**UNIVERSITÀ POLITECNICA DELLE MARCHE**

**FACOLTÀ DI INGEGNERIA**

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea in Ingegneria dell'Informazione per Videogame e Realtà Virtuale





**TESI DI LAUREA**

# Progettazione e Sviluppo di una Pipeline Multimodale per Applicazioni di Realtà Virtuale

# Design and Development of a Multimodal Pipeline for Virtual Reality Applications


Relatore

Prof. Riccardo Rosati

Correlatore

Dott. Tommaso Pantaloni

Candidato

Samuele d'Arienzo



**ANNO ACCADEMICO 2025/2026**

# Indice

**Elenco delle figure** **iii**

**Elenco delle tabelle** **iv**

**Elenco dei listati** **v**

**1 Introduzione** **1**
1.1 Contesto del progetto . . . 1
1.2 Obiettivi . . . 2
1.3 Struttura della tesi . . . 2

**2 Stato dell'arte** **3**
2.1 Metodi tradizionali di creazione di contenuti 3D . . . 3
2.1.1 Modellazione manuale . . . 3
2.1.2 Generazione procedurale . . . 4
2.1.3 Scansione 3D e fotogrammetria . . . 5
2.1.4 Asset store e librerie predefinite . . . 5
2.2 Generazione di contenuti 3D tramite intelligenza artificiale . . . 6
2.2.1 Approcci alla generazione text-to-3D . . . 6
2.2.2 Servizi e piattaforme commerciali . . . 8
2.2.3 Rappresentazioni e formati di output . . . 9
2.3 Lavori correlati e posizionamento del lavoro . . . 10

**3 Requisiti e progettazione del sistema** **13**
3.1 Scenario applicativo, requisiti e vincoli . . . 13
3.2 Scelte tecnologiche e criteri progettuali . . . 15
3.2.1 Tecnologie open-source e controllo della pipeline . . . 16
3.2.2 Riconoscimento vocale e Whisper . . . 16
3.2.3 Scelta del modello generativo . . . 18
3.3 Architettura del sistema . . . 19

**4 Implementazione della pipeline multimodale** **21**
4.1 Acquisizione dell'input testuale . . . 21
4.2 Acquisizione dell'input vocale . . . 21
4.3 Servizio di generazione 3D . . . 22
4.3.1 Containerizzazione con Docker . . . 23

4.3.2 Esposizione tramite API REST . . . . . 23
4.4 Comunicazione client-server . . . . . 24
4.4.1 Invio del prompt . . . . . 24
4.4.2 Ricezione del modello . . . . . 24
4.5 Importazione runtime del modello . . . . . 26
4.5.1 Evoluzione della soluzione di importazione: da OBJ a GLB . . . . . 26
4.5.2 Integrazione di glTFast . . . . . 27

**5 Gestione dei contenuti e interazione in Unity** **29**
5.1 Persistenza dei contenuti . . . . . 29
5.2 Organizzazione in stanze . . . . . 30
5.3 Manipolazione dei modelli . . . . . 31
5.4 Adattamento multipiattaforma . . . . . 32

**6 Valutazione sperimentale e discussione** **33**
6.1 Obiettivi della valutazione . . . . . 33
6.2 Ambiente sperimentale . . . . . 33
6.3 Scenari di prova . . . . . 34
6.4 Metriche e protocollo sperimentale . . . . . 34
6.5 Analisi quantitativa dei risultati . . . . . 35
6.6 Analisi qualitativa dei modelli generati . . . . . 37
6.7 Discussione dei risultati . . . . . 38

**7 Conclusioni** **40**

**Bibliografia** **41**

# Elenco delle figure


2.1 Confronto visivo tra diverse rappresentazioni tridimensionali: mesh poligonali, nuvole di punti, NeRF e 3D Gaussian Splatting. . . . . 9
2.2 Principali formati per la memorizzazione e lo scambio di contenuti tridimensionali. . . . . 10

3.1 Schermata iniziale per la creazione, selezione e gestione delle stanze. . . . . 14
3.2 Esempio di modello 3D generato e inserito nell'ambiente virtuale dell'applicazione. . . . . 14
3.3 Flusso di utilizzo dell'applicazione . . . . . 14
3.4 Architettura generale del sistema e flusso di comunicazione tra il client Unity e il backend di generazione. . . . . 20

4.1 Flusso di elaborazione dell'input vocale. . . . . 22
4.2 Flusso di invio del prompt dal client Unity al backend di generazione. . . . . 24
4.3 Distribuzione della memoria allocata prima e dopo il caricamento di un modello OBJ. . . . . 26
4.4 Dettaglio dell'utilizzo del Managed Heap prima e dopo il caricamento del modello OBJ. . . . . 27

5.1 Esempio di selezione e manipolazione di un modello 3D mediante il gizmo di trasformazione. . . . . 31

6.1 Tempo medio di elaborazione del backend per ciascuno scenario di prova. Le linee verticali indicano l'intervallo compreso tra il tempo minimo e il tempo massimo osservati. . . . . 36
6.2 Scomposizione del tempo end-to-end per ciascuno scenario della pipeline integrata. . . . . 37
6.3 Esempi di output generati da Shap-E: P01 (a simple chair), P08 (a hammer), P07 (a teapot with a handle and a spout) e P10 (a bicycle). . . . . 38

# Elenco delle tabelle


3.1 Requisiti funzionali del sistema . . . . . 15
3.2 Requisiti non funzionali del sistema . . . . . 15
3.3 Configurazioni della famiglia Whisper descritte da Radford et al. . . . . 18

4.1 Parametri principali della configurazione di campionamento utilizzata con Shap-E . . . . . 22

6.1 Scenari di prova utilizzati per la valutazione sperimentale. . . . . 34
6.2 Tempi di elaborazione del backend Shap-E per i dieci scenari di prova, calcolati su tre esecuzioni per ciascun prompt. . . . . 35
6.3 Tempi misurati nella pipeline integrata. . . . . 36
6.4 Statistiche riepilogative dei tempi. . . . . 36

# Elenco dei listati


4.1 Implementazione dell'endpoint HTTP per la generazione e la restituzione del modello 3D . . . . . 23
4.2 Corpo JSON della richiesta HTTP inviata al backend . . . . . 24
4.3 Gestione della richiesta HTTP e ricezione del modello . . . . . 25
4.4 Caricamento a runtime di un modello GLB e creazione del relativo oggetto Unity 28
5.1 Salvataggio dello stato di un modello . . . . . 30
5.2 Caricamento dei modelli associati alla stanza corrente . . . . . 30
5.3 Selezione di un modello e associazione al gizmo di trasformazione . . . . . 31

CAPITOLO 1

# Introduzione

## 1.1 Contesto del progetto

Il presente lavoro descrive il progetto sviluppato nell'ambito del tirocinio curricolare svolto presso Ubisive S.r.l., azienda attiva in ambiti che comprendono lo sviluppo di videogiochi e di applicazioni di realtà virtuale (VR) e realtà aumentata (AR) e l'impiego di tecnologie di intelligenza artificiale. L'attività si è concentrata in particolare sull'integrazione di tecniche di generazione automatica di contenuti tridimensionali all'interno di un'applicazione sviluppata con Unity e destinata all'utilizzo anche in VR.

La produzione di contenuti tridimensionali costituisce una componente centrale nello sviluppo di applicazioni interattive e immersive. Nelle pipeline tradizionali di sviluppo, gli asset utilizzati all'interno di applicazioni interattive vengono generalmente creati e preparati prima dell'esecuzione, per essere successivamente convertiti e integrati nell'ambiente runtime [Lear et al., 2019]. Tali contenuti possono essere ottenuti mediante la modellazione, l'acquisizione da sorgenti reali o l'utilizzo di contenuti già disponibili.

Negli ultimi anni, lo sviluppo dell'intelligenza artificiale generativa ha introdotto nuove modalità di produzione dei contenuti digitali, estendendosi progressivamente anche all'ambito tridimensionale. I modelli generativi possono infatti utilizzare informazioni di diversa natura, tra cui descrizioni testuali e immagini, per produrre nuove rappresentazioni 3D, riducendo la necessità che ogni contenuto venga realizzato interamente attraverso una pipeline di modellazione tradizionale [Liu et al., 2024]. Questo sviluppo rende possibile considerare la generazione non soltanto come uno strumento utilizzato durante la produzione dell'applicazione, ma anche come una componente capace di rispondere a richieste formulate dinamicamente dall'utente.

L'incontro tra queste tecniche e le tecnologie immersive apre quindi la possibilità di rendere gli ambienti virtuali maggiormente adattabili al contesto e alle richieste degli utenti. La letteratura recente sull'integrazione tra intelligenza artificiale generativa e VR evidenzia infatti un crescente interesse verso la creazione dinamica, multimodale e personalizzata dei contenuti [Rahimi et al., 2025]. Nell'ambito della realtà estesa (XR) sono già state sperimentate pipeline nelle quali modelli generativi vengono utilizzati direttamente durante l'esperienza immersiva per produrre contenuti tridimensionali coerenti con le richieste o con il contesto dell'utente [Santarnecchi et al., 2025].

In questo quadro si inserisce il progetto di tirocinio, nel quale è stata esplorata la possibilità di introdurre un processo di generazione dinamica in cui un nuovo modello 3D possa

essere richiesto dall'utente durante l'esecuzione, a partire da una descrizione dell'oggetto desiderato.

La possibilità di generare un contenuto tridimensionale durante l'esecuzione modifica il ruolo tradizionale degli asset all'interno dell'applicazione. Affinché questo processo possa essere utilizzato concretamente, la generazione deve quindi inserirsi all'interno del normale funzionamento dell'applicazione, fino a rendere il contenuto prodotto disponibile e utilizzabile dall'utente.

## 1.2 Obiettivi

L'obiettivo generale del progetto è stato la realizzazione di un prototipo che consentisse di integrare la generazione dinamica di contenuti tridimensionali all'interno di un'applicazione Unity destinata alla VR.

A partire da questo obiettivo generale, il lavoro ha riguardato la progettazione e lo sviluppo di un flusso applicativo completo, dalla formulazione della richiesta fino alla disponibilità del modello generato nell'ambiente virtuale. Particolare attenzione è stata dedicata alle modalità di interazione con il sistema, sia nella formulazione della richiesta sia nell'utilizzo dei contenuti generati all'interno dell'ambiente virtuale.

L'attività non ha quindi avuto come obiettivo lo sviluppo di nuovi modelli di intelligenza artificiale, ma l'integrazione di tecnologie esistenti all'interno di una soluzione applicativa. Le scelte architetturali, i requisiti del sistema e le modalità concrete con cui tali obiettivi sono stati tradotti nell'implementazione vengono approfonditi nei capitoli successivi.

## 1.3 Struttura della tesi

La tesi è organizzata in modo da descrivere progressivamente il contesto tecnologico, la progettazione del sistema, la sua implementazione e la successiva valutazione sperimentale. Dopo il presente capitolo introduttivo, il Capitolo 2 presenta lo stato dell'arte relativo alla creazione di contenuti tridimensionali, considerando sia gli approcci tradizionali sia le più recenti tecniche di generazione basate sull'intelligenza artificiale, e analizza alcuni lavori correlati al progetto. Il Capitolo 3 descrive lo scenario applicativo, i requisiti e i principali vincoli del sistema, approfondendo le scelte tecnologiche adottate e l'architettura complessiva della soluzione. Il Capitolo 4 approfondisce l'implementazione della pipeline multimodale, analizzando l'acquisizione dell'input testuale e vocale, il servizio di generazione, la comunicazione tra client e backend e il caricamento a runtime dei modelli. Il Capitolo 5 è dedicato alla gestione dei contenuti all'interno di Unity, comprendendo l'inserimento nella scena, la persistenza dei contenuti, l'organizzazione dei modelli e le modalità di manipolazione nei diversi ambienti di esecuzione. Il Capitolo 6 presenta la valutazione sperimentale del sistema, integrando misure quantitative dei tempi di elaborazione con un'analisi qualitativa dei modelli generati. Infine, il Capitolo 7 riassume i risultati raggiunti, evidenzia i limiti della soluzione sviluppata e individua possibili sviluppi futuri.

# CAPITOLO 2

# Stato dell'arte

La creazione di contenuti tridimensionali rappresenta una fase centrale nello sviluppo di applicazioni grafiche, interattive e immersive. Tradizionalmente, la produzione di asset 3D è stata affrontata attraverso tecniche quali la modellazione manuale e la generazione procedurale [Fukaya et al., 2025; Smelik et al., 2014], la scansione tridimensionale e la fotogrammetria [Bernardini e Rushmeier, 2002; Remondino e El-Hakim, 2006], oppure mediante l'utilizzo di contenuti disponibili in librerie e marketplace [Foxman, 2019]. Questi approcci rispondono a esigenze differenti e rimangono ampiamente utilizzati, ma presentano caratteristiche che possono limitarne l'impiego quando il contenuto deve essere prodotto dinamicamente durante l'esecuzione di un'applicazione a partire da una richiesta formulata direttamente dall'utente.

Negli ultimi anni lo sviluppo di modelli generativi ha introdotto modalità differenti di produzione dei contenuti 3D, nelle quali descrizioni testuali o altre forme di condizionamento possono essere utilizzate per ottenere automaticamente nuove rappresentazioni tridimensionali [Lee et al., 2024]. Tali sistemi modificano quindi il rapporto tra la richiesta dell'utente e la produzione dell'asset: invece di selezionare un contenuto già disponibile o definirne esplicitamente geometria e parametri, diventa possibile delegare parte del processo a un modello generativo. Tuttavia, la generazione del contenuto costituisce soltanto una fase del problema quando il risultato deve essere utilizzato all'interno di un'applicazione interattiva, poiché assumono rilevanza altre caratteristiche come il tipo di output prodotto e la successiva gestione del modello.

## 2.1 Metodi tradizionali di creazione di contenuti 3D

### 2.1.1 Modellazione manuale

La modellazione manuale rappresenta uno degli approcci tradizionali alla creazione di contenuti tridimensionali. Attraverso software dedicati, come Blender, Autodesk Maya o 3ds Max [Blender Foundation, s.d.; Autodesk, s.d.-a; Autodesk, s.d.-b], l'utente costruisce e modifica direttamente la geometria del modello mediante gli strumenti messi a disposizione dall'ambiente di modellazione. Nel caso delle mesh poligonali, ad esempio, la superficie è definita attraverso vertici, spigoli e facce che possono essere progressivamente creati e modificati fino a ottenere la forma desiderata. La modellazione può partire da primitive geometriche semplici, che vengono successivamente trasformate e dettagliate attraverso operazioni di editing.

Questo approccio permette un controllo diretto sulla geometria e consente al modellatore di adattare il contenuto alle esigenze specifiche dell'applicazione. Tale controllo richiede tuttavia un intervento umano significativo. È un'attività dispendiosa in termini di tempo e richiede competenze specialistiche [Hossain et al., 2023].

### 2.1.2 Generazione procedurale

La generazione procedurale costituisce un approccio alla creazione di contenuti tridimensionali nel quale la geometria viene prodotta automaticamente attraverso un insieme di procedure, regole e parametri definiti dallo sviluppatore. Un modello procedurale può essere descritto come un programma che, a partire da un insieme di parametri in ingresso, genera una determinata forma; modificando tali parametri è quindi possibile ottenere numerose varianti senza modellare manualmente ciascun asset [Hossain et al., 2023]. Questo principio è stato applicato a diversi tipi di contenuti, tra cui terreni, vegetazione, edifici e interi ambienti virtuali [Smelik et al., 2014].

Le modalità con cui tali procedure vengono definite dipendono dal contenuto da generare. Un esempio è costituito dai sistemi basati su grammatiche, nei quali un insieme di regole determina progressivamente la struttura del risultato. Tra questi rientrano gli L-system, originariamente introdotti per descrivere matematicamente i processi di crescita degli organismi vegetali. Un L-system parte da una configurazione iniziale e applica ripetutamente un insieme di regole di riscrittura: a ogni iterazione alcuni elementi vengono sostituiti secondo regole prestabilite, producendo strutture progressivamente più articolate. La sequenza ottenuta può successivamente essere interpretata geometricamente, associando agli elementi alcune operazioni di disegno e trasformazione. Questo meccanismo rende gli L-system particolarmente adatti alla generazione procedurale di forme caratterizzate da processi di crescita e da strutture gerarchiche come alberi, rami e altre forme di vegetazione [Prusinkiewicz e Lindenmayer, 1990].

