital Twin, gode del supporto finanziario e della promozione da parte del Comune di Bologna, dell'Università degli Studi di Bologna, di Cineca e della Fondazione Bruno Kessler, e mira a creare un Digital Twin avanzato dell'intera area metropolitana di Bologna. L'obiettivo principale è offrire uno strumento avanzato per il planning urbano, la gestione delle infrastrutture, il coinvolgimento della cittadinanza e l'analisi e previsione delle reazioni della città di fronte a eventi sia naturali che causati dall'uomo.

Questo lavoro di tesi è stato sviluppato in collaborazione con un altro studente, rendendo i due lavori interdipendenti e complementari. Di conseguenza, la struttura è stata organizzata in modo da condividere i capitoli introduttivi, mentre le parti riguardanti l'implementazione sono state sviluppate in modo indipendente e specifico per ciascuno.

Nel contesto del progetto Bologna Digital Twin, abbiamo mirato a creare un prototipo di Digital Twin partendo dall'elaborazione di dati grezzi fino alla realizzazione di un ambiente 3D interattivo, con un'enfasi particolare sull'acquisizione di modelli 3D dettagliati e completi da dati eterogenei, e lo sviluppo di un Digital Twin basato su questi modelli. Un ulteriore obiet-

tivo era rendere il Digital Twin un sistema dinamico, capace di aggiornarsi continuamente con dati in tempo reale.

L'elaborato inizia con un'analisi dello stato dell'arte dei Digital Twin Urbani esistenti, per poi focalizzarsi sull'analisi dei dati iniziali forniti da Cineca, evidenziando sfide e ostacoli da superare. Dopo i capitoli introduttivi, i due lavori di tesi si dividono per trattare separatamente l'elaborazione dei dati iniziali con l'intento di ottenere modelli 3D utilizzabili per il Digital Twin di Bologna. In particolare, il mio lavoro si è concentrato sulla creazioni di modelli con vari livelli di dettaglio per il terreno, la modellazione dei dati LiDAR per la costruzione delle mesh dei tetti della città e il relativo texturing di entrambe le componenti. Nell'ultimo capitolo, l'attenzione si sposta verso lo sviluppo del prototipo di Digital Twin, esplorando la creazione di una scena 3D esplorabile e lo sviluppo di servizi per integrare il Digital Twin con dati esterni.

Per far comprendere al meglio il progetto presentato nella sua interezza, verranno brevemente inseriti anche alcuni dei risultati del mio collega, sorvolando sulle metodologie per le quali si consiglia di consultare il suo elaborato di tesi[3].

Indice

Elenco delle figure

Capitolo 1

Digital Twin

Contents

1.1 Che cos'è un Digital Twin

Il Digital Twin, o Gemello Digitale, costituisce una delle novità più rilevanti e influenti nel campo della trasformazione digitale e dell'industria 4.0[26]. Considerarlo come un semplice duplicato digitale di un elemento fisico sarebbe una semplificazione eccessiva; in realtà, il Digital Twin è un modello dinamico che replica con fedeltà le proprietà, i processi e il comportamento di sistemi fisici appartenenti al mondo reale. Questa tipologia di modello ha guadagnato importanza crescente con l'emergere dell'Internet of Things (IoT)[58] e del Cloud Computing[38], facilitando l'integrazione e l'analisi di enormi quantità di dati generati da sensori IoT collocati sul sistema fisico, fornendone un monitoraggio dello stato attuale, la previsione delle prestazioni future mediante simulazioni e l'ottimizzazione delle decisioni in tempo reale[34].

Il valore distintivo del Digital Twin sta nella sua abilità di collegare il mondo fisico a quello digitale, fornendo un ambiente simulato per testare teo-

rie, simulare scenari, prevedere risultati e orientare decisioni senza influenzare direttamente l'elemento fisico[25]. Questo è particolarmente utile nella gestione del ciclo di vita dei prodotti, dalla concezione alla manutenzione, fino alla dismissione[37], oltre che nell'ottimizzazione di sistemi complessi quali reti di energia e acqua, sistemi di trasporto e infrastrutture cittadine[83].

La nozione di Digital Twin è stata introdotta nei primi anni 2000 da Michael Grieves[26], sebbene le sue origini si estendano a lavori e concetti anteriori riguardanti la modellazione e simulazione di sistemi complessi. Per esempio, la NASA, già negli anni '60 per missioni spaziali, elaborava modelli simulati di navicelle spaziali per esaminare e valutare le prestazioni in condizioni critiche, evitando pericoli per la sicurezza per gli astronauti durante le missioni[2].

Oggigiorno, con l'evoluzione delle tecnologie legate all'IoT, all'analisi dei big data e al cloud computing, la capacità di sviluppare e impiegare i Digital Twins è aumentata in modo esponenziale, rendendo questo strumento cruciale per l'innovazione in vari ambiti. I Digital Twins, evolvendosi da semplici duplicati di elementi fisici, sono diventati sistemi complessi che possono simulare intere catene produttive, infrastrutture urbane e persino intere città, aprendo possibilità ancora inesplorate per una gestione più efficiente delle risorse, la pianificazione urbana e la sostenibilità ambientale.

1.2 Digital Twin urbani

Fra le numerose applicazioni del Digital Twin, quella dei Digital Twin Urbani emerge come particolarmente innovativa e potenzialmente trasformativa. Tali modelli digitali dinamici di città o porzioni urbane si avvalgono di dati in tempo reale e storici, raccolti da una vasta rete di sensori ambientali urbani. Questi dati, che coprono aspetti come il flusso del traffico, il consumo energetico, la qualità dell'atmosfera, le infrastrutture e i servizi pubblici, sono fondamentali per alimentare i Digital Twin Urbani, permettendo a urbanisti, politici e cittadini di disporre di strumenti avanzati per una pianificazione urbana e una gestione della città più efficiente.

La forza dei Digital Twin Urbani (UDT) sta nella loro capacità di modellare con precisione e in modo complesso l'ecosistema urbano, offrendo la possibilità di eseguire analisi approfondite sull'interazione di diversi fattori e il loro impatto sulla vita urbana. Grazie all'impiego di tecnologie all'avan-

guardia, come l'intelligenza artificiale per l'elaborazione e l'analisi dei dati, gli UDT possono anticipare le conseguenze di modifiche infrastrutturali, politiche urbane, eventi straordinari e tendenze di sviluppo a lungo termine.

Dal punto di vista ambientale, gli UDT assumono un ruolo fondamentale nel favorire la sostenibilità e nell'attenuare gli effetti dei cambiamenti climatici. Attraverso simulazioni precise, facilitano l'ottimizzazione delle risorse e la riduzione delle emissioni di CO2, incrementando la capacità delle città di resistere agli effetti dei cambiamenti climatici e orientandole verso un'economia più green.

Un ulteriore vantaggio degli UDT riguarda il rafforzamento della partecipazione dei cittadini e della governance collaborativa. Offrendo piattaforme visive e interattive, gli UDT promuovono un coinvolgimento diretto dei cittadini nella progettazione e gestione urbana, permettendo loro di fornire feedback e suggerimenti. Questo non solo eleva la trasparenza e l'efficacia delle politiche urbane, ma stimola anche un maggiore senso di appartenenza e responsabilità comunitaria.

In conclusione, gli UDT si pongono come strumenti fondamentali per rivoluzionare le città, rendendole luoghi più abitabili, ecologici e inclusivi. Rappresentano l'essenza del concetto di "città intelligente", dove l'uso strategico di tecnologia e dati mira ad elevare la qualità della vita urbana, a conservare l'ambiente e a costruire comunità resilienti, pronte ad adattarsi alle sfide future[4].

1.3 Stato dell'arte dei Digital Twin

Negli ultimi decenni, l'adozione di Urban Digital Twins è cresciuta esponenzialmente, con molte metropoli che hanno abbracciato questa tecnologia per migliorare la gestione urbana e la pianificazione. Alcune delle implementazioni più avanzate sono esempi lampanti di come queste soluzioni possono trasformare la vivibilità e la sostenibilità delle città.

1.3.1 Architettura

La tecnologia dei digital twin urbani richiede una convergenza di varie fonti di dati, strumenti analitici e tecniche di visualizzazione per creare una replica digitale in tempo reale degli ambienti urbani. L'architettura di un digital twin urbano può essere strutturata in diversi strati e componenti chiave,

ognuno dei quali svolge funzioni distinte operando al contempo in modo integrato (fig. 1.1).

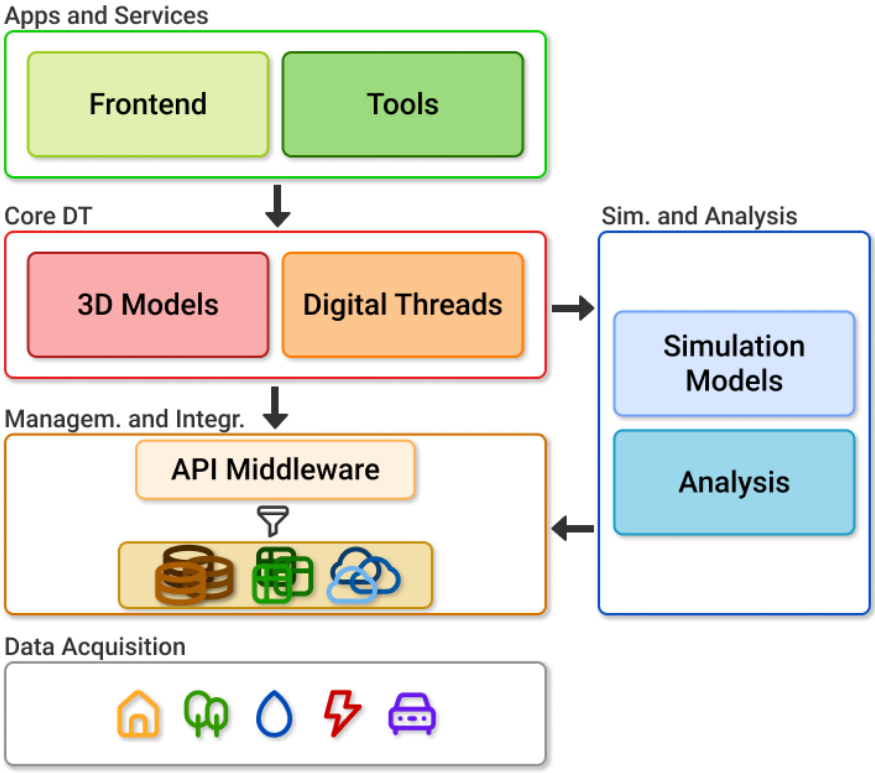


Figura 1.1: Modello architettura

1. Acquisizione Dati

La base di qualsiasi digital twin urbano sono i dati utilizzati per replicare l'ambiente reale. Questo strato coinvolge la raccolta di dati da varie fonti, tra cui:

- **Sensori e Dispositivi IoT**: dislocati in tutta la città, questi dispositivi si occupano della raccolta dati in tempo reale su traffico, condizioni meteorologiche, qualità dell'aria e molto altro.
- **Dati Geospaziali**: comprendono mappe, immagini satellitari e altre forme di dati spaziali che forniscono il layout fisico e le caratteristiche dell'area urbana.
- **Dati Operativi**: dati provenienti dai dipartimenti della città (come servizi pubblici, trasporti e sicurezza pubblica) e altri sistemi operativi.

2. Gestione ed Integrazione Dati

Questo strato gestisce l'archiviazione, l'elaborazione e l'integrazione dei dati raccolti. Garantisce che il digital twin abbia accesso a set di dati tempestivi, accurati e completi. Componenti chiave includono:

- **Data Lakes/Warehouses**: permettono l'archiviazzione dell'enorme quantità di dati strutturati e non strutturati.
- **Framework di Elaborazione**: strumenti ed algoritmi che si occupano di pulire, trasformare ed integrare dati provenienti da fonti disparate, rendendoli pronti all'utilizzo.
- **API e Middleware**: facilitano lo scambio dei dati tra sistemi e componenti diversi del DT.

3. Simulazione ed Analisi

Un insieme di modelli computazionali ed algoritmi che si occupa di simulare i processi urbani ad analizzare i dati, includendo:

- **Modelli di Simulazione**: mirano a replicare e prevedere il comportamento dei sistemi urbani sotto varie condizioni e scenari.
- **Analisi di Dati**: strumenti e tecniche per analizzare i dati, identificare modelli e generare osservazioni. Basandosi su modelli di machine learning si possono così prevedere tendenze future basandosi su dati storici.

4. Nucleo del DT

Uno strato centrale che rappresenta il cuore del Digital Twin, integrando dati e osservazioni dei livelli precedenti per costruire e aggiornare la replica digitale dell'ambiente urbano tramite:

- **Modelli e Visualizzazione 3D**: permettono una rappresentazione 3D dinamica della città, includendo edifici, infrastrutture e caratteristiche ambientali.
- **Digital Threads**: flussi continui di dati che collegano ambienti digitali e fisici, aggiornando il DT in tempo reale.

5. Applicazioni e Servizi

Posto al di sopra del nucleo, fornisce applicazioni e servizi specifici per gli utenti finali. Traduce i complessi dati e i modelli derivanti in osservazioni e interfacce utente di semplice comprensione, includendo:

- **Strumenti di Pianificazione Urbana**: per la pianificazione di scenari, zonizzazione e sviluppo infrastrutturale.
- **Sistemi di Riposta alle Emergenze**: utilizzando dati in tempo reale permettono di ottimizzare sia i tempi che le qualità delle risposte a disastri ed emergenze.
- **Piattaforme per i Cittadini**: permettono di coinvolgere i residenti facendoli interfecciare con il Digital Twin, così che possano comprendere meglio le politiche urbane e partecipare attivamente alla pianificazione della città.

Per aumentarne l'efficacia e l'intuitività, possono includere mappe interattive, realtà aumentata (AR) e realtà virtuale (VR).

6. Strumenti di Supporto Decisionale

L'obbiettivo finale di un DT è di supportare un migliore processo decisionale. Il twin fornisce informazioni necessarie per permettere di fare scelte informate sulla gestione e lo sviluppo urbano. Ciò permette:

- **Simulazione Procedure**: testa i potenziali impatti delle decisioni nell'ambiente virtuale, così da analizzare vantaggi e svantaggi.
- **Strumenti di Ottimizzazione**: permette una migliore allocazione delle risorse, migliorando il flusso del traffico e altre operazioni urbane.

1.3.2 Alcuni Esempi

Mostriamo quindi alcuni esempi di Urban Digital Twin sviluppati. Buona parte di essi sono tuttavia chiusi al pubblico, limitando molto le informazioni su di essi trovabili online.

Shanghai (fig. 1.2) si distingue come uno dei Digital Twin urbani più avanzati ad oggi[5], creato dalla compagnia di Pechino 51World[1], un punto di riferimento nel settore dei gemelli digitali. Coprendo un'area di circa 3750 km quadrati, sfrutta Unreal Engine 5[21] per la visualizzazione, rendendo la rappresentazione della città incredibilmente dettagliata e dinamica. La modellazione di Shanghai è stata realizzata combinando dati da satelliti, droni e sensori accompagnati da quello che pensiamo sia un BIM completo per la visualizzazione degli edifici. Per quanto riguarda invece le informazioni sui dati sensoristici IoT sono limitate e non sappiamo quante e quali informazioni siano gestite.

Figura 1.2: Digital Twin della città di Shangai[5]

Wellington (fig. 1.3), grazie al lavoro di Buildmedia, ha sviluppato il suo Digital Twin[13] utilizzando anche in questo caso Unreal Engine 5. Questo modello offre dati in tempo reale su diversi aspetti del traffico urbano, inclusi bus, treni, traghetti, automobili e biciclette. Sono fornite inoltre informazioni sul traffico aereo e sulla disponibilità dei parcheggi, evidenziando la capacità di integrare varie fonti di dati per una gestione urbana più efficiente.

Herrenberg (fig. 1.4), una piccola città tedesca con circa 30.000 abitanti, ha messo in campo un Digital Twin notevolmente avanzato, facendo uso

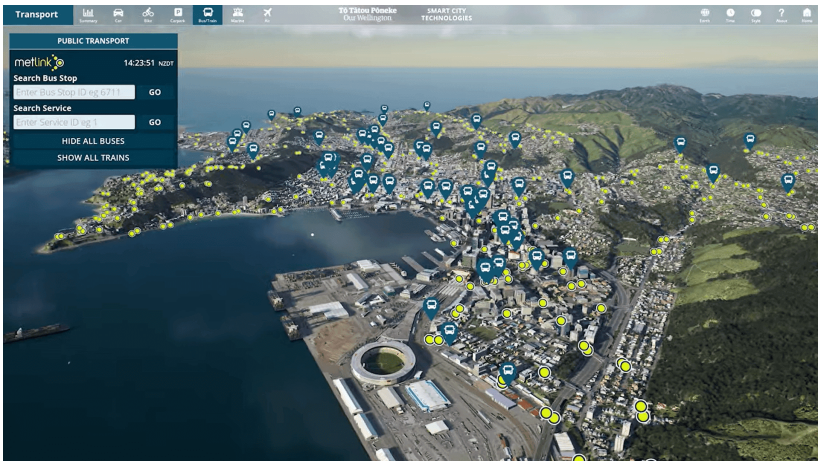


Figura 1.3: Digital Twin della città di Wellington[13]

di dati eterogenei[82]. Per la modellazione 3D, si sono avvalsi di LiDAR e dati BIM, arricchendo il loro sistema con informazioni sul modello stradale, pattern di movimento cittadini, qualità dell'aria e urban emotions. Le simulazioni, effettuate tramite calcolo su cluster HPC, forniscono previsioni precise su traffico e flusso del vento.

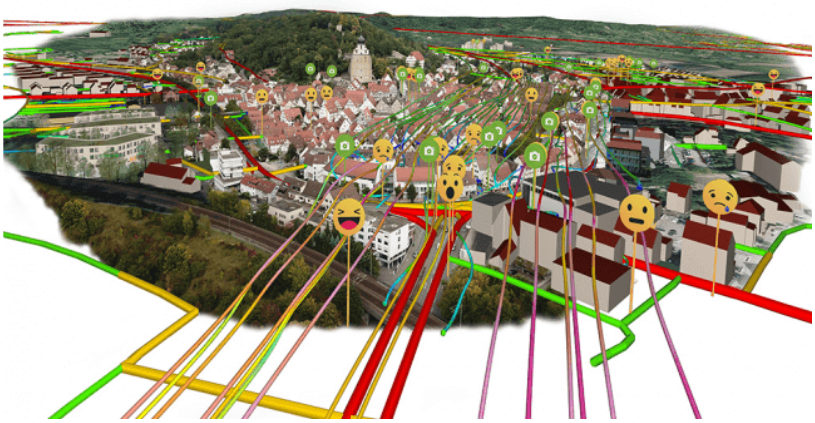


Figura 1.4: Digital Twin della città di Herrenberg[82]

Perugia (fig. 1.5), con il suo Digital Twin presentato nel 2023 e sviluppato da Wisetown[81], utilizza nuovamente Unreal Engine 5, ma si distacca dagli altri per un focus minore sulla resa visiva e maggiore sui dati ambientali, promuovendo l'interazione tra amministrazione comunale e cittadini[31]. Questo Digital Twin permette la visualizzazione della distribuzione demografica,

analisi del verde urbano per monitorare lo stato di salute della vegetazione e un sistema di issue tracking che consente ai cittadini di segnalare problemi relativi alle infrastrutture urbane.

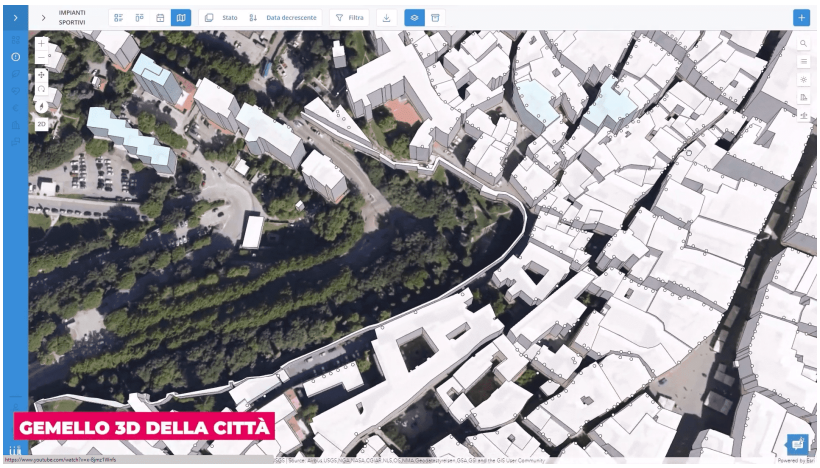


Figura 1.5: Digital Twin della città di Perugia[32]

L'analisi degli Urban Digital Twins attraverso gli esempi di Shanghai, Wellington, Herrenberg e Perugia rivela il vasto potenziale che questa tecnologia detiene per le città globali. Ciascun esempio illustra un approccio distintivo nell'uso dei Digital Twins per risolvere specifiche problematiche urbane, che vanno dalla gestione del traffico e della pianificazione, alla mitigazione degli effetti dei cambiamenti climatici, fino al potenziamento dell'engagement dei cittadini.

Questi casi di studio mettono in luce insegnamenti essenziali per il futuro sviluppo dei Digital Twins urbani, che verranno esplorati più approfonditamente nei capitoli successivi:

- **L'importanza dei dati**: La qualità, la vastità e la facilità di accesso ai dati sono determinanti per la riuscita e l'efficienza dei Digital Twins. Affrontare la sfida di armonizzare dati da fonti disparate necessita l'utilizzo di approcci innovativi per quanto riguarda la standardizzazione e la tutela della privacy. Questa evidenzia la necessità di una gestione dati sofisticata che possa supportare l'integrazione e l'analisi senza compromettere la sicurezza o la riservatezza delle informazioni.
- **Adattabilità e scalabilità**: È cruciale che i Digital Twins siano concepiti per essere flessibili e scalabili, consentendogli di adattarsi alle

trasformazioni urbane e alle emergenti esigenze cittadine. Questa elasticità assicura che tali strumenti mantengano la loro rilevanza e utilità nel tempo, adeguandosi all'evoluzione del tessuto urbano e alle dinamiche sociali. La progettazione orientata alla scalabilità permette inoltre di estendere l'implementazione dei Digital Twins a nuove aree e funzionalità, garantendo una copertura più ampia e un impatto più significativo sulla vita urbana.

Incorporando questi apprendimenti, è possibile delineare un percorso per l'avanzamento e l'applicazione dei Digital Twins urbani che non solo affronti le sfide attuali delle metropoli ma che si proietti anche verso la creazione di città intelligenti, resilienti e sostenibili. Questo approccio integrato ed evolutivo apre la strada a una nuova era della pianificazione urbana, in cui la tecnologia dei Digital Twins gioca un ruolo chiave nel modellare l'infrastruttura e la società del futuro.

Capitolo 2

Bologna Digital Twin

Contents

2.1 Obiettivi del progetto

L'obiettivo principale è creare un Digital Twin della città, che serva come strumento avanzato per il supporto alla pianificazione urbana, l'analisi ambientale (comprese simulazioni sugli effetti del cambiamento climatico), la gestione del traffico, la valutazione dei consumi energetici e l'interazione con i cittadini.

Il ruolo che ci è stato affidato in questo progetto è la realizzazione dell'ambiente 3D di Bologna, basato sui dati forniti, con un'attenzione particolare su alcuni aspetti chiave. Questi includono l'estrazione di informazioni tridimensionali dai dati LiDAR, lo sviluppo di vari layer di visualizzazione, l'analisi del verde urbano e la creazione di un'interfaccia utente interattiva, adattata in base al tipo di utente che si interfaccia con il DT.

Il nostro approccio sarà strettamente di tipo data-driven, cioè guidato dai dati, in quanto è necessario fondare le nostre decisioni progettuali sui dati disponibili e su cosa essi ci permettono di realizzare.

Durante la realizzazione di questo progetto ci siamo potuti confrontare con altri dipartimenti di sviluppo e abbiamo tratto da queste conversazioni gli obiettivi principali del nostro lavoro.

2.2 Analisi di progetto

Lo sviluppo di un Digital Twin 3D per la città di Bologna implica una serie di complessità sia tecniche che organizzative. Queste difficoltà nascono principalmente dalla qualità e dalla disponibilità dei dati esistenti, oltre che dalla mancanza di alcune categorie di dati necessari.

L'obbiettivo, ispirato dall'analisi delle migliori pratiche nei Digital Twins già realizzati, è quello di creare un modello 3D completamente esplorabile di tutta la città, arricchito da diversi livelli di visualizzazione che variano a seconda del livello di dettaglio richiesto.

Un elemento fondamentale è rappresentato dalla possibilità di interagire con i componenti del modello digitale, consentendo agli utenti di accedere a informazioni specifiche su elementi come edifici, alberi, strade e altre caratteristiche urbane semplicemente selezionandoli nel modello 3D. Un altro ruolo importante è l'aggiornamento e la manutenzione nel tempo del Digital Twin. Data la natura dinamica di una città, è essenziale che il modello digitale possa riflettere fedelmente e tempestivamente le trasformazioni urbane. Ciò implica la necessità di stabilire processi efficienti per l'aggiornamento dei dati e la loro integrazione nel modello 3D.

Infine, la realizzazione di un Digital Twin 3D dovrebbe rispondere alle esigenze dei diversi attori coinvolti, inclusi urbanisti, ingegneri, amministratori e cittadini, rendendo cruciale lo sviluppo di un'interfaccia utente intuitiva e facilmente navigabile che favorisca l'esplorazione e l'interazione con il modello tridimensionale.

Riassumendo, i criteri fondamentali per la realizzazione del DT di Bologna includono:

- **Immersività**: il Digital Twin deve offrire un'esperienza il più possibile vicina alla realtà, obiettivo raggiungibile attraverso l'uso di mesh e texture ad alta definizione, insieme a un sistema di illuminazione che emuli fedelmente quello reale.

- **Performance**: per assicurare un'esperienza di navigazione fluida e un utilizzo efficace, è critico mantenere elevate le prestazioni, garantendo un framerate alto e costante.

- **Data-driven**: l'aspetto forse più cruciale considerando che l'obiettivo è quello di creare un Urban Digital Twin, richiedendo la costruzione

del modello 3D e di tutte le visualizzazioni ad esso correlate basandosi fedelmente sui dati reali della città.

- **UX intuitiva**: per assicurarsi che qualsiasi tipo di utente possa interagire in maniera naturale con il Twin.

Segue un'analisi preliminare che identifica e categorizza le sfide principali in base alla loro natura.

Dati i tipi di dati a disposizione, la prima grande sfida è la creazione del modello 3D dell'intera città. Come si può dedurre, gli unici elementi direttamente convertibili in mesh sono i modelli del DTM (Modello Digitale del Terreno). Per tutti gli altri tipi di dati, è necessario un elaborato processing per ottenere dei modelli tridimensionali.

2.2.1 Raccolta dati

I dati fornitici provengono dal **Comune di Bologna** in due distinte sorgenti:

- **Bologna Open Data**[7]
- **Rilievi aerei**

Bologna Open Data è un progetto del Comune di Bologna che mira a rendere accessibili al pubblico una serie di dati in formato aperto (Open). L'obiettivo è quello di permettere a cittadini, aziende e associazioni di utilizzare e valorizzare questi dati. Il catalogo presente sul portale è in costante aggiornamento.

Inoltre ci sono stati forniti dei dataset risultanti dall'ultima serie di **rilievi aerei** commissionati dal Comune di Bologna. Questi dati sono stati forniti a Cineca in seguito all'accordo per lo sviluppo del Digital Twin, sono tutt'ora caricati sul supercomputer Leonardo e contengono dati **LiDAR** e **fotografici**.