Un'altra tecnica ampiamente utilizzata è il rumore procedurale, impiegato per introdurre variazioni irregolari ma controllate all'interno di superfici e forme generate algoritmicamente. Un esempio è il rumore di Perlin, introdotto come funzione capace di produrre variazioni continue e dall'aspetto naturale, evitando l'effetto completamente disordinato tipico di valori casuali indipendenti [Perlin, 1985]. In termini intuitivi, punti vicini dello spazio tendono infatti ad assumere valori correlati, producendo transizioni graduali anziché variazioni brusche. Questa proprietà lo rende adatto a simulare fenomeni e strutture naturali, come irregolarità del terreno, nuvole, rocce o variazioni superficiali [Lagae et al., 2010].

Il paradigma procedurale è utilizzato anche in strumenti professionali dedicati alla produzione di contenuti digitali, tra cui Houdini. Il software adotta un approccio basato su delle reti di nodi, nelle quali ogni nodo rappresenta un'operazione di generazione o trasformazione della geometria e il collegamento tra i nodi definisce il flusso con cui tali operazioni vengono applicate. In questo modo il processo di modellazione non viene rappresentato soltanto dal risultato finale, ma anche dalla sequenza di operazioni che lo ha prodotto. Modificando i parametri di un nodo o una fase precedente della rete, le modifiche possono propagarsi automaticamente alle operazioni successive, consentendo di rigenerare il risultato senza ricostruirlo da zero [SideFX, s.d.]. Houdini, quindi, rappresenta bene il vantaggio della modellazione procedurale rispetto alla costruzione manuale di ogni singolo asset.

Rispetto alla modellazione manuale, la generazione procedurale consente quindi di automatizzare una parte significativa della produzione e risulta particolarmente efficace quando è possibile descrivere preventivamente la famiglia di contenuti da ottenere. Il limite rispetto allo scenario considerato nel presente lavoro risiede però proprio in questa necessità: regole, procedure e parametri devono essere definiti a priori dallo sviluppatore. Il sistema

può generare variazioni all'interno dello spazio previsto dalla procedura, ma non interpreta autonomamente una richiesta libera dell'utente.

### 2.1.3 Scansione 3D e fotogrammetria

Un approccio differente rispetto alla modellazione manuale e alla generazione procedurale consiste nell'acquisire la geometria di un oggetto o di un ambiente reale e ricostruirne la rappresentazione tridimensionale digitale. Tra le tecniche impiegate a questo scopo rientrano la scansione 3D e la fotogrammetria. In entrambi i casi il contenuto non viene creato a partire da una descrizione astratta, ma deriva dall'osservazione di un soggetto fisicamente esistente.

La scansione 3D comprende diverse tecniche di acquisizione della forma di una superficie. Sistemi basati, ad esempio, su laser, luce strutturata o misure del tempo di volo permettono di ottenere informazioni sulla distanza dei punti della superficie rispetto al sensore. Un sistema laser è un dispositivo che emette un fascio di luce direzionale verso un oggetto e ne rileva la risposta per ricavare informazioni. Nei sistemi a luce strutturata, un pattern luminoso noto viene proiettato sulla superficie dell'oggetto; la deformazione osservata da una o più telecamere consente di stimare la profondità e ricostruire la geometria tridimensionale. Il tempo di volo, invece, è una tecnica che stima la distanza misurando quanto tempo un impulso luminoso impiega a raggiungere una superficie e tornare indietro. Poiché una singola acquisizione può descrivere soltanto la parte dell'oggetto visibile dal punto di osservazione, la ricostruzione completa può richiedere più scansioni effettuate da posizioni differenti. I dati così ottenuti devono quindi essere allineati e integrati per costruire una rappresentazione unica della superficie [Bernardini e Rushmeier, 2002].

La fotogrammetria ricostruisce invece la struttura tridimensionale a partire da immagini bidimensionali acquisite da punti di vista differenti. La presenza degli stessi elementi in più fotografie consente di determinare la posizione nello spazio attraverso relazioni geometriche tra le diverse viste. Una tipica pipeline di ricostruzione image-based comprende quindi l'acquisizione di più immagini, l'individuazione delle corrispondenze e la successiva ricostruzione della geometria tridimensionale. Rispetto a sistemi di scansione specializzati, questo approccio può essere realizzato anche utilizzando comuni fotocamere digitali e presenta quindi vantaggi in termini di portabilità e costo dell'acquisizione [Remondino e El-Hakim, 2006].

Scansione 3D e fotogrammetria risultano particolarmente utili quando l'obiettivo è ottenere una rappresentazione digitale di oggetti e ambienti reali. Questa caratteristica costituisce però anche il limite principale rispetto allo scenario considerato nel presente lavoro: entrambe presuppongono l'esistenza di un oggetto reale da acquisire. Non consentono quindi, da sole, di produrre durante l'esecuzione dell'applicazione un oggetto nuovo e arbitrario a partire da una richiesta formulata dall'utente.

### 2.1.4 Asset store e librerie predefinite

Un'ulteriore modalità di reperimento dei contenuti tridimensionali consiste nel riutilizzo di asset già realizzati e distribuiti attraverso asset store e librerie online. Queste piattaforme consentono agli sviluppatori di cercare e acquisire modelli esistenti, evitando di dover produrre da zero ogni contenuto necessario all'applicazione. Un esempio direttamente legato allo sviluppo con Unity è lo Unity Asset Store, che mette a disposizione asset 3D appartenenti a categorie quali personaggi, ambienti, oggetti di scena, vegetazione e veicoli [Unity Technologies, s.d.].

Un'altra piattaforma rilevante è Sketchfab, utilizzata per la pubblicazione, visualizzazione e distribuzione di modelli tridimensionali. La piattaforma mette a disposizione un'ampia rac-

colta di contenuti scaricabili e supporta anche l'accesso programmatico ai modelli [Sketchfab, s.d.].

Un'impostazione analoga è adottata da TurboSquid, un marketplace specializzato nella distribuzione di modelli 3D destinati, tra gli altri ambiti, a videogiochi e applicazioni VR/AR. La piattaforma raccoglie modelli gratuiti o a pagamento, disponibili in diversi formati utilizzati negli strumenti di produzione 3D e nei motori grafici [TurboSquid, s.d.].

Il principale vantaggio di queste soluzioni consiste quindi nella possibilità di riutilizzare rapidamente contenuti già disponibili, riducendo il lavoro necessario alla loro produzione. Rispetto allo scenario considerato nel presente lavoro, tuttavia, la scelta dell'utente rimane vincolata all'insieme degli asset pubblicati nel catalogo della piattaforma.

Nel complesso, gli approcci tradizionali analizzati presentano caratteristiche e vantaggi differenti, ma richiedono una o più di queste condizioni: tempo di produzione, competenze specialistiche, procedure definite preventivamente, un soggetto reale da acquisire oppure la disponibilità di un asset già esistente. Tali condizioni non si conciliano con l'obiettivo di produrre dinamicamente, durante l'esecuzione dell'applicazione, un nuovo contenuto tridimensionale a partire da una richiesta dell'utente. Questo limite motiva l'interesse verso gli approcci di generazione 3D basati sull'intelligenza artificiale, descritti nella sezione successiva.

## 2.2 Generazione di contenuti 3D tramite intelligenza artificiale

Lo sviluppo recente dei modelli generativi ha introdotto nuove modalità di produzione dei contenuti tridimensionali, nelle quali il risultato può essere ottenuto a partire da informazioni di input di diversa natura, tra cui descrizioni testuali o immagini [Liu et al., 2024]. Rispetto agli approcci tradizionali descritti nella sezione precedente, questi sistemi non richiedono necessariamente che la geometria venga costruita manualmente, che sia disponibile un oggetto reale da acquisire o che la varietà dei risultati sia interamente determinata da regole procedurali definite a priori. Il contenuto viene invece prodotto da un modello generativo condizionato dall'informazione fornita in ingresso.

Nello stesso panorama delle tecniche recenti per la rappresentazione tridimensionale rientrano anche approcci orientati principalmente alla ricostruzione di scene a partire da osservazioni visive, come i Neural Radiance Fields (NeRF) e il 3D Gaussian Splatting [Mildenhall et al., 2020; Kerbl et al., 2023]. Tali tecniche rispondono quindi a un problema differente rispetto alla generazione di nuovi contenuti a partire da una descrizione testuale, che costituisce il principale ambito di interesse del presente lavoro. Le relative rappresentazioni verranno riprese nella Sezione 2.2.3 in relazione ai possibili output tridimensionali.

Nel contesto di questo lavoro assume particolare rilevanza la generazione text-to-3D, nella quale una descrizione in linguaggio naturale viene utilizzata per guidare la produzione di un contenuto tridimensionale [Lee et al., 2024]. Questo paradigma permette di avvicinare la richiesta dell'utente alla generazione dell'asset, rendendo possibile produrre nuovi contenuti durante l'esecuzione di un'applicazione senza doverli predisporre singolarmente in fase di sviluppo. Le soluzioni disponibili differiscono tuttavia per il processo generativo adottato, per le modalità con cui vengono rese accessibili e per il tipo di rappresentazione tridimensionale prodotta; aspetti che assumono particolare importanza quando il risultato deve essere successivamente utilizzato all'interno di un'applicazione interattiva.

### 2.2.1 Approcci alla generazione text-to-3D

La generazione text-to-3D presenta difficoltà specifiche rispetto alla sintesi di immagini bidimensionali. Il contenuto prodotto deve infatti descrivere una struttura coerente nello

spazio tridimensionale. Per i modelli text-to-image sono disponibili grandi quantità di coppie immagini-testo con cui addestrare i modelli, mentre la disponibilità di dati tridimensionali associati a descrizioni testuali è più limitata [Lee et al., 2024]. Questa differenza ha favorito lo sviluppo di strategie diverse: alcuni metodi sfruttano modelli text-to-image già addestrati come guida per costruire una rappresentazione tridimensionale, mentre altri apprendono direttamente a generare particolari rappresentazioni 3D.

Dal punto di vista del processo generativo, una famiglia di metodi particolarmente rilevante per i sistemi più recenti è costituita dai modelli di diffusione. I Denoising Diffusion Probabilistic Models descritti da Ho et al. sono modelli generativi nei quali, durante l'addestramento, viene considerato un processo che perturba progressivamente i dati introducendo rumore. Il modello apprende quindi un processo inverso capace di rimuovere gradualmente tale rumore e di ricostruire campioni appartenenti alla distribuzione dei dati. In fase di generazione è così possibile partire da una configurazione iniziale di rumore e ottenere progressivamente un nuovo campione. Il lavoro originale di Ho, Jain e Abbeel riguarda principalmente la generazione di immagini [Ho et al., 2020]. Il principio della diffusione è stato successivamente applicato anche ad altre rappresentazioni e costituisce una componente importante di diversi sistemi per la generazione tridimensionale.

Un esempio significativo è DreamFusion, proposto da Poole et al., che utilizza un modello di diffusione text-to-image preaddestrato come conoscenza a priori per ottimizzare una rappresentazione tridimensionale. Il metodo parte da una rappresentazione 3D inizializzata casualmente e ne genera viste da differenti angolazioni; tali immagini vengono confrontate indirettamente con la distribuzione appresa dal modello di diffusione 2D, e l'informazione ottenuta viene utilizzata per aggiornare progressivamente la rappresentazione tridimensionale. In questo modo il sistema può realizzare una sintesi text-to-3D senza richiedere l'addestramento di un modello su un grande dataset tridimensionale [Poole et al., 2023]. L'utilizzo di un modello 2D come guida presenta il vantaggio di poter sfruttare la disponibilità e la capacità semantica dei moderni sistemi text-to-image, ma richiede un processo di ottimizzazione della rappresentazione tridimensionale per ogni nuovo contenuto.

Una strategia alternativa ai metodi basati sull'ottimizzazione consiste nell'utilizzare modelli generativi addestrati a produrre una rappresentazione tridimensionale mediante un processo di campionamento, senza dover ottimizzare una nuova rappresentazione per ogni contenuto. Un esempio è Point-E, che nella sua configurazione principale genera inizialmente un'immagine a partire dal testo e utilizza successivamente un modello di diffusione condizionato dall'immagine per produrre una nuvola di punti tridimensionale [Nichol et al., 2022].

Nella stessa direzione, ma adottando una rappresentazione differente, si colloca Shap-E, sviluppato da Jun e Nichol, già coautori di Point-E. Il modello è progettato per produrre contenuti tridimensionali a partire da un condizionamento testuale oppure visivo e, a differenza di Point-E, non genera direttamente una nuvola di punti. Shap-E produce invece i parametri di una funzione implicita che rappresenta il contenuto tridimensionale e che può successivamente essere utilizzata per ricavarne differenti rappresentazioni, tra cui mesh o NeRF [Jun e Nichol, 2023].

Il funzionamento di Shap-E può essere descritto, in modo semplificato, come un processo articolato in due fasi. Durante l'addestramento, un encoder impara innanzitutto ad associare un asset tridimensionale ai parametri di una funzione implicita che ne rappresenta il contenuto. Successivamente, un modello di diffusione condizionale viene addestrato sulle rappresentazioni prodotte dall'encoder. In fase di generazione il modello può quindi produrre direttamente una nuova rappresentazione di questo tipo, senza richiedere un processo di ottimizzazione specifico per ogni singolo prompt [Jun e Nichol, 2023].

Nel caso della generazione text-to-3D, il prompt fornito dall'utente viene utilizzato

come condizionamento del processo di diffusione. Il testo orienta quindi la generazione verso una rappresentazione tridimensionale semanticamente compatibile con la descrizione richiesta. Il risultato del campionamento non corrisponde ancora al file geometrico finale, ma a una rappresentazione interna dell'asset dalla quale è possibile ricavare il contenuto 3D. Questa caratteristica è particolarmente rilevante per l'utilizzo del modello all'interno di una pipeline applicativa, poiché introduce una fase di decodifica tra la generazione e l'ottenimento dell'asset effettivamente utilizzabile.

Un approccio più recente è rappresentato da TRELLIS, proposto da Xiang et al. Il modello introduce una rappresentazione denominata Structured LATent (SLAT), progettata per descrivere all'interno di una struttura unificata informazioni relative sia alla geometria sia all'aspetto dell'asset. La generazione può essere condizionata da testo o da immagini. La rappresentazione SLAT può essere successivamente decodificata in differenti tipi di rappresentazioni, tra cui mesh, Radiance Fields e 3D Gaussian Splatting [Xiang et al., 2025]. TRELLIS rappresenta quindi un esempio dell'evoluzione recente verso sistemi capaci di combinare generazione condizionata a una maggiore flessibilità nella rappresentazione finale del contenuto.

### 2.2.2 Servizi e piattaforme commerciali

Accanto ai modelli resi disponibili open source dalla comunità scientifica, la generazione automatica di contenuti tridimensionali è offerta anche tramite servizi commerciali accessibili attraverso interfacce web o API (Application Programming Interface). In questo caso l'utente non deve necessariamente configurare ed eseguire localmente il modello generativo: l'elaborazione viene effettuata dall'infrastruttura del fornitore e il risultato viene successivamente reso disponibile all'utente o all'applicazione che ha effettuato la richiesta. Questo approccio può semplificare l'accesso alla generazione 3D e l'integrazione nei flussi applicativi, a fronte di una maggiore dipendenza dal servizio esterno e dalle modalità di utilizzo previste dal fornitore.

Un esempio è Meshy, piattaforma che mette a disposizione funzionalità di generazione sia text-to-3D sia image-to-3D. Nel caso di generazione da testo, l'utente inserisce una descrizione dell'oggetto e il servizio produce inizialmente una mesh, che può essere successivamente sottoposta a una fase di raffinamento o texturing. Le stesse funzionalità sono accessibili anche tramite API basate sullo stile architetturale Representational State Transfer (REST), consentendo di integrare il servizio all'interno di applicazioni esterne [Meshy, s.d.].