2.2.1.1 Opendata Comune di Bologna

Alcuni esempi dei dati disponibili includono la rilevazione del flusso di veicoli, le posizioni delle fermate dei mezzi pubblici, i flussi di biciclette in vari punti della città e persino il registro dei defibrillatori. Inoltre, sono disponibili dati climatici come temperature, precipitazioni e le misurazioni della qualità dell'aria.

Tra di essi troviamo anche alcuni dati per noi essenziali ai fini di questo progetto, essi sono **edifici volumetrici**[9] e **alberi in manutenzione**[8].

Edifici Volumetrici

Gli edifici volumetrici fanno parte della **Carta Tecnica Comunale** (CTC), una cartografia tecnica in scala 1:200 che il Sistema Informativo Territoriale (SIT) del Comune di Bologna ha fatto realizzare nel 2001 tramite il metodo fotogrammetrico diretto. In questa restituzione sono state inserite anche la **Quota di Gronda** e la **Quota al Piede** degli edifici, con una tolleranza di 54 cm. Questo significa che la pianta di ogni edificio è stata divisa in aree multiple le quali parametri di altezza comuni (fig. 2.1).

L'aggiornamento continuo della CTC avviene utilizzando fonti primarie come progetti edilizi e fonti secondarie come ortofoto di precisione, librerie di immagini oblique e dati LiDAR aerei come quelli fornitici.

Il **Sistema di Riferimento** utilizzato è il ***WGS84*** [80], in formato *Gradi Decimali*, e sono tutt'ora presenti 66095 record in totale.

Nello specifico, abbiamo un dataset tabellare con i seguenti dati:

- **CODICE_OGGETTO** (int): un codice univoco per ciascun edificio.
- **DATA_ISTITUZIONE** (date): data dell'inserimento nel CTC.
- **DATA_VARIAZIONE** (date): data dell'ultima modifica.
- **NOTEOGG** (string): testo con eventuali note.
- **PERIMETRO** (int): lunghezza del perimetro.
- **DESCRIZIONE** (string): descrizione/classe tipologia dell'edificio.
- **ORIGINE** (string): quale tipo di dato è stato consultato per l'inserimento.
- **QUOTA_GRONDA** (double): quota media della gronda sul livello del mare.
- **QUOTA_PIEDE** (double): quota media del piede sul livello del mare.
- **ALTEZZA_GRONDA** (double): differenza tra le quote di gronda e di piede.
- **AREA_OGG** (double): area dell'edificio.
- **VOLUME** (double): volume dell'edificio.
- **Geo Point** (geo_point_2d): coordinate geografiche (latitudine e longitudine) del centroide dell'edificio.
- **Geo Shape** (geo_shape): coordinate geografiche (latitudine e longitudine) che definiscono la forma poligonale dell'edificio.

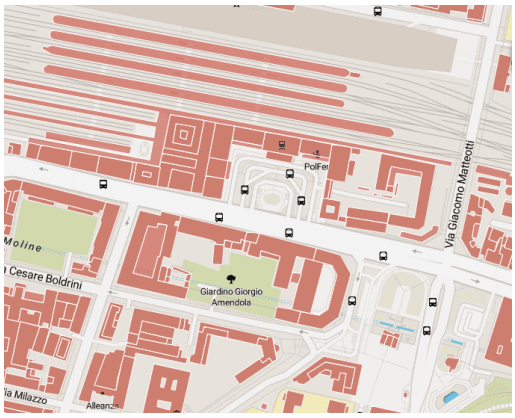


Figura 2.1: Open Data: edifici volumetrici (zona stazione)

Si nota che anche se viene utilizzato un double per i valori di quota la precisione massima al momento apprezzabile sul dataset è al decimetro.

Alberi in manutenzione

Il dataset degli Alberi in manutenzione del Comune di Bologna contiene i dati sugli alberi soggetti a manutenzione all'interno del territorio comunale di Bologna. Sono quindi presenti i soli esemplari arborei rilevati attraverso il sistema di gestione della relativa manutenzione. Sono tutt'ora presenti 85043 record in totale. Le coordinate delle loro posizioni rispettano lo stesso formato precedentemente descritto (*WGS84 in Gradi Decimali*)

Nello specifico, questo dataset tabellare presenta i seguenti dati:

- **Progressivo pianta** (string): un codice univoco per ciascuna pianta.
- **Codice albero** (string): un codice non univoco che può rappresentare gruppi di piante.
- **Pianta isolata** (string): una stringa binaria (S/N) che indica se l'albero è isolato o meno.
- **Specie arborea** (string): specie della pianta.
- **Classe di altezza** (string): classe di altezza della pianta, divisa nelle seguenti categorie:

- Cl1: <6mt
- Cl2: 6mt - 12mt
- Cl3: 12mt - 16mt
- Cl4: 16mt - 23mt
- Cl5: >23mt

- **Classe circonferenza (diametro)** (string): classe di circonferenza della pianta, divisa nelle seguenti categorie:
  - Cl1: <15 (<5 cm)
  - Cl2: 15 - 30 (5-10 cm)
  - Cl3: 30 - 45 (10-15 cm)
  - Cl4: 45 - 60 (15-19 cm)
  - Cl5: 60 - 90 (19-28 cm)
  - Cl6: 90 - 110 (28-35 cm)
  - Cl7: 110 - 140 (35-45 cm)
  - Cl8: 140 - 170 (45-54 cm)
  - Cl9: 170 - 200 (54-64 cm)
  - Cl10: 200 - 230 (64-73 cm)
  - Cl11: 230 - 260 (73-80 cm)
  - Cl12: >260 (>80 cm)

- **Dimora** (string): indica la tipologia di dimora nella quale è inserita la pianta, questa può essere delle seguenti categorie:
  - Prato
  - Terra
  - Fioriera
  - Formella (e sotto tipologie)

- **Irrigazione** (string): una stringa binaria (S/N) che indica se l'albero è irrigato.
- **Albero di pregio** (string): una stringa binaria (S/N) che indica se l'albero è di pregio o meno.
- **Data inserimento inventario** (date): data dell'inserimento nell'inventario.
- **Data ultimo aggiornamento** (date): data dell'ultima modifica.
- **Distanza dai fabbricati** (string): indica la distanza dai fabbricati della pianta, può ricadere nelle seguenti categorie:
  - Da 0 a 3mt
  - Da 3mt a 6mt
  - Oltre 6mt

- **Data impianto** (date): data in cui è stato piantato l'albero.
- **Geo Point** (geo_point_2d): coordinate geografiche (latitudine e longitudine) del centro del tronco dell'albero.
- **Geo Shape** (geo_shape): coordinate geografiche (latitudine e longitudine) dello shape dell'albero (punto al centro del tronco).
- **Campagna** (string): indica in quale campagna di rimboscamento appartiene.
- **IN_PATRIM** (string): una stringa binaria (0/1) che indica se l'albero è in patrimonio o meno.
- **Zone di prossimità** (string): indica la zona alla quale appartiene l'albero.
- **Area statistica** (string): indica una zona più specifica alla "Zona di prossimità".

Le date di piantumazione sono presenti solo in maniera parziale e sono state inserite in inventario dal 2014 in poi.

Ne è possibile vedere la distribuzione in alcune aree del centro tramite la visualizzazione in figura 2.2.

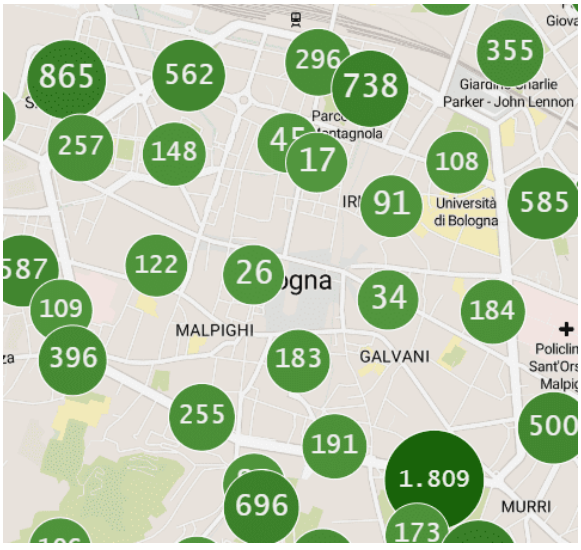


Figura 2.2: Open Data: visualizzazione distribuzione alberi

2.2.1.2 Rilievi aerei

I dati risultanti da questi rilievi sono di vario tipo, principalmente si dividono in derivati da pointcloud LiDAR e fotografie.

Dati LiDAR

La tecnologia **LiDAR** (Light Detection and Ranging)[64] è un metodo di rilevamento remoto che misura le distanze tra il sensore e gli oggetti tramite un laser pulsato. Questa tecnologia è ampiamente utilizzata in numerose applicazioni come la mappatura topografica, la forestazione, l'urbanistica e recentemente la creazione di modelli 3D per creare i gemelli digitali di città.

Spesso i dati ottenuti durante la rilevazione richiedono una fase di elaborazione prima di essere effettivamente utilizzati. Questi processi includono principalmente processi come la georeferenziazione, la triangolazione aerea e la classificazione dei punti.

Nel nostro caso questi dati sono archiviati all'interno del supercomputer **Leonardo**[62], al momento 2° supercomputer più potente in Europa e 6° al mondo, ospitato al tecnopolo di Bologna e gestito dal consorzio interuniversitario CINECA. Abbiamo quindi fatto richiesta di accesso ad esso per il nostro progetto e, dopo la avvenuta approvazione dei nostri username, siamo stati aggiunti al gruppo risorse contenente i dati. Tramite connessione **SSH**[73] abbiamo inizialmente trasferito i nostri dati alle nostre directory remote e quindi li abbiamo ottenuti utilizzando **FTP**[52].

I dati presenti nel nostro dataset sono i seguenti (tabella 2.1):

	Dimensione	**Num. tot. file**	**Formato**	**Tipo di dato**
Raw Data	947 G	1738	.las	Dati LiDAR senza classificazione
Nuvola Classificata	198 G	654	.las	Dati LiDAR con classificazione
DSM	4.1 G	654	.asc	Digital Surface Model
DTM	4.1 G	654	.asc	Digital Terrain Model

Tabella 2.1: Tabella dati LiDAR

Con **Raw Data** si intendono i dati raccolti al loro stato puro. Essi vengono memorizzati in un formato di file denominato **LAS** (LASer)[61], che è un open standard specificato dalla American Society for Photogrammetry and Remote Sensing (ASPRS)[41]. Questo formato binario è progettato per l'archiviazione delle nuvole di punti ottenute durante i rilievi e supporta la

rappresentazione di dati 3D con l'aggiunta di altri attributi associati ad ogni punto.

Purtroppo l'unico documento in nostro possesso riguardante questi dati illustra unicamente gli strumenti utilizzati e le metodologie con i quali i dati di partenza (Raw Data) sono stati rilevati, tralasciando ogni informazione sul processing utilizzato per ottenere i dati finali, suddivisi in:

- **Nuvola Classificata**
- **Digital Surface Model** (DSM)[17]
- **Digital Terrain Model** (DTM)[17]

Questi file sono stati organizzati in modo da dividere il comune di Bologna in una griglia, suddividendolo in **Tile** di dimensione 500x500m. Come si può capire dalla tabella, questa suddivisione genera in totale 654 tile organizzati nel seguente modo.

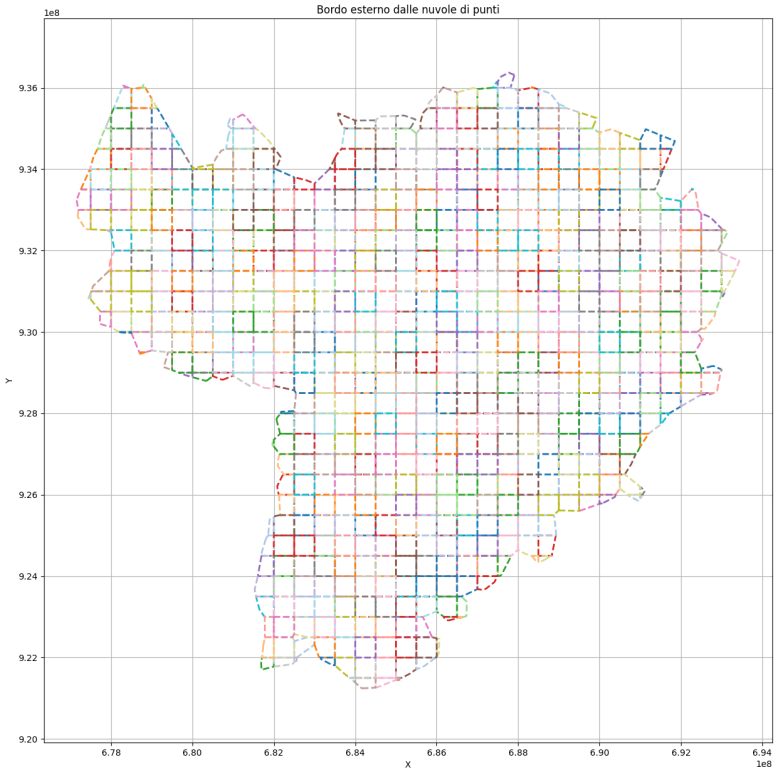


Figura 2.3: Comune di Bologna: suddivisione in tile 500x500

Come si può notare dalla figura 2.3 le nuvole di punti seguono i confini del Comune di Bologna, perciò sono presenti unicamente i punti con coordinate all'interno di esso. Di conseguenza, i tile di confine non sono tutti di forma quadrata, anche se il nome rimane in riferimento all'LLC come se il tile fosse riempito interamente.

La convenzione di denominazione dei nomi di questi file deriva dalle coordinate dell'angolo in basso a sinistra, ovvero l'**LLC** (Lower Left Corner), in formato **UTM (WGS84)**. Questo sistema divide la terra in 60 bande longitudinali, rappresentate con numeri, ognuna di esse suddivisa a sua volta in 20 bande latitudinali, rappresentate con lettere. All'interno di queste zone le coordinate dei punti vengono rappresentate utilizzando valori metrici suddivisi in Est (E) e Nord (N).

Nel nostro caso, Bologna ricade nel quadrante 32T, perciò i nostri file sono nominati con la stringa **"32_E_N"** (la T viene omessa) dove **E** e **T** sono appunto le coordinate dell'LLC del nostro tile. Per esempio, il file contenente i dati su Piazza Maggiore avrà il nome *32_686000_4929000*.

Andando nel dettaglio, i file della **Nuvola Classificata** sono nuvole di punti non organizzate contenute in file di tipo LAS (fig. 2.4). Oltre ai campi base possiamo trovare anche diversi campi calcolati in un momento successivo rispetto al rilievo. In particolare possiamo trovare il campo di **Classificazione** (*classification*) nel quale si determina se quel punto appartiene o meno al ground. Anche in questo caso non sappiamo come questa classificazione è avvenuta, tuttavia sembra essere una stima con una buona attendibilità.

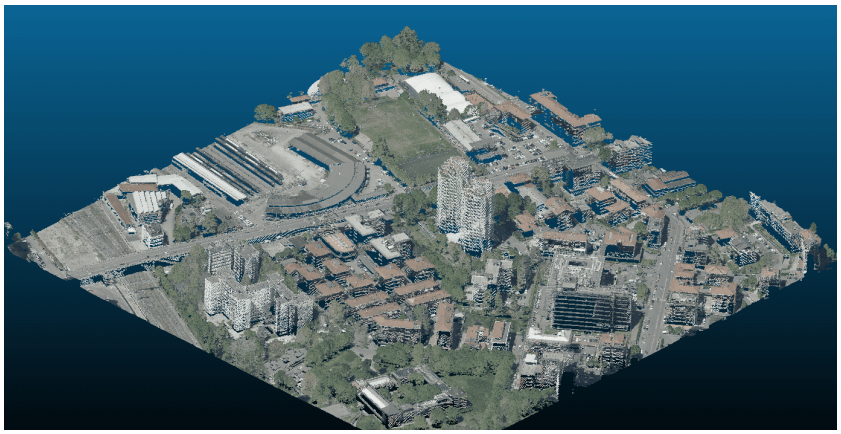


Figura 2.4: File LAS visualizzato su CloudCompare (32_687000_4930500)

Ogni file di questo tipo presenta **dai 9 ai 10 milioni** di punti, avendo quindi una densità che si aggira intorno ai **38 punti al m^2**.

Per ogni punto abbiamo le seguenti informazioni (tabella 2.2):

Dimensione	Descrizione
x, y, z	Coordinate (UTC WGS84)
intensity	Misura dell'intensità di ritorno dell'impulso laser
return_number	Numero di ritorno dell'impulso per un dato impulso
number_of_returns	Numero totale di ritorni per un dato impulso
synthetic	Se un punto è stato creato con un metodo diverso dalla raccolta lidar, ad esempio digitalizzato da un modello stereo fotogrammetrico
key_point	Se un punto è considerato un punto-chiave del modello e non deve essere traslaciato in nessun algoritmo di assestamento
withheld	Se il punto non deve essere incluso nell'elaborazione
overlap	Se un punto si trova all'interno della regione di sovrapposizione di due o più linee di volo o strisciate
scanner_channel	Il canale (testa dello scanner) di un sistema multicanale
scan_direction_flag	La direzione in cui viaggiava lo specchio dello scanner al momento dell'impulso di uscita
edge_of_flight_line	Ha valore 1 solo quando il punto si trova alla fine di una scansione
classification	Può essere 1 (Unassigned) o 2 (Ground)
user_data	Può essere utilizzata a discrezione dell'utente per dati extra
scan_angle	Angolo di uscita del punto laser dal sistema laser (compreso il rullo dell'aeromobile)
point_source_id	Il file da cui proviene questo punto
gps_time	Il valore del time tag a virgola mobile doppia in cui il punto è stato acquisito
red, green, blue	Colore del punto
nir	La regione del vicino infrarosso dello spettro

Tabella 2.2: Descrizione dei dati nel file LAS.

Successivamente sono stati creati anche i file del **Digital Surface Model (DSM)** e del **Digital Terrain Model (DTM)**. Il primo di essi contiene punti appartenenti a qualsiasi classificazione, mentre il secondo si basa solamente sui punti classificati come terreno (Ground).

Questi file sono di tipo binario rappresentati nel formato ASCII Grid (.asc)[50] e contengono nuvole di punti molto più sparse ma organizzate in una griglia 1000x1000, avendo quindi un punto ogni 0.5m. In questo formato vengono mantenuti solo i valori di posizione, andando a specificare solo la coordinata z per ogni punto, permettendo di avere file di piccole dimensioni.

Dati fotografici

I dati fotografici che ci sono stati passati sono foto aeree[40] scattate durante i rilievi LiDAR (**Oblique**) e **Ortofoto** aeree geometricamente corrette e georeferenziate (tabella 2.3).

	Dimensione	**Num. tot. file**	**Formato**	**Tipo di dato**
Oblique	3.6 T	4749	.jpg	right
		4788	.jpg	left
		4826	.jpg	back
		4811	.jpg	forward
		4651*	.tif	nadir**
Ortofoto	658 G	199***	.tif	Tavole CIR
		199***	.tif	Tavole RGBN

* per ogni file .tif[54] ci sono i corrispondenti file .dbf, .shp, .shx[74], .prj[20], .tfw
** punto del terreno che si trova verticalmente sotto il centro prospettico dell'obiettivo della fotocamera o dei rilevatori dello scanner
*** per ogni file .tif ci sono i corrispondenti file .tfw e .prj

Tabella 2.3: Tabella dati fotografici

Le foto *oblique* (fig. 2.5) sono state ottenute da 4 camere inclinate a 45° nelle quattro direzioni rispetto all'andamento dell'aeromobile (.jpg)[59] e sono ad altissima risoluzione (14192x10640, 240 dpi).

Figura 2.5: Esempio foto obliqua

Ad esse si aggiunge l'immagine nadir (.tif) che inquadra il terreno al momento dello scatto, presentano la stessa dimensione delle loro compagne ma hanno una risoluzione diversa (72x72).

Per quanto riguarda la convenzione di denominazione di queste immagini, esse presentano nomi particolari in cui è inclusa la data di acquisizione, alcuni numeri che indicano in modo vago la posizione di acquisizione e il tipo di orientamento della camera.

Basandoci sul numero totale dei file, possiamo dedurre che abbiamo all'incirca 7 quintuple di immagini per ogni nostro tile.

Le *Ortofoto* (fig. 2.6) sono invece immagini di formato GeoTiff (tif + file extra di georeferenziazione)[54] ad altissima risoluzione (20802x20802) scattate in modo ortogonale al terreno. Ognuna di queste foto ci fornisce una visualizzazione di vasta area di territorio, poco più di $\mathbf{2km^2}$. Nel seguente esempio possiamo vedere parte del quartiere Saffi.

Figura 2.6: Esempio ortofoto

Per capire la convenzione di denominazione di queste immagini dobbiamo prima capire come esse sono organizzate. Ognuna di esse appartiene a una quadrupla disposta nel seguente modo (fig. 2.7):

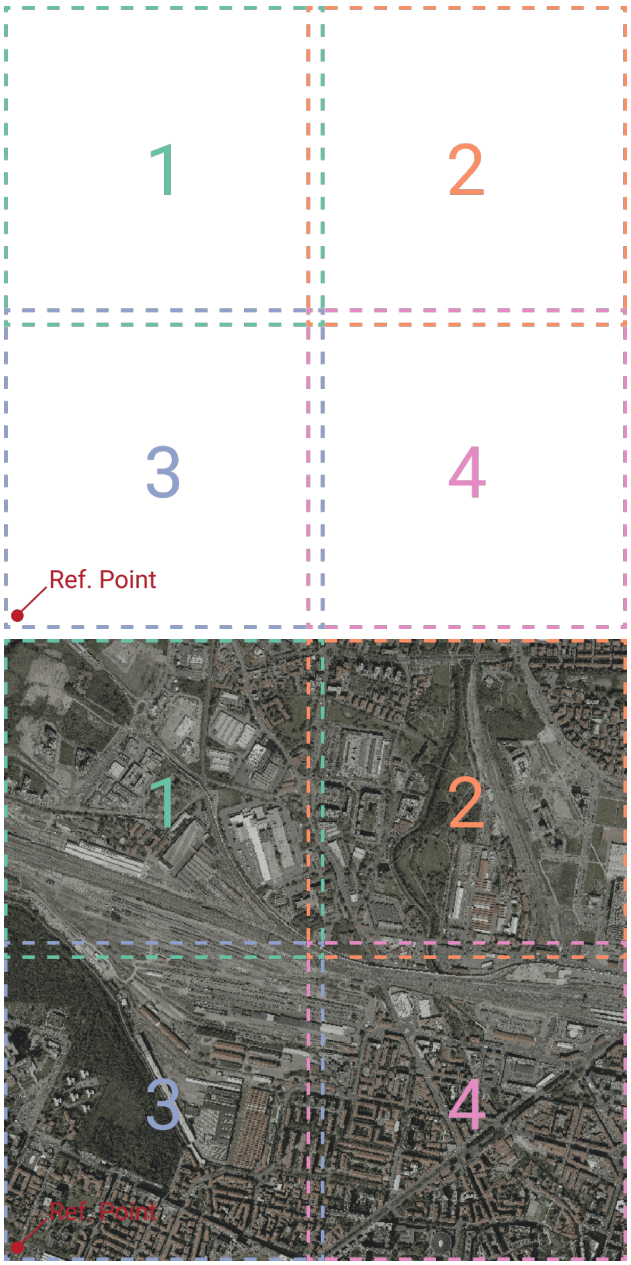


Figura 2.7: Esempio disposizione ortofoto

Come si può notare che c'è una leggera sovrapposizione tra i bordi delle immagini. Oltre a questo il punto di riferimento (Ref. point) non è all'esatto

angolo ma all'LLC del rettangolo interno che arriva a metà del bordo di sovrapposizione.

Quindi, la convenzione di denominazione utilizzata utilizza la zona (sempre 32), le prime 4 cifre delle coordinate Est e Nord del punto di riferimento e il la posizione dell'immagine all'interno del gruppo per denominarla. Nel nostro caso la ortofoto visualizzata in figura 2.6 avrà il nome *32_6840493 0_4.tif*.

Queste ortofoto sono quindi suddivise in due cartelle in base alla tipologia di dati rappresentati:

- **RGBN**: a quattro canali (8 bit ognuno), rispettivamente Red-Green-Blue-Near Infrared.
- **CIR**: a tre canali (8 bit ognuno), rispettivamente Near Infrared-Red-Green. Se ne può vedere un esempio in figura 2.8.

Figura 2.8: Esempio ortofoto (versione CIR)

2.2.2 Creazione del modello geometrico 3D della città

A causa della metodologia con cui sono stati raccolti i dati LiDAR, ossia mediante sorvolo aereo, è presente una significativa assenza di dati relativi alle facciate verticali degli edifici nella nuvola di punti. Di conseguenza, le informazioni disponibili attraverso la nuvola di punti riguardano principalmente le superfici orizzontali, con solo occasionali e disomogenee indicazioni sulle componenti verticali degli edifici (fig. 2.9).

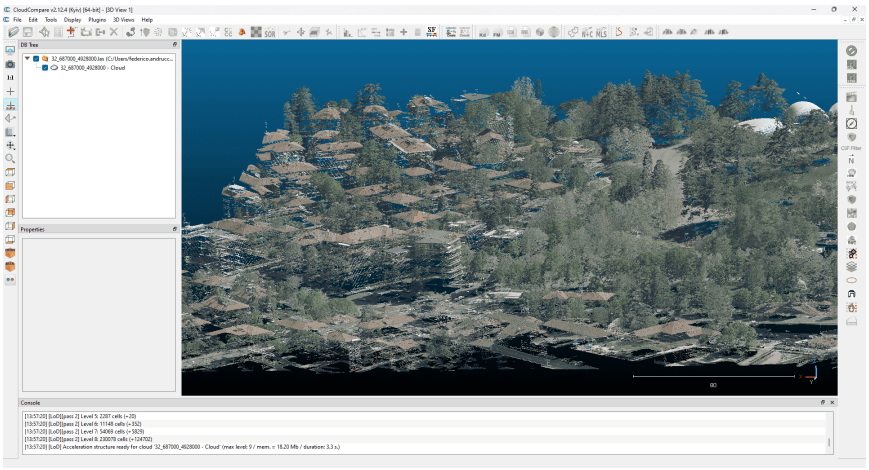


Figura 2.9: Esempio distribuzione punti LiDAR (CloudCompare)[24]

Pertanto, emerge con chiarezza la necessità di intraprendere un processo di elaborazione dei dati LiDAR particolarmente accurato e sofisticato, al fine di generare le mesh tridimensionali degli edifici, trasformando le limitate informazioni iniziali in modelli il più possibile dettagliati e completi, includendo sia le strutture orizzontali che quelle verticali.

Aggregazione di fonti di dati multiple

Per affrontare le lacune presenti nei dati LiDAR, diventa cruciale l'impiego integrato di altre fonti di dati geospaziali. La combinazione di immagini satellitari, fotografie aeree ortorettificate e database con informazioni specifiche sugli edifici potrebbe colmare le carenze informative, fornendo un quadro più ampio e dettagliato della struttura urbana. Con questa aggregazione di dati si mira ad arricchire il set di dati a disposizione, facilitando la creazione dei modelli 3D necessari al Twin.