Una soluzione analoga è offerta da Tripo, che supporta la generazione di modelli a partire da descrizioni testuali o immagini. La piattaforma permette di inserire un prompt testuale, ottenere un modello tridimensionale e successivamente visualizzarlo, modificarlo o esportarlo. Anche Tripo mette a disposizione API dedicate alla generazione, rendendo possibile richiamare il servizio direttamente da codice invece di utilizzare esclusivamente l'interfaccia web [Tripo AI, s.d.]. Un altro servizio dello stesso tipo è Rodin, sviluppato da Hyper3D, che espone tramite API funzionalità text-to-3D e image-to-3D e permette di controllare diversi aspetti del processo, tra cui livello di dettaglio, caratteristiche della mesh e generazione dei materiali [Hyper3D, s.d.].

Questi servizi mostrano come la generazione 3D basata su intelligenza artificiale sia ormai disponibile non soltanto sotto forma di modelli sperimentali, ma anche attraverso piattaforme orientate all'utilizzo applicativo. L'esecuzione remota riduce la necessità di predisporre localmente l'infrastruttura computazionale richiesta dai modelli e le API possono facilitare l'integrazione in software esterni. Al tempo stesso, rispetto a una soluzione eseguita localmente, l'utilizzo di un servizio proprietario introduce una dipendenza dalle API, dalle condizioni economiche e operative del fornitore e dal livello di configurabilità reso disponibile.

Questi aspetti assumono rilevanza quando la generazione deve essere inserita stabilmente all'interno di una pipeline applicativa.

### 2.2.3 Rappresentazioni e formati di output

I sistemi di generazione tridimensionale possono produrre risultati espressi attraverso rappresentazioni differenti, e questa scelta influenza direttamente le modalità con cui il contenuto può essere trasferito, visualizzato e utilizzato all'interno di un'applicazione interattiva. Tra le rappresentazioni più comuni rientrano mesh poligonali, nuvole di punti e rappresentazioni neurali come i NeRF [Lee et al., 2024]. Modelli più recenti possono inoltre supportare più tipi di output a partire dalla stessa rappresentazione interna, come avviene in Shap-E e TRELLIS.

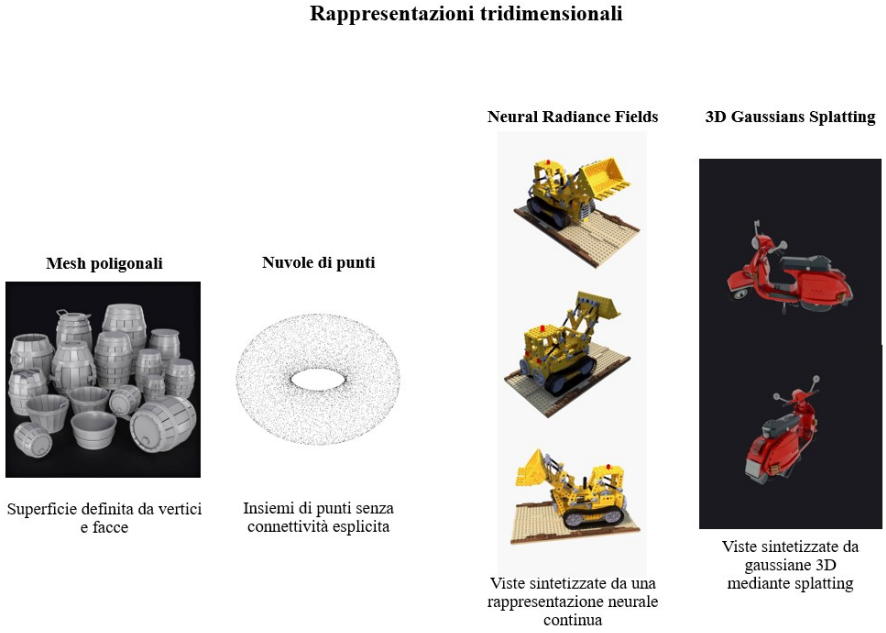



**Figura 2.1:** Confronto visivo tra diverse rappresentazioni tridimensionali: mesh poligonali, nuvole di punti, NeRF e 3D Gaussian Splatting.

**Fonti delle immagini, da sinistra:** enterstudio, Wooden Barrel | Base Mesh-VOL01, DeviantArt, licenza CC BY-NC-ND 3.0; Lucas Vieira, Point cloud torus, Wikimedia Commons, pubblico dominio; Mildenhall et al. (2020), sequenza Lego, fotogrammi estratti, modello 3D di Heinzelnisse, CC BY-NC; Renderbricks, LEGO® MOC Vespa P200 by The Arvo Brothers, SuperSplat, CC BY 4.0; viste acquisite dalla scena.

Le mesh poligonali descrivono la superficie di un oggetto attraverso vertici collegati in facce, generalmente triangolari. Questa rappresentazione è direttamente compatibile con il modello geometrico utilizzato dai motori grafici e consente di associare alla superficie materiali, texture e altre informazioni necessarie al rendering.

Una nuvola di punti rappresenta invece il contenuto tridimensionale mediante un insieme di posizioni nello spazio, eventualmente associate a proprietà aggiuntive. Rispetto a una mesh, questa rappresentazione non descrive esplicitamente la connettività tra i punti e quindi non definisce direttamente una superficie composta da facce.

Un approccio differente è rappresentato dai NeRF. In questo caso la scena non è descritta attraverso una geometria poligonale esplicita, ma mediante una funzione neurale continua che, a partire da una posizione tridimensionale e da una direzione di osservazione, restituisce informazioni relative alla densità volumetrica e al colore. Le nuove viste vengono generate interrogando tale funzione lungo i raggi della camera e applicando tecniche di volume

rendering [Mildenhall et al., 2020]. Una rappresentazione di questo tipo è particolarmente adatta alla sintesi di nuove viste, ma non corrisponde direttamente a un asset poligonale tradizionale. Di conseguenza, il suo utilizzo in un motore grafico richiede un sistema di rendering specifico oppure una conversione verso una rappresentazione geometrica più direttamente compatibile con le funzionalità del motore.

Anche rappresentazioni più recenti, come quelle basate su gaussiane 3D, seguono una logica diversa da quelle delle mesh: la scena viene descritta attraverso primitive distribuite nello spazio e renderizzate mediante tecniche di splatting [Kerbl et al., 2023].

Accanto al tipo di rappresentazione tridimensionale, assume rilevanza anche il formato utilizzato per memorizzare e scambiare il contenuto. La Figura 2.2 riassume alcuni dei principali formati utilizzati per questo scopo.

**Principali formati per la memorizzazione e lo scambio di contenuti tridimensionali**

| | |
|---|---|
| **OBJ** | Mesh e geometria 3D; materiali tramite file MTL separato. È un formato di scambio per modelli 3D statici |
| **STL** | Geometria triangolare, senza materiali o animazioni. È usato nella stampa 3D |
| **PLY** | Mesh e nuvole di punti con attributi; è diffuso in scansione 3D e computer vision. È usato in alcune implementazioni del 3D Gaussian Splatting |
| **PCD** | Nuvole di punti con attributi; usato in robotica, LiDAR e computer vision |
| **glTF** | Formato di scambio per asset e scene 3D; supporta geometria, materiali, texture, gerarchie e animazioni |
| **GLB** | Variante binaria di glTF; raccoglie asset e risorse associate in un unico file |

**Figura 2.2:** Principali formati per la memorizzazione e lo scambio di contenuti tridimensionali.

Per lo scambio di asset tra applicazioni può essere utilizzato, ad esempio, il formato GL Transmission Format (glTF), progettato per rappresentare scene tridimensionali comprendenti geometria, materiali, texture, trasformazioni e animazioni. La variante binaria GLB permette di raccogliere le diverse risorse dell'asset all'interno di un singolo file [Khronos Group, 2021]. In Unity, librerie come glTFast consentono inoltre di importare asset glTF e GLB direttamente durante l'esecuzione dell'applicazione.

Nel contesto di un'applicazione Unity, in cui il contenuto deve essere caricato durante l'esecuzione, inserito nella scena e successivamente manipolato, una rappresentazione basata su mesh risulta particolarmente adatta, poiché è direttamente compatibile con le strutture geometriche e con i normali meccanismi di rendering e interazione del motore. Rappresentazioni quali nuvole di punti, NeRF o gaussiane 3D richiedono invece modalità di rendering specifiche o ulteriori elaborazioni qualora sia necessario ottenere un asset poligonale tradizionale.

## 2.3 Lavori correlati e posizionamento del lavoro

L'integrazione tra tecniche di generazione automatica di contenuti tridimensionali e ambienti immersivi costituisce un ambito di ricerca recente, nel quale diversi lavori hanno iniziato ad affrontare il problema della produzione di asset durante l'utilizzo di applicazioni XR [Weng et al., 2024; Tong et al., 2025; Behravan e Gračanin, 2025; Santarnecchi et al., 2025; Kim et al., 2026]. Le soluzioni proposte differiscono per modalità di input, tecniche generative, dispositivi di destinazione e modalità di interazione con i contenuti prodotti. Alcuni sistemi risultano particolarmente rilevanti per il presente lavoro perché non si limitano alla genera-

zione isolata di un modello 3D, ma considerano anche il suo trasferimento, caricamento e utilizzo all'interno dell'ambiente immersivo.

Un precedente particolarmente vicino è Dream Mesh, proposto da Weng, Chiou e Do nel 2024. Il sistema realizza una pipeline speech-to-3D per un ambiente di Mixed Reality sviluppato in Unity e destinato a Meta Quest Pro. L'utente formula verbalmente la richiesta, che viene convertita in testo attraverso Meta Voice SDK e inviata a una macchina esterna responsabile della generazione mediante DreamFusion. Una volta completata l'elaborazione, il modello viene scaricato dal visore e inserito nell'ambiente, dove l'utente può modificarne posizione, dimensione e trasparenza. La separazione tra dispositivo XR e server generativo deriva dai requisiti computazionali della generazione. Una delle principali limitazioni riportate dagli autori è il tempo necessario alla produzione dell'asset, pari a circa 30-40 minuti nella configurazione sperimentata; inoltre, la possibilità di conservare i modelli generati per reimportarli successivamente viene indicata come possibile evoluzione del sistema [Weng et al., 2024].

Un approccio differente è adottato da MS2Mesh-XR, presentato da Tong et al. nel 2025. Il sistema combina schizzi realizzati direttamente nell'ambiente XR con informazioni vocali: lo schizzo fornisce indicazioni sulla struttura dell'oggetto, mentre l'input vocale contribuisce alla descrizione semantica. I due segnali vengono utilizzati per produrre un'immagine di riferimento, dalla quale viene successivamente ricostruita una mesh tridimensionale. Gli autori riportano tempi inferiori a 20 secondi e consentono la visualizzazione e manipolazione del risultato nella scena XR durante l'esecuzione. Rispetto a un'interazione basata esclusivamente sul linguaggio naturale, MS2Mesh-XR richiede quindi anche una bozza, offrendo in cambio un maggiore controllo sulla struttura desiderata dell'oggetto [Tong et al., 2025].

Un sistema ancora più vicino alla pipeline considerata nel presente lavoro è Matrix, presentato da Behravan e Gračanin nel 2025. Il framework è progettato per consentire la generazione di oggetti tridimensionali all'interno di un ambiente AR a partire da comandi vocali multilingua. L'input viene trascritto o tradotto mediante SeamlessM4T e successivamente elaborato da un modello linguistico, utilizzato per individuare gli oggetti e le relative caratteristiche descritte dall'utente. Il sistema verifica quindi la disponibilità di contenuti già presenti in un repository e, quando è necessaria una nuova generazione, utilizza Shap-E per produrre il modello tridimensionale. Dopo la generazione viene inoltre applicata una procedura di semplificazione della mesh, con l'obiettivo di ridurne la complessità e facilitarne l'utilizzo sul dispositivo AR. Il framework è stato sperimentato su Microsoft HoloLens 2; gli autori riportano tempi di generazione inferiori a 50 secondi. Una volta inseriti nell'ambiente, gli oggetti possono essere spostati, ruotati e ridimensionati mediante gesti delle mani [Behravan e Gračanin, 2025].

Un approccio complementare è proposto da MineVRA, un framework per la generazione di contenuti tridimensionali direttamente all'interno di ambienti VR. Il sistema adotta un approccio Human-in-the-Loop e context-aware, nel quale l'utente può formulare mediante testo o voce la descrizione dell'ambiente virtuale corrente. La pipeline generativa utilizza Stable Diffusion XL per produrre immagini candidate e InstantMesh per trasformare l'immagine selezionata dall'utente in un modello tridimensionale. Il contenuto ottenuto viene successivamente inserito nell'ambiente VR e può essere manipolato dall'utente. Gli autori confrontano inoltre gli asset generati con modelli recuperati da Sketchfab mediante uno studio con utenti, evidenziando i vantaggi della generazione contestuale in termini di personalizzazione e coerenza con l'ambiente, ma anche alcuni limiti rispetto agli asset tradizionali in termini di realismo e tempi di generazione [Santarnecchi et al., 2025].

L'evoluzione molto recente del settore ha portato a sistemi ancora più simili alla pipeline qui considerata. Voice-to-Mesh, pubblicato nel 2026 da Kim et al., realizza una pipeline end-to-end nella quale comandi vocali multilingua vengono trascritti mediante Whisper

e interpretati attraverso un modello linguistico; la richiesta risultante viene utilizzata per generare una mesh, successivamente normalizzata e caricata in un ambiente Mixed Reality su HoloLens 2. Il sistema permette inoltre di manipolare gli oggetti generati e di conservarli per un utilizzo successivo. Rispetto a MS2Mesh-XR non richiede uno schizzo come informazione aggiuntiva, avvicinandosi quindi maggiormente a una pipeline nella quale il linguaggio naturale costituisce direttamente l'origine della generazione [Kim et al., 2026].

Il confronto mostra quindi che l'integrazione tra generazione automatica di asset tridimensionali e tecnologie XR non costituisce un problema ancora privo di soluzioni. Al contrario, i lavori pubblicati negli ultimi anni indicano l'emergere di una linea di ricerca orientata alla creazione di contenuti durante l'utilizzo dell'ambiente immersivo, spesso attraverso modalità di interazione basate sul linguaggio naturale. Le differenze tra le soluzioni riguardano soprattutto la tecnica generativa adottata, la modalità di acquisizione dell'input, il dispositivo target e le funzionalità offerte dopo la generazione.

Il presente lavoro si colloca all'interno di questo scenario come progettazione e realizzazione di una specifica pipeline end-to-end nella quale input testuale e vocale convergono verso la generazione tramite Shap-E, eseguita esternamente al client Unity, e il modello risultante viene trasferito e importato a runtime nell'applicazione. Il sistema affronta inoltre la persistenza dei contenuti e delle relative trasformazioni, la loro organizzazione in insiemi distinti e la manipolazione in ambiente VR su Meta Quest 2. Il contributo non consiste pertanto nell'introduzione di un nuovo metodo di generazione 3D o di riconoscimento vocale, ma nell'integrazione delle diverse componenti in un prototipo funzionante e nell'analisi delle scelte e dei vincoli emersi durante la sua realizzazione.

# CAPITOLO 3

# Requisiti e progettazione del sistema

La progettazione del sistema è stata guidata dall'obiettivo di integrare la generazione dinamica di contenuti tridimensionali all'interno di un'applicazione Unity utilizzabile anche in VR. A partire dallo scenario applicativo sono stati quindi definiti i requisiti funzionali e i principali vincoli del prototipo, considerando in particolare le modalità di acquisizione della richiesta, le risorse disponibili sul dispositivo client e la necessità di rendere i contenuti generati utilizzabili durante l'esecuzione dell'applicazione. Tali esigenze hanno orientato sia la scelta delle tecnologie impiegate, con particolare riferimento al riconoscimento vocale e alla componente generativa, sia la definizione dell'architettura complessiva del sistema.

## 3.1 Scenario applicativo, requisiti e vincoli

Lo scenario applicativo prevede che l'utente possa creare e gestire contenuti tridimensionali senza che questi debbano essere predisposti durante lo sviluppo dell'applicazione. I modelli vengono invece generati durante l'esecuzione a partire da una descrizione dell'oggetto desiderato e successivamente inseriti nell'ambiente Unity. L'utilizzo dell'applicazione ha inizio con la selezione di una stanza esistente oppure con la creazione di una nuova stanza, intesa come contenitore logico attraverso cui mantenere separati insiemi differenti di modelli e il relativo stato persistente.

All'interno dell'ambiente, l'utente può formulare la richiesta attraverso due modalità alternative: inserendo direttamente una descrizione testuale oppure pronunciandola tramite il microfono. In entrambi i casi l'obiettivo è ottenere una rappresentazione testuale comune, utilizzabile dalle fasi successive della pipeline. Il prompt viene quindi inviato alla componente di generazione 3D, che è eseguita esternamente al client Unity; al termine dell'elaborazione, il modello prodotto viene trasferito all'applicazione e importato durante l'esecuzione. Una volta inserito nella scena, il contenuto può essere selezionato e manipolato mediante operazioni di traslazione, rotazione e ridimensionamento. Il modello e lo stato associato alle relative trasformazioni possono essere inoltre conservati, così da consentirne il ripristino nelle successive esecuzioni dell'applicazione.

Il flusso complessivo è sintetizzato nella Figura 3.3, che rappresenta le principali operazioni disponibili dal punto di vista dell'utente.

A partire da questo scenario sono stati definiti i requisiti funzionali del prototipo, cioè le funzionalità che il sistema deve mettere a disposizione per consentire il completamento dell'intero processo di utilizzo.




**Figura 3.1:** Schermata iniziale per la creazione, selezione e gestione delle stanze.

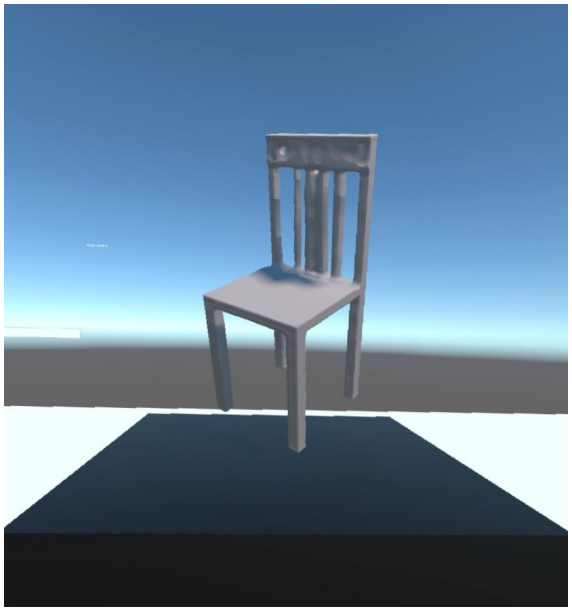


**Figura 3.2:** Esempio di modello 3D generato e inserito nell'ambiente virtuale dell'applicazione.

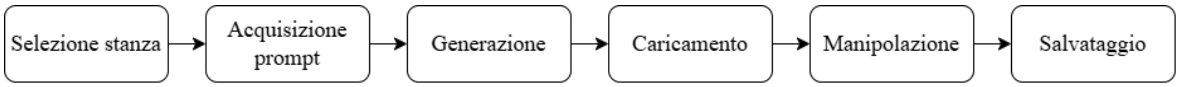



**Figura 3.3:** Flusso di utilizzo dell'applicazione

**Tabella 3.1:** Requisiti funzionali del sistema

| ID | Requisito funzionale |
|---|---|
| RF01 | Il sistema deve consentire all'utente di descrivere mediante testo l'oggetto da generare. |
| RF02 | Il sistema deve consentire di acquisire una richiesta vocale e convertirla in una rappresentazione testuale utilizzabile dalla pipeline. |
| RF03 | Il sistema deve consentire l'invio del prompt dal client Unity al servizio di generazione. |
| RF04 | Il sistema deve consentire il caricamento del modello 3D nella scena durante l'esecuzione dell'applicazione. |
| RF05 | Il sistema deve consentire la selezione e la manipolazione dei modelli mediante traslazione, rotazione e ridimensionamento. |
| RF06 | Il sistema deve consentire il salvataggio e il ripristino dello stato di una stanza tra esecuzioni diverse dell'applicazione. |
| RF07 | Il sistema deve consentire la creazione, la selezione e la rimozione di stanze distinte, ciascuna associata ai relativi contenuti. |

Accanto alle funzionalità descritte, la progettazione ha dovuto tenere conto di alcuni requisiti non funzionali e vincoli legati all'ambiente di esecuzione del prototipo. Tali esigenze sono riassunte nella Tabella 3.2.

**Tabella 3.2:** Requisiti non funzionali del sistema

| ID | Requisito non funzionale |
|---|---|
| RNF01 | Il client deve essere sviluppato in Unity e deve poter essere adattato all'esecuzione su Meta Quest 2. |
| RNF02 | Le operazioni computazionalmente onerose necessarie alla generazione non devono dipendere dalle risorse disponibili sul dispositivo VR. |
| RNF03 | Il client e la componente generativa devono poter comunicare tramite rete, mantenendo separate le rispettive responsabilità. |
| RNF04 | Il formato utilizzato per lo scambio dei modelli deve essere compatibile con il trasferimento in rete e con l'importazione a runtime in Unity. |
| RNF05 | I dati persistenti devono essere memorizzati in un'area accessibile all'applicazione anche nelle build destinate al dispositivo finale. |
| RNF06 | L'interazione deve essere adattabile alle modalità di input previste nell'ambiente immersivo. |

Tra i vincoli individuati, particolare rilevanza assume la necessità di separare la componente generativa dal client Unity, scelta che incide direttamente sulla successiva definizione dell'architettura del sistema.

## 3.2 Scelte tecnologiche e criteri progettuali

La definizione dei requisiti e dei vincoli ha orientato alcune scelte tecnologiche che hanno inciso direttamente sulla struttura del sistema. In particolare, è stato necessario individuare una soluzione per l'acquisizione vocale compatibile con il flusso basato su prompt testuali, una componente generativa che potesse essere integrata e controllata direttamente e, più in generale, tecnologie adatte ad una pipeline eseguibile anche in un ambiente separato dal client

Unity. Le decisioni adottate sono state quindi valutate non soltanto in base alle funzionalità offerte, ma anche considerando fattibilità dell'esecuzione, possibilità di integrazione, controllo della configurazione e dipendenza da servizi esterni.

### 3.2.1 Tecnologie open-source e controllo della pipeline

L'impiego di componenti open source ha avuto un ruolo rilevante nella realizzazione del sistema. Secondo la definizione dell'Open Source Initiative, la disponibilità del codice sorgente costituisce soltanto uno degli aspetti dell'open source: la licenza deve consentire anche l'utilizzo, la modifica e la realizzazione e distribuzione di opere derivate secondo le condizioni previste dalla licenza stessa. Questo distingue un progetto open source da un software proprietario il cui codice può eventualmente essere disponibile o utilizzabile gratuitamente, ma senza concedere gli stessi diritti di modifica e redistribuzione [Open Source Initiative, 2007].

Sia Shap-E sia Whisper sono distribuiti attraverso repository pubblici con licenza MIT [OpenAI, 2023; OpenAI, 2022]. Per il lavoro svolto, il vantaggio principale di questa disponibilità non consiste semplicemente nella possibilità di utilizzare gratuitamente i modelli, ma nella possibilità di eseguirli e integrarli direttamente nell'ambiente applicativo. In particolare, l'accesso al codice di Shap-E ha consentito di adattare il processo di generazione alle esigenze della pipeline, intervenendo sulla configurazione e sulla produzione dell'output senza dipendere dall'interfaccia resa disponibile da un servizio proprietario.

Il controllo diretto delle componenti contribuisce inoltre alla riproducibilità dell'ambiente software, poiché rende espliciti il codice utilizzato, le dipendenze e i parametri di configurazione. Tale caratteristica non garantisce da sola la completa riproducibilità dell'esecuzione, che rimane influenzata anche dall'hardware e dalle versioni delle librerie, ma riduce la dipendenza da elementi controllati esclusivamente da fornitori esterni.

Il vantaggio dell'open source va quindi interpretato, nel presente lavoro, soprattutto in termini di possibilità di modifica, controllo dell'esecuzione, riproducibilità e minore dipendenza da servizi proprietari, aspetti che hanno contribuito anche alla scelta della componente generativa adottata nella pipeline finale.

### 3.2.2 Riconoscimento vocale e Whisper

Il riconoscimento automatico del parlato, generalmente indicato come Automatic Speech Recognition (ASR), comprende le tecniche utilizzate per convertire un segnale contenente linguaggio parlato in una rappresentazione testuale. Il problema consiste quindi nell'individuare, a partire dalla sequenza temporale che costituisce il segnale audio, la sequenza di parole o unità linguistiche che meglio rappresenta quanto pronunciato. Nel sistema sviluppato questa elaborazione consente di ricondurre l'input vocale alla stessa rappresentazione testuale utilizzata dall'inserimento manuale, mantenendo invariato il resto della pipeline di generazione.

Nei sistemi tradizionali, gli Hidden Markov Models (HMM) venivano utilizzati per modellare l'evoluzione temporale del parlato attraverso una sequenza di stati, consentendo di rappresentare anche le variazioni nella durata con cui i suoni vengono pronunciati. I Gaussian Mixture Models (GMM) venivano invece impiegati nella modellazione acustica per stimare quanto le caratteristiche osservate nel segnale fossero compatibili con i diversi stati del modello. Con l'introduzione del deep learning, le Deep Neural Network (DNN) hanno progressivamente sostituito i GMM in questa funzione. Nei sistemi DNN-HMM, gli HMM continuano a descrivere la successione temporale dei suoni, mentre le DNN valutano quali stati siano più compatibili con il segnale audio osservato [Hinton et al., 2012].

L'evoluzione successiva ha portato allo sviluppo di sistemi end-to-end, nei quali una parte maggiore del processo di riconoscimento viene appresa all'interno di un'architettura neurale integrata. Rispetto alle architetture tradizionali, questi sistemi riducono la necessità di progettare e addestrare separatamente numerosi componenti specifici del riconoscimento vocale e apprendono a trasformare direttamente il segnale audio in una sequenza testuale, riducendo il numero di componenti intermedi separati. Le architetture end-to-end sono diventate uno degli approcci principali nell'ASR moderno e comprendono differenti strategie per gestire la relazione tra una sequenza audio e la relativa trascrizione [Prabhavalkar et al., 2024].

Uno dei problemi fondamentali in questo ambito riguarda l'allineamento tra segnale audio e testo. Durante l'addestramento solitamente è disponibile la trascrizione complessiva di un segmento audio, ma non necessariamente l'istante preciso in cui viene pronunciato ciascun carattere o ciascuna parola. Un contributo importante in questa direzione è rappresentato dalla Connectionist Temporal Classification (CTC), introdotta da Graves et al., che consente di addestrare reti neurali su sequenze non segmentate senza richiedere un allineamento esplicito tra ogni elemento dell'input e la corrispondente etichetta testuale. Questo principio ha contribuito allo sviluppo di sistemi nei quali l'allineamento tra audio e trascrizione può essere appreso direttamente durante l'addestramento [Graves et al., 2006].

La qualità di un sistema ASR dipende inoltre dalle condizioni nelle quali viene acquisito il segnale. Differenze nella pronuncia, velocità del parlato, caratteristiche dei parlanti, rumore ambientale e qualità del dispositivo di registrazione possono modificare significativamente l'input acustico pur lasciando invariato il contenuto linguistico. Un sistema di riconoscimento deve quindi essere in grado di estrarre informazioni sufficientemente robuste da produrre una trascrizione corretta anche in presenza di tale variabilità. Nei sistemi moderni questa capacità dipende in larga misura dai dati utilizzati per l'addestramento e dalla capacità del modello di generalizzare a condizioni differenti da quelle osservate durante l'apprendimento [Prabhavalkar et al., 2024].

Whisper è una famiglia di modelli ASR sviluppata da OpenAI. Il modello è stato presentato con il lavoro Robust Speech Recognition via Large-Scale Weak Supervision di Radford et al. L'obiettivo principale del lavoro è ottenere un sistema capace di mantenere buone prestazioni anche su audio proveniente da domini, parlanti e condizioni di registrazione differenti, senza richiedere un adattamento specifico per ciascun dataset. A questo scopo Whisper è stato addestrato su circa 680.000 ore di audio associato a trascrizioni, raccolte da fonti disponibili sul web e comprendenti sia dati in lingua inglese sia contenuti multilingua e dati per la traduzione vocale verso l'inglese [Radford et al., 2023].

Gli autori definiscono questo approccio large-scale weak supervision. Il termine weak supervision indica in questo caso che i dati di addestramento non provengono esclusivamente da dataset accuratamente annotati e verificati manualmente, ma comprendono grandi quantità di coppie audio-trascrizione reperite dal web e sottoposte a procedure automatiche di filtraggio. Questa scelta permette di aumentare notevolmente la quantità e la varietà dei dati disponibili, pur utilizzando annotazioni meno controllate rispetto a quelle di dataset costruiti interamente mediante trascrizioni verificate manualmente. La diversità dei dati comprende differenti condizioni di registrazione, parlanti, ambienti e lingue ed è utilizzata dagli autori con l'obiettivo di migliorare la capacità del modello di generalizzare a input non osservati durante l'addestramento [Radford et al., 2023].

Dal punto di vista architetturale, Whisper utilizza un Transformer di tipo encoder-decoder. Il segnale audio viene prima ricampionato a 16 kHz e trasformato in uno spettrogramma log-Mel, una rappresentazione che descrive come cambiano nel tempo le frequenze presenti nell'audio. L'encoder analizza questa rappresentazione e ne ricava le informazioni acustiche principali. Il decoder utilizza poi tali informazioni per generare progressivamente il testo

della trascrizione. Whisper elabora l'audio mediante segmenti della durata massima di circa 30 secondi [Radford et al., 2023].

Una caratteristica di Whisper è l'utilizzo della stessa architettura per più compiti di elaborazione del parlato. Oltre alla trascrizione, il modello è addestrato per riconoscere la lingua parlata, tradurre il parlato verso l'inglese e individuare segmenti privi di parlato. Il compito da svolgere viene rappresentato attraverso token speciali forniti al decoder: ad esempio, token distinti specificano la lingua e indicano se il modello deve effettuare una trascrizione oppure una traduzione [Radford et al., 2023].

Whisper è distribuito in diverse dimensioni, progettate per offrire diversi compromessi tra la capacità del modello e il costo computazionale. L'aumento della dimensione del modello comporta generalmente una maggiore capacità rappresentativa, ma richiede anche più memoria e una maggiore quantità di risorse computazionali durante l'inferenza. La disponibilità di più configurazioni consente quindi di scegliere il modello in funzione dei vincoli dell'ambiente di esecuzione e del compromesso desiderato tra prestazioni e costo computazionale [Radford et al., 2023].

**Tabella 3.3:** Configurazioni della famiglia Whisper descritte da Radford et al.

| Modello | Parametri |
|---|---|
| tiny | 39 milioni |
| base | 74 milioni |
| small | 244 milioni |
| medium | 769 milioni |
| large | 1.55 miliardi |

Nel prototipo sviluppato, è stato impiegato il modello tiny di Whisper. Il suo ruolo nella pipeline è limitato alla conversione del segnale audio in testo: una volta ottenuta la trascrizione, il risultato viene trattato come un normale prompt e può essere inoltrato alle successive fasi di generazione.

### 3.2.3 Scelta del modello generativo

La scelta del modello generativo è stata effettuata considerando requisiti più ampi della sola qualità visiva degli output. Il modello selezionato doveva infatti poter essere eseguito con le risorse disponibili, integrato in un servizio separato dal client Unity e inserito in un flusso capace di restituire un contenuto tridimensionale utilizzabile dall'applicazione. La valutazione ha quindi considerato congiuntamente fattibilità computazionale, possibilità di integrazione, controllo dell'esecuzione e caratteristiche dell'output prodotto.