Utilizzo di tecniche di Computer Vision

L'adozione di metodologie avanzate di elaborazione delle immagini (Image Processing)[49] e di visione artificiale (Computer Vision)[44] rappresenta un'ulteriore tattica cruciale per l'analisi e la manipolazione efficace delle nuvole di punti. Miriamo ad utilizzare queste tecnologie per interpretare e convertire i dati grezzi in rappresentazioni tridimensionali fedeli della realtà urbana.

Gestione della complessità computazionale

Affrontare la complessità computazionale è un altro aspetto fondamentale per assicurare che i modelli 3D ottenuti non siano soltanto ricchi di dettagli ma anche efficienti dal punto di vista del rendering e della simulazione in tempo reale. L'applicazione di tecniche di ottimizzazione, come la diminuzione del numero di poligoni e l'introduzione di vari livelli di dettaglio (LOD), consente di trovare un equilibrio tra la qualità visiva e le prestazioni complessive del sistema, rendendo il modello 3D accessibile su dispositivi con differenti potenzialità di calcolo.

Standard e formati di modellazione

Inoltre, rimane importante l'osservanza di standard e formati di modellazione riconosciuti a livello internazionale, facilitando l'integrazione del modello 3D in diverse applicazioni. In questo modo vengono sostenute l'interoperabilità e la collaborazione tra svariate piattaforme e strumenti.

Ciò non solo aumenta l'utilità e l'efficacia del Digital Twin ma promuove anche un ambiente di lavoro condiviso tra professionisti aventi skill set e obiettivi più mirati.

2.2.3 Il problema del texturing

Una delle questioni più critiche si manifesta nella realizzazione del texturing delle mesh degli edifici urbani, dovuto principalmente alla mancanza di dati sulle superfici verticali degli edifici. Se la nuvola di punti LiDAR avesse incluso anche queste superfici verticali, la problematica sarebbe stata parzialmente attenuata. Infatti, ogni punto della nuvola di punti, oltre a fornire coordinate spaziali (x, y, z), contiene anche dati cromatici (R, G, B), che avrebbero potuto facilitare notevolmente la derivazione di texture direttamente da queste informazioni.

Questa difficoltà si aggrava ulteriormente tenendo in considerazione gli altri tipi di dati a disposizione. Le ortofoto, per definizione, sono scatti ortogonali al suolo e, di conseguenza, non includono le facciate degli edifici.

Per quanto riguarda invece le fotografie oblique fornite, benché in alcuni casi possano offrire dettagli sulle facciate, la loro variabilità e la necessità di un'intensa elaborazione rendono molto complesso il loro utilizzo per un texturing coerente e omogeneo degli edifici.

Strategie di texturing degli edifici

Data l'assenza di informazioni ad-hoc per le facciate degli edifici, è richiesto un approccio più creativo per affrontare questa parte di texturing. Le possibili soluzioni da noi esplorate sono state le seguenti:

- **Applicazione di Texture Generiche**: L'uso di texture che imitano materiali edilizi comuni, quali mattoni, intonaco, e vetro, potrebbe essere una via praticabile. Queste texture potrebbero essere applicate proceduralmente attraverso algoritmi specifici che le adattano ai diversi tipi di facciate, tenendo conto anche degli stili architettonici. Sebbene questa tecnica possa risultare efficace in diversi contesti, per un Digital Twin che mira a una corrispondenza esatta con la realtà urbana, questo approccio non sarebbe l'ideale, a causa della perdita di fedeltà specifica per ogni edificio.

- **Sfruttamento di Database di Texture Online**: L'utilizzo di risorse online come Google Maps[56], Google Earth[55] o Bing Maps[42] rappresentano una possibile soluzione. Questi servizi possono offrire un vasto archivio di texture realistiche che, se disponibili, potrebbero essere applicate direttamente agli edifici del nostro modello 3D. Questo metodo promette un alto grado di precisione e realismo, permettendo di mantenere una stretta corrispondenza visiva con le effettive facciate degli edifici nella città fisica.

In breve, entrambe le soluzioni offrono vantaggi e limitazioni: mentre l'applicazione di texture generiche offre un controllo più ampio e la possibilità di coprire ampie aree senza dati specifici, l'uso di database di texture online potrebbe garantire una maggiore accuratezza visiva per gli edifici.

Strategie di texturing del terreno

Per quanto riguarda il texturing del Digital Terrain Model (DTM), le fotografie aeree si rivelano un'ottima fonte di dati. Attraverso l'elaborazione di queste immagini per adattarle alla mesh del terreno, è possibile raggiungere un alto livello di dettaglio e realismo. Questo processo implica l'allineamen-

to preciso delle immagini alle caratteristiche topografiche del modello 3D e l'aggiustamento delle texture per rappresentare con fedeltà le strade, la vegetazione, i corsi d'acqua e le altre caratteristiche sia naturali che artificiali del paesaggio urbano.

Data la vasta estensione del territorio da modellare e la complessità intrinseca delle texture del terreno, diventa essenziale l'ottimizzazione. Strategie come il mipmapping e il tiling delle texture emergono come soluzioni efficaci per bilanciare le prestazioni del sistema di visualizzazione con la necessità di conservare un dettaglio visivo elevato nelle aree di interesse. Queste tecniche permettono di ridurre il carico computazionale mantenendo al contempo una qualità grafica elevata, assicurando che il modello 3D sia visualizzabile fluidamente su diversi dispositivi senza compromettere l'esperienza utente.

2.2.4 Motore o framework di visualizzazione

La realizzazione efficace di un Digital Twin per la città di Bologna, che integri modelli 3D dettagliati con avanzate informazioni visive, necessita di una scelta accurata riguardo al motore o al framework di visualizzazione da utilizzare. Tale decisione è fondamentale non soltanto per assicurare una rappresentazione grafica fedele e precisa del modello virtuale, ma anche per garantire un'esperienza utente coinvolgente e fluida.

Considerando la diversità delle opzioni disponibili, ciascuna con specifici vantaggi e limitazioni, è cruciale valutare quale strumento si adatti meglio agli obiettivi del progetto, ai bisogni tecnici e alle aspettative di chi interagirà con il sistema.

Esaminando i motori di visualizzazione disponibili, si distinguono particolarmente:

- **Unreal Engine 5**[21]:
  - **Pro**: alta qualità visiva, supporto eccellente per l'illuminazione dinamica e la precisione architettonica, ampie possibilità di interazione e coinvolgimento, vasta disponibilità di plugin per simulazioni, forte supporto per realtà aumentata e virtuale. Si distingue per la qualità grafica elevata e il suo intuitivo sistema di scripting visuale.

  - **Contro**: curva di apprendimento impegnativa, requisiti hardware alti per utilizzare tutte le funzionalità, mancanza di esportazione diretta per ambienti web.

- **Unity**[39]:
  - **Pro**: ampio sostegno dalla community, flessibile e compatibile con un'ampia varietà di piattaforme, buona integrazione con tecnologie AR e VR, accessibilità per i principianti.
  - **Contro**: qualità di rendering e dettagli grafici leggermente inferiori rispetto a UE5 in certi aspetti.

- **Godot**[28]:
  - **Pro**: open source e gratuito, leggero e adattabile a progetti di diverse dimensioni, supporta grafica 3D e 2D.
  - **Contro**: community molto più ridotta rispetto ad altri motori, con conseguenti minori risorse e supporto, limitazioni nelle funzionalità avanzate di rendering a confronto di Unity e UE5.

- **WebGL e Three.js**[79][77]:
  - **Pro**: compatibilità nativa con il web, accessibile senza l'installazione di software aggiuntivi, Three.js facilita lo sviluppo con API di alto livello.
  - **Contro**: prestazioni dipendenti dalle capacità del browser e del dispositivo, limitazioni significative nei dettagli grafici e nelle funzionalità avanzate rispetto ai motori per desktop.

La selezione del motore di visualizzazione più appropriato richiede l'analisi di vari aspetti, inclusi il livello di qualità grafica richiesto, la complessità del modello, le opzioni interattive necessarie e l'accessibilità per l'utente finale.

Altri elementi, come il supporto della community, la disponibilità di documentazione e la sostenibilità del progetto a lungo termine, sono altresì decisivi nella scelta della piattaforma più adatta per lo sviluppo del Digital Twin.

2.2.5 Rendere vivo il Digital Twin

Nel momento in cui avremo completato la modellazione della città, l'attenzione si concentrerà sull'interattività del Digital Twin e sull'integrazione di

dati in tempo reale. Questo elemento è cruciale poiché un Digital Twin che non rifletta dinamicamente i cambiamenti esterni mediante dati aggiornati non può essere considerato pienamente funzionale.

Identificare e analizzare le diverse fonti di dati che alimentano il Digital Twin diventa quindi prioritario. Questo include **sensori IoT** posizionati in vari punti della città, dati provenienti dai sistemi di gestione del traffico, informazioni ambientali e, potenzialmente, tanto altro. La scelta delle fonti di dati sarà guidata dagli obiettivi specifici del Digital Twin e dalle aree di interesse prioritario.

Successivamente, si presenta la sfida tecnologica dell'integrazione di queste fonti nel modello 3D. L'utilizzo di tecnologie come **RestAPI** per il trasferimento di dati diventa fondamentale in questo contesto. La modalità di rappresentazione di tali dati all'interno del Digital Twin varia a seconda della natura delle informazioni. Ad esempio, per dati parametrici quali l'intensità del traffico, potrebbero essere impiegati indicatori visivi come variazioni cromatiche; per l'integrazione di elementi fisici nuovi, come la piantumazione di un albero, sarebbe necessario inserire un nuovo modello 3D specifico all'interno del Digital Twin.

L'architettura del sistema di visualizzazione necessita di una progettazione attenta per assicurare la sua flessibilità e scalabilità, specialmente per quanto riguarda l'integrazione e la visualizzazione di dati dinamici. Un approccio modulare, unito all'adozione di interfacce (standardizzate per le varie fonti) e tipologie di dati, diventa cruciale per il successo del progetto. Tale strategia non solo semplifica l'aggiunta di nuovi flussi di dati ma garantisce anche che il Digital Twin possa essere aggiornato e adattato a futuri sviluppi senza la necessità di ristrutturazioni profonde dell'infrastruttura esistente.

2.3 Workflow e Strumenti

Dopo un'approfondita analisi delle sfide incontrate, questa sezione illustra sinteticamente il flusso di lavoro adottato nello sviluppo del progetto di tesi relativo al Digital Twin della città di Bologna (fig. 2.10).

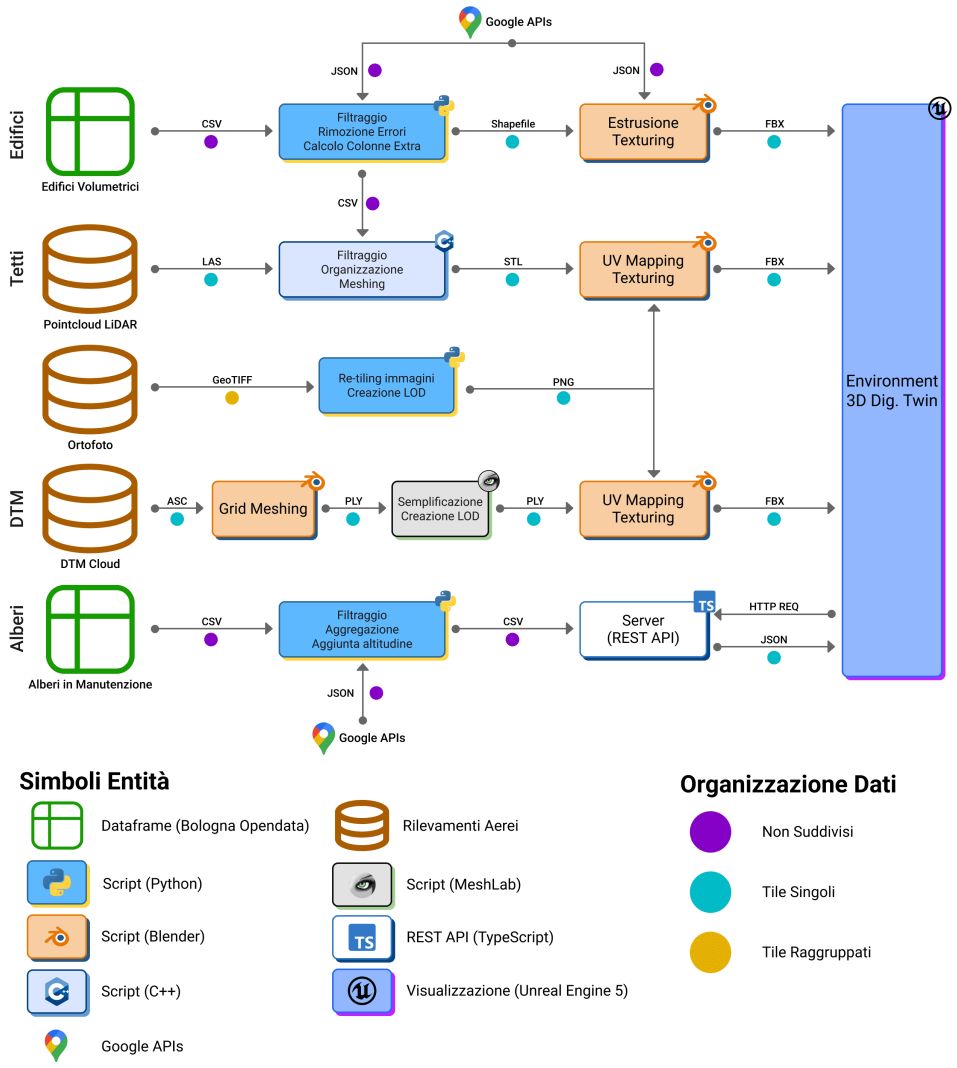


Figura 2.10: Diagramma di workflow completo

Come illustrato dall'immagine di riferimento, per ciascun tipo di dato è stata definita una pipeline specifica, operante in maniera indipendente dalle altre. Ogni pipeline inizia con l'elaborazione dei dati grezzi e si conclude con la conversione di questi ultimi nel formato comune FBX[51], per poi essere

integrati nell'ambiente 3D del Digital Twin.

Ecco una breve descrizione delle diverse pipeline implementate:

- **Edifici**: partendo dai dati catastali forniti dal Comune di Bologna, si procede alla costruzione delle mesh e successivamente al texturing utilizzando Blender[33].
- **Nuvola di punti LiDAR**: utilizzata per ricavare dettagli sui tetti e per la costruzione del modello 3D degli edifici. Il processo di texturing viene anch'esso realizzato in Blender.
- **Ortofoto**: mediante script Python[71], le ortofoto vengono sezionate seguendo lo standard dei tile 500x500. Questi frammenti sono poi impiegati per il texturing dei tetti e delle mesh del terreno.
- **Modello del terreno**: le mesh vengono semplificate tramite Meshlab[16], mentre in Blender si costruiscono i vari livelli di dettaglio (LOD)[63] e conseguentemente vengono applicate le texture adeguate.
- **Alberi**: i dati forniti dal comune vengono utilizzati per creare un servizio Cloud che invia informazioni in tempo reale sugli alberi al DT.

Tutto ciò termina con Unreal Engine 5, che ci permetterà di visualizzare i dati generati e di gestirli al meglio.

Questo schema offre una panoramica completa dei vari compiti eseguiti per realizzare un prototipo funzionale del Digital Twin di Bologna, evidenziando l'approccio metodologico e le tecnologie impiegate per ogni fase del progetto.

2.4 Tool e Software utilizzati

Mostriamo quindi brevemente gli strumenti e i software utilizzati nel nostro processo:

- **Blender**[33] (fig. 2.11): un software open source dedicato alla realizzazione di contenuti in 3D, offrendo un insieme completo di strumenti per la pipeline di creazione dei modelli, includendo modellazione, texturing, animazione, simulazione, rendering, compositing e motion tracking.

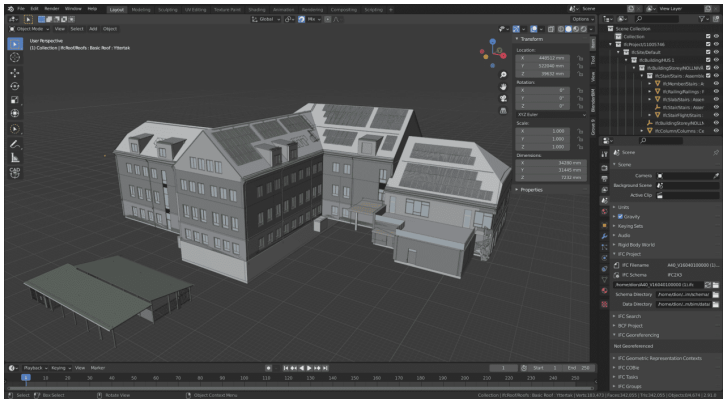


Figura 2.11: Software Blender

- **Meshlab**[16] (fig. 2.12): anch'esso open source, è un software essenziale per la manipolazione e l'editing di mesh 3D, perfetta per operazioni di pulizia, semplificazione e ricostruzione delle mesh.

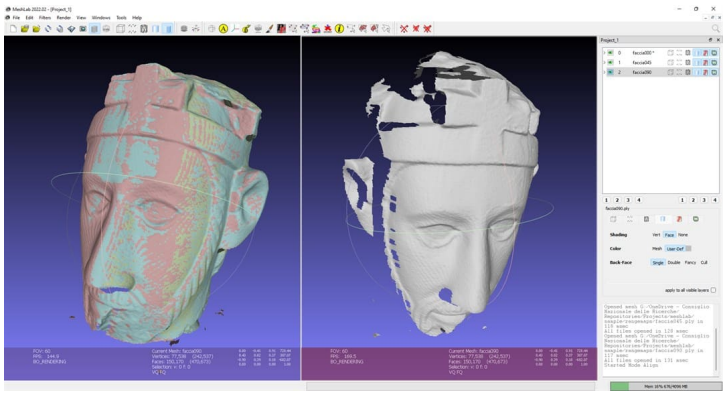


Figura 2.12: Software Meshlab

- **Unreal Engine 5**[21] (fig. 2.13): che è stato scelto per le possibilità che permette di esplorare, unendo una visualizzazione realistica alle alte prestazioni. Più in dettaglio, Unreal Engine è riconosciuto per le sue capacità avanzate di rendering in tempo reale, che consentono di produrre immagini di alta risoluzione e ricche di dettagli, fondamentali per descrivere con precisione la complessità visuale degli spazi urbani. La sua struttura modulabile e l'abilità di gestire ampi ambienti open-world lo rendono ideale per la creazione di modelli di aree urbane vaste e dettagliate. In aggiunta, Unreal Engine offre una vasta gamma di strumenti, inclusi il sistema di scripting visuale *Blueprint* e la programmazione in C++, che facilitano lo sviluppo di logiche complesse e l'implementazione di interazioni dinamiche nel DT.

Figura 2.13: Software Unreal Engine 5

Un altro software che abbiamo utilizzato solo per la mera visualizzazione di dati durante le fasi di progettazione è **Cloud Compare**[24], un software open source progettato per la gestione e il processamento di nuvole di punti 3D e modelli mesh. Esso è risultato particolarmente comodo per vedere le relazioni tra i nostri dati LiDAR e altri formati geo-localizzati.

Capitolo 3

Elaborazione dei dati grezzi

Contents

3.1 Tiling Ortofoto e creazione textures

Come mostrato gia nell'introduzione ai dati di partenza, in figura 2.7, le ortofoto sono al momento disposte in uno standard articolato e difficilmente utilizzabili per gli scopi di texturing che ci siamo posti. La sfida principale risiede nel rendere questi dati perfettamente sovrapponibili ai tile del modello 3D, in modo da poterli utilizzare senza difficoltà come texture nei vari Livelli di Dettaglio (LOD) del modello.

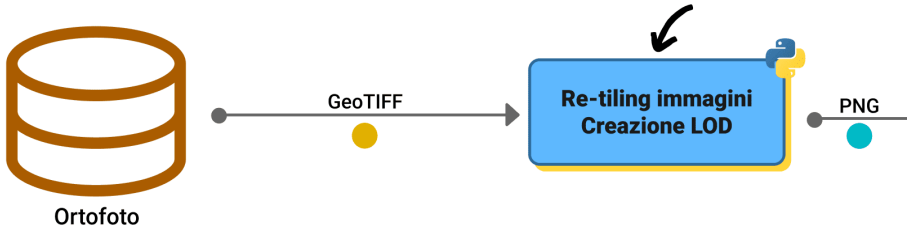


Figura 3.1: Sezione workflow: ortofoto e textures

Per superare questa sfida, abbiamo sviluppato uno script Python dedicato (fig. 3.1), che sfrutta la libreria *OSGeo*[23], per trasformare le ortofoto iniziali in immagini a risoluzioni variabili, ottimizzate per il texturing dei nostri LOD. Questo script rappresenta un ponte cruciale tra i dati grezzi e la loro applicazione pratica nel contesto del Digital Twin, garantendo che ogni texture si allinei perfettamente con le corrispondenti mesh 3D.

3.1.1 Ritaglio

Ognuna delle quattro immagini .tif, disposta come mostrato in figura 2.7, è a sua volta composta da 4 tile 500x500 m (fig. 3.2).

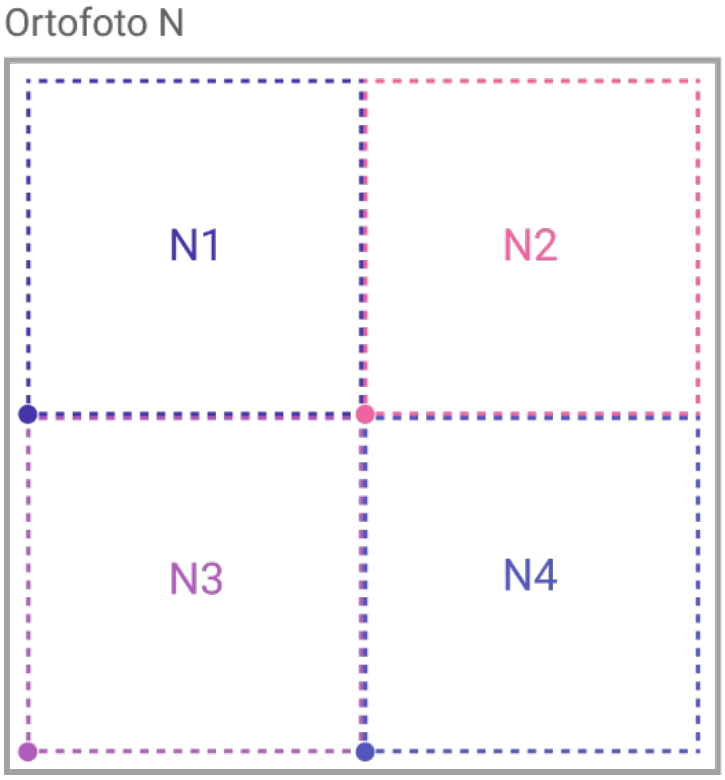


Figura 3.2: Visualizzazione separazione interna ortofoto

Il processo inizia con l'identificazione precisa delle coordinate del Lower Left Corner (LLC) di ogni file, un passaggio fondamentale per garantire l'allineamento accurato dei dati geospaziali. Utilizzando le informazioni georeferenziate e i metadati integrati nei file GeoTiff, il nostro script procede poi alla suddivisione delle immagini in tile, seguendo un approccio metodico che tiene conto delle coordinate specifiche e dei valori extra forniti dai dati.

Garantiamo così che ogni texture generata sia perfettamente dimensionata e posizionata, rendendo il processo di texturing non solo più efficiente ma anche più accurato. Anche in questo caso il naming dei file dipende dalle coordinare dell'LLC della texture creata.

3.1.2 LOD texture

Una volta avvenuta la suddivisione ci occupiamo di generare le texture. Lo script permette di modificare la lista delle risoluzioni che si vogliono avere in output e in base a ciò salva l'immagine nella cartella apposita, seguendo la stessa convenzione di naming utilizzata per tutti gli altri dati basata sull'LLC del tile che rappresenta.

Nel dettaglio, abbiamo optato per generare texture con risoluzioni di **4k** (4096x4096 pixel), **2k** (2048x2048 pixel), **512** (512x512 pixel) e **256** (256x256 pixel). Questa scelta di risoluzioni diverse risponde alla necessità di bilanciare qualità visiva e prestazioni del sistema, consentendo al Digital Twin di essere visualizzato efficacemente su diverse piattaforme e dispositivi.

Attraverso questo approccio, possiamo assicurare che ogni LOD del DTM disponga della texture più adatta, ottimizzando la rappresentazione visiva del territorio a vari gradi di zoom e dettaglio.

Queste texture verranno inoltre utilizzate per il texturing dei tetti nel capitolo dedicato.

3.2 Digital Terrain Model: DTM

Un **DTM** (Digital Terrain Model) è un modello digitale che rappresenta la topografia di una porzione di superficie terrestre escludendo tutti gli elementi sovrastanti, come alberi e costruzioni. Esso utilizza punti, che possono essere ordinati/organizzati o meno, con coordinate geografiche e altitudini per creare una rappresentazione tridimensionale del terreno.

Nel nostro caso, come descritto precedentemente, l'azienda che si è occupata della rilevazione dei dati ha già fornito un insieme di file del DTM in formato ASCII. Il processo di generazione di questi file a partire dai dati LiDAR *raw* può essere un processo complesso che richiede diverse fasi di elaborazione. Basandoci sui file risultanti e sulle informazioni forniteci in un documento tecnico, che non spiega per nulla il processing dei dati, presumiamo che in seguito alle fasi di pre-elaborazione (pulizia) e classificazione, sia stato effettuato un processo di interpolazione e modellazione simile a quello che abbiamo intrapreso per la generazione delle mesh dei nostri tetti.

Per la loro natura, questi dati sono ovviamente poco precisi al di sotto degli edifici. Tuttavia, dato che essi saranno al di sotto dei nostri modelli e che utilizzare i dati *raw* per costruire il nostro DTM da capo sarebbe complesso e richiederebbe molto tempo, abbiamo preferito continuare a utilizzare i dati che ci sono stati forniti.

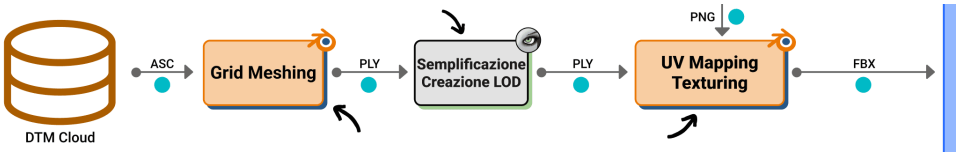


Figura 3.3: Sezione workflow: DTM

La pipeline che si occupa di gestire questi dati (fig. 3.3) inizia con la generazioni di mesh tramite questi punti, quindi avviene uno step di semplificazione (dal quale ci aspettiamo una riduzione di alcune delle lacune presenti nei dati) e infine avviene la fase di texturing.

3.2.1 Ottenimento mesh DTM da file ASCII

Come accennato nel precedente capitolo, i file che ci sono stati forniti sono ti tipo ASCII e ognuno di loro rappresenta un tile 500x500m con una matrice 1000x1000.

Innanzitutto, dobbiamo determinare quale tecnica di meshing sia più adatta a questi dati. Dopo aver considerato diverse tecniche popolari, come la Triangolazione di Delaunay[48], Marching Cubes[65], Poisson Surface Reconstruction[70] e Greedy Projection Triangulation[57], abbiamo optato per un approccio basato sulla 4-connectivity (4C)[69] (fig. 3.4).