Tra le alternative considerate, TRELLIS ha ricevuto particolare attenzione in quanto soluzione più recente e qualitativamente promettente, oltre che capace di supportare rappresentazioni tridimensionali differenti. Le prove effettuate durante il progetto hanno tuttavia evidenziato difficoltà nel completare l'intero processo nelle configurazioni hardware e software disponibili, in particolare per quanto riguarda le risorse richieste e le fasi successive alla generazione.

Sono state considerate anche soluzioni basate su servizi esterni di generazione 3D. Questi avrebbero permesso di delegare l'elaborazione all'infrastruttura del fornitore, riducendo i requisiti necessari per l'esecuzione locale. Tale impostazione avrebbe però introdotto una maggiore dipendenza da API, condizioni operative e modalità di utilizzo definite esternamente, riducendo il controllo diretto sulla componente generativa. Alla luce dei criteri discussi

nella sezione precedente, questa caratteristica risultava meno adatta all'impostazione scelta per il prototipo.

La soluzione finale è stata quindi basata su Shap-E. Il modello ha consentito di realizzare una pipeline end-to-end nella quale la generazione può essere eseguita all'interno di un backend direttamente controllato e il risultato può essere trasformato in una mesh successivamente trasferibile al client Unity. La scelta deve essere interpretata come un compromesso progettuale tra qualità del contenuto, requisiti computazionali e possibilità di integrazione: Shap-E non rappresenta necessariamente la soluzione più avanzata dal punto di vista della qualità geometrica, ma si è dimostrato compatibile con i vincoli del progetto e ha permesso di completare l'intero flusso applicativo.

## 3.3 Architettura del sistema

A partire dai requisiti individuati e dalle scelte tecnologiche descritte nelle sezioni precedenti, il sistema è stato organizzato secondo un'architettura client-server. La separazione riguarda in particolare la generazione tridimensionale, che richiede risorse computazionali non disponibili sul dispositivo VR, mentre le funzionalità direttamente legate all'interazione con l'utente e alla gestione dell'ambiente rimangono affidate al client Unity. L'architettura risultante è quindi composta da due macrocomponenti principali: il client Unity, responsabile dell'acquisizione dell'input, della gestione dell'interfaccia e dei contenuti tridimensionali nell'ambiente, e il backend generativo, responsabile dell'esecuzione del modello text-to-3D e della produzione dell'asset da restituire all'applicazione.

Il client costituisce il punto di accesso dell'utente alla pipeline. Le due modalità di input previste seguono inizialmente percorsi differenti ma convergono sulla stessa rappresentazione testuale: l'inserimento manuale produce direttamente il prompt, mentre nel percorso vocale la registrazione viene trascritta tramite Whisper prima di proseguire. Dopo questa convergenza, le fasi successive non dipendono più dalla modalità con cui la richiesta è formulata. Questa organizzazione consente di mantenere separata la responsabilità del riconoscimento vocale dalla generazione tridimensionale e di utilizzare un'unica interfaccia logica verso il backend.

Il backend generativo viene eseguito su una macchina esterna al dispositivo client e rende disponibile la generazione attraverso un'interfaccia di comunicazione HTTP (HyperText Transfer Protocol). Dal punto di vista architetturale, il contratto tra le due parti rimane volutamente semplice: il client invia una descrizione testuale dell'oggetto richiesto e il backend restituisce il contenuto tridimensionale prodotto. La generazione vera e propria è affidata a Shap-E, mentre le modalità concrete con cui il modello viene configurato, esposto come servizio e richiamato attraverso la rete appartengono al livello implementativo e vengono descritte nel Capitolo 4.

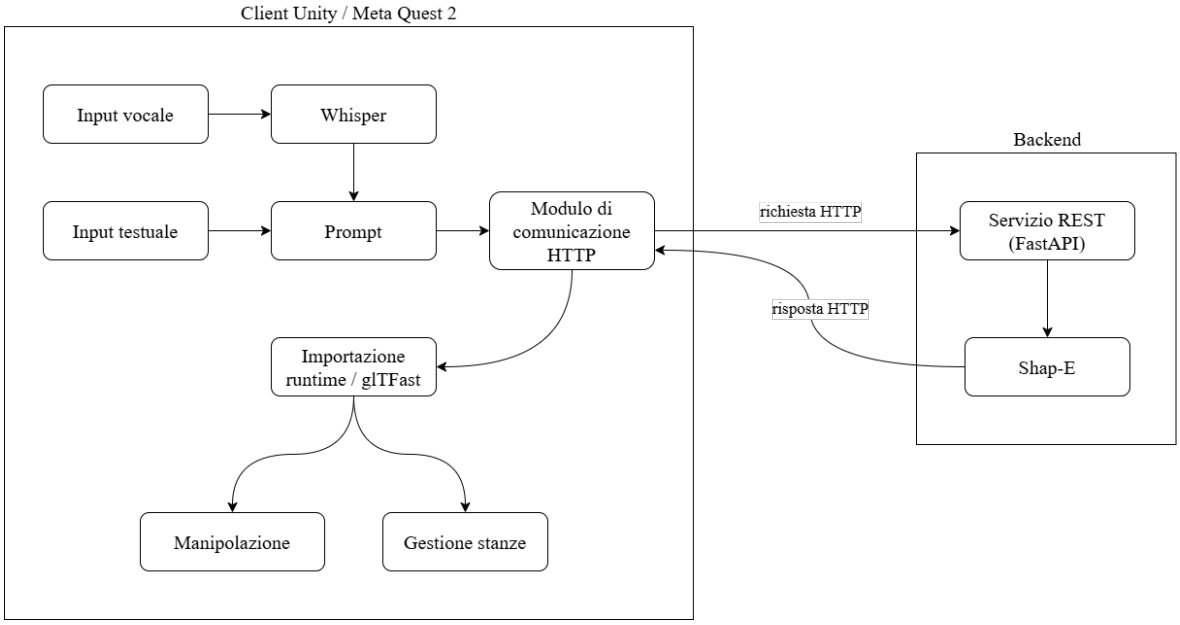



**Figura 3.4:** Architettura generale del sistema e flusso di comunicazione tra il client Unity e il backend di generazione.

# CAPITOLO 4

# Implementazione della pipeline multimodale

L'implementazione del sistema traduce l'architettura descritta nel capitolo precedente in una sequenza di componenti software. Il presente capitolo segue il percorso della richiesta dell'utente attraverso diverse fasi della pipeline, dall'acquisizione dell'input fino al caricamento del modello tridimensionale nell'applicazione. Per ciascuna fase vengono descritte le soluzioni adottate per realizzare concretamente le responsabilità individuate in fase di progettazione.

## 4.1 Acquisizione dell'input testuale

L'input testuale rappresenta la modalità più diretta per avviare la generazione. L'utente inserisce nell'apposito campo dell'interfaccia una descrizione dell'oggetto desiderato. Il contenuto viene memorizzato nello stato condiviso dell'applicazione, gestito dal GameManager, in modo da renderlo disponibile al componente responsabile della comunicazione con il backend. La modalità testuale produce quindi direttamente il prompt utilizzato nelle successive fasi della pipeline. La modalità vocale, descritta nella sezione seguente, converge sulla stessa rappresentazione testuale.

## 4.2 Acquisizione dell'input vocale

La modalità vocale estende il percorso testuale introducendo una fase di acquisizione e trascrizione prima della generazione. La sua funzione è produrre un prompt nella stessa forma prevista per l'inserimento diretto, permettendo così di riutilizzare senza modifiche le fasi successive della pipeline.

Nel client Unity l'interazione con la componente speech-to-text è gestita da WhisperClient. Lo script mantiene i riferimenti al componente WhisperManager, responsabile della trascrizione, e a un componente MicrophoneRecord, utilizzato per acquisire il segnale dal microfono. La logica di registrazione rimane quindi separata da Shap_eClient, che non deve distinguere tra prompt digitati e prompt ottenuti dalla voce. Nella configurazione utilizzata dal prototipo la lingua di trascrizione è impostata su inglese. La registrazione viene controllata esplicitamente dall'utente attraverso lo stesso pulsante, che alterna le operazioni di avvio e arresto.

All'arresto della registrazione, i campioni audio vengono passati in modo asincrono a WhisperManager per ottenere la trascrizione. Una volta completata l'elaborazione, il testo risultante viene mostrato nell'interfaccia e memorizzato nel GameManager, rendendolo disponibile come prompt per la generazione tridimensionale. Il percorso implementato è rappresentato nella figura 4.1.

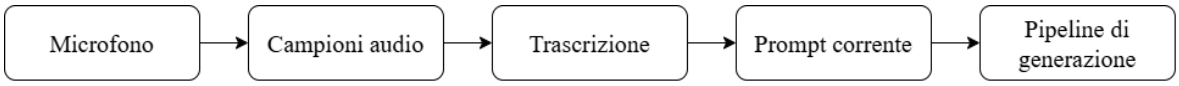



**Figura 4.1:** Flusso di elaborazione dell'input vocale.

## 4.3 Servizio di generazione 3D

La generazione tridimensionale viene eseguita attraverso un servizio separato dall'applicazione Unity. Shap-E, inizialmente utilizzabile come componente Python indipendente, è stato adattato affinché la generazione potesse essere avviata mediante una richiesta proveniente dal client e il risultato potesse essere restituito sotto forma di file GLB.

La funzione utilizzata dal backend riceve come parametro una stringa contenente il prompt e seleziona CUDA quando è disponibile una GPU compatibile. La configurazione utilizza il modello testuale text300M, il modello transmitter e la configurazione di diffusione prevista da Shap-E. Il prompt viene quindi utilizzato dal processo di campionamento per ottenere una rappresentazione latente, successivamente decodificata in una mesh triangolare. Nella configurazione finale utilizzata dal progetto, i principali parametri di generazione sono:

**Tabella 4.1:** Parametri principali della configurazione di campionamento utilizzata con Shap-E

| **Parametro** | **Valore adottato** |
|---|---|
| Batch size | 1 |
| Guidance scale | 15 |
| Karras steps | 64 |
| FP16 | Abilitato |
| sigma_min | $10^{-3}$ |
| sigma_max | 160 |

I parametri riportati in Tabella 4.1 definiscono la configurazione utilizzata durante il campionamento. Il batch size pari a 1 indica che per ogni richiesta viene generato un singolo modello, mentre guidance scale controlla l'intensità del condizionamento testuale durante il campionamento: valori maggiori tendono a indirizzare maggiormente la generazione verso il contenuto descritto dal prompt, riducendo però potenzialmente la qualità o la varietà del risultato. Nella configurazione adottata è stato utilizzato un valore pari a 15. Il parametro Karras steps determina il numero di passi utilizzati durante il processo di campionamento. Nella configurazione adottata è stato impostato a 64. Un numero maggiore di passi comporta un processo di campionamento più articolato e un conseguente aumento del tempo di elaborazione. L'opzione FP16 abilita l'utilizzo della precisione numerica a 16 bit durante l'elaborazione, riducendo il consumo di memoria rispetto alla precisione a 32 bit e risultando quindi adatta all'esecuzione del modello su GPU. I parametri sigma_min e sigma_max delimitano l'intervallo dei valori utilizzati durante il campionamento.

Una volta concluso il campionamento, la rappresentazione latente viene decodificata mediante decode_latent_mesh() e successivamente convertita in una mesh triangolare. Dalla

mesh risultante vengono estratti vertici e facce, utilizzati per costruire un oggetto Trimesh, che viene infine esportato in formato GLB.

### 4.3.1 Containerizzazione con Docker

Il backend è stato organizzato all'interno di un container Docker che costituisce l'ambiente di esecuzione completo della componente generativa. Il container comprende il codice utilizzato per eseguire Shap-E, il servizio FastAPI e le relative dipendenze software. L'ambiente utilizzato è basato su CUDA 11.8, tecnologia NVIDIA che consente di sfruttare l'accelerazione della GPU per le operazioni di calcolo richieste dal modello. La containerizzazione permette quindi di mantenere il backend separato dal client Unity e di raggruppare in un unico ambiente controllato sia l'applicazione server sia le relative dipendenze. Il client non deve conoscere la configurazione interna del container, ma interagisce con il backend esclusivamente attraverso l'interfaccia HTTP esposta da FastAPI.

### 4.3.2 Esposizione tramite API REST

Per rendere la componente generativa accessibile al client Unity senza introdurre una dipendenza diretta dall'implementazione Python, il backend espone un'interfaccia HTTP realizzata mediante FastAPI. Il servizio viene eseguito tramite Uvicorn, server utilizzato per rendere disponibile l'applicazione FastAPI sulla rete. L'interfaccia è stata mantenuta volutamente semplice e prevede un endpoint HTTP POST denominato /generate. Il corpo della richiesta contiene il campo testuale prompt. Nel backend tale struttura è rappresentata dalla classe PromptInput, definita tramite Pydantic e composta da un singolo campo di tipo stringa. Una volta ricevuto un input valido, l'endpoint estrae il contenuto del campo prompt e lo passa alla funzione generate_3d_model(). La funzione restituisce il percorso del file GLB prodotto; tale percorso viene quindi utilizzato per costruire una risposta mediante la classe FileResponse. Il contenuto tridimensionale viene pertanto restituito direttamente come corpo della risposta HTTP, con tipo application/octet-stream, cioè come file binario.

**Listato 4.1:** Implementazione dell'endpoint HTTP per la generazione e la restituzione del modello 3D

```
from fastapi import FastAPI
from pydantic import BaseModel
from shap_e.examples.sample_text_to_3d import generate_3d_model
from fastapi.responses import FileResponse
from time import perf_counter
import os

app = FastAPI()

class PromptInput(BaseModel):
    prompt: str

@app.post("/generate")
def generate(data: PromptInput):
    backend_start = perf_counter()
    output_path = generate_3d_model(data.prompt)
    backend_s = (perf_counter() - backend_start)
    print(f"[BENCHMARK] prompt={data.prompt} backend_s={backend_s:.2f}")
    return FileResponse(
        path=output_path,
        media_type="application/octet-stream",
        filename=os.path.basename(output_path)
    )
```

## 4.4 Comunicazione client-server

La comunicazione tra Unity e il backend è gestita da Shap_eClient attraverso UnityWebRequest. Il componente realizza sia l'invio del prompt sia la ricezione del modello e collega quindi la parte applicativa Unity al servizio FastAPI.

### 4.4.1 Invio del prompt

Una volta ottenuto un prompt valido, il client Unity deve trasferirlo al servizio di generazione. Questa operazione è gestita dal componente Shap_eClient, che utilizza la classe UnityWebRequest per costruire e inviare una richiesta HTTP di tipo POST. Il prompt viene inserito in una struttura dati con un unico campo testuale e successivamente serializzato in formato JavaScript Object Notation (JSON), come mostrato di seguito.

**Listato 4.2:** Corpo JSON della richiesta HTTP inviata al backend

```
{
    "prompt": "a simple chair"
}
```

Tale struttura corrisponde a quella prevista dal backend. Nel client, la rappresentazione JSON viene convertita in una sequenza di byte mediante codifica UTF-8 e associata alla richiesta attraverso la classe UploadHandlerRaw. Viene impostato l'header HTTP Content-Type al valore application/json, in modo che il servizio FastAPI possa interpretare correttamente il contenuto ricevuto. L'invio viene eseguito in Unity attraverso una coroutine. In questo modo l'applicazione può attendere il completamento della richiesta senza bloccare direttamente il normale ciclo di aggiornamento della scena.