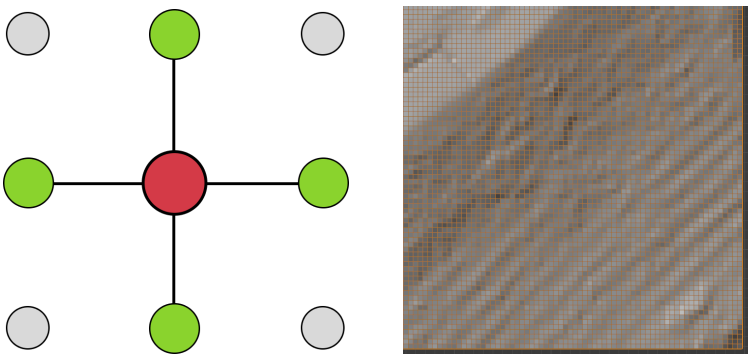


Figura 3.4: Rappresentazione grafica 4-connectivity e dettaglio mesh risultante

Sebbene gli altri metodi producano mesh più dettagliate e accurate, tendono anche a generare un numero elevato di edge e richiedono significative risorse computazionali. Questo elevato dettaglio è meno rilevante per noi poiché miriamo a un equilibrio tra accuratezza e prestazioni per simulazioni e analisi urbane. L'approccio 4C riduce i requisiti computazionali mantenendo una rappresentazione adeguata del terreno, essenziale per il nostro Digital Twin. Ciò garantisce efficienza e gestibilità dei dati senza impattare negativamente sulle funzionalità del modello.

I vantaggi offerti dal nostro approccio sono i seguenti:

- **Velocità di elaborazione**: l'uso di un metodo così semplice può ridurre significativamente i tempi di elaborazione, soprattutto quando si deve lavorare con dataset di grandi dimensioni.
- **Semplificazione successiva**: avendo di base 1 milione di punti per ogni tile, il volume dell'intero DTM del Comune di Bologna (654 tile)

raggiunge quota 654 milioni. Considerando un livello di connettività basso come la 4C, verrebbero prodotti oltre 855 bilioni di edge, che come si può intuire è un numero fin troppo alto per rappresentare solo il terreno del nostro DT. Per questo motivo reputiamo essenziale uno step di semplificazione e di gerarchia di livelli di dettaglio (LOD) per la rappresentazione di queste mesh. Poiché la semplificazione è mirata a ridurre la complessità della mesh mantenendo la fedeltà al modello, iniziare con una struttura meno complessa faciliterebbe questo passaggio impattando poco la qualità finale.

- **Gestione dei dati**: dato che la nostra è una point cloud già organizzata in una matrice regolare, l'approccio basato sulla connettività sfrutta tale struttura aiutando a mantenere un buon equilibrio tra dettaglio geometrico e gestibilità dei dati.

Da qualche veloce test su piccola scala, abbiamo inoltre notato come partendo dagli stessi punti, applicando diversi metodi di meshing e applicando infine lo stesso tipo e livello di semplificazione, le mesh ottenute erano molto simili tra loro e diventavano quasi indistinguibili una volta applicata la texture ad esse.

Dato che il primo punto è l'LLC (Lower Left Corner) del tile, ed è colui che fornisce il nome al file, i punti appartenenti a questa matrice coprono un area effettiva di 495,5x495,5m in quanto il 1001esimo punto appartiene ai tile adiacenti. Per questo motivo, se facessimo meshing dei punti allo stato attuale, troveremmo un buco di mezzo metro tra un tile e l'altro (fig. 3.5).

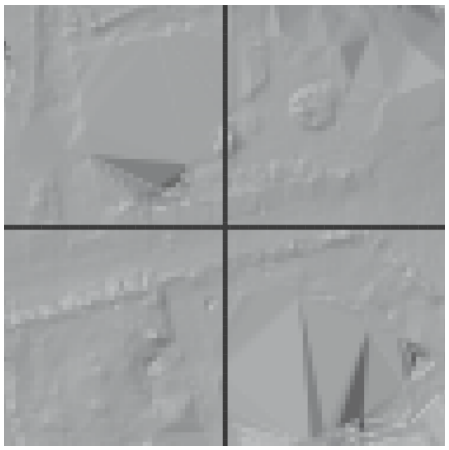


Figura 3.5: Visualizzazione divario tra tile contigui

Per risolvere questo problema, la nostra soluzione prevede l'aggiustamento dei dati prima del processo di meshing. Specificamente, integriamo la prima colonna di dati dal tile immediatamente a destra di quello in analisi per colmare il gap sul lato destro e l'ultima riga dal tile direttamente sopra per chiudere il vuoto nella parte superiore. Questo metodo di "pre-elaborazione" dei dati assicura una continuità senza interruzioni tra i tile adiacenti, garantendo continuità nel modello finale e mantenendo contemporaneamente la precisione e l'integrità del DTM (fig. 3.6).

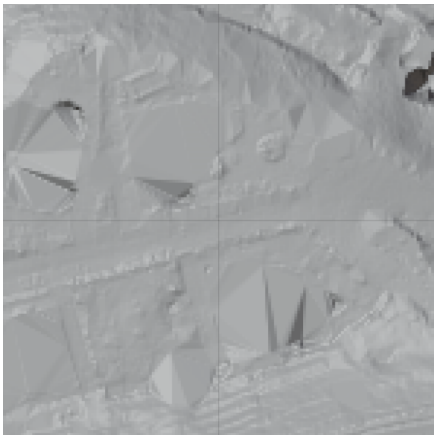


Figura 3.6: Visualizzazione seamless

Tutto ciò è stato realizzato in Python come estensione Blender, automatizzando il processo di creazione di terreni. Le mesh risultanti, che rimangono separate per tile, sono quindi già visualizzabili all'interno dell'ambiente 3D (fig. 3.7). Viene quindi resa possibile l'esportazione di esse in formato PLY per passare al seguente step di semplificazine.

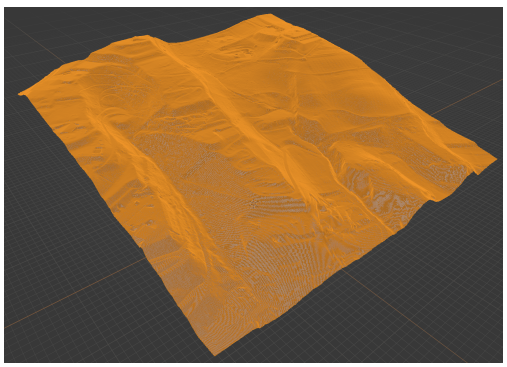


Figura 3.7: Visualizzazione mesh completa

3.2.2 Riduzione mesh per costruire LOD

Per quanto riguarda la riduzione del numero di poligoni della mesh, vogliamo ottenere risultati con il minor impatto possibile sulla sua apparenza esteriore. Per questo motivo abbiamo adottato il metodo **Quadric Edge Collapse Decimation**[22]. Questa tecnica di semplificazione è infatti ampiamente utilizzata per ridurre il numero di vertici e poligoni mantenendo la forma generale e le caratteristiche della mesh originale, specialmente utile per i nostri scopi dato che vogliamo creare vari livelli di dettaglio per rappresentare la nostra mesh al meglio.

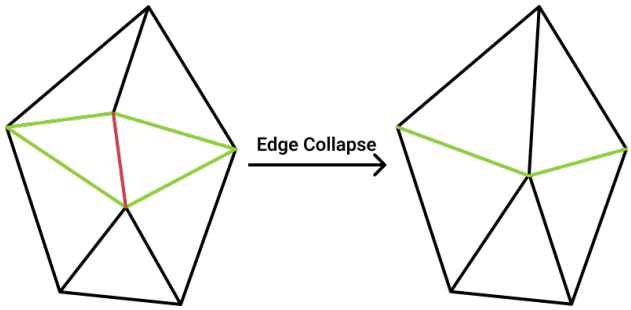


Figura 3.8: Esempio di Edge Collapse

Inoltre è particolarmente potente perché si adatta bene a mesh di qualsiasi complessità e può essere facilmente controllata dall'utente, che può decidere il grado di semplificazione desiderato. In base al target di edge dato in input (fig. 3.8), essa cerca di preservare la mesh originale cercando di minimizzare il criterio di costo di collasso, appunto calcolato da una funzione quadratica che misura l'errore geometrico introdotto dalla rimozione di quell'edge.

Questo tipo di semplificazione è presente nel software **Meshlab**, perciò possiamo scrivere uno script *Python* che sfrutta il package *pymeshlab* per automatizzare il processo.

Tuttavia è importante sottolineare come questa tecnica possa causare problemi se non usata con accortezza, soprattutto per quanto riguarda i bordi della mesh che potrebbero diventare incoerenti con i tile adiacenti o addirittura bucati. Preservare inalterati i bordi è cruciale per mantenere la continuità del modello, ma può limitare l'efficacia della semplificazione.

Per questo motivo abbiamo fatto alcuni test iterativi cercando di bilanciare precisione e qualità della semplificazione. La decisione derivata da essi

è stata di creare 3 differenti livelli di dettaglio (LOD):

- **LOD0** (fig. 3.9): 50k edge, senza diminuire i bordi

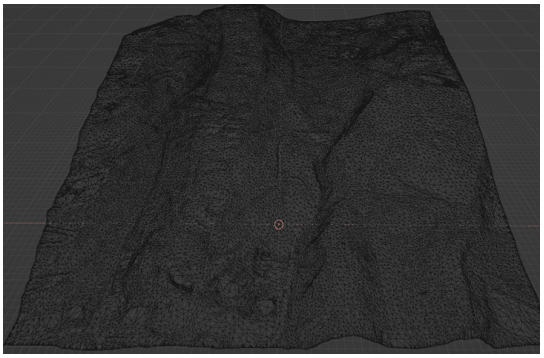


Figura 3.9: Visualizzazione risultato LOD0

- **LOD1** (fig. 3.10): 16k edge, diminuendo i bordi a 1600 edge

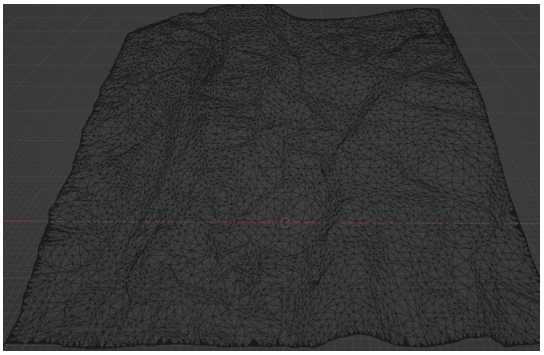


Figura 3.10: Visualizzazione risultato LOD1

- **LOD2** (fig. 3.11): 5k edge, diminuendo i bordi a 400 edge

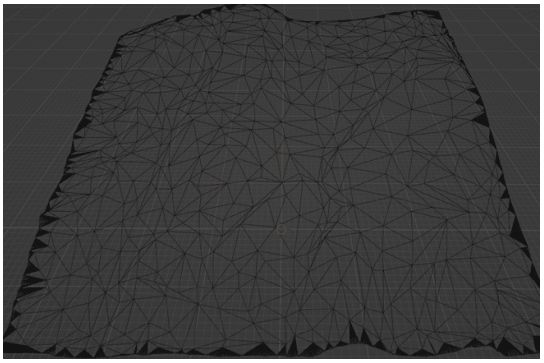


Figura 3.11: Visualizzazione risultato LOD2

Nel nostro caso, il formato di export utilizzato è rimasto il PLY.

3.2.3 Texturing DTM

Le mesh ottenute vengono ora utilizzate insieme alle textures precedentemente preparate per generare oggetti 3D avanzati che possono contenere livelli di dettaglio multipli. Per far questo ci avvaliamo di Blender e del formato **FBX**, uno standard molto diffuso per la rappresentazione di mesh 3D che può contenere non solo i dati riguardanti la geometria ma anche materiali/texture, skeleton, animazioni e potenzialmente altri dati extra (metadati).

Il processo di creazione di questi file viene fatto da uno script in Python che sfrutta le API di Blender per automatizzare completamente il processo. Infatti, una volta forniti i percorsi alle cartelle dei file, il codice esegue per ogni tile i seguenti passaggi:

- **importare** i vari livelli di dettaglio.
- generare l'**UV mapping**[78] utilizzando il metodo di proiezione dopo aver impostato la camera al di sopra di esso in modalità ortografica.
- crea i **materiali** utilizzando Principled BSDF e assegna le diverse **texture** in base al livello di dettaglio: 2k (LOD0), 512 (LOD1) e 254 (LOD2).
- **esporta** in gruppo i vari livelli di dettaglio nel formato desiderato.

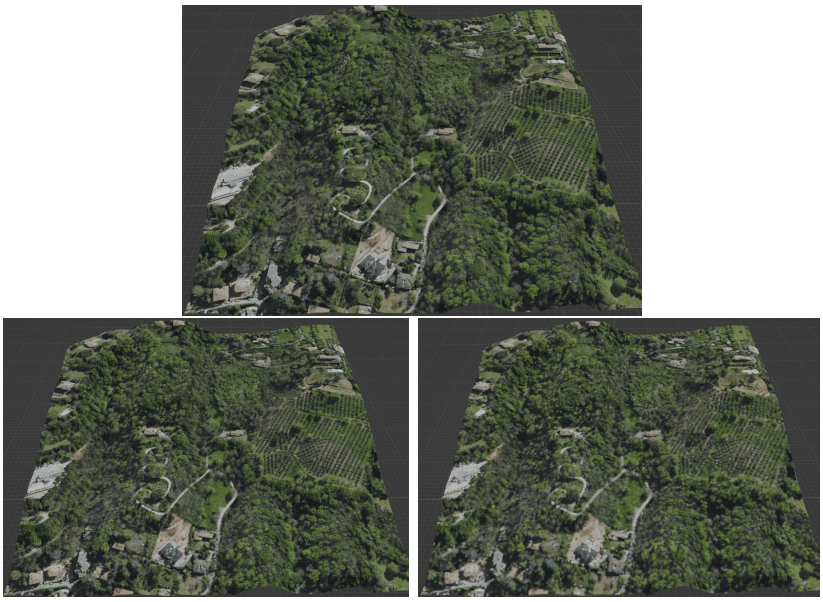


Figura 3.12: Visualizzazione LOD risultanti

3.3 Edifici

Vengono qui presentati brevemente i risultati ottenuti dal mio collega per ciò che riguarda la creazione delle mesh degli edifici e il relativo texturing.

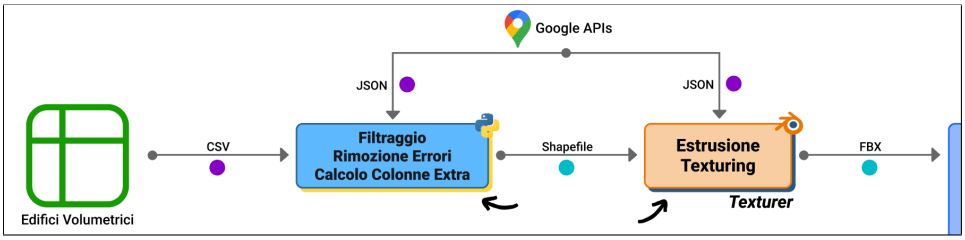


Figura 3.13: Sezione workflow: edifici

La pipeline di processing (fig. 3.13) di questi dati inizia con un accurato filtraggio e modifica dei dati degli edifici. Per questa prima sezione sono stati anche utilizzate le API di Google[15] che ci hanno permesso di correggere alcuni degli errori presenti nel dataset, infatti circa 400 edifici risultavano avere una quota di piede allo zero assoluto, posizionandoli al di sotto del terreno.

Gli shapefile risultanti ci hanno quindi permesso di ottenere le relative mesh tridimensionali su Blender dove, insieme ad altri dati ottenibili tramite le API di Google, è stata automatizzata una fase di texturing e di generazione del corrispettivo atlas. Ne vediamo i risultati in figura 3.14.

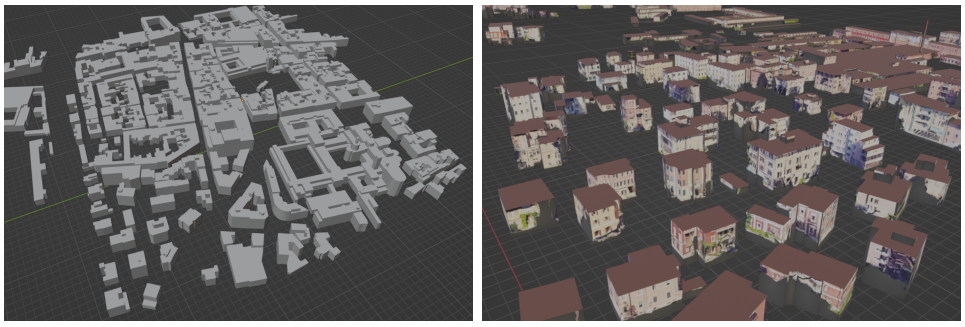


Figura 3.14: Risultati edifici

3.4 Tetti: meshing Point Cloud

In questa sezione del nostro workflow intendiamo creare delle mesh con texture che aiutino non solo ad avere una migliore visualizzazione della città, ma possano anche (in futuro) essere utilizzate per fare delle proiezioni accurate per quanto riguarda le superfici utili dei tetti.

Infatti, uno degli obiettivi del DT della sezione *Energia* comprende la possibilità di poter fare delle proiezioni, in base alle superfici di tetto utilizzabili, su quanti pannelli solari sia possbile installare. Di conseguenza, verrebbe permesso di prevedere la loro produzione di energia media in base alla loro esposizione, agli edifici e agli alberi intorno ad essi e alle previsioni delle condizioni meteo basandosi sui dati storici.

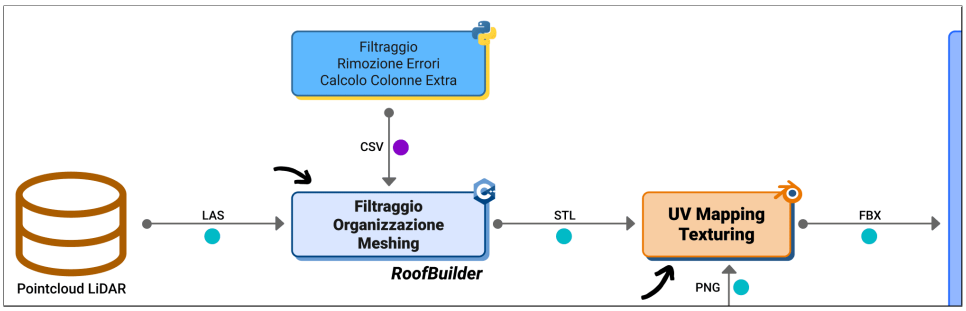


Figura 3.15: Sezione workflow: tetti

Come è possibile vedere in figura 3.15, la pipeline inizia con la sezione di meshing da parte del nostro software nominato **RoofBuilder**, che utizza contemporaneamente i dati LiDAR e una versione creata ad-hoc dei dati riguardanti gli edifici. In seguito le mesh risultanti sono state texturizzate su Blender utilizzando le immagini precedentemente ottenute (capitolo 3.1). Ne risultano dei modelli in formato FBX utilizzabili su qualsiasi piattaforma.

Prima di delineare il nostro algoritmo di meshing, abbiamo iniziato con un analisi più approfondita dei nostri dati. Elenchiamo qui le nostre più importanti considerazioni derivate dalla analisi:

- i **dati fornitici** che possono darci informazioni a riguardo comprendono unicamente gli edifici volumetrici, le Point Cloud LiDAR e le ortofoto. Per motivi di tempo e complessità abbiamo deciso di fare **focus sulle prime due fonti** di informazioni. Riteniamo che sia possibile introdurre l'utilizzo delle ortofoto all'interno di esso per generare mesh più accurate, anche se ciò aumenterebbe al contempo la complessità del processo, l'overhead e i tempi di calcolo.

- come si può notare in figura 3.16, in cui avviene la sovrapposizione dei tra GeoShape degli edifici (contorni bianchi) e la nuvola LiDAR, la poca precisione dei dati dei GeoShape è notabile dal un **disallineamento**, sia in traslazione che in rotazione, rispetto ai punti della nuvola LiDAR. Osservando una parte dati, ci aspettiamo che questo la distanza tra i dati possa arrivare **fino a circa 1,5m** e varia da edificio a edificio. Questo è un peccato dato che un allineamento e una sovrapposizione migliori avrebbero permesso di sviluppare un metodo di procesazione dei dati più semplice e, potenzialmente, ottenere risultati migliori.

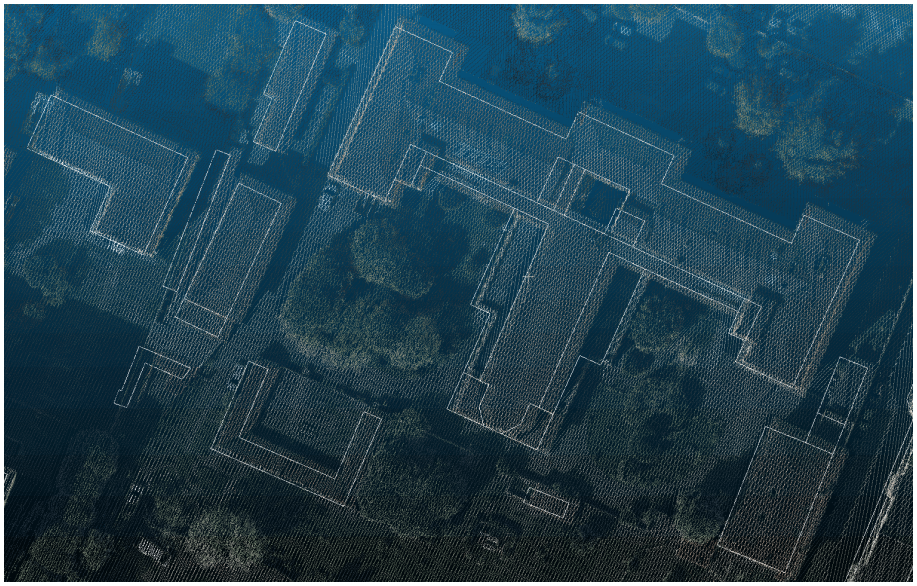


Figura 3.16: Sovrapposizione dati LiDAR e GeoShape (CloudCompare)

- le **tipologie di tetti** in Italia sono molteplici e disparate, tra le più comuni possiamo trovare quelli a due e quattro falde, tetti a mansarda (due falde con colmo non centrato per avere sottotetti abitabili), tetti a falda unica (edifici industriali e capannoni), tetti piani/terrazzati. In particolare, a Bologna troviamo anche alcune cupole di varie tipologie, per esempio emisferiche o geodetiche (PalaDozza). Questa ampia scelta viene ampliata drasticamente dal fatto che molti tetti sono **composizioni di queste forme basiche**, dando nascita a tetti unici e di forme particolarmente **complesse** (fig. 3.17).

Figura 3.17: Confronto GeoShape (blu) e tetto sovrastante (verde)

- il riconoscimento di questi tetti viene ulteriormente complicato dalla **aggregazione di edifici per altezza** che si può notare all'interno degli Edifici Volumetrici (figura 3.18). Questo avviene soprattutto in centro dove molti edifici avendo una quota di gronda più o meno si-

mile vengono aggregati in un unico volume. In questa figura è inoltre possibile notare come alcuni dei cortili interni sono stati correttamente inseriti creando dei fori negli shape, mentre altri non sono stati registrati in alcun modo. Analizzando il file abbiamo notato che questo problema non accade solo in centro ed è presente per cortili sia di piccole che di grandi dimensioni, rendendoli poco consistenti.

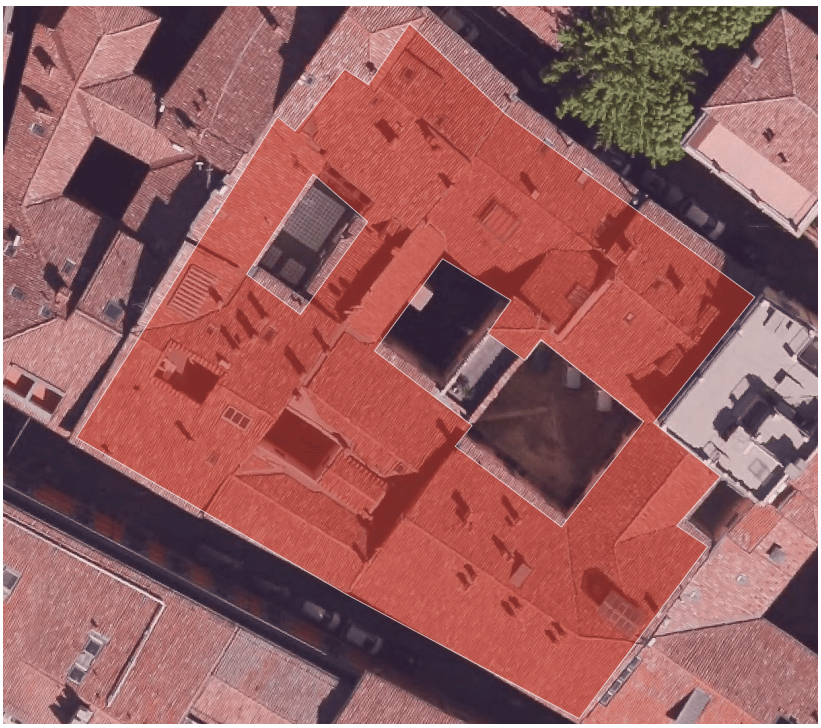


Figura 3.18: Esempio aggregazione tetti

- un indicatore della possibile complessità del tetto può essere banalmente il **numero di vertici** che definiscono il **GeoShape** dell'edificio. Abbiamo quindi analizzato questo dato all'interno del nostro CSV, già filtrato di dati a noi non utili come precedentemente descritto, trovando risultati che confermano le nostre considerazioni. Il grafico ad albero in figura 3.19 mostra con chiarezza quanto siano rari gli edifici a 4 vertici (0,1%), preannunciando in questo modo quanto siano pochi i casi considerabili come "semplici". Anche se la maggior parte degli edifici ha tra i 5 e i 10 vertici, ciò non preclude che questi possano avere forme particolarmente complicate. Sono infine presenti, anche se pochi, tetti che superano le centinaia di punti.

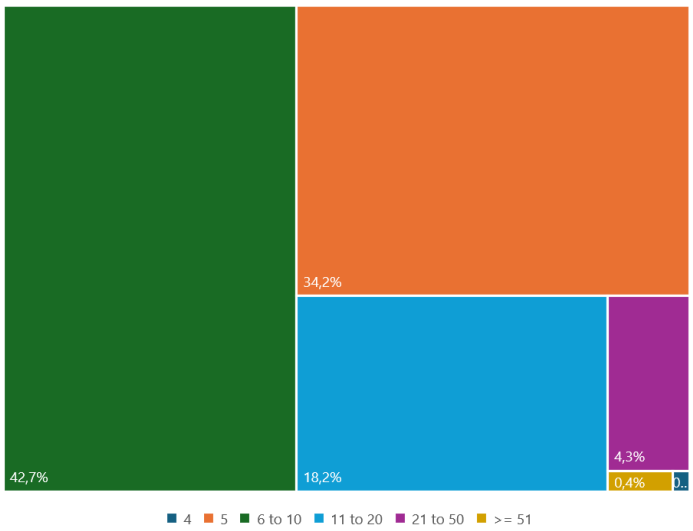


Figura 3.19: Grafico distribuzione numero vertici dataset GeoShape

- l'ultima considerazione riguarda la **poca precisione** per quanto riguarda sia le **quote di piede e di gronda** degli edifici (dal sito *Opendata* riportata avere una tolleranza di 54 cm). Questo va sommato al fatto che, dalle nostre osservazioni, per gli edifici dal tetto piano la quota di gronda sembra essere talvolta presa al livello del piano del tetto e altre volte a quello del parapetto che lo circonda.