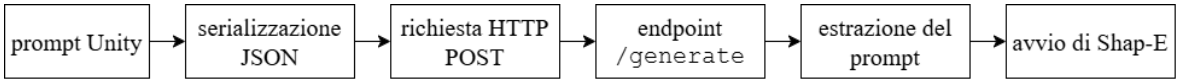



**Figura 4.2:** Flusso di invio del prompt dal client Unity al backend di generazione.

### 4.4.2 Ricezione del modello

La risposta restituita dal backend contiene il file GLB prodotto dalla generazione. Per gestire tale risposta, Shap_eClient associa al componente UnityWebRequest un componente DownloadHandlerFile, al quale viene fornito il percorso nel quale memorizzare il file ricevuto. Il file viene inizialmente salvato in una directory dedicata ai download dell'applicazione. La separazione tra questa directory temporanea e quella associata alla stanza corrente consente di distinguere la fase di ricezione dalla memorizzazione definitiva del contenuto: soltanto dopo che la richiesta HTTP è terminata con successo il file viene considerato valido e trasferito nella posizione prevista per i modelli della stanza. Questa organizzazione evita inoltre che un trasferimento non completato venga trattato come un modello già appartenente allo stato persistente dell'applicazione. La ricezione e il salvataggio del GLB costituiscono infatti una delle responsabilità specifiche attribuite alla comunicazione client-server nell'architettura di progetto.

Prima del trasferimento nella directory definitiva, al modello viene assegnato un GUID (Globally Unique Identifier), utilizzato per costruire un nome univoco per il file. Questo passaggio è necessario perché il backend esporta ogni generazione con lo stesso nome, generated_mesh.glb; conservando direttamente tale nome, una nuova richiesta potrebbe quindi

sovrascrivere il risultato precedente. L'identificatore generato dal client permette invece di mantenere contemporaneamente più modelli nella stessa stanza senza dover ricavare il nome del file dal prompt o da altre informazioni fornite dall'utente. Lo stesso identificativo può inoltre essere utilizzato dalle successive componenti dell'applicazione per associare in modo stabile il file GLB al relativo oggetto Unity e ai dati persistenti che ne descrivono lo stato.

Le directory utilizzate per il download e per la conservazione dei modelli sono costruite a partire da Application.persistentDataPath. Questa proprietà fornita da Unity restituisce un percorso destinato ai dati che devono essere conservati tra diverse esecuzioni dell'applicazione, evitando di dipendere dalla struttura della cartella del progetto utilizzata durante lo sviluppo. La scelta è particolarmente importante in un'applicazione multipiattaforma: mentre nell'Editor il percorso fa riferimento al filesystem del computer, in una build Android o Meta Quest, Unity restituisce una directory appartenente allo spazio persistente assegnato all'applicazione. Il resto della logica può quindi utilizzare lo stesso meccanismo di gestione dei file senza inserire nel codice percorsi assoluti specifici per ciascuna piattaforma.

Durante una richiesta in corso il pulsante di generazione viene disabilitato e viene resa disponibile l'operazione di annullamento. Quest'ultima richiama la funzione Abort() sul componente UnityWebRequest se l'operazione non è ancora terminata. Al termine, i pulsanti vengono riportati allo stato iniziale. La gestione degli errori implementata è essenziale. Il codice distingue il completamento corretto dagli altri risultati della richiesta, ma nei casi di fallimento restituisce all'interfaccia un messaggio generico.

**Listato 4.3:** Gestione della richiesta HTTP e ricezione del modello

```
COROUTINE SendRequest(validPrompt)
    url = serverAddress + "/generate"
    jsonData = serialize validPrompt as JSON

    create a POST request to url

    create an UploadHandler from jsonData
    assign it to the request

    set downloadPath to the temporary path for "generated_mesh.glb"
    create a DownloadHandlerFile using downloadPath
    assign it to the request

    set the "Content-Type" header to "application/json"

    WAIT FOR request.SendWebRequest()

    IF request was successful
        newFileName = generate a GUID + ".glb"
        rename the downloaded file

        newPath = move the file to the current room directory

        LoadModel(newPath, true)

    ELSE
        show a generation error message
    END IF

END COROUTINE
```

## 4.5 Importazione runtime del modello

La ricezione del file non conclude la pipeline. Il modello deve essere trasformato da contenuto presente nel filesystem a oggetto appartenente alla scena Unity. Questa responsabilità è affidata al componente GlbManager, che utilizza glTFast per importare il file durante l'esecuzione.

### 4.5.1 Evoluzione della soluzione di importazione: da OBJ a GLB

La prima soluzione utilizzata per il caricamento runtime dei contenuti generati era basata sul formato OBJ. Il backend produceva un file generated_mesh.obj, che veniva trasferito al client e importato in Unity mediante una libreria dedicata al caricamento degli OBJ durante l'esecuzione. Dalle prove è emerso che il caricamento degli OBJ prodotti da Shap-E non riusciva a terminare a causa dell'esaurimento della memoria disponibile.

Inizialmente sono state considerate alcune possibili cause legate direttamente ai file prodotti. In particolare, gli OBJ generati da Shap-E non contenevano informazioni relative alle normali, circostanza segnalata anche da Unity durante l'importazione. Per verificare se il problema dipendesse più in generale dal formato o dalla libreria utilizzata, sono stati caricati con lo stesso importer anche modelli OBJ provenienti da altre fonti. Le prove effettuate con modelli ottenuti da Meshy e TRELLIS hanno completato correttamente il caricamento, nonostante per alcuni di essi venissero segnalati problemi relativi alle normali. Inoltre, tali modelli potevano presentare un numero di vertici e facce superiore a quello degli output di Shap-E. Non risultava quindi possibile ricondurre il comportamento osservato unicamente alla complessità geometrica del modello o all'assenza delle normali.

Per analizzare il consumo di memoria sono stati utilizzati Unity Profiler e Memory Profiler. Sono stati acquisiti due snapshot, rispettivamente prima del caricamento e dopo l'avvio dell'operazione di caricamento. Il confronto ha evidenziato una variazione particolarmente marcata nella sezione Managed Heap Utilization. Prima del caricamento il Memory Profiler riportava circa 0.96 GB di memoria gestita complessiva, di cui circa 0.84 GB classificati come Empty Heap Space. Nello snapshot successivo al caricamento, gli stessi valori raggiungevano rispettivamente circa 13.53 GB e 13.40 GB.

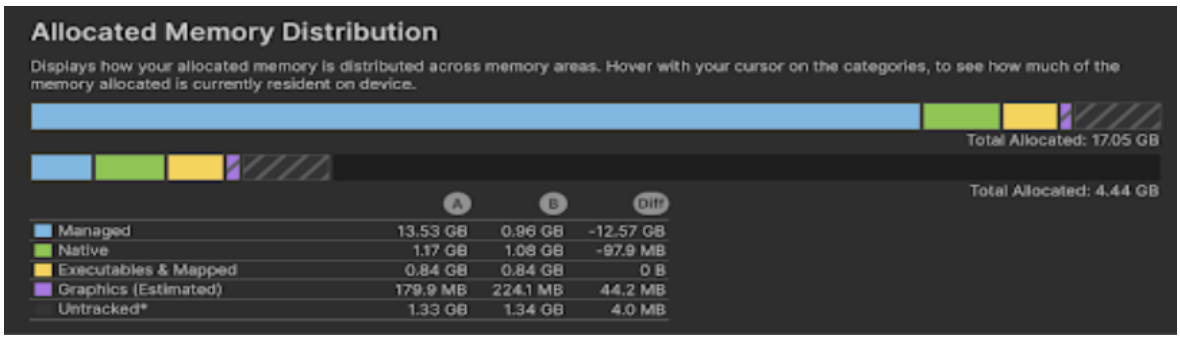



**Figura 4.3:** Distribuzione della memoria allocata prima e dopo il caricamento di un modello OBJ.

Nello snapshot successivo al caricamento, la memoria attribuita agli oggetti era circa 29.1 MB, rispetto ai 28.1 MB dello snapshot precedente. La principale differenza era quindi costituita dall'Empty Heap Space, aumentato di circa 12.56 GB.

A seguito del profiling è stato esaminato anche il funzionamento della libreria utilizzata per l'importazione OBJ. L'analisi ha permesso di ricostruire le principali operazioni svolte dall'importer, ma non ha consentito di isolare con certezza la causa dell'espansione dell'heap osservata precedentemente. A causa di questo problema, è stata scelta una soluzione alternativa di importazione runtime. Il formato OBJ e la relativa libreria di caricamento sono stati




**Figura 4.4:** Dettaglio dell'utilizzo del Managed Heap prima e dopo il caricamento del modello OBJ.

sostituiti rispettivamente da GLB e glTFast. Il backend è stato quindi modificato affinché il risultato prodotto da Shap-E venisse esportato in GLB.

### 4.5.2 Integrazione di glTFast

L'importazione dei modelli GLB durante l'esecuzione è gestita dal componente GlbManager attraverso la libreria glTFast. A differenza degli asset inclusi nel progetto durante lo sviluppo, i modelli generati da Shap-E non sono disponibili al momento della relativa compilazione dell'applicazione; il client deve quindi costruire dinamicamente la relativa rappresentazione nella scena a partire dal file ricevuto dal backend. Il caricamento viene avviato dal metodo LoadModel, al quale viene fornito il percorso assoluto del file GLB e un parametro che permette di distinguere un modello appena generato da uno ripristinato da una precedente esecuzione.

Come primo passo viene creato un nuovo GameObject, denominato glbModel, utilizzato come oggetto contenitore per il modello da importare. Tale oggetto viene inserito come figlio del GlbManager, mantenendo così i modelli caricati all'interno di una gerarchia comune. Sullo stesso oggetto viene inoltre aggiunto dinamicamente un componente GlbModel, responsabile delle operazioni che devono essere eseguite dopo il completamento dell'importazione.

Prima di avviare il caricamento viene recuperato dal nome del file l'identificativo assegnato nella fase precedente. In questo modo l'identità del modello viene mantenuta anche nelle successive esecuzioni. Una volta verificato l'identificativo, a glbModel viene aggiunto un componente GltfAsset fornito da glTFast. La proprietà LoadOnStartup viene impostata a false, poiché il modello da importare non è disponibile all'avvio della scena ma viene generato e ricevuto durante l'esecuzione dell'applicazione. Il caricamento viene quindi avviato esplicitamente in modo asincrono, passando il percorso assoluto del file GLB. Solo al termine di questa operazione GlbManager richiama il metodo OnModelLoad del componente GlbModel.

GlbModel ricerca nella gerarchia appena generata un componente MeshFilter contenente la geometria del modello importato. Quando questo componente viene trovato il modello può essere configurato per l'utilizzo nell'applicazione. La configurazione prevede la creazione di un componente MeshCollider e di un XRSimpleInteractable, necessario per rendere l'oggetto selezionabile attraverso il sistema di interazione XR. Se esistono dati persistenti associati allo stesso GUID, questi vengono caricati per ripristinare la trasformazione precedentemente salvata; nel caso di un modello appena generato viene invece eseguita una fase iniziale di posizionamento e ridimensionamento rispetto ad un'area iniziale per i nuovi modelli.

**Listato 4.4:** Caricamento a runtime di un modello GLB e creazione del relativo oggetto Unity

```
ASYNC PROCEDURE LoadModel(absolutePath, isFirstLoad)

    modelObject = create a new GameObject named "glbModel"
    set modelObject as a child of GlbManager

    glbModel = add a GlbModel component to modelObject

    fileName = extract the file name from absolutePath

    IF fileName cannot be parsed as a GUID
        RETURN
    END IF

    modelGuid = GUID parsed from fileName

    add a GltfAsset component to modelObject

    AWAIT gltf.Load(absolutePath)

    glbModel.OnModelLoad(modelGuid, isFirstLoad)

END PROCEDURE
```

# CAPITOLO 5

## Gestione dei contenuti e interazione in Unity

Il completamento della pipeline di generazione non coincide con la sola importazione del file GLB all'interno di Unity. Una volta caricato, il contenuto deve poter essere conservato tra diverse esecuzioni, organizzato insieme agli altri modelli e modificato dall'utente. Il presente capitolo descrive quindi le funzionalità relative alla persistenza dei contenuti, alla loro organizzazione in stanze, alla manipolazione dei modelli e all'adattamento dell'interazione ai diversi ambienti di esecuzione.

### 5.1 Persistenza dei contenuti

La persistenza dei contenuti consente di mantenere disponibili i modelli generati e il relativo stato anche dopo la chiusura dell'applicazione. Nel sistema sviluppato, il file GLB rappresenta il contenuto tridimensionale vero e proprio, mentre le informazioni necessarie a ricostruirne la configurazione nella scena vengono gestite separatamente. In particolare, a ciascun modello sono stati associati dati relativi a posizione, rotazione e scala, così da poter ripristinare l'oggetto nello stesso stato in cui era stato lasciato. Questa separazione tra geometria e stato applicativo permette di conservare i modelli senza modificarne direttamente il file GLB e di ricostruire una stanza nelle esecuzioni successive caricando i contenuti memorizzati e applicando le trasformazioni salvate. La gestione della persistenza comprende quindi tre aspetti principali: la memorizzazione dei modelli, la serializzazione dello stato e il successivo ripristino dei contenuti associati a una stanza.

La memorizzazione dei modelli è organizzata in modo da riflettere la struttura logica dell'applicazione. Ogni stanza dispone di una propria area dedicata, all'interno della quale vengono conservati i file GLB associati ai contenuti appartenenti a quella stanza. In questo modo l'appartenenza di un modello a una stanza è rappresentata direttamente anche a livello di filesystem, semplificando le successive operazioni di caricamento, eliminazione e ripristino. Il modello viene quindi mantenuto come file persistente indipendente dall'oggetto Unity creato durante l'esecuzione. Questa distinzione è rilevante perché l'istanza presente nella scena ha natura temporanea, mentre il file GLB costituisce la rappresentazione da cui il contenuto può essere ricostruito in una successiva esecuzione.

Oltre al file GLB, per ciascun modello vengono conservate le informazioni necessarie a ricostruirne la configurazione all'interno della stanza. La gestione di questi dati è affidata a JsonManager, che serializza lo stato del modello in formato JSON e ne consente la successiva deserializzazione. Il file geometrico e le informazioni applicative rimangono quindi separati:

il primo conserva il contenuto tridimensionale generato, mentre il secondo descrive lo stato assunto dall'oggetto durante l'utilizzo dell'applicazione. In particolare, lo stato persistente comprende le trasformazioni applicate al modello, cioè posizione, rotazione e scala. Quando l'utente modifica un oggetto, tali valori possono essere salvati senza intervenire direttamente sul file GLB. L'associazione tra il modello e i relativi dati serializzati permette così di conservare le modifiche effettuate dall'utente e costituisce la base per il successivo ripristino della stanza.

**Listato 5.1:** Salvataggio dello stato di un modello

```
PROCEDURE SaveModel()
    IF model is loaded
        transformData.SetTransformData(model.transform)

        fileName = modelGuid + ".json"
        JsonManager.Serialize(transformData, fileName)
    END IF
END PROCEDURE
```

La memorizzazione dei modelli e la serializzazione delle relative trasformazioni consentono di ricostruire il contenuto di una stanza nelle esecuzioni successive dell'applicazione. Quando una stanza già esistente viene aperta, i modelli ad essa associati vengono nuovamente caricati a partire dai file GLB conservati e, qualora presenti, vengono recuperati i corrispondenti dati sullo stato. Dopo il caricamento di ciascun modello vengono applicati i valori di posizione, rotazione e scala precedentemente salvati dall'utente, ripristinando così l'ultima configurazione memorizzata dell'oggetto.

## 5.2 Organizzazione in stanze

Per consentire la gestione separata di più insiemi di contenuti, l'applicazione introduce il concetto di stanza. Una stanza non corrisponde a una scena Unity distinta, ma rappresenta un raggruppamento logico di modelli generati dall'utente. All'avvio dell'applicazione l'utente può quindi selezionare una stanza esistente oppure crearne una nuova, determinando l'insieme di modelli sul quale verranno effettuate le successive operazioni. La gestione delle stanze è coordinata dal GameManager, che mantiene lo stato necessario a identificare la stanza corrente. Oltre alla selezione e alla creazione, il sistema prevede la possibilità di rimuovere le stanze non più necessarie. Le operazioni di creazione, selezione e rimozione costituiscono quindi il meccanismo attraverso cui l'utente organizza i contenuti prodotti, soddisfacendo il requisito funzionale di mantenere distinti più insiemi di modelli all'interno della stessa applicazione.