Questa analisi dei dati ci ha dato una buona idea sulle condizioni nelle quali dovrà eseguire il nostro algoritmo. Sulla base di ciò, il nostro intento è stato quello di creare una procedura che generalizzi il più possibile il problema, in modo da poter trattare il numero più alto di casi possibile.

Strategie inizialmente pensate che sfruttavano tecniche di straight skeletoning sugli shape sono state scartate dato che qualche veloce test aveva dimostrato quanto esse potessero essere sia poco controllabili che spesso lontane dalla realtà.

In breve, la metodologia di processing pensata filtra i punti della nuvola non organizzata appartenenti ad un edificio e li utilizza per genera una versione organizzata in griglia. È quindi possibile trattare il problema con tecniche di Image Processing e Computer Vision allo scopo di trovare i punti di controllo che ci possano permettere di generare la Mesh. Infine, la triangolazione di essa avviene tramite la triangolazione di Delaunay vincolata dal poligono esterno.

3.4.1 Introduzione colonne utili al filtraggio

Siamo partiti dal chiederci come potessimo creare un sistema efficace per il filtraggio dei punti appartenenti alle nuvole LiDAR considerabili come tetti. Abbiamo quindi deciso di creare delle bounding box utilizzando i dati riguardanti gli edifici a nostra disposizione. Questa fase è stata realizzata in **Python**.

Come visto in analisi, anche se disallineati, abbiamo un match tra i nostri GeoShape e le nuvole di punti. Per ovviare a questo fatto abbiamo generato dei nuovi poligoni a uno specifico offset (2,5 m), così da andare incontro sia al disallineamento che al normale sbalzo che in genere hanno i tetti rispetto alle mura dell'edificio (fig. 3.20). Essi sono stati inseriti in una nuova colonna denominata "Geo Shape UTM_OFF".

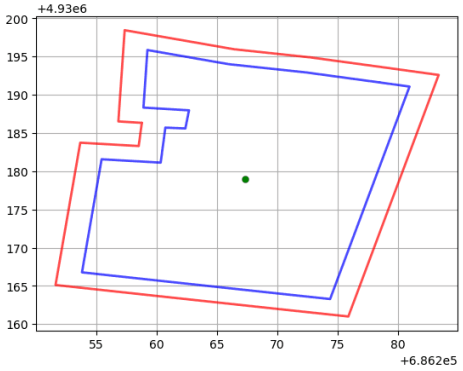


Figura 3.20: Esempio di GeoShape (blu) con il relativo offset (rosso)

Ora che abbiamo creato delle aree di delimitazione per gli assi orizzontali dobbiamo occuparci dell'asse verticale. Potremmo banalmente prendere ogni punto al di sopra della quota di gronda, tuttavia questo può causare un certo numero di problemi in casi come:

- alberi che sovrastano un edificio
- antenne e camini particolarmente alti
- balconi sopra tetti terrazzati
- tetti al di sopra di altri tetti

Per cercare di limitare i problemi che possono comparire in queste casistiche abbiamo introdotto una tolleranza che stima una quanto possa essere alto il tetto basandosi sul poligono appena creato.

In generale l'inclinazione dei tetti può variare ampiamente in base a molti fattori, per semplicità abbiamo considerato una pendenza media del 30% e calcolato la tolleranza con il lato più lungo di ogni edificio. Per i casi speciali in cui si hanno molti e piccoli lati (tetti dal bordo circolare/ovale) in cui questa stima non può funzionare, abbiamo invece calcolato la tolleranza utilizzando la stessa pendenza sulla diagonale maggiore dell'inviluppo convesso dell'edificio.

Queste tolleranze sono quindi state inserite in una nuova colonna nominata "Toll_Tetto".

Per testare i risultati derivanti da questo tipo di filtraggio era stato sviluppato uno script in **Rust**[18] che eliminava dalla nuvola tutti i punti che non appartenevano alla bounding box di nessun edificio.

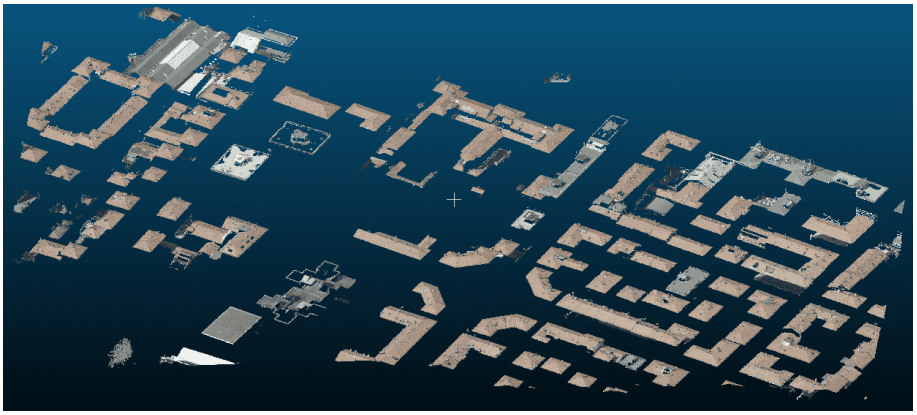


Figura 3.21: Risultati test filtraggio tetti (CloudCompare)

Come è possibile vedere nella figura 3.21 otteniamo un isolamento dei punti dei tetti generalmente buono, con inconsistenze solamente sui tetti piani per i motivi precedentemente menzionati.

Le colonne utili a questo filtraggio sono:

- "CODICE_OGGETTO"
- "QUOTA_GRONDA"
- "Geo Shape UTM_OFF"
- "Toll_Tetto"
- "Tiles"

Di conseguenza è stato creato un file contenente solamente queste colonne per trattare con più comodità questi dati.

3.4.2 RoofBuilder

L'architettura di RoofBuilder è stata creata con l'intento di verificare la fattibilità della creazione delle mesh e le possibilità che questo tipo di algoritmo può offrire. Per motivi di tempo non è stato ottimizzato ogni singolo aspetto del nostro codice, tuttavia riporteremo in alcuni casi come lo avremmo corretto/modificato tempo permettendo.

Avevamo inizialmente valutato l'utilizzo del linguaggio di programmazione *Rust*, continuando quindi la fase di filtraggio precedentemente utilizzato. Tuttavia, dopo una più attenta analisi delle necessità del nostro metodo, ci siamo resi conto che esso non forniva, data la poca maturità delle sue librerie di terze parti, molti dei tool che avremmo necessitato utilizzare. Per esso sono spesso presenti alcuni porting di librerie sviluppate in altri linguaggi, tuttavia sono in buona parte parziali e poco aggiornati.

Abbiamo perciò deciso di passare a **C++** che ci permette di sfruttare librerie quali:

- **OpenCV**[12] per l'Image Processing, i task di Computer Vision e la visualizzazione.
- **PDAL**[30] per il processing di PointCloud.
- **FLANN** per la ricerca di punti. Nel nostro caso è stata utilizzata la versione header-only **nanoflann**[29].
- **Triangle**[35] per la triangolazione con il metodo di Delaunay vincolato.

L'architettura del software creato è la seguente:

- **MyPoint** è una classe che rappresenta un punto nello spazio tridimensionale. È inoltre presente la versione bidimensionale di questa classe rappresentata con il nome **MyPoint2**.
- **MyTriangle** è una classe che rappresenta un triangolo come una tripletta di MyPoint. È inoltre presente la versione bidimensionale, **MyTriangle2**, che utilizza come punti *MyPoint2*.
- **MyMesh** è una classe che rappresenta una mesh come un vettore di *MyTriangle*.
- **PointCloud** è una classe che ospita un vettore di *MyPoint* ed espone metodi utili per la KDTreeSearch[60].

Figura 3.22: UML Geometria

- **Readers\ReaderCsv** si occupa di leggere i file CSV.
- **Readers\ReaderLas** si occupa di leggere i file LAS immagazzinando i dati nelle nostre strutture.
- **Writer** si occupa di scrivere file.
- **Exporter** permette di scrivere su file le Mesh, al momento in formato STL.

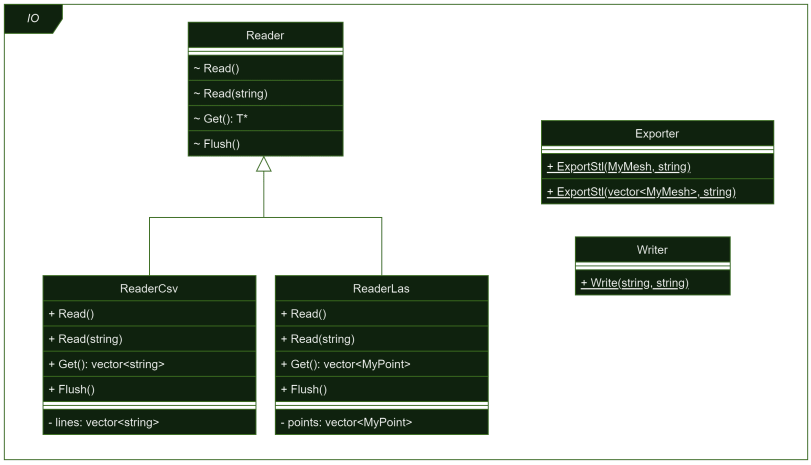


Figura 3.23: UML Input-Output

- **Building** è una factory che costruisce gli oggetti *Building* a partire da una stringa che li rappresenta in formato CSV.
- **Grid** gestisce il passaggio delle nuvole da unorganized a organized, fa ciò tramite la creazione di uno spazio di coordinate locali e quindi tramite Nearest Neighbour Search utilizzando *nanoflann* per ottimizzare la ricerca. Essa permette poi anche di utilizzare il procedimento inverso per trovare il punto globale corrispondente al punto locale voluto.
- **Dbscan** si occupa del clustering dei punti utilizzando *nanoflann*, è stato utilizzato un adattamento dell'implementazione presente su *GitHub*[19].
- **TriangleWrap** è un wrapper per la libreria C *Triangle*.

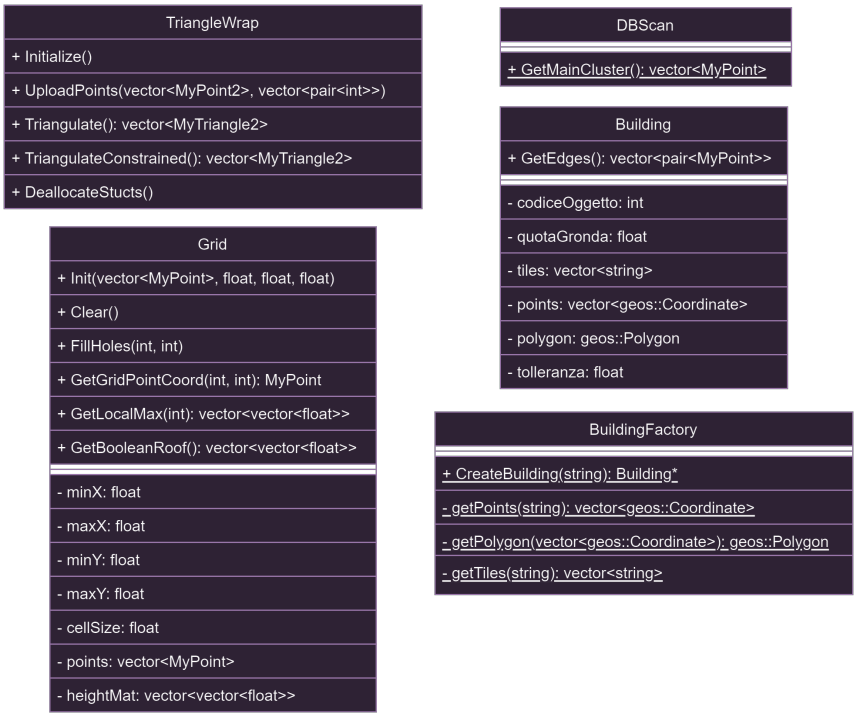


Figura 3.24: UML Utility

- **ImageProcessing\ImageProcessingUnit** implementa tutte le unità di calcolo utilizzate per il processing delle immagini, utilizzando *OpenCV*.

- **ImageProcessing\UtilsCV** fornisce (oltre a metodi per la visualizzazione) il nostro **ImageProcesser** che si occupa, con la relativa factory, di gestire la creazione e il calcolo delle pipeline di processing di *OpenCV* utilizzando le nostre *ImageProcessingUnit*.

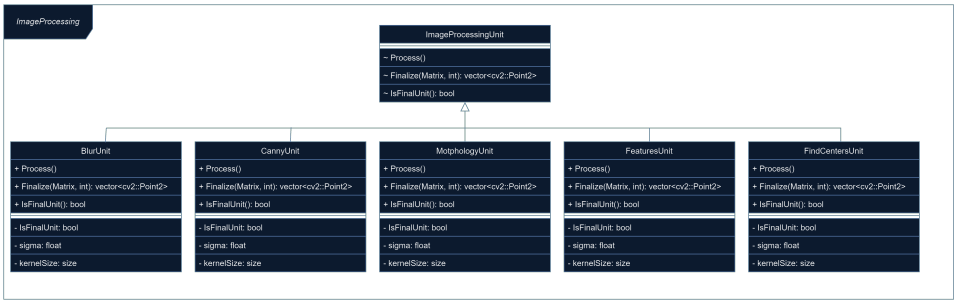


Figura 3.25: UML Image Processing

L'architettura è stata organizzata in questo modo per essere il più modulare e indipendente possibile dalle effettive implementazioni.

3.4.2.1 Procedimento di meshing

L'algoritmo di meshing delle geometrie dei tetti, dopo la fase di caricamento dati, considera tutti gli edifici presenti. Per ognuno di essi avvengono i seguenti passaggi:

1: Filtraggio punti

2: Clustering e selezione cluster primario

3: Organizzazione punti su griglia (coordinate locali)

4: Ottenimento punti di controllo del bordo esterno

5: Prima triangolazione con Delaunay senza vincoli

6: Rimozione triangoli esterni allo shape e filtraggio dei soli edge esterni

7: Ricostruzione del poligono di contenimento

8: Ottenimento dei punti di controllo di feature importanti (punti di massimo locale)

9: Rimozione punti duplicati

10: Seconda triangolazione con tutti i punti utilizzando Delaunay con vincoli

11: Trasformazione dei triangoli ottenuti in coordinate mondo

Infine avviene un passaggio di esportazione aggregando in un unico file tutte le geometrie calcolate.

Ovviamente ci sono alcune condizioni per le quali questo metodo ignora il corrente edificio passando al seguente, per esempio se nessuno dei file LAS associati ad esso è stato caricato (non si ha accesso ai suoi punti) oppure la nuvola di punti trovata è talmente piccola che può essere ignorata.

Caricamento dati

Come già discusso, le informazioni caricate si dividono in nuvole di punti LiDAR (file LAS) e dataframe degli edifici (file CSV). Le letture avvengono tramite gli appositi *reader*.

I vettori di punti risultanti sono stati quindi inseriti in una mappa insieme al nome del file, così da poterci accedere in maniera veloce durante l'esecuzione del ciclo di meshing. Abbiamo deciso di utilizzare questa strategia perchè

velocizza sensibilmente l'esecuzione al prezzo di avere più occupazione di memoria RAM a runtime.

Dopo di ciò vengono caricati gli edifici dal file CSV e viene utilizzata la factory *Building* per costruire gli oggetti.

1: Filtraggio punti

In questa prima fase ci occupiamo del primo isolamento dei punti che riteniamo appartenenti al nostro edificio. Per fare ciò utilizziamo i dati precedentemente calcolati tramite il CSV e quindi inseriti nell'oggetto Building, utilizzando un adattamento del codice scritto in *Rust*.

Cicliamo su ogni Tile a cui appartiene l'edificio (utile per gli shape al confine tra più tile) e per ognuno di questi andiamo a ricercare i punti che sono all'interno del poligono dell'edificio e che hanno una quota compresa tra la sua Quota di Gronda e la relativa somma con il valore di Tolleranza calcolato.

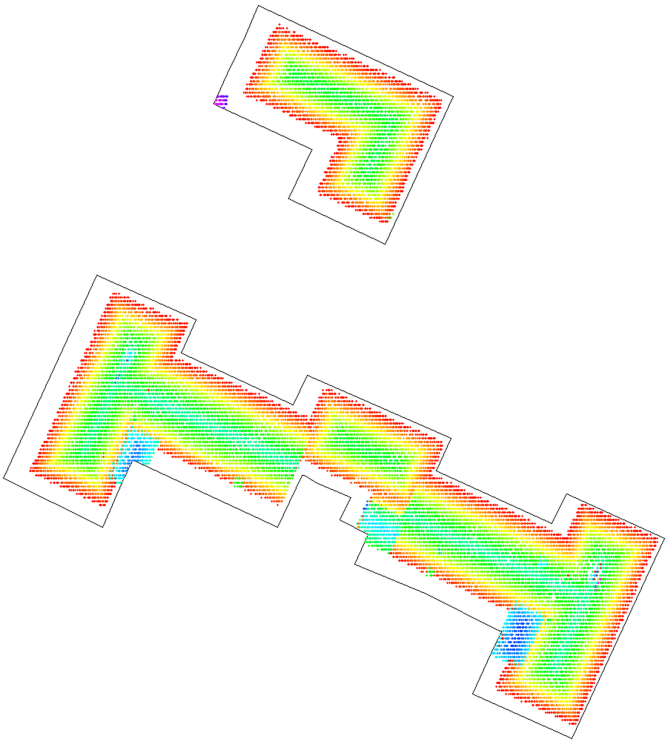


Figura 3.26: Esempi di filtraggio punti (edifici 52578 e 24069)

In figura 3.26 possiamo notare i punti filtrati colorati in base alla loro quota, partendo dai punti più bassi in rosso fino a quelli più alti in viola.

2: Clustering e selezione cluster primario

Viene utilizzata la metodologia DBSCAN[47] per il clustering, viene quindi identificato il cluster primario (quello con un numero maggiore di punti) e i restanti punti vengono scartati. Facciamo questo passaggio per eliminare eventuali outlier come punti di tetti, alberi o altri oggetti molto vicini ma comunque considerati separati a livello di cluster.

Come parametri sono stati utilizzati $epsilon = 0.8$ e $minPoints = 10$.

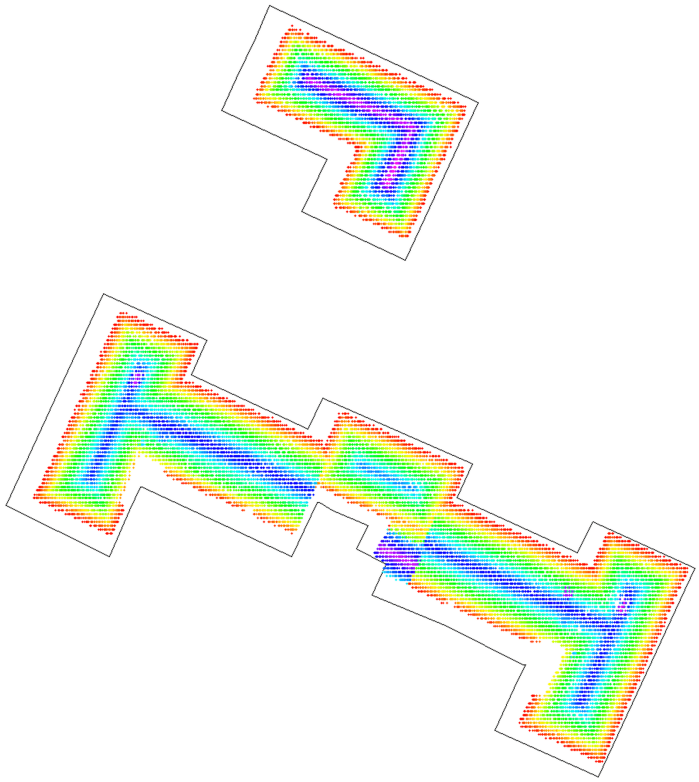


Figura 3.27: Esempi di filtraggio tramite clustering (edifici 52578 e 24069)

Dalla figura 3.27 si può vedere come i punti di tetti vicini e appartenenti ad antenne particolarmente alte sono stati esclusi rispetto al passaggio precedente.

3: Organizzazione punti su griglia

Si passa la PointCloud ottenuta a Grid, così da ottenere una versione semplificata e organizzata in una griglia della nostra nuvola, generando così una matrice altimetrica. Come parametri per la griglia generata abbiamo scelto una distanza 0,1 m, una tolleranza esterna rispetto alla bounding box

dei punti di 2,0 m e un raggio massimo di ricerca di 0,2 m. La ricerca dei punti avviene tramite Nearest Neighbour[67] search utilizzando *nanoflann* per la KD Tree Search e, dove non vengono trovati punti a una distanza minore del raggio di ricerca, viene impostato il valore altimetrico a 0,00.

Abbiamo tuttavia notato che questa metodologia, data la non uniformità della distribuzione dei punti nella nuvola di punti, talvolta lascia qualche buco dove non dovrebbe esserci (valori a 0,0 in mezzo a valori positivi). Per ovviare a questo, sottoponiamo la matrice a una operazione di riempimento che trova questi punti e ne calcola l'altimetria facendo una media dei loro neighbour con valori positivi (fig. 3.28).

Questo Grid manterrà oltre alla matrice anche i punti originali, necessari per l'ultimo step di riconversione in coordinate mondo.

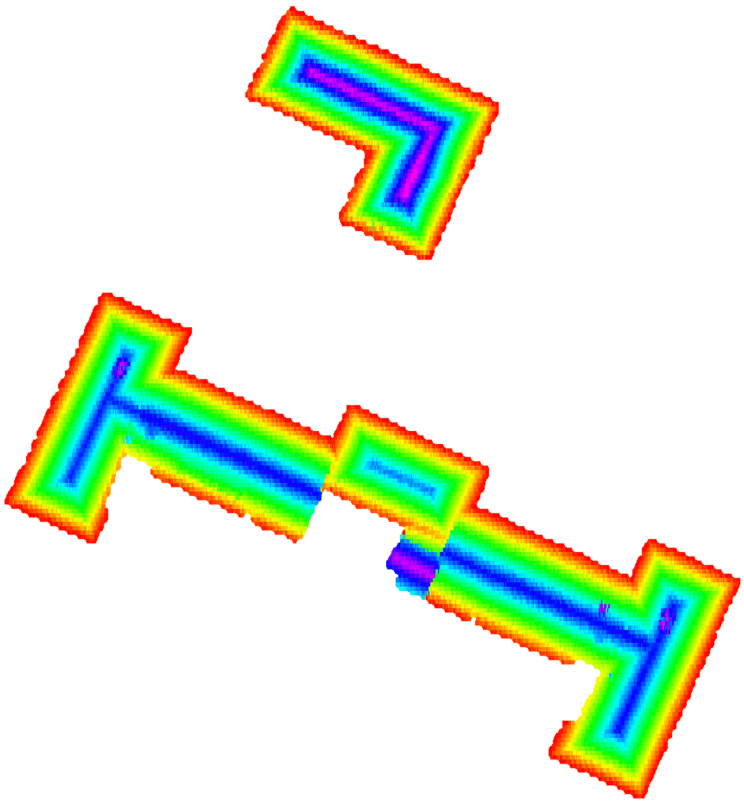


Figura 3.28: Esempi versione organizzata dei dati (edifici 52578 e 24069)

Anche in questo caso, il colore dipende dall'altimetria del punto, dove il bianco viene associato alla quota 0,00.

4: Ottenimento punti di controllo del bordo esterno

L'ottenimento di questi punti parte con la richiesta al nostro Grid di ottenere il tetto in versione booleana (solo due valori). Viene quindi utilizzata la prima pipeline di processing con OpenCV che, dopo aver creato l'immagine dalla matrice, esegue nell'ordine descritto le seguenti unità:

- **Blur** si occupa di applicare un Gaussian Blur[53] (parametro sigma 3,5), passaggio sempre necessario prima del prossimo step;
- **Canny** utilizza l'omonimo algoritmo[43] per la rilevazione dei bordi. Esso si basa sul calcolo del gradiente e quindi raffina i risultati con Non Maxima Suppression e sogliatura d'isteresi. I parametri di soglia utilizzati sono 20,0 e 80,0.
- **Morphology** (fig. 3.29) esegue una operazione di morfologia[66] di chiusura con elemento a croce, kernel di dimensione 5x5 e 4 iterazioni. Questo passaggio ci permette di modificare leggermente il bordo ottenuto con Canny per assicurarsi che esso sia continuo e non ci siano interruzioni. Conseguentemente viene eseguita una ricerca dei contorni e vengono rimossi tutti i contorni interni.

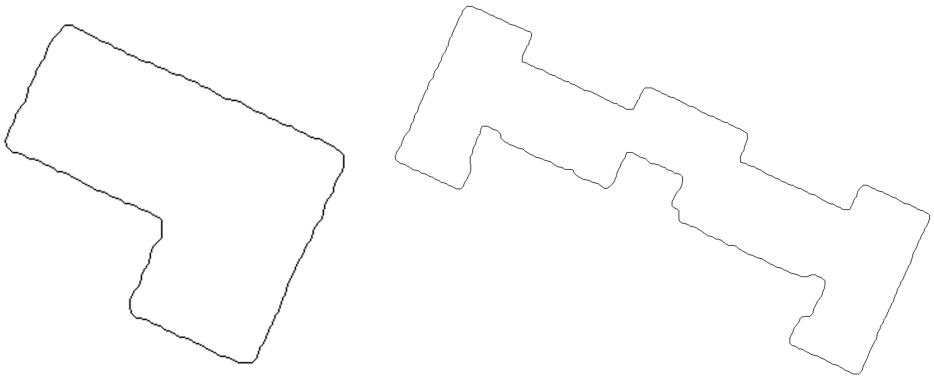


Figura 3.29: Esempi bordi tetti dopo morfologia (edifici 52578 e 24069)

- **Features** finalizza la pipeline utilizzando il metodo chiamato goodFeaturesToTrack che si basa sull'approccio di corner detection Shi-Tomasi[45] per selezionare i punti più caratteristici del bordo. I parametri utilizzati sono 0,1 come livello di qualità, 10,0 di distanza minima e 9 come dimensione di blocco/kernel per la ricerca. I punti restituiti

(fig. 3.30) sono in ordine di score, ciò ci permetterà quindi di poterli filtrare in seguito.

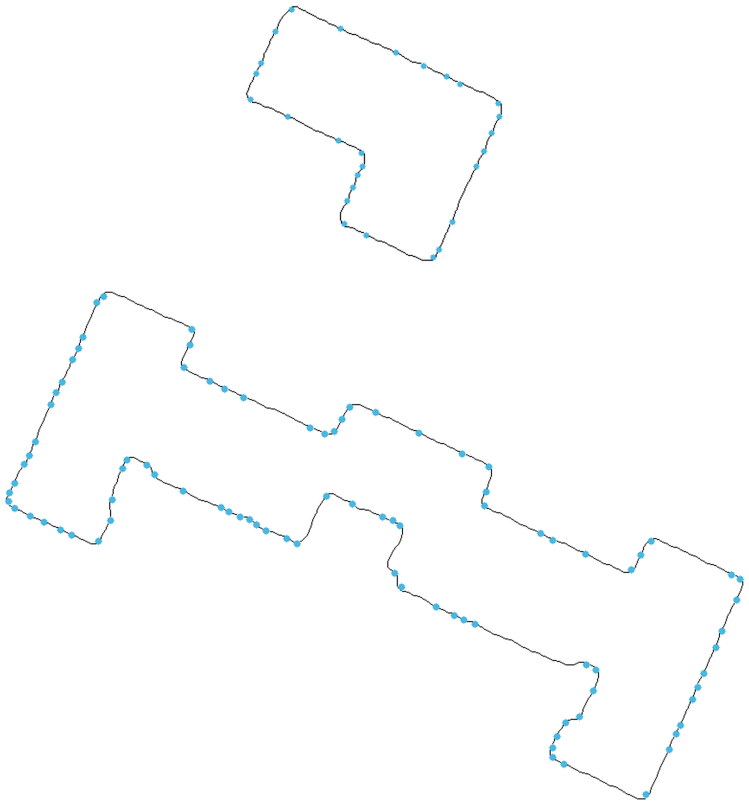


Figura 3.30: Esempi features ottenute (edifici 52578 e 24069)

Al momento della chiamata della pipeline viene passato come variabile il numero *MaxFeatures* che indica quanti punti si vogliono avere. Questo valore viene calcolato in base al numero di vertici presenti nello shape dell'edificio in analisi moltiplicato per un *safetyFactor*, che nel nostro caso è 3,0. Il perchè di questa operazione verrà spiegato all'inizio del passaggio seguente.