**Listato 5.2:** Caricamento dei modelli associati alla stanza corrente

```
PROCEDURE LoadRoom()

    set roomDirectory to the directory associated with currentRoom

    FOR EACH .glb file in roomDirectory
        LoadModel(file, false)
    END FOR

END PROCEDURE
```

## 5.3 Manipolazione dei modelli

Una volta inseriti nella stanza, i modelli possono essere modificati direttamente dall'utente attraverso un sistema di manipolazione basato su gizmo. La gestione di queste operazioni è affidata al componente RuntimeGizmo, che consente di intervenire sulle principali trasformazioni dell'oggetto selezionato. Le modifiche riguardano traslazione, rotazione e ridimensionamento e agiscono direttamente sul componente Transform del modello selezionato nella scena. Il sistema di manipolazione completa quindi il passaggio dal semplice caricamento del contenuto alla sua effettiva organizzazione nell'ambiente virtuale.

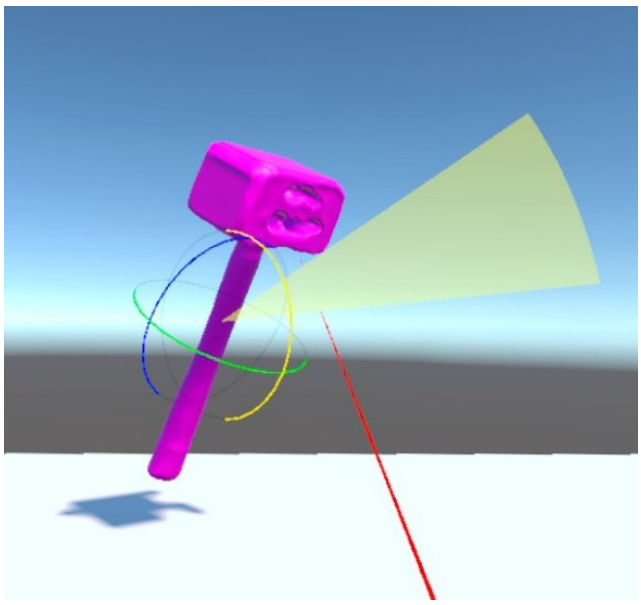


**Figura 5.1:** Esempio di selezione e manipolazione di un modello 3D mediante il gizmo di trasformazione.

**Listato 5.3:** Selezione di un modello e associazione al gizmo di trasformazione

```
PROCEDURE OnSelectEntered(hitTransform)

    IF hitTransform does not identify a selectable object
        RETURN
    END IF

    IF another model is already selected
        restore the original material of the previously selected model
    END IF

    selectedModel = hitTransform
    apply the selection material to selectedModel

    transformHandle.target = selectedModel
    activate the transform handle

END PROCEDURE
```

La manipolazione richiede innanzitutto la selezione del modello sul quale applicare la trasformazione. Una volta individuato l'oggetto, il sistema di manipolazione ne mantiene il riferimento come elemento attivo, in modo che le successive operazioni siano applicate esclusivamente al modello selezionato. La traslazione permette di modificare la posizione

del modello nello spazio tridimensionale. Una volta selezionato l'oggetto, il sistema di manipolazione permette di intervenire sul relativo Transform, modificandone le coordinate spaziali senza alterarne orientamento o dimensioni. La rotazione permette di modificare l'orientamento del modello mantenendone invariata la posizione e la scala. L'operazione agisce sui parametri di rotazione del componente Transform dell'oggetto selezionato e consente quindi di adattarne l'orientamento rispetto agli altri elementi presenti nell'ambiente. Il ridimensionamento consente infine di modificare la scala del modello dopo il suo inserimento nella stanza. L'operazione interviene sui valori di scala del relativo Transform e permette di adattare le dimensioni dell'oggetto alle esigenze dell'ambiente senza modificare direttamente la geometria contenuta nel file GLB.

## 5.4 Adattamento multipiattaforma

L'applicazione è stata progettata mantenendo separata, per quanto possibile, la logica principale della pipeline dalle modalità di interazione proprie della piattaforma di esecuzione. Le funzionalità di generazione, caricamento, organizzazione e persistenza dei contenuti rimangono infatti sostanzialmente comuni, mentre cambiano soprattutto i meccanismi attraverso cui l'utente interagisce con l'interfaccia e con i modelli tridimensionali. Nel corso del progetto il client è stato utilizzato in ambiente desktop e successivamente adattato a dispositivi Android e alla VR su Meta Quest 2.

Nell'ambiente desktop il client Unity utilizza le modalità di interazione tradizionali disponibili su computer, senza richiedere i componenti specifici introdotti successivamente per l'esperienza immersiva. La versione desktop ha rappresentato una configurazione completa del client sulla quale verificare l'integrazione tra i diversi componenti della pipeline prima dell'adattamento a dispositivi caratterizzati da modalità di input differenti. Per la manipolazione dei modelli, il sistema utilizzato nel progetto era originariamente concepito per modalità d'interazione basate su mouse e tastiera.

L'adattamento ad Android ha richiesto principalmente di adeguare l'interazione e la comunicazione di rete alle caratteristiche di un dispositivo mobile. L'interfaccia e i controlli sono stati adattati all'utilizzo su tablet, sostituendo le modalità di interazione tipiche dell'ambiente desktop con input basati su touchscreen. La logica principale dell'applicazione è rimasta invece invariata: acquisizione del prompt, gestione dei contenuti e caricamento dei modelli continuano a essere gestiti dagli stessi componenti. Anche nella configurazione Android la generazione tridimensionale rimane separata dal dispositivo client. Il dispositivo Android comunica quindi attraverso la rete locale con la macchina sulla quale è in esecuzione il servizio Shap-E.

L'ultima configurazione del client è destinata all'esecuzione su Meta Quest 2. L'adattamento alla VR ha richiesto l'integrazione di OpenXR e XR Interaction Toolkit, utilizzati rispettivamente per il supporto dell'ambiente XR e per la gestione delle modalità di interazione disponibili nel visore. La struttura generale dell'applicazione è rimasta per lo più invariata rispetto alla versione Android. Il cambiamento principale riguarda la manipolazione dei modelli. In ambiente immersivo, la selezione e la manipolazione dei modelli devono essere effettuate attraverso i controller VR.

Per rendere possibile la manipolazione anche in VR è stato quindi necessario intervenire sulla libreria utilizzata, adattandone il comportamento alle modalità di interazione offerte dai controller del visore. Le modifiche hanno consentito di mantenere le principali operazioni di selezione e trasformazione dei modelli anche nella versione destinata a Meta Quest 2, senza sostituire completamente il sistema già adottato nelle configurazioni precedenti.

CAPITOLO 6

# Valutazione sperimentale e discussione

## 6.1 Obiettivi della valutazione

La valutazione sperimentale ha lo scopo di caratterizzare il comportamento della pipeline sviluppata attraverso misure delle sue prestazioni, affiancate da un'analisi qualitativa dei contenuti tridimensionali prodotti. L'obiettivo non è valutare separatamente le prestazioni del modello Shap-E in termini generali, ma verificare il comportamento della configurazione effettivamente adottata nel prototipo e il suo impatto sull'utilizzo dell'applicazione.

La sperimentazione è stata articolata su due livelli. Una prima serie di prove è stata dedicata al backend generativo, isolato dal client Unity, allo scopo di misurare in condizioni controllate il tempo necessario alla produzione del file GLB e la variabilità tra esecuzioni successive. Questa analisi permette di individuare il peso della generazione tridimensionale, che costituisce la fase computazionalmente più onerosa della pipeline.

Una seconda valutazione è stata invece condotta sulla pipeline integrata nella configurazione finale dell'applicazione. In questo caso l'interesse si sposta dal solo processo generativo al tempo necessario affinché una richiesta formulata dal client venga elaborata dal backend e il modello risultante diventi disponibile nell'ambiente Unity. In questo modo è possibile confrontare l'ordine di grandezza del tempo di generazione con il tempo complessivamente necessario al caricamento del contenuto.

## 6.2 Ambiente sperimentale

Le prove sul servizio di generazione sono state eseguite su un notebook MSI Thin GF63 12VF, equipaggiato con un processore Intel Core i7-12650H, 16 GB di memoria RAM DDR4 e una GPU NVIDIA GeForce RTX 4060 con 8 GB di memoria video. Il backend è stato eseguito all'interno di un container Docker costruito a partire dall'immagine nvidia/cuda:11.8.0-runtime-ubuntu22.04, con Docker 28.3.2 e PyTorch 2.7.1 compilato per CUDA 11.8. La generazione viene quindi eseguita sfruttando l'accelerazione hardware della GPU attraverso CUDA. Durante l'acquisizione dei dati sono state mantenute condizioni di esecuzione il più possibile costanti, lasciando in esecuzione solamente il container Docker e i terminali utilizzati per inviare le richieste e controllare i log. Nelle prove end-to-end, Meta Quest 2 e il notebook su cui era eseguito il backend erano collegati alla stessa rete locale, attraverso la quale avveniva la comunicazione HTTP tra client e server. Le prove end-to-end sono state eseguite direttamente su Meta Quest 2, utilizzando la build standalone finale dell'applicazione.

## 6.3 Scenari di prova

Per evitare che la valutazione fosse legata a un singolo tipo di oggetto, sono stati definiti dieci prompt testuali differenti. I casi comprendono oggetti con caratteristiche geometriche diverse, includendo forme relativamente semplici, oggetti composti da più elementi e forme organiche.

**Tabella 6.1:** Scenari di prova utilizzati per la valutazione sperimentale.

| ID | Prompt testuale |
|---|---|
| P01 | *a simple chair* |
| P02 | *a round table with four legs* |
| P03 | *a desk lamp with a curved neck* |
| P04 | *a toy car with four wheels* |
| P05 | *a cactus in a flower pot* |
| P06 | *a penguin* |
| P07 | *a teapot with a handle and a spout* |
| P08 | *a hammer* |
| P09 | *a treasure chest with a lid* |
| P10 | *a bicycle* |

I prompt sono stati formulati in lingua inglese per mantenere costante la lingua di input durante tutte le prove. Sono inoltre stati privilegiati attributi relativi alla forma e alla struttura degli oggetti, evitando di utilizzare colore o materiale come caratteristiche principali del test. Nella configurazione finale adottata, infatti, il processo di esportazione conserva la geometria generata senza trasferire texture o materiali, rendendo tali proprietà poco significative ai fini della valutazione dell'output. Gli stessi dieci scenari sono stati utilizzati sia per la valutazione del backend generativo sia per quella della pipeline integrata.

## 6.4 Metriche e protocollo sperimentale

La prima fase della valutazione considera il tempo di elaborazione del backend, definito come l'intervallo compreso tra l'avvio della funzione di generazione e il completamento della produzione del file GLB. La misura è stata effettuata direttamente nel codice del backend, all'interno dell'endpoint FastAPI dedicato alla generazione, utilizzando la funzione perf_counter() per rilevare il tempo trascorso.

Prima di definire il protocollo definitivo sono state eseguite alcune prove preliminari. Queste hanno evidenziato una maggiore variabilità e, in alcuni casi, tempi superiori nelle prime richieste effettuate dopo l'avvio del backend. Per ridurre l'influenza di questa fase iniziale, prima delle misure ufficiali sono state eseguite tre generazioni di warm-up con prompt differenti, escluse dall'analisi successiva. Le trenta prove sono state suddivise in tre round, eseguendo quindi ciascuno dei dieci scenari tre volte. Per evitare che un'eventuale variazione progressiva delle prestazioni penalizzasse sistematicamente gli stessi prompt, l'ordine degli scenari è stato modificato ad ogni ripetizione: il primo round ha seguito l'ordine crescente da P01 a P10, il secondo l'ordine inverso, mentre il terzo ha utilizzato un ordine prefissato diverso dai precedenti. Per ciascun prompt sono stati quindi calcolati media, minimo, massimo e deviazione standard campionaria del tempo di elaborazione.

La valutazione della pipeline integrata è stata effettuata misurando i tempi direttamente sul client Unity. Sono state considerate tre grandezze: il tempo della richiesta HTTP; il

tempo necessario al caricamento del file GLB mediante glTFast; e il tempo end-to-end, che comprende entrambe le fasi. Le misure iniziano dalla richiesta di generazione di un prompt già disponibile in forma testuale e non comprendono quindi l'eventuale acquisizione e trascrizione dell'input vocale.

Anche per questa prova sono state eseguite tre richieste preliminari di warm-up, escluse dall'analisi successiva. Dopo tale fase, ciascuno dei dieci scenari definiti nella Tabella 6.1 è stato eseguito una volta nella configurazione finale su Meta Quest 2. Questa seconda serie di misure è stata utilizzata principalmente per quantificare il tempo complessivamente richiesto dalla pipeline e valutare l'incidenza del caricamento del modello rispetto alla durata della richiesta.

## 6.5 Analisi quantitativa dei risultati

I risultati delle trenta generazioni sono riportati nella tabella 6.2. Tutte le richieste hanno completato l'elaborazione e prodotto un file GLB importabile e visualizzabile correttamente, senza rilevare file vuoti o corrotti. Il tasso di successo tecnico osservato è stato quindi del 100%.

**Tabella 6.2:** Tempi di elaborazione del backend Shap-E per i dieci scenari di prova, calcolati su tre esecuzioni per ciascun prompt.

| ID | Media [s] | Min [s] | Max [s] | Dev. std. [s] |
|---|---|---|---|---|
| P01 | 36.01 | 35.54 | 36.25 | 0.41 |
| P02 | 36.30 | 36.18 | 36.39 | 0.11 |
| P03 | 35.75 | 35.43 | 35.94 | 0.28 |
| P04 | 36.23 | 35.51 | 37.00 | 0.75 |
| P05 | 38.15 | 36.02 | 40.79 | 2.43 |
| P06 | 39.02 | 36.89 | 40.88 | 2.01 |
| P07 | 37.08 | 36.06 | 39.02 | 1.68 |
| P08 | 35.73 | 35.46 | 36.02 | 0.28 |
| P09 | 35.38 | 34.95 | 36.14 | 0.66 |
| P10 | 37.99 | 37.31 | 39.26 | 1.10 |

Considerando complessivamente le trenta esecuzioni, il tempo medio di elaborazione del backend è risultato pari a circa 36.76 s, mentre la deviazione standard complessiva è risultata pari a circa 1.57 s. Le differenze tra le medie dei singoli prompt sono relativamente contenute. Non emerge dai risultati una relazione immediata tra la complessità apparente della descrizione e il tempo richiesto dalla generazione.

I risultati della prova sulla pipeline integrata sono riportati nelle tabelle 6.3 e 6.4. Tutti i dieci scenari hanno completato la richiesta e il successivo caricamento del modello nel client.

Il tempo medio della richiesta è risultato pari a 36.74 s, con valori compresi tra 35.05 s e 38.47 s e una deviazione standard di 1.00 s. Il caricamento del file GLB ha richiesto invece mediamente 0.11 s, con un minimo di 0.04 s e un massimo di 0.28 s. Il tempo end-to-end medio è risultato pari a 36.85 s, con valori compresi tra 35.20 s e 38.53 s e una deviazione standard di 0.98 s.

Nei casi osservati, il tempo di caricamento del modello rappresenta quindi una frazione molto ridotta del tempo end-to-end, mentre la durata complessiva rimane sostanzialmente determinata dalla fase compresa nella richiesta HTTP.