5: Prima triangolazione con Delaunay senza vincoli

Dobbiamo innanzitutto considerare il fatto che la triangolazione di Delaunay[48] di un gruppo di punti permette di ottenere l'inviluppo convesso dei punti, tuttavia il nostro obbiettivo è di creare mesh di poligoni spesso contenenti sia concavità che convessità.

Abbiamo notato che meno punti sono presenti e più è probabile che Delaunay generi triangoli che siano in parte dentro e in parte fuori al nostro

poligono, cosa che vogliamo evitare dato che dovremo scartare i triangoli all'esterno del nostro poligono. La moltiplicazione per il *safetyFactor* ci garantisce con una percentuale molto più alta che ciò non accada.

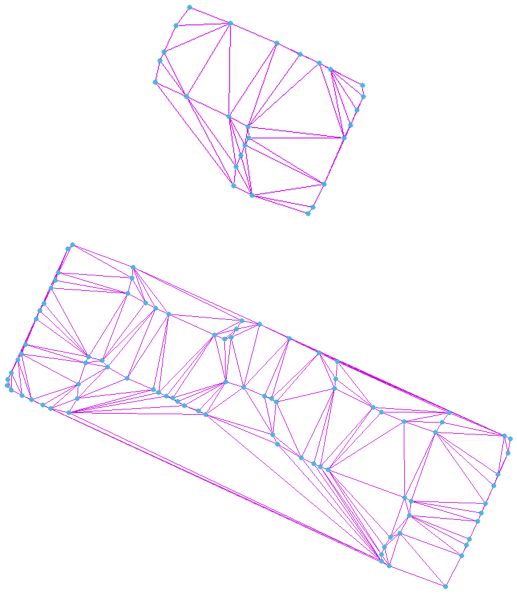


Figura 3.31: Esempi triang. Delaunay (senza vincoli - edifici 52578 e 24069)

6: Rimozione triangoli esterni allo shape e filtraggio dei soli edge esterni

Per identificare i triangoli esterni al nostro poligono utilizziamo una maschera derivante dal bordo precedentemente ottenuto. Tramite l'utilizzo di questa matrice, possiamo scartare tutti i triangoli il cui centroide cade al di fuori della maschera binaria, ottenendo i risultati visibili in figura 3.32.

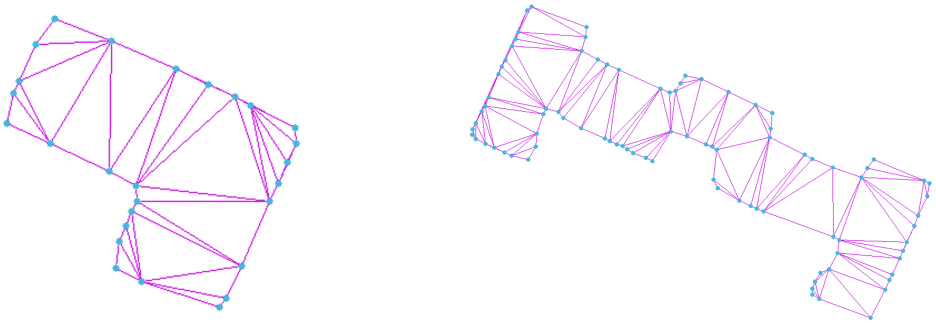


Figura 3.32: Esempi pulizia triangoli (edifici 52578 e 24069)

Avviene quindi un filtraggio che mantiene solo i bordi presenti in un unico triangolo, ottenendo così i soli lati esterni (fig. 3.33).

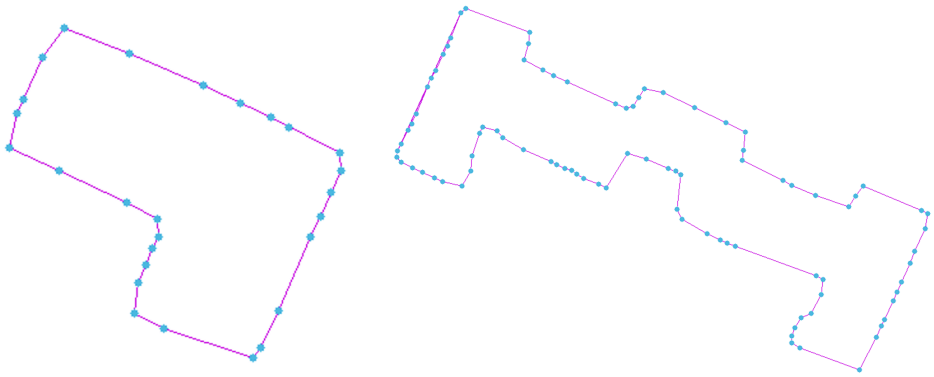


Figura 3.33: Esempi filtraggio bordi esterni (edifici 52578 e 24069)

7: Ricostruzione del poligono di contenimento semplificato

Come precedentemente menzionato, l'ordine dei nostri punti esterni è in base a quanto essi sono considerati angoli dal metodo di Shi-Tomasi. Possiamo utilizzare ciò a nostro vantaggio impostando un valore di precisione che ci identifica quali di essi possiamo considerare come "vertici primari" che funziona come spartiacque. Questo step mira appunto a costruire il poligono di contenimento esterno semplificando i punti di controllo ai soli vertici che rispettano questa condizione.

Per fare ciò abbiamo scritto un procedimento ad-hoc che prima pulisce il vettore di lati ottenuto nello step precedente, quindi si occupa della poligonizzazione di essi.

Nella fase di pulizia si utilizza il vettore dei lati esterni e, connessione per connessione, si vanno a ricostruire le connessioni tra i vertici primari passando per i secondari. Per ottimizzare il processo, ogni volta che un edge viene utilizzato per creare una connessione esso viene eliminato dal vettore, così da ridurre il numero di iterazioni progressivamente. Al termine di ciò abbiamo quindi un vettore non ordinato di connessioni che ci va a identificare con chi è collegato ogni vertice primario.

Possiamo quindi passare alla fase di poligonizzazione, che si occupa di ordinare questi edge in modo tale da avere un poligono consecutivo chiuso. Questa fase è importante per quanto riguarda l'utilizzo di Delaunay con

vincoli, dato che il poligono esterno sarà ciò con cui vogliamo confinare la nostra mesh finale.

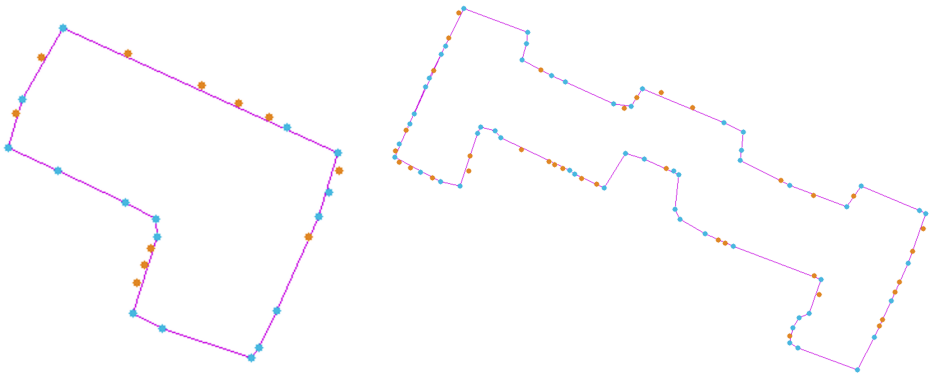


Figura 3.34: Esempi semplificazione bordi (edifici 52578 e 24069)

In figura 3.34 si può notare come vengano mantenuti solo i punti azzurri, scartando quelli arancioni, ricostruendo al contempo il nuovo poligono esterno.

8: Ottenimento dei punti di controllo di feature importanti (punti di massimo locale)

In questo passaggio ci occupiamo della ricerca delle feature di massimo locale dei tetti, questo ci può aiutare a isolare alcuni punti corrispondenti alle gronde all'interno del tetto, così da poterli utilizzare all'interno della nostra mesh.

Per la ricerca di questi punti utilizziamo il Grid precedentemente creato che, iterando all'interno della sua matrice altimetrica con una dimensione di kernel fornita (nel nostro caso 11), ci permette di filtrare i valori all'interno dell'intorno e di vedere in quella sotto-matrice qual'è il valore massimo trovato.

Questo step viene seguito da una specie di NMS (Non-Maximum Suppression), che si occupa di isolare i soli punti della matrice in cui si hanno effettivamente dei punti di massimo, ottenendo così gli intorni delle nostre feature.

In figura 3.35 mostriamo il risultato di questa ricerca sovrapposto al bordo per dare più contesto. Come si può notare, data la natura della nostra organizzazione dati spesso questi intorni possono essere composti da

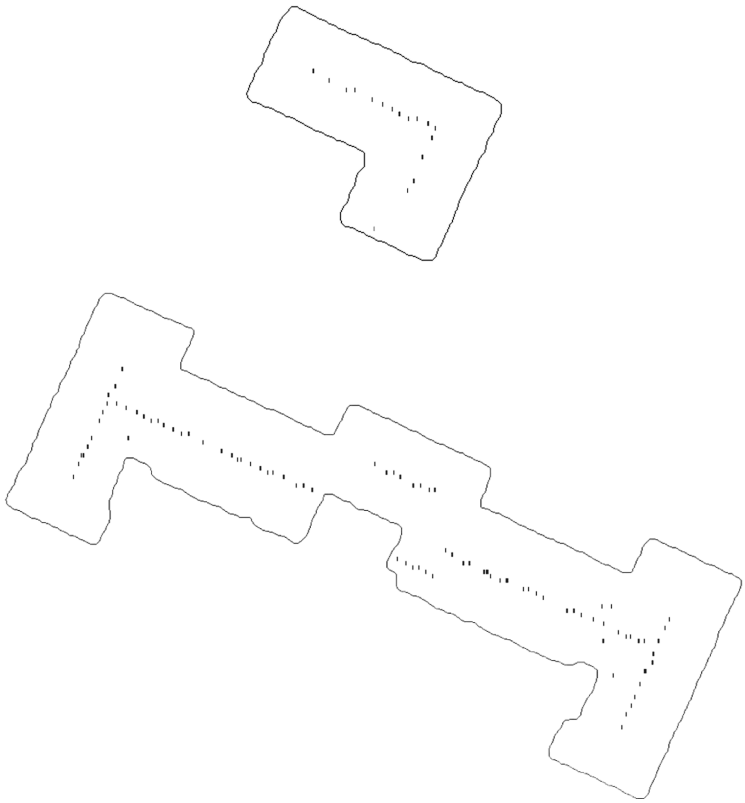


Figura 3.35: Esempi estrazione punti (edifici 52578 e 24069)

più pixel. Per ottenere punti singoli da ciascun intorno abbiamo selezionato in seguito come punto di interesse il loro centroide.

9: Rimozione punti duplicati

Si itera sui punti delle feature e vengono scartati tutti i punti a una distanza minore di una certa tolleranza da a un punto già presente tra i punti di controllo esterni, così da evitare duplicati e punti pressoché attaccati.

10: Seconda triangolazione con tutti i punti utilizzando Delaunay con vincoli

Ora che abbiamo a disposizione tutti i punti della mesh e abbiamo il poligono esterno semplificato e ordinato, possiamo procedere con la triangolazione della Mesh finale. Questo avviene nuovamente tramite la libreria *Triangle* tramite un apposito setup necessario per ottenere la triangolazione di Delaunay vincolando il perimetro esterno.

In figura 3.36 sono stati evidenziati i punti del bordo in azzurro e i punti

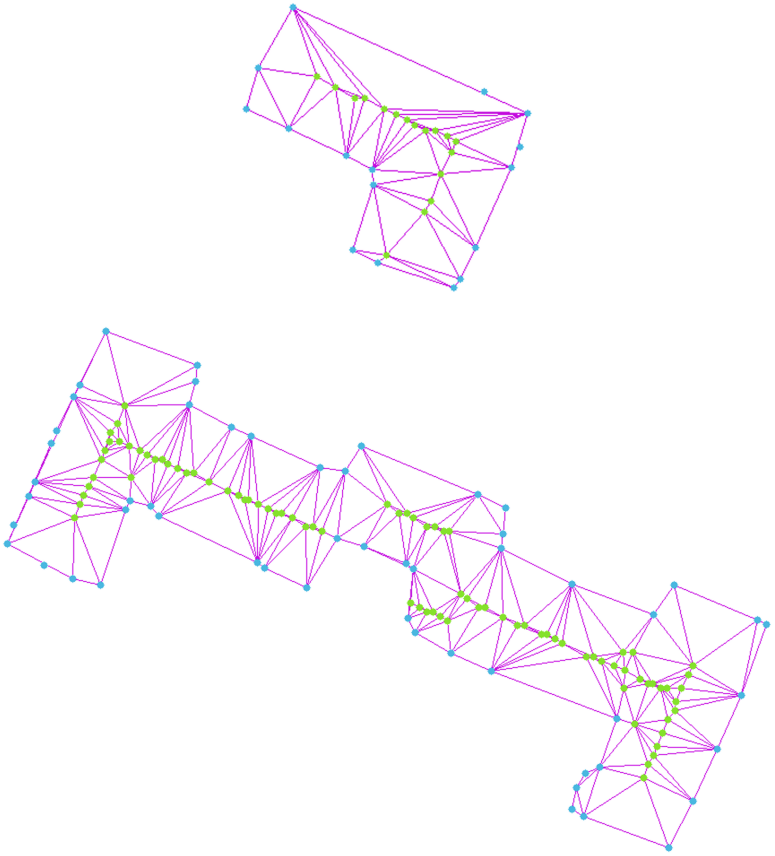


Figura 3.36: Esempi mesh ottenute con Delaunay vincolato dal poligono esterno (edifici 52578 e 24069)

interni in verde. Come si può vedere i triangoli della mesh risultante rimangono all'interno del poligono di esterno, rispettando il vincolo e garantendoci di non avere triangoli al contempo interni ed esterni alla nostra nuvola. Ovviamente non è necessario lo step di pulizia precedentemente utilizzato.

11: Trasformazione dei triangoli ottenuti in coordinate mondo

Tramite il Grid, si trasformano le coordinate locali di ogni vertice in coordinate mondo e quindi si fa una ricerca del Nearest Neighbour tra i punti della nuvola inizialmente filtrati, reintegrando in questo modo anche la coordinata Z "persa" durante il processing.

Le mesh risultanti vengono quindi aggiunte a un vettore pronte per l'esportazione.

Esportazione mesh

La fase terminale si occupa quindi di eseguire l'esportazione delle mesh create in formato STL[75] (modalità ASCII). Esse vengono raggruppate in un unico file per velocizzare sia il salvataggio che il futuro caricamento dei dati. Abbiamo utilizzato questo formato per la sua compattezza e semplicità, permettendo la scrittura del file tempi brevi.

Infatti, questo formato ci permette di rappresentare ogni triangolo definendo le coordinate dei suoi vertici nel seguente formato:

```
solid obj_name
    outer loop
        vertex v1x v1y v1z
        vertex v2x v2y v2z
        vertex v3x v3y v3z
    endloop
    [...]
endsolid obj_name
```

Avendo molti thread a disposizione, sarebbe anche possibile destinare uno di essi alla scrittura del file man mano che vengono calcolate le mesh, tuttavia reputiamo che questo non porti necessariamente a un miglioramento drastico delle prestazioni.

3.4.2.2 Risultati

I tetti mostrati come esempio nelle figure mostrate sono stati il nostro focus principale durante lo sviluppo dell'algoritmo presentato. Ci teniamo tuttavia a mostrare altri tre esempi di tetti che il nostro metodo ricostruisce in maniera molto accurata. I passaggi mostrati saranno: 1(a), 3(b), 6(c), 7(d), 8(e) e 10(f).

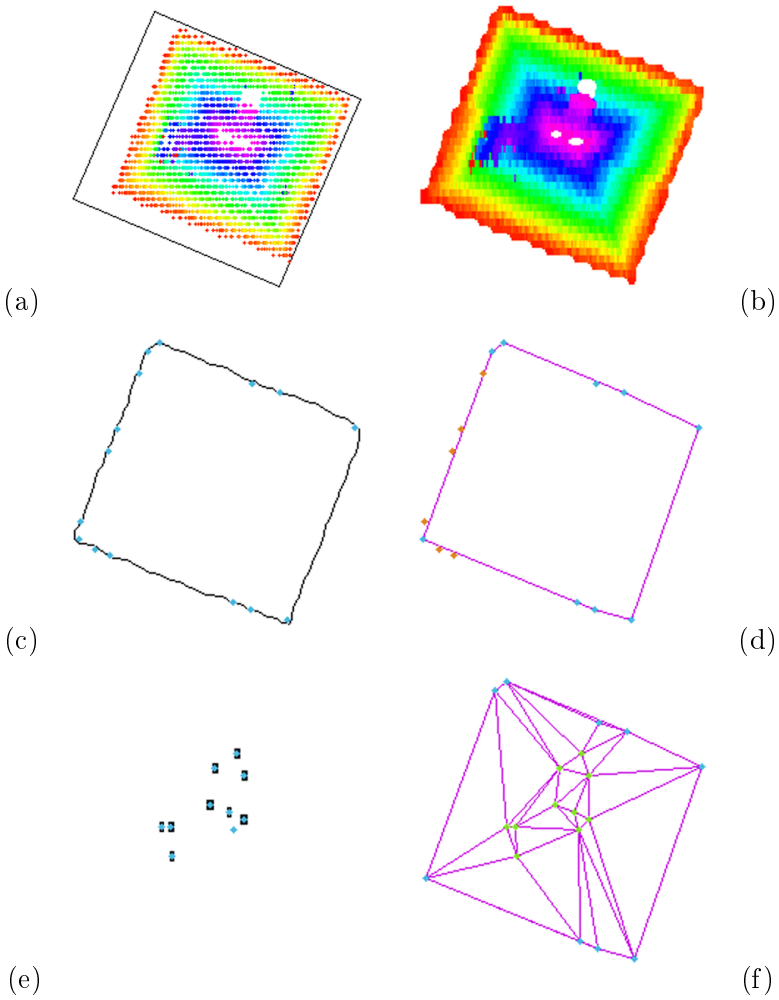


Figura 3.37: Esempio risultati - edificio 52148

Dalla figura 3.37 possiamo vedere il caso che ci aspetteremmo come basico per la maggior parte dei tetti presenti in provincia. Il tetto è di forma circa quadrata e presenta sia un camino alto che varie antenne/parabole che vengono esclusi dagli step di tolleranza e clustering. I risultati ottenuti in questa casistica ci sembrano molto promettenti.

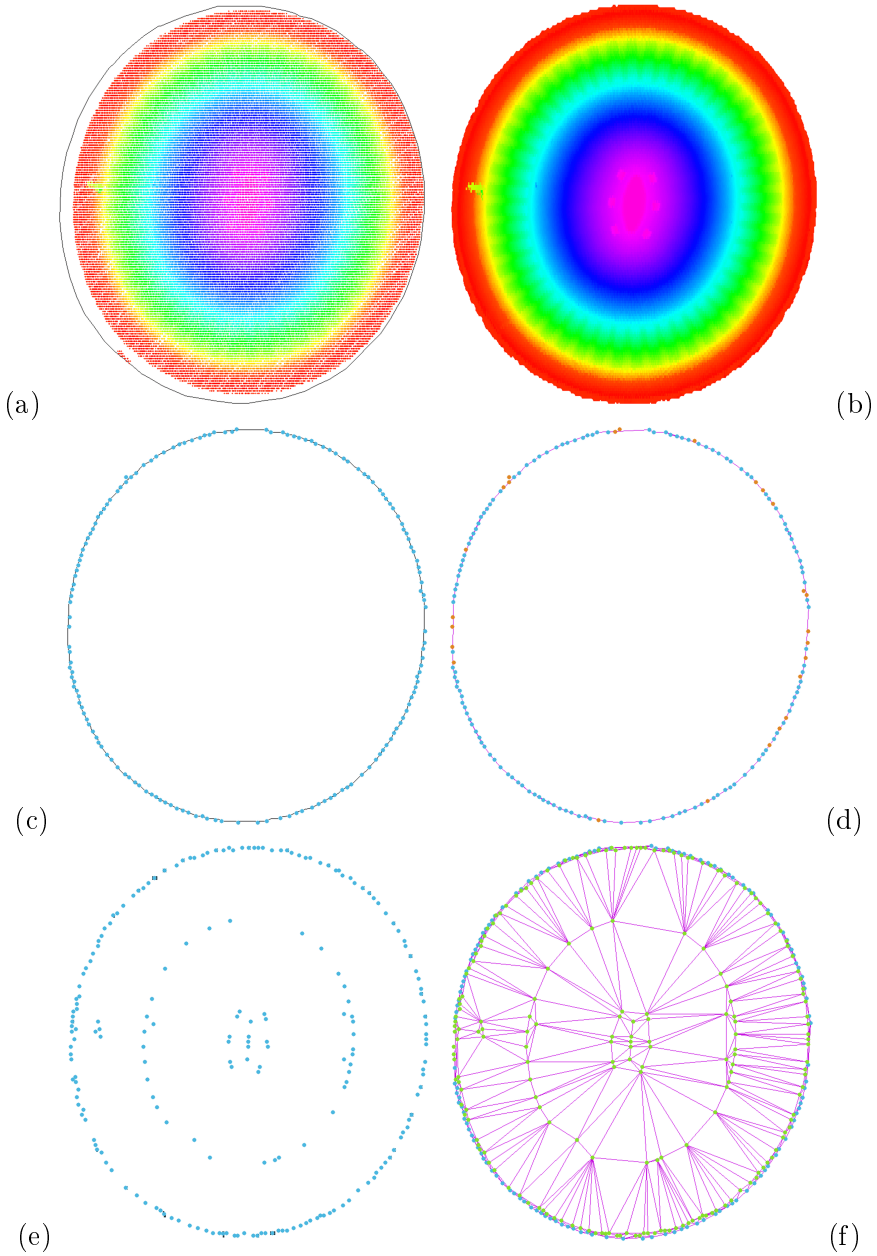


Figura 3.38: Esempio risultati - edificio 47924 (PalaDozza)

Nella figura 3.38 analizziamo come il nostro metodo si comporta nel caso di un tetto a cupola geodetica, in particolare quello del PalaDozza[68]. Si può notare come il numero di punti nel contorno esterno sia ancora abbastanza alto, questo deriva dal fatto che il poligono del GeoShape utilizzato ha anch'esso un numero molto alto di vertici.

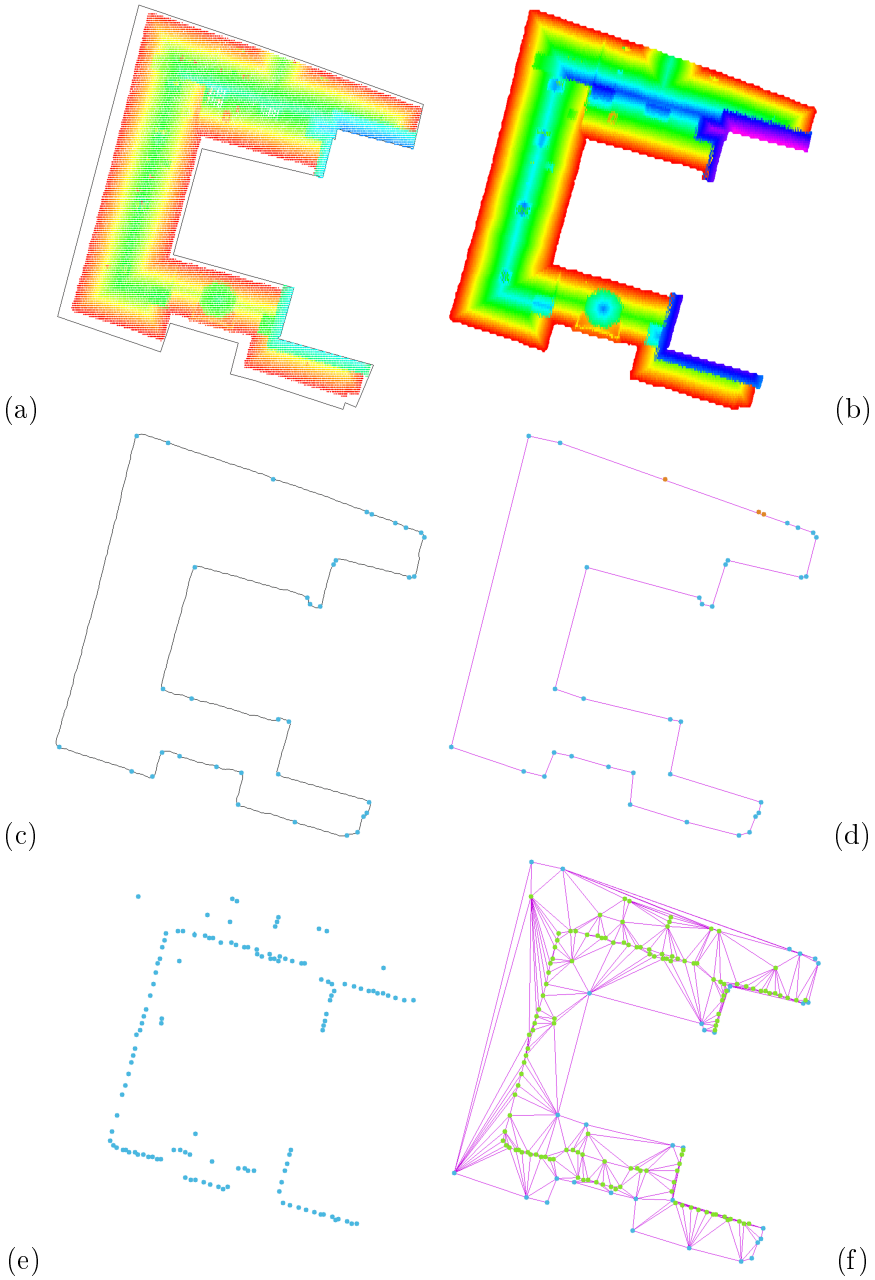


Figura 3.39: Esempio risultati - edificio 24921

Infine, in figura 3.39 vediamo come il nostro approccio si comporti con uno shape particolarmente complesso nel quale sono presenti varie sotto-tipologie di tetti, identificabili dalla matrice altimetrica raffigurata.

Nei casi mostrati si può notare come spesso lo shape utilizzato per a ricerca iniziale sia "abbondante" nella parte inferiore sinistra. Questo è si un fatto ricorrente, tuttavia da un analisi più approfondita della distribuzione degli shape rispetto ai punti è risultato che ci sono molte altre casistiche che si comportano in maniera diversa e anche opposta.

Analizzando invece i fallimenti, nei quali il nostro procedimento da risultati sbagliati o solo parziali dell'area del tetto, abbiamo tratto le seguenti motivazioni:

- disallineamenti molto ampi tra il GeoShape e i punti del relativo edificio, che ovviamente non ci permettono di trovare i punti in modo corretto.
- alta disparità tra la complessità del tetto e il numero di vertici del GeoShape che lo contiene, dato che il numero di feature del contorno estratte dipende da esso questo comporta il filtraggio di troppi pochi punti per rappresentare il bordo del tetto, dando aree solo parziali.
- punti estranei sui bordi e al di sopra dei tetti, possono a volte complicare le computazioni introducendo rumori difficili da trattare.
- tetti piani con quota di gronda al parapetto, dove come già detto vengono filtrati i punti della base piana perchè al di sotto della nostra quota di filtraggio. Abbassare questa quota per includerli comporterebbe più problemi di quelli che risolverebbe.

Proponiamo quindi alcune soluzioni, non esplorate per motivi di tempo, per migliorare questi aspetti:

- utilizzo di ortofoto che, se coincidono meglio con le nuvole di punti, potrebbero migliorare sia il posizionamento degli edifici che potenzialmente l'allineamento tra essi e i relativi tetti. Oltre a ciò queste potrebbero essere utilizzate per riconoscere i tetti piani per trattarli in maniera migliore e anche per segmentare ulteriormente gli shape basandosi sulla composizione di sotto-categorie di tetti più semplici, così da suddividere shape troppo complessi in sotto-shape più semplici.
- miglioramento della semplificazione e del trattamento dei punti di interesse, risultando in mesh più semplici da calcolare e più leggere in termini di poligoni.

- introduzione di punti di minimo locale e la possibilità di avere "fori" nelle mesh, non banale dato che bisognerebbe distinguere la mancanza di punti causata da cortili interni rispetto a quella data dal filtraggio di punti al di sopra della tolleranza. Questo potrebbe però generare mesh decisamente più accurate.

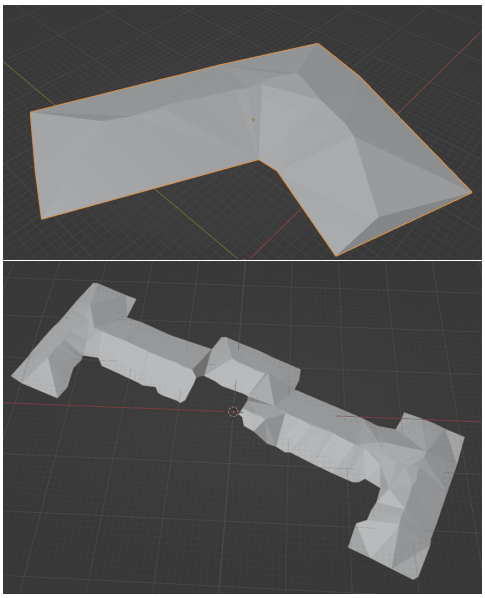


Figura 3.40: Esempi mesh ottenute edifici 52578 e 24069 (Blender)

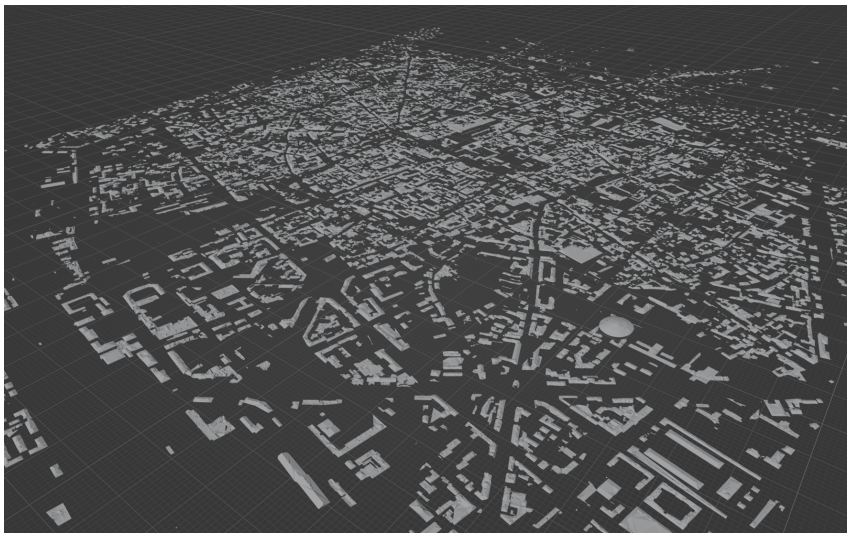


Figura 3.41: Risultati tetti del centro (Blender)

3.4.2.3 Prestazioni

Abbiamo infine fatto una piccola analisi delle prestazioni per vedere quali altri aspetti possono essere migliorati. Questi test sono stati fatti sul portatile fornitoci da *CINECA* avente le seguenti caratteristiche hardware:

- CPU: i5-8250U - Quad Core (1.60 GHz)
- RAM: 16 GB

I dati di seguito riportati ignorano i tempi di caricamento dati (presenti nella fase iniziale) e di salvataggio su file (in terminazione), della durata complessiva di qualche secondo.

Iniziando con i dati relativi ai singoli tetti mostrati precedentemente, essi hanno avuto i seguenti tempi di computazione:

Info	Codice Oggetto edificio	52578	24069	52148	47924	24921
	Numero vertici GeoShape	9	30	5	84	27
	Numero punti filtrati	8278	36489	5268	136132	91677
	Numero punti dopo Clustering	8204	34897	5240	136085	91411
	Triangoli mesh risultante	46	223	29	564	283
Tempi di Esecuzione (ms)	Filtraggio dalla nuvola	139,316	946,509	76,942	1931,09	2676,96
	Clustering	21,644	85,042	15,036	38,257	263,072
	Organizzazione (Grid)	22,635	168,523	11,72	183,433	287,185
	Rilevamento punti bordo	13,669	29,977	18,214	33,7838	39,265
	Triangolazione (Non Vinc.)	0,135	0,246	0,141	0,494	0,206
	Rimoz. trian. esterni + isolamento bordi esterni	278,326	4878,64	98,076	36891,1	4159,37
	Poligono contenimento	0,005	0,015	0,003	0,032	0,009
	Rilevamento punti loc. max	10,423	66,35	5,955	79,07	92,484
	Pulizia punti	0,032	0,354	0,018	2,117	0,323
	Triangolazione (Vincolata)	0,077	0,175	0,048	0,364	0,204
	Conversione coordinate	460,625	11248,9	176,472	105633	31658,2
	Totale	**946,887**	**17424,731**	**402,625**	**144792,7408**	**39177,278**

Figura 3.42: Risultati performance tetti singoli (single thread)

Come si può vedere dalla tabella presentata in figura 3.42 le tempistiche di computazione dipendono molto dal tetto che si sta calcolando, generalmente richiedendo un tempo più ampio per i tetti più complessi. Dai tempi di computazione possiamo vedere come i vari step della metodologia utilizzata impattino sul tempo di calcolo.

Si può notare subito come la conversione delle coordinate della nostra mesh sia la cosa più impattante a livello di tempo. Questo perchè, per semplificare le cose, abbiamo utilizzato come struttura dati un vettore di triangoli dove ogni triangolo è formato da 3 punti. Ovviamente questa struttura rende

semplice la gestione della mesh ma risulta poco ottimizzata per quest'ultimo passaggio. La nostra proposta per ottimizzare questa struttura e la sua conversione consiste nel renderla composta da un vettore di punti insieme a un vettore di triple che indica i triangoli usando gli indici dei punti.

Per quanto riguarda i tempi di computazione degli altri step riteniamo al momento che non ci siano soluzioni immediate per un ottimizzazione migliore.

Invece riguardo ai tile completi, sui 30 tile del centro di cui abbiamo generato i tetti, sono state calcolate le performance su 10 di essi in aggiunta a uno esterno al centro. Abbiamo fatto ciò per vedere quanta differenza ci fosse dato che la complessità dei tetti del centro decade drasticamente nella maggior parte delle zone al di fuori di esso, dove si trovano generalmente tetti più sparsi e di più facile computazione.

Nei **10 tile del centro** il valore medio di esecuzione per un singolo edificio è di **1,837 sec**, anche in questo caso troviamo che i tempi di calcolo principali rimangono appartenenti alla conversione di coordinate (67%), il tempo di rimozione triangoli esterni e estrazione dei bordi esterni (16%) e quindi il tempo di filtraggio (12%). Il tempo di esecuzione medio cala a **1,152 sec** nel **tile esterno** al centro.

Per quanto riguarda l'occupazione di memoria RAM, in single thread con un solo tile come target ci si aggira a circa 170 MB con picchi a circa 220 MB.

In **multithread** viene ottenuto un incremento generale della velocità di computazione sui tile, ottenendo nel nostro caso un **incremento di velocità medio del 256%**. Conseguentemente anche i picchi in memoria RAM hanno un incremento, aggirandosi in media intorno ai 560 MB.

3.4.3 Texturing tetti

Per il texturing dei tetti il procedimento si fa più complesso rispetto a quello utilizzato per il DTM. Il processo sarà comunque gestito tramite l'applicazione Blender.

Allineamento

Il passo iniziale e cruciale del pre-processing consiste nell'**allineare** i dati con il DTM e gli edifici. Sebbene questi dati siano georeferenziati, una volta importati in Blender, si trovano a una distanza eccessiva dall'origine, rendendo la loro gestione difficile a causa di limitazioni specifiche del software.

Per risolvere questa problematica, abbiamo raggruppato tutte le mesh sotto un oggetto *Empty*, posizionato alle coordinate 685000x, 4928000y e 0z. Questa strategia ci permette di spostarli facilmente, tramite l'oggetto padre, all'origine di Blender, garantendo così che le mesh si posizionino correttamente rispetto agli altri elementi.

Texturing

Il processo di texturing è stato concepito per affrontare la sfida rappresentata dalla discrepanza tra le dimensioni reali dei tetti e le dimensioni delle texture disponibili, che coprono un'area di 500x500.

La strategia adottata ha incluso l'uso di un piano posizionato al di sotto delle mesh dei tetti della dimensione della texture. In tal modo, le mesh dei tetti hanno potuto utilizzare esso come piano di proiezione per la texture associata al loro materiale. Questo approccio ha consentito di mappare in modo efficace e preciso le texture 2k su di essi, garantendo che l'aspetto visivo dei tetti fosse realistico e coerente.

Ottimizzazione tramite Atlas

Data la limitata estensione dei tetti rispetto alle vaste aree rappresentate dalle ortofoto, è stato adottato un approccio che prevede la creazione di un **Atlas**[76] di texture. Questa soluzione ha permesso di organizzare le texture in gruppi di 5 tile, sfruttando texture 4K per immagazzinare un volume di informazioni significativamente maggiore rispetto a quello che sarebbe stato possibile con le texture derivanti direttamente dalle ortofoto.

La metodologia implementata ha inizio la selezione dei tile raggruppati, seguita da uno Smart UV Mapping delle mesh su texture 4k vuote, utilizzando un valore di margine di 0,0001 unità, ottimizzando l'applicazione

della texture. Successivamente, avviene il processo di aggregazione delle isole (Islands Packing), mantenendo lo stesso valore di margine, che migliorato ulteriormente la disposizione delle texture, massimizzando l'efficienza dello spazio disponibile.

Il passo finale comprende il baking delle texture originali sulla texture 4k generata, mantenendo il raggruppamento di 5 tile, con un valore di margine di 4 pixel. Questa fase assicura che i dettagli e la qualità visiva delle texture originali vengano preservati, trasferendoli efficacemente sulla nuova texture. L'atlas così creato viene applicato ai materiali delle mesh corrispondenti, rendendo il modello non solo più accurato ma anche più facile da gestire. Questo approccio consente di conservare un'elevata fedeltà visiva pur ottimizzando lo spazio di archiviazione delle texture.

Export

L'esportazione delle mesh avviene anche in questo caso tramite il formato FBX. Per mantenere lo standard di modularità e sovrapposizione, i tetti appartenenti a cascun tile vengono esportati separatamente, condividendo tuttavia il materiale in base al raggruppamento utilizzato.

Texture e densità

Come si evince dall'approccio adottato, l'accuratezza e la densità di pixel associata a ciascun tetto sono direttamente influenzate dalla densità di area dei tetti nei 5 tile raggruppati. In altre parole, maggiore è la copertura dei tetti all'interno di questi tile, minore sarà la precisione texture. Questo perché una texture 4K deve rappresentare un'area significativamente più ampia di dettagli rispetto a zone meno dense di tetti, dove l'area effettiva coperta è minore.

Per mitigare questa problematica, si propone di adottare strategie differenziate a seconda della densità degli edifici nelle varie zone. Dove la presenza di edifici è minore, si possono utilizzare texture di dimensioni ridotte o raggruppare un numero maggiore di tile, mentre nelle zone con una maggiore concentrazione di tetti, sarebbe opportuno ridurre il numero di tile per gruppo, al fine di mantenere una definizione più alta delle texture.

Questa necessità di adattare l'approccio in base alla densità di copertura dei tetti giustifica la decisione di non automatizzare il processo attraverso script. In futuro riteniamo automatizzabile il processo trovando il rapporto superficie-aggregazione necessario per avere risultati ottimali considerando il livello di dettaglio desiderato.

Figura 3.43: Tetti con texture su DTM ed edifici (UE5)

3.5 Verde Urbano

Vengono qui presentati brevemente i risultati ottenuti dal mio collega per ciò che riguarda la preparazione dei dati del verde urbano.

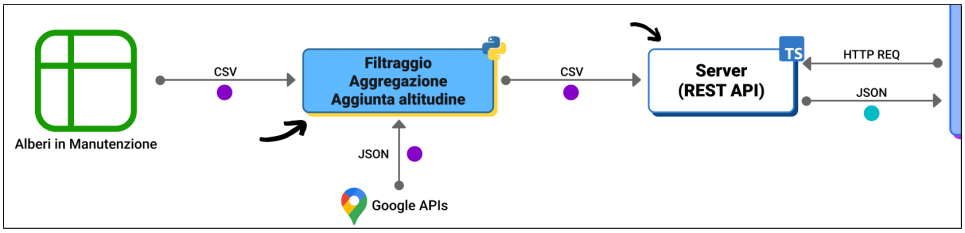


Figura 3.44: Sezione workflow: alberi

La pipeline (fig. 3.44) inizia con una fase di processing che si occupa prima di ogni cosa del filtraggio in base alle coordinate (fig. 3.45), dato che miriamo alla visualizzazione dei soli alberi presenti nel centro. Successivamento viene fatta un'aggregazione delle specie arboree, visto che il numero di mesh 3D degli alberi disponibili è molto più limitato rispetto alla biodiversità degli alberi presenti, e quindi è stata aggiunta la quota del terreno per ogni albero tramite le API di Google.

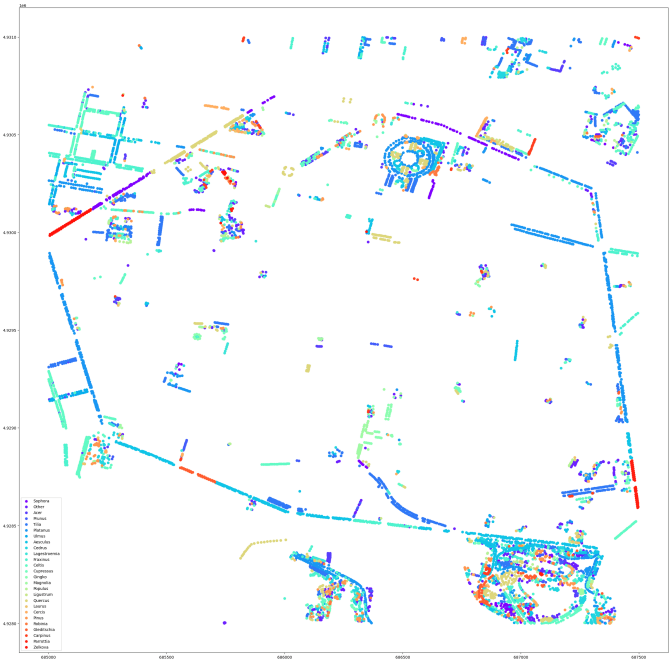
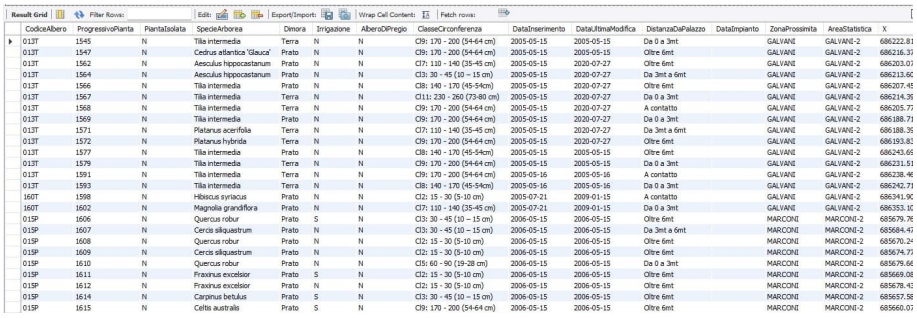


Figura 3.45: Mappa distribuzione alberi del centro

Il dataframe risultante è quindi stato utilizzato per la creazione di un database[46] SQL (fig. 3.46) con il quale è possibile interagire tramite delle REST[72] API esposte da un Server Cloud.



CodiceAlbero	ProgressivoPianta	PiantaIsolata	SpecieArborea	Dimora	Irrigazione	AlberoDiPregio	ClasseCirconferenza	DataInserimento	DataUltimaModifica	DistanzaDaPalazzo	DataImpianto	ZonaProssimita	AreaStatistica	X
013T	1545	N	Tilia intermedia	Terra	N	N	C9: 170 - 200 (54-64 cm)	2005-05-15	2005-05-15	Da 0 a 3mt		GALVANI	GALVANI-2	686222.81
013T	1547	N	Cedrus atlantica 'Glauca'	Prato	N	N	C9: 170 - 200 (54-64 cm)	2005-05-15	2005-05-15	Oltre 6mt		GALVANI	GALVANI-2	686216.37
013T	1562	N	Aesculus hippocastanum	Prato	N	N	C7: 110 - 140 (35-45 cm)	2005-05-15	2020-07-27	Oltre 6mt		GALVANI	GALVANI-2	686203.07
013T	1564	N	Aesculus hippocastanum	Prato	N	N	C3: 30 - 45 (10 – 15 cm)	2005-05-15	2020-07-27	Da 3mt a 6mt		GALVANI	GALVANI-2	686213.66
013T	1566	N	Tilia intermedia	Prato	N	N	C8: 140 - 170 (45-54cm)	2005-05-15	2020-07-27	Oltre 6mt		GALVANI	GALVANI-2	686207.45
013T	1567	N	Tilia intermedia	Terra	N	N	C11: 230 - 260 (73-80 cm)	2005-05-15	2020-07-27	Da 0 a 3mt		GALVANI	GALVANI-2	686214.36
013T	1568	N	Tilia intermedia	Terra	N	N	C9: 170 - 200 (54-64 cm)	2005-05-15	2020-07-27	A contatto		GALVANI	GALVANI-2	686205.77
013T	1569	N	Tilia intermedia	Prato	N	N	C9: 170 - 200 (54-64 cm)	2005-05-15	2020-07-27	Da 0 a 3mt		GALVANI	GALVANI-2	686188.71
013T	1571	N	Platanus acerifolia	Terra	N	N	C7: 110 - 140 (35-45 cm)	2005-05-15	2020-07-27	Da 3mt a 6mt		GALVANI	GALVANI-2	686188.35
013T	1572	N	Platanus hybrida	Terra	N	N	C9: 170 - 200 (54-64 cm)	2005-05-15	2020-07-27	Oltre 6mt		GALVANI	GALVANI-2	686193.81
013T	1577	N	Tilia intermedia	Prato	N	N	C8: 140 - 170 (45-54cm)	2005-05-15	2005-05-15	Oltre 6mt		GALVANI	GALVANI-2	686243.65
013T	1579	N	Tilia intermedia	Terra	N	N	C9: 170 - 200 (54-64 cm)	2005-05-15	2005-05-15	Da 0 a 3mt		GALVANI	GALVANI-2	686231.51
013T	1591	N	Tilia intermedia	Terra	N	N	C9: 170 - 200 (54-64 cm)	2005-05-16	2005-05-16	A contatto		GALVANI	GALVANI-2	686238.49
013T	1593	N	Tilia intermedia	Terra	N	N	C8: 140 - 170 (45-54cm)	2005-05-16	2005-05-16	Da 0 a 3mt		GALVANI	GALVANI-2	686242.71
160T	1598	N	Hibiscus syriacus	Prato	N	N	C2: 15 - 30 (5-10 cm)	2005-07-21	2009-01-15	A contatto		GALVANI	GALVANI-2	686341.90
160T	1602	N	Magnolia grandiflora	Prato	N	N	C7: 110 - 140 (35-45 cm)	2005-07-21	2009-01-15	Da 0 a 3mt		GALVANI	GALVANI-2	686353.10
015P	1606	N	Quercus robur	Prato	S	N	C3: 30 - 45 (10 – 15 cm)	2006-05-15	2006-05-15	Oltre 6mt		MARCONI	MARCONI-2	685679.76
015P	1607	N	Cercis siliquastrum	Prato	N	N	C3: 30 - 45 (10 – 15 cm)	2006-05-15	2006-05-15	Da 3mt a 6mt		MARCONI	MARCONI-2	685684.47
015P	1608	N	Quercus robur	Prato	N	N	C2: 15 - 30 (5-10 cm)	2006-05-15	2006-05-15	Oltre 6mt		MARCONI	MARCONI-2	685670.24
015P	1609	N	Cercis siliquastrum	Prato	N	N	C2: 15 - 30 (5-10 cm)	2006-05-15	2006-05-15	Oltre 6mt		MARCONI	MARCONI-2	685674.71
015P	1610	N	Quercus robur	Prato	N	N	C5: 60 - 90 (19-28 cm)	2006-05-15	2006-05-15	Da 0 a 3mt		MARCONI	MARCONI-2	685679.66
015P	1611	N	Fraxinus excelsior	Prato	S	N	C2: 15 - 30 (5-10 cm)	2006-05-15	2006-05-15	Oltre 6mt		MARCONI	MARCONI-2	685669.08
015P	1612	N	Fraxinus excelsior	Prato	N	N	C2: 15 - 30 (5-10 cm)	2006-05-15	2006-05-15	Oltre 6mt		MARCONI	MARCONI-2	685678.43
015P	1614	N	Carpinus betulus	Prato	S	N	C3: 30 - 45 (10 – 15 cm)	2006-05-15	2006-05-15	Oltre 6mt		MARCONI	MARCONI-2	685657.58
015P	1615	N	Celtis australis	Prato	S	N	C9: 170 - 200 (54-64 cm)	2006-05-15	2006-05-15	Oltre 6mt		MARCONI	MARCONI-2	685660.07

Figura 3.46: Snapshot Database

Capitolo 4

Sviluppo del Digital Twin statico

Contents

Mostriamo brevemente com'è stata sviluppata la parte statica del Digital Twin, sintetizzando il lavoro svolto dal mio collega.

4.1 Modellazione degli alberi

Utilizzando il software gratuito Tree It[36] (fig. 4.1), è stato possibile modellare alberi realistici e dettagliati per rappresentare le specie più comuni trovate nella città.

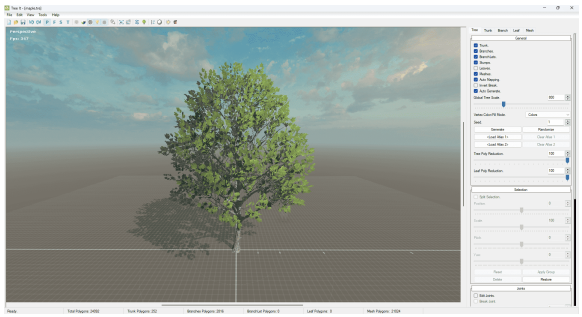
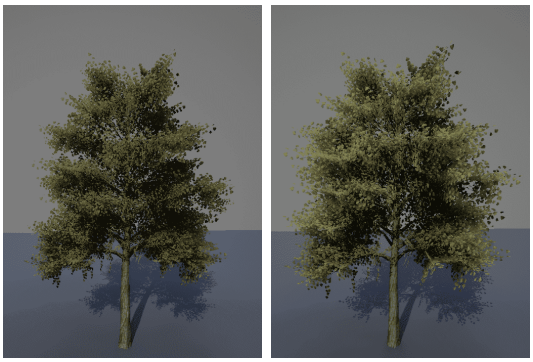
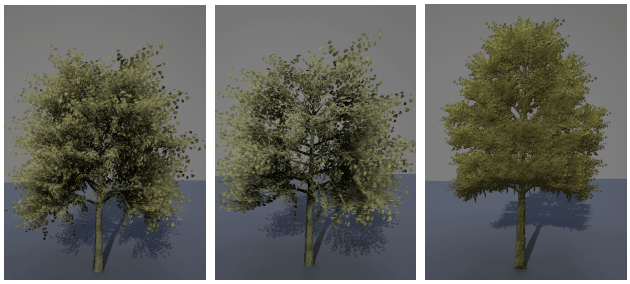


Figura 4.1: Tree It

Sono stati generati modelli a diverse risoluzioni poligonali per costruire LOD efficienti (fig. 4.2), fondamentali per ridurre il carico di rendering dato dalla complessa geometria degli alberi.



(a) LOD 0 (b) LOD 1

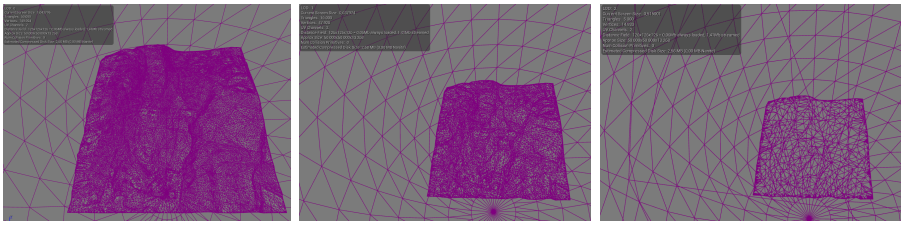


(c) LOD 2 (d) LOD 3 (e) LOD 4

Figura 4.2: LOD degli alberi

4.2 Importazione dei Modelli 3D della Città

Dopo la creazione di un progetto vuoto in Unreal Engine, si procede con l'importazione delle mesh 3D relative al terreno (DTM), degli edifici, dei tetti e dei modelli degli alberi. Per le mesh del terreno e degli edifici, si effettuano operazioni di tuning, impostazione dei LOD (Level of Detail) (fig. 4.3) e configurazione delle collisioni (fig. 4.4) per consentire una navigazione immersiva e realistica all'interno del Digital Twin.



(a) LOD 0 (b) LOD 1 (c) LOD 2

Figura 4.3: LOD del DTM in Unreal

Nel nostro caso, dato che per i DTM era impossibile utilizzare collisioni semplici, è stata utilizzata la mesh del LOD 0 come modello poligonale per il calcolo delle collisioni.