**Tabella 6.3:** Tempi misurati nella pipeline integrata.

| ID | Richiesta [s] | Caricamento GLB [s] | End-to-end [s] |
|---|---|---|---|
| P01 | 37.26 | 0.08 | 37.34 |
| P02 | 36.84 | 0.09 | 36.93 |
| P03 | 38.47 | 0.06 | 38.53 |
| P04 | 37.44 | 0.09 | 37.53 |
| P05 | 37.10 | 0.10 | 37.20 |
| P06 | 36.21 | 0.17 | 36.38 |
| P07 | 35.05 | 0.15 | 35.20 |
| P08 | 35.39 | 0.05 | 35.44 |
| P09 | 36.53 | 0.28 | 36.81 |
| P10 | 37.12 | 0.04 | 37.17 |

**Tabella 6.4:** Statistiche riepilogative dei tempi.

| Metrica | Media [s] | Min [s] | Max [s] | Dev. std. [s] |
|---|---|---|---|---|
| Richiesta | 36.74 | 35.05 | 38.47 | 1.00 |
| Caricamento GLB | 0.11 | 0.04 | 0.28 | 0.07 |
| End-to-end | 36.85 | 35.20 | 38.53 | 0.98 |

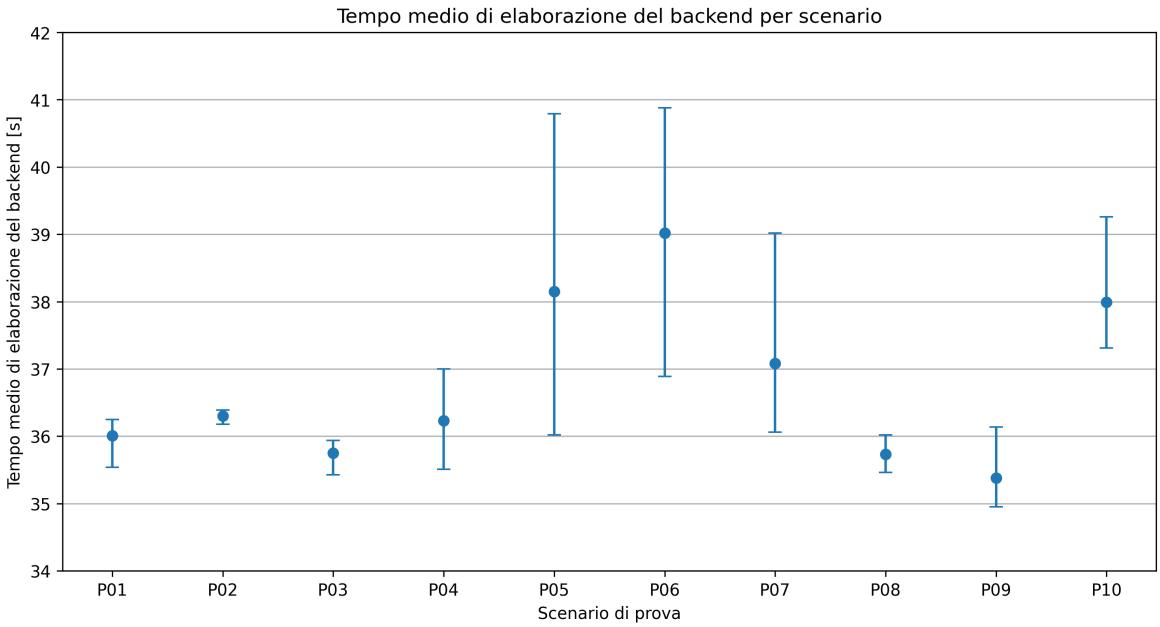



**Figura 6.1:** Tempo medio di elaborazione del backend per ciascuno scenario di prova. Le linee verticali indicano l'intervallo compreso tra il tempo minimo e il tempo massimo osservati.

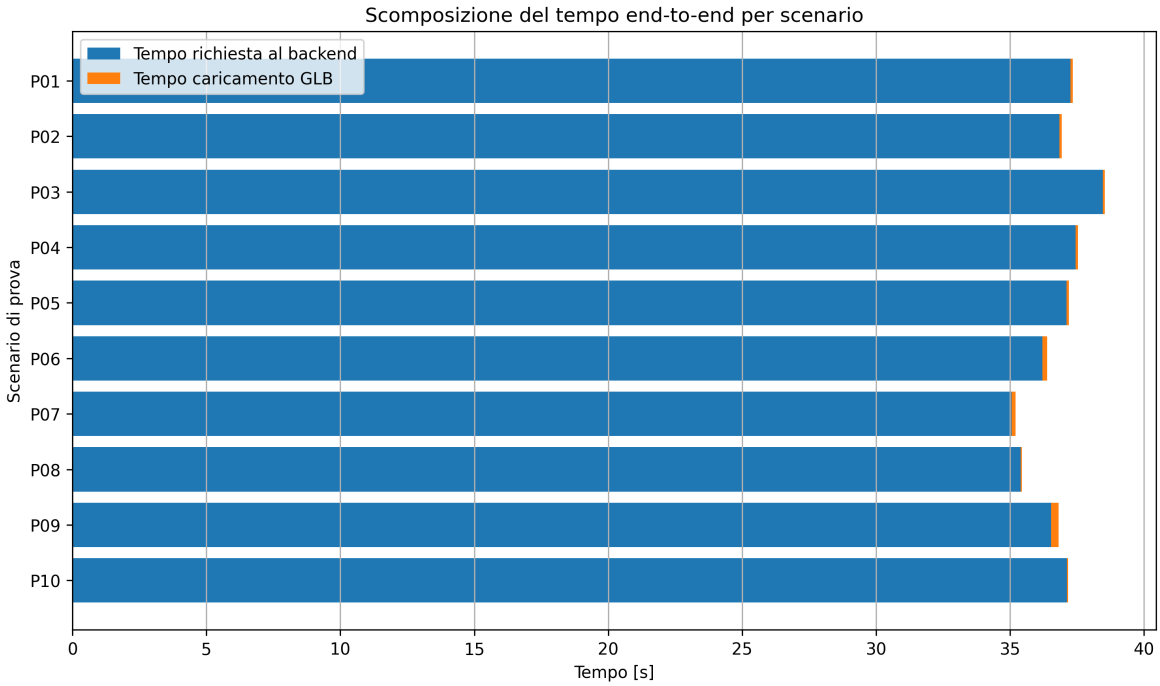



**Figura 6.2:** Scomposizione del tempo end-to-end per ciascuno scenario della pipeline integrata.

## 6.6 Analisi qualitativa dei modelli generati

L'analisi qualitativa è stata condotta sui trenta modelli prodotti durante le prove del backend. L'obiettivo di questa analisi non è attribuire un punteggio formale alla qualità dei modelli, ma verificare in modo descrittivo se la geometria generata risulti riconoscibile, coerente con il prompt fornito e sufficientemente integra da poter essere utilizzata nelle fasi successive della pipeline. I file GLB sono stati importati e ispezionati in Blender, considerando in particolare la riconoscibilità dell'oggetto, la presenza degli elementi richiesti e l'eventuale comparsa di deformazioni, parti disconnesse o elementi mancanti. Le ripetizioni dello stesso scenario hanno generalmente mantenuto caratteristiche qualitative dello stesso ordine, pur mostrando in alcuni casi differenze rilevanti nella presenza degli elementi richiesti e nella coerenza della geometria.

La qualità osservata non è risultata uniforme tra i diversi scenari. Alcuni oggetti caratterizzati da una struttura relativamente compatta sono stati riprodotti in maniera chiaramente riconoscibile. Nel caso P01 (*a simple chair*), ad esempio, risultano distinguibili la seduta, le gambe e lo schienale; analogamente, il caso P08 (*a hammer*) presenta una geometria complessivamente coerente, nella quale la testa e l'impugnatura risultano chiaramente distinguibili e correttamente connesse. Il caso P09 (*a treasure chest with a lid*) conserva una forma complessiva compatibile con quella dell'oggetto richiesto. Anche P06 (*a penguin*) e P05 (*a cactus in a flower pot*) risultano immediatamente identificabili, pur presentando una geometria semplificata e irregolare.

In alcuni casi è emersa una marcata variabilità anche tra generazioni ottenute a partire dallo stesso prompt. Un esempio significativo è P02 (*a round table with four legs*). Nel primo round il modello è chiaramente riconoscibile come un tavolo rotondo, ma la struttura inferiore assume la forma di un sostegno centrale articolato invece delle quattro gambe richieste. Nel secondo round la geometria risulta invece fortemente deformata e la struttura dell'oggetto è difficilmente riconoscibile. La terza generazione presenta quattro elementi assimilabili alle gambe del tavolo, seppur con la presenza anche di un sostegno centrale. Le

maggiori criticità sono state osservate negli oggetti la cui struttura richiede elementi sottili, separati o fortemente articolati. Ad esempio, P03 (*a desk lamp with a curved neck*) mantiene alcuni elementi riconducibili alla forma richiesta, ma presenta componenti molto sottili e parzialmente disconnesse. Nel caso P07 (*a teapot with a handle and a spout*) il corpo principale e il beccuccio risultano riconoscibili, ma il manico appare frammentato. Il caso più critico tra quelli esaminati è P10 (*a bicycle*): le diverse componenti risultano fortemente frammentate e la geometria complessiva non permette di ricostruire con chiarezza la struttura dell'oggetto.

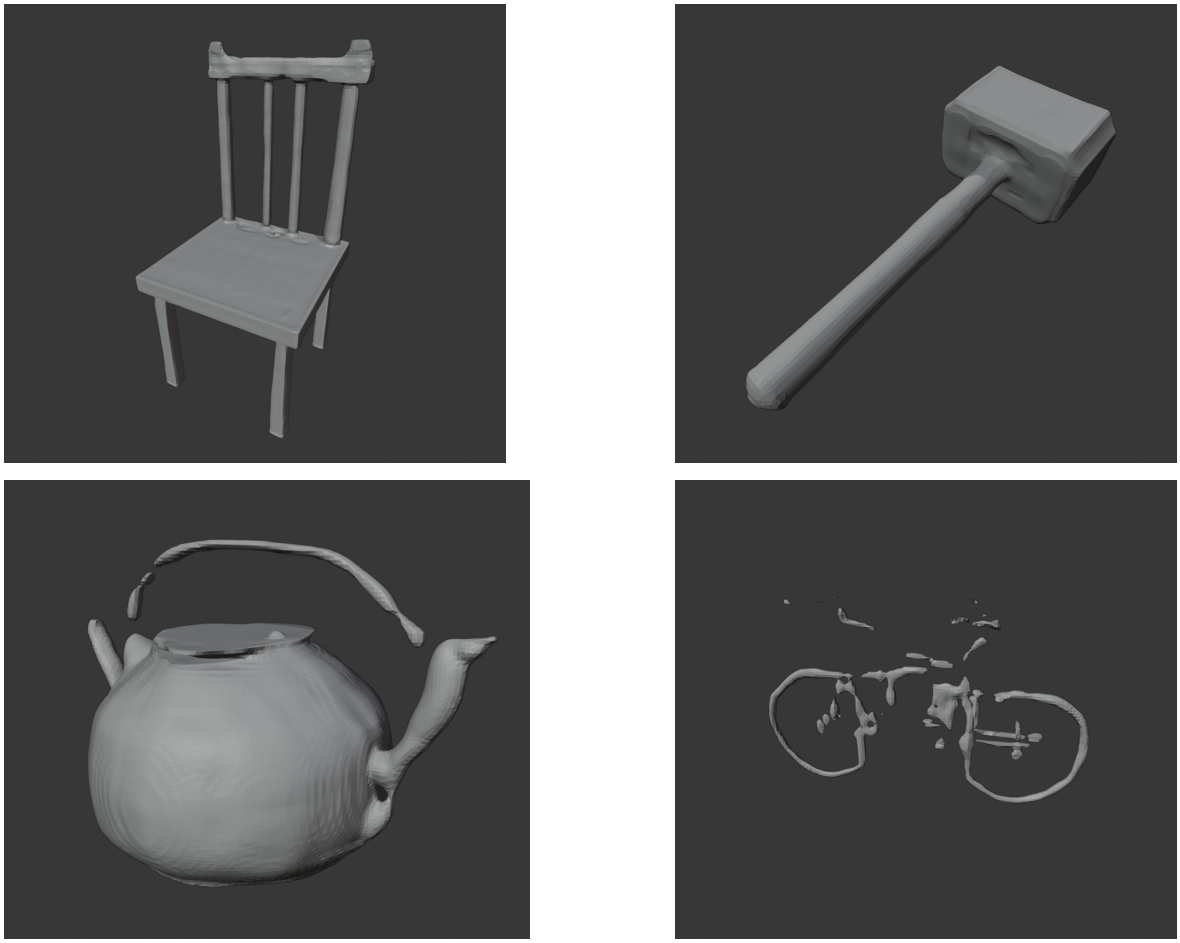


**Figura 6.3:** Esempi di output generati da Shap-E: P01 (a simple chair), P08 (a hammer), P07 (a teapot with a handle and a spout) e P10 (a bicycle).

Queste osservazioni suggeriscono, limitatamente ai casi analizzati, una maggiore difficoltà nella generazione di strutture costituite da numerosi elementi sottili o separati rispetto a oggetti con una forma più compatta. Tale risultato deve tuttavia essere interpretato come un'osservazione qualitativa sul campione utilizzato e non come una caratterizzazione generale delle prestazioni di Shap-E.

## 6.7 Discussione dei risultati

I risultati quantitativi indicano che il principale vincolo prestazionale della soluzione è riconducibile alla fase di generazione tridimensionale. Nel test della pipeline integrata, il caricamento del file GLB sul client richiede infatti una frazione minima del tempo complessivo, mentre la durata della richiesta rimane dello stesso ordine di grandezza dei tempi osservati nelle prove isolate del backend. Poiché le due serie di misure sono state ottenute in esecuzioni

differenti, il loro confronto non consente di quantificare separatamente il contributo della comunicazione di rete; permette tuttavia di osservare che l'importazione runtime del modello non costituisce un collo di bottiglia rilevante nella configurazione sperimentata.

La variabilità osservata rispetto al tempo medio di generazione è complessivamente contenuta. Questo suggerisce che, nella configurazione adottata, la durata della generazione sia poco influenzata dalle caratteristiche apparenti dell'oggetto richiesto. Tale interpretazione rimane comunque limitata agli scenari considerati.

Dal punto di vista dell'affidabilità tecnica, tutte le trenta generazioni utilizzate per la valutazione del backend hanno prodotto file GLB validi, mentre i dieci scenari eseguiti sulla pipeline integrata hanno completato sia la richiesta sia il successivo caricamento nel client. Nel campione osservato, quindi, il passaggio dalla generazione al trasferimento e all'importazione runtime non ha introdotto fallimenti. Questo risultato è rilevante per l'obiettivo del progetto, poiché mostra che il contenuto prodotto può attraversare l'intera pipeline.

La validità tecnica del risultato non coincide tuttavia con la sua qualità geometrica. L'analisi qualitativa ha evidenziato output chiaramente riconoscibili accanto a modelli caratterizzati da deformazioni, elementi mancanti o strutture incoerenti, con differenze anche tra generazioni dello stesso prompt. La soluzione adottata deve quindi essere interpretata come un compromesso: Shap-E ha reso possibile completare una pipeline controllabile e utilizzabile con le risorse disponibili, ma la qualità del contenuto generato rimane meno prevedibile rispetto alla correttezza del processo di integrazione.

Dal punto di vista dell'applicazione finale, il dato temporale assume rilevanza anche rispetto al flusso di interazione. Il nuovo modello diventa disponibile soltanto al termine di un'attesa dell'ordine di decine di secondi; durante questo intervallo, tuttavia, la richiesta viene gestita senza bloccare il normale aggiornamento della scena. L'utente può quindi continuare a muoversi nell'ambiente e a interagire con i contenuti già presenti. Il tempo di elaborazione rappresenta pertanto un vincolo solamente alla disponibilità del nuovo modello, ma non determina un'interruzione dell'esperienza applicativa. I dati raccolti permettono di quantificare tale attesa, ma non di stabilire quanto essa sia accettabile dal punto di vista dell'esperienza utente.

CAPITOLO 7

# Conclusioni

Il lavoro svolto ha portato alla realizzazione di un prototipo funzionante per la generazione e l'utilizzo dinamico di contenuti tridimensionali all'interno di un'applicazione Unity destinata alla VR. L'obiettivo principale non era sviluppare un nuovo modello generativo, ma verificare la possibilità di integrare tecnologie esistenti in una pipeline end-to-end utilizzabile durante l'esecuzione dell'applicazione. Il risultato ottenuto mostra che tale integrazione è tecnicamente realizzabile nella configurazione sperimentata: i contenuti generati dal backend possono essere trasferiti al client, caricati a runtime, manipolati