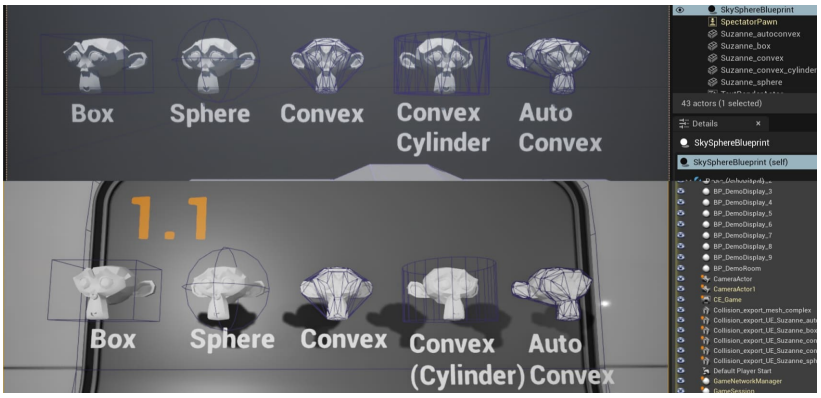


Figura 4.4: Diverse tipologie di Collisioni

Inoltre, sono stati adattati i materiali delle mesh (fig. 4.5) per garantire la massima realtà visiva.

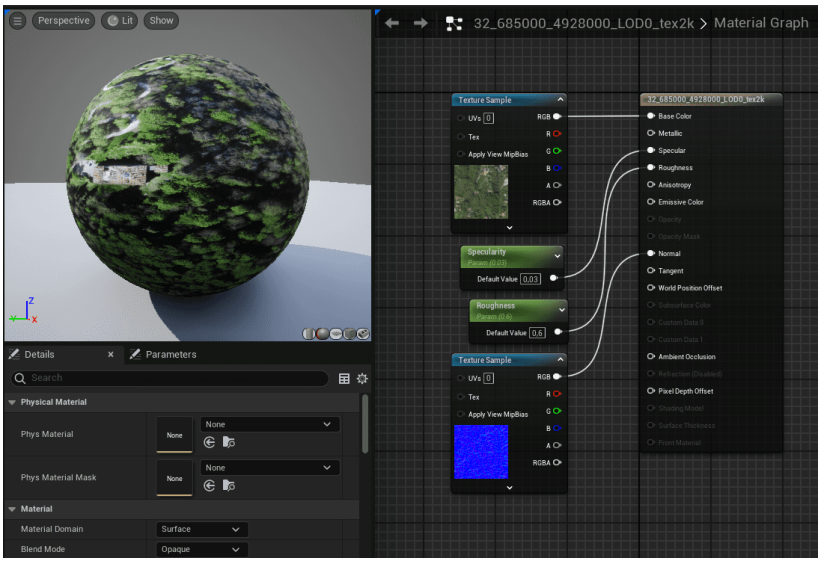


Figura 4.5: Materiale esempio del DTM

4.3 Creazione della Scena Statica del DT

Con tutte le mesh 3D importate, si procede con la composizione della scena che rappresenta il Digital Twin statico della città (fig. 4.6). Grazie alla georeferenziazione delle mesh e alla suddivisione del modello in tile di 500x500 metri, l'importazione e l'allineamento delle mesh avvengono senza necessità di trasformazioni aggiuntive. Questo passaggio consolida la base per la visualizzazione del Digital Twin, rendendo l'ambiente 3D pronto per ulteriori sviluppi e implementazioni dinamiche.

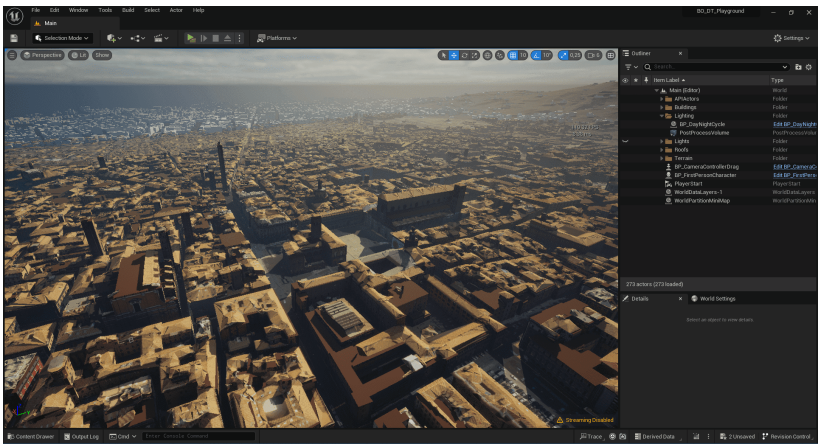


Figura 4.6: Previsualizzazione centro Bologna in Unreal

Capitolo 5

Sviluppo della dinamicità del Digital Twin

Contents

Mostriamo brevemente com'è stata sviluppata la parte dinamica del Digital Twin, sintetizzando il lavoro svolto dal mio collega.

5.1 Infrastruttura di Unreal e architettura del progetto

L'infrastruttura di Unreal Engine fornisce il fondamento su cui si costruisce il Digital Twin, introducendo concetti chiave come GameMode, Controller e GameState per gestire la logica di "gioco", l'input dell'utente e lo stato globale del "gioco".

Il design utilizzato (fig. 5.1) sfrutta la potente architettura di Unreal per supportare un'ampia gamma di funzionalità interattive e dinamiche all'interno del Digital Twin, dalla gestione della navigazione alla simulazione del ciclo giorno/notte, fino all'integrazione di dati esterni.

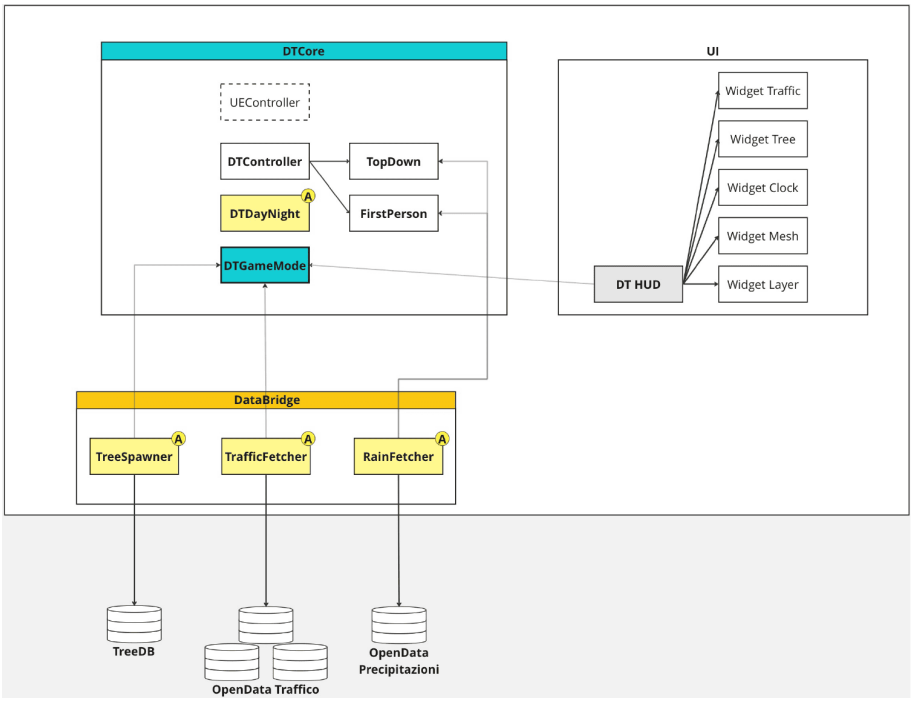


Figura 5.1: Diagramma dell'architettura

5.1.1 Navigazione

La navigazione è uno degli aspetti più importanti del Digital Twin, in quanto deve offrire agli utenti la possibilità di esplorare virtualmente la città di Bologna nella maniera più semplice e intuitiva possibile. L'implementazione prevista comprende sia una navigazione in prima persona (fig. 5.2), per una maggiore immersività, che una navigazione dall'alto (fig. 5.3), simile a quella offerta da applicazioni di mappe online.

Figura 5.2: Navigazione in prima persona

Figura 5.3: Navigazione dall'alto

Questa doppia modalità permette agli utenti di interagire con l'ambiente virtuale in modi diversi, aumentando l'accessibilità e l'usabilità del Digital Twin.

5.1.2 Ciclo Giorno/Notte

L'introduzione del ciclo giorno/notte dinamico arricchisce ulteriormente l'esperienza virtuale, aggiungendo realismo all'ambiente simulato (fig. 5.4). Questa caratteristica non solo migliora l'aspetto visivo del Digital Twin ma apre anche la strada a simulazioni e studi sull'impatto dell'illuminazione naturale e sulle performance energetiche degli edifici, fornendo così uno strumento prezioso per la pianificazione urbana e lo studio degli spazi cittadini.

(a) Alba (b) Mezzogiorno

(c) Tramonto (d) Sera

Figura 5.4: Ciclo giorno/notte

Per rendere l'ambiente più realistico, sono stati aggiunti elementi come **SkyLight**, **SkyAtmosphere**, **ExponentialHeightFog**, **VolumetricCloud** e una **DirectionalLight** per simulare la luna.

5.2 Collegamento con dati esterni

L'abilità di integrare dati esterni è fondamentale per mantenere il Digital Twin aggiornato e dinamico. Attraverso l'uso di API REST e altri meccanismi di integrazione, il Digital Twin può riflettere cambiamenti reali nell'ambiente urbano, come la crescita degli alberi, le condizioni meteorologiche e il traffico cittadino. Questo collegamento diretto con dati reali trasforma il Digital Twin da un semplice modello statico a un sistema vivo e in costante evoluzione, che può servire da piattaforma per simulazioni ambientali, studi sull'impatto urbano e la gestione delle emergenze.

5.2.1 Alberi

L'integrazione degli alberi è realizzata tramite un attore denominato *TreeSpawner*, che si interfaccia con il database precedentemente menzionato (3.5) per recuperare dati su di essi, principalmente la loro tipologia, la posizione e la dimensione. Ciò permette di rappresentare fedelmente gli alberi all'interno della scena digitale di Bologna, posizionandoli in base alle loro coordinate reali. In figura 5.5 possiamo vedere una visualizzazione di una parte dei Giardini Margherita.

Figura 5.5: Visualizzazione alberi spawnati

L'uso di **InstancedStaticMeshComponent** minimizza l'impatto sulle prestazioni, consentendo la renderizzazione efficiente di un grande numero di alberi.

Gli alberi non sono solo statici ma possono essere interrogati per ottenere informazioni dettagliate (fig. 5.6), migliorando l'interattività e la ricchezza informativa del modello.

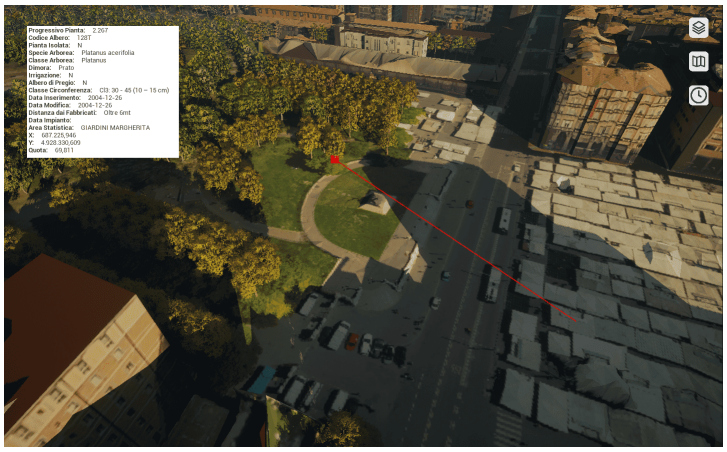


Figura 5.6: Ottenimento albero selezionato e widget

5.2.2 Precipitazioni

La rappresentazione delle precipitazioni nel Digital Twin utilizza il sistema di particelle Niagara di Unreal Engine. Questo approccio dinamico permette di simulare pioggia realistica che segue la camera dell'utente, creando l'illusione che piova sull'intera città (fig. 5.7).

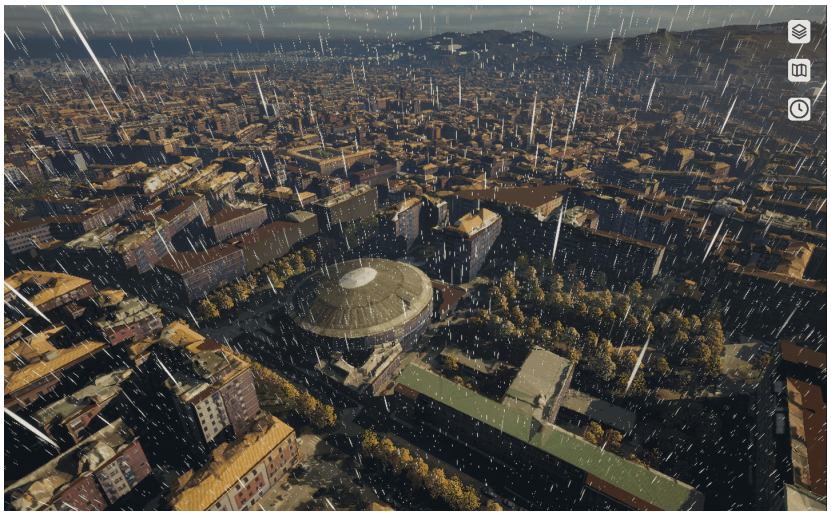


Figura 5.7: Visualizzazione della pioggia

L'attore responsabile della gestione delle precipitazioni, *RainFetcher*, ottiene dati aggiornati sulle condizioni meteorologiche da Bologna OpenData. Questi dati vengono poi utilizzati per regolare l'intensità e la distribuzione della pioggia nel Digital Twin, arricchendo ulteriormente la simulazione con aspetti climatici realistici.

5.2.3 Traffico

Il traffico viene simulato nel Digital Twin attraverso un attore chiamato *TrafficFetcher*, che raccoglie dati sul traffico in tempo reale da Bologna OpenData. Questi dati consentono di visualizzare lo stato del traffico nelle varie zone della città, offrendo una rappresentazione aggiornata della mobilità urbana (fig. 5.8).

Figura 5.8: Visualizzazione dati traffico: Via Rizzoli

La simulazione del traffico non si limita alla visualizzazione statica ma prevede la possibilità di estensioni future per prevedere i flussi di traffico e condurre simulazioni di vario tipo. Questo apre a nuove prospettive per l'analisi e la pianificazione urbana all'interno del Digital Twin.

Conclusioni

Il progetto Bologna Digital Twin ha dimostrato con successo il potenziale rivoluzionario della tecnologia Digital Twin applicata al contesto urbano. Attraverso un'approfondita analisi dei dati, l'adozione di tecniche all'avanguardia nella modellazione 3D e la creazione di un'interfaccia utente intuitiva, siamo stati in grado di sviluppare un prototipo che non solo permette una visualizzazione accuratamente della città di Bologna in un formato digitale, ma offre anche una piattaforma dinamica per la simulazione e l'analisi di vari scenari urbani.

Nonostante le sfide poste dalla disponibilità e qualità dei dati, nonché dalla complessità tecnica intrinseca nel replicare una città complessa come Bologna, il lavoro svolto ha gettato le basi per futuri sviluppi e miglioramenti.

Guardando al futuro, il Bologna Digital Twin si pone come un modello di riferimento per la realizzazione di gemelli digitali urbani, con l'obiettivo di rendere le nostre città più resilienti, vivibili e sostenibili.

Bibliografia

[1] 51World. *51World*. URL: https://www.51vr.com.au/.

[2] B Danette Allen. *Digital Twins and Living Models at NASA*. Presentazione al Digital Twin Summit. 2021. URL: https://ntrs.nasa.gov/citations/20210023699.

[3] Federico Andrucci. *Sviluppo ed implementazione di un Digital Twin della città di Bologna*. In Tesi Magistrale in Ingegneria Informatica, Università di Bologna. 2024.

[4] M. Angelidou. «The role of smart city characteristics in the plans of fifteen cities». In: *Journal of Urban Technology* 24.4 (2017), pp. 3–28. DOI: 10.1080/10630732.2017.1348880.

[5] The B1M. *How China Cloned Shanghai*. 2020. URL: https://www.youtube.com/watch?v=hOJZhsNtB6g.

[6] Comune di Bologna. *Bologna avrà un Gemello digitale*. 2023. URL: https://www.comune.bologna.it/notizie/gemello-digitale.

[7] Comune di Bologna. *Bologna Open Data*. URL: https://opendata.comune.bologna.it/pages/home/.

[8] Comune di Bologna. *Bologna Open Data - Alberi in Manutenzione*. URL: https://opendata.comune.bologna.it/explore/dataset/alberi-manutenzioni.

[9] Comune di Bologna. *Bologna Open Data - Edifici Volumetrici*. URL: https://opendata.comune.bologna.it/explore/dataset/c_a944ctc_edifici_pl.

[10] Comune di Bologna. *Comune di Bologna*. URL: https://www.comune.bologna.it/home.

[11] Università di Bologna. *Università di Bologna*. URL: https://www.unibo.it/it.

[12] Gary Bradski. *OpenCV Library*. https://opencv.org/. 2023.

[13] Buildmedia. *Buildmedia's Digital Twin of Wellington | Build: Architecture 2021*. 2021. URL: https://www.youtube.com/watch?v=Y-0m9GHl86I.

[14] CINECA. *CINECA*. URL: https://www.cineca.it/.

[15] Google Cloud. *Elevation API*. URL: https://developers.google.com/maps/documentation/elevation/overview.

[16] ISTI - CNR. *MeshLab*. Ver. 2023.12. 2023. URL: https://www.meshlab.net/.

[17] Paolo Corradeghini. *DTM VS DSM VS DEM*. URL: https://3dmetrica.it/dtm-dsm-dem/.

[18] The Rust Project Developers. *Rust Programming Language*. 2023. URL: https://www.rust-lang.org/.

[19] Eleobert. *dbscan: A C++ Implementation of the DBSCAN Algorithm*. https://github.com/Eleobert/dbscan. 2023.

[20] ESRI. *Projection files for CAD datasets*. URL: https://desktop.arcgis.com/en/arcmap/latest/manage-data/cad/projection-files-for-cad-datasets.htm.

[21] Epic Games. *Unreal Engine*. Ver. 5.3. 2024. URL: https://www.unrealengine.com.

[22] Michael Garland. «Surface Simplification Using Quadric Error Metrics». In: (1997).

[23] GDAL. *OSGeo - Python*. URL: https://gdal.org/api/python/osgeo.html.

[24] Daniel Girardeau-Montaut. *Cloud Compare*. Ver. 2.12.4. 2022. URL: https://github.com/CloudCompare/CloudCompare.

[25] Edward Glaessgen e D. Stargel. «The Digital Twin Paradigm for Future NASA and U.S. Air Force Vehicles». In: *53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*. 2012. DOI: 10.2514/6.2012-1818.

[26] Michael Grieves. *Digital Twin: Manufacturing Excellence through Virtual Factory Replication*. Presentato alla conferenza University of Michigan. White paper. 2002.

[27] Fondazione Bruno Kessler. *Fondazione Bruno Kessler*. URL: https://www.fbk.eu/it/.

[28] Juan Linietsky; Ariel Manzur. *Godot*. Ver. 4.2.1. 2023. URL: https://godotengine.org/.

[29] NanoFLANN Developers. *NanoFLANN: a C++ header-only library for Nearest Neighbor (NN) search with KD-trees*. https://github.com/jlblancoc/nanoflann. 2023.

[30] PDAL Contributors. *PDAL - Point Data Abstraction Library*. https://pdal.io/. 2023.

[31] Comune di Perugia. *Digital Twin Comune di Perugia*. 2023. URL: https://digitaltwin.comune.perugia.it/.

[32] Comune di Perugia. *Digital Twin Comune di Perugia*. 2023. URL: https://www.youtube.com/watch?v=2LA_rqaQbb8.

[33] Ton Roosendaal. *Blender*. Ver. 4.1.0. 2023. URL: https://www.blender.org/.

[34] Benjamin Schleich et al. «Shaping the digital twin for design and production engineering». In: *CIRP Annals* 66.1 (2017), pp. 141–144. DOI: 10.1016/j.cirp.2017.04.040.

[35] Jonathan Richard Shewchuk. *Triangle: A Two-Dimensional Quality Mesh Generator and Delaunay Triangulator*. https://www.cs.cmu.edu/~quake/triangle.html. 2005.

[36] Evolved software. *TreeIt*. URL: http://www.evolved-software.com/treeit/treeit.

[37] Fei Tao et al. «Digital Twin-driven Product Design, Manufacturing and Service with Big Data». In: *The International Journal of Advanced Manufacturing Technology* 94.9-12 (2019), pp. 3563–3576. DOI: 10.1007/s00170-017-0233-1.

[38] Fei Tao et al. «Digital Twin-driven product design, manufacturing and service with big data». In: *The International Journal of Advanced Manufacturing Technology* 94.9-12 (2018), pp. 3563–3576. DOI: 10.1007/s00170-017-0233-1.

[39] Unity Technology. *Unity*. Ver. 2023.1.5. 2023. URL: https://unity.com/.

[40] Wikipedia. *Aerial photography*. URL: https://en.wikipedia.org/wiki/Aerial_photography.

[41] Wikipedia. *American Society for Photogrammetry and Remote Sensing*. URL: https://en.wikipedia.org/wiki/American_Society_for_Photogrammetry_and_Remote_Sensing.

[42] Wikipedia. *Bing Maps*. URL: https://en.wikipedia.org/wiki/Bing_Maps.

[43] Wikipedia. *Canny edge detector*. URL: https://en.wikipedia.org/wiki/Canny_edge_detector.

[44] Wikipedia. *Computer vision*. URL: https://en.wikipedia.org/wiki/Level_of_detail_(computer_graphics).

[45] Wikipedia. *Corner detection*. URL: https://en.wikipedia.org/wiki/Corner_detection.

[46] Wikipedia. *Database*. URL: https://en.wikipedia.org/wiki/Database.

[47] Wikipedia. *DBSCAN*. URL: https://en.wikipedia.org/wiki/DBSCAN.

[48] Wikipedia. *Delaunay triangulation*. URL: https://en.wikipedia.org/wiki/Delaunay_triangulation.

[49] Wikipedia. *Digital image processing*. URL: https://en.wikipedia.org/wiki/Digital_image_processing.

[50] Wikipedia. *Esri grid*. URL: https://en.wikipedia.org/wiki/Esri_grid.

[51] Wikipedia. *FBX*. URL: https://en.wikipedia.org/wiki/FBX.

[52] Wikipedia. *File Transfer Protocol*. URL: https://en.wikipedia.org/wiki/File_Transfer_Protocol.

[53] Wikipedia. *Gaussian blur*. URL: https://en.wikipedia.org/wiki/Gaussian_blur.

[54] Wikipedia. *GeoTIFF*. URL: https://en.wikipedia.org/wiki/GeoTIFF.

[55] Wikipedia. *Google Earth*. URL: https://en.wikipedia.org/wiki/Google_Earth.

[56] Wikipedia. *Google Maps*. URL: https://en.wikipedia.org/wiki/Google_Maps.

[57] Wikipedia. *Greedy triangulation*. URL: https://en.wikipedia.org/wiki/Greedy_triangulation.

[58] Wikipedia. *Internet of things*. URL: https://en.wikipedia.org/wiki/Internet_of_things.

[59] Wikipedia. *JPEG*. URL: https://en.wikipedia.org/wiki/JPEG.

[60] Wikipedia. *k-d tree*. URL: https://en.wikipedia.org/wiki/K-d_tree.

[61] Wikipedia. *LAS file format*. URL: https://en.wikipedia.org/wiki/LAS_file_format.

[62] Wikipedia. *Leonardo (supercomputer)*. URL: https://it.wikipedia.org/wiki/Leonardo_(supercomputer).

[63] Wikipedia. *Level of detail (computer graphics)*. URL: https://en.wikipedia.org/wiki/Level_of_detail_(computer_graphics).

[64] Wikipedia. *Lidar*. URL: https://en.wikipedia.org/wiki/Lidar.

[65] Wikipedia. *Marching cubes*. URL: https://en.wikipedia.org/wiki/Marching_cubes.

[66] Wikipedia. *Mathematical morphology*. URL: https://en.wikipedia.org/wiki/Mathematical_morphology.

[67] Wikipedia. *Nearest neighbor search*. URL: https://en.wikipedia.org/wiki/Nearest_neighbor_search.

[68] Wikipedia. *PalaDozza*. URL: https://it.wikipedia.org/wiki/PalaDozza.

[69] Wikipedia. *Pixel connectivity*. URL: https://en.wikipedia.org/wiki/Pixel_connectivity.

[70] Wikipedia. *Poisson's equation*. URL: https://en.wikipedia.org/wiki/Poisson%27s_equation.

[71] Wikipedia. *Python (programming language)*. URL: https://en.wikipedia.org/wiki/Python_(programming_language).

[72] Wikipedia. *REST*. URL: https://en.wikipedia.org/wiki/REST.

[73] Wikipedia. *Secure Shell*. URL: https://en.wikipedia.org/wiki/Secure_Shell.

[74] Wikipedia. *Shapefile*. URL: https://en.wikipedia.org/wiki/Shapefile.

[75] Wikipedia. *STL (file format)*. URL: https://en.wikipedia.org/wiki/STL_(file_format).

[76] Wikipedia. *Texture atlas*. URL: https://en.wikipedia.org/wiki/Texture_atlas.

[77] Wikipedia. *Three.js*. URL: https://en.wikipedia.org/wiki/Three.js.

[78] Wikipedia. *UV mapping*. URL: https://en.wikipedia.org/wiki/UV_mapping.

[79] Wikipedia. *WebGL*. URL: https://en.wikipedia.org/wiki/WebGL.

[80] Wikipedia. *World Geodetic System*. URL: https://en.wikipedia.org/wiki/World_Geodetic_System.

[81] WiseTown. *WiseTown*. URL: https://wise.town/.

[82] Sarah Wray. *How a small German town is using an advanced digital twin*. 2020. URL: https://cities-today.com/how-a-small-german-town-is-using-an-advanced-digital-twin/.

[83] Y. Zheng, S. Yang e J. C. P. Cheng. «Digital Twin-based Smart Production Management and Control Framework for the Complex Product Assembly Plant». In: *Advanced Engineering Informatics* 43 (2020). DOI: 10.1016/j.aei.2019.101043.

Ringraziamenti

Desidero esprimere la mia profonda gratitudine alla Prof.ssa Serena Morigi, non solo per avermi offerto l'opportunità di lavorare a questo progetto ma anche per il suo indispensabile sostegno durante il suo svolgimento.

Un sentito ringraziamento va alla mia tutor, Ing. Antonella Guidazzoli, per aver creduto nel nostro lavoro e per averci motivato a conseguire risultati sempre migliori. Ringrazio anche Silvano e Daniele per il loro supporto in diverse fasi del progetto; collaborare con colleghi tanto capaci quanto simpatici è stato un vero piacere. Un ringraziamento speciale va a tutti gli altri membri del team del DT e alle nostre compagne d'ufficio, Simona e Giorgia, per il loro prezioso contributo.

Con tutto il cuore, voglio ringraziare la mia famiglia e, in particolare, mia madre Nirvana, per il loro incondizionato sostegno in questi anni di studio. Un grazie speciale va anche alla mia ragazza Melissa che mi ha aiutato più di ogni altro durante i momenti difficili, dimostrandosi un sostegno fondamentale.

Un ulteriore ringraziamento speciale è destinato al mio amico e collega, di questo e di molti altri progetti, Federico così agli altri miei compagni universitari Gabriele, Michele, Lorenzo e Andrea, che ormai considero come fratelli.

Infine, desidero ringraziare i miei amici di lunga data Giorgia, Debora, Daniela e Massimiliano per il loro sostegno psicologico e per essere sempre stati pronti a condividere un momento di relax e conversazione davanti a una birra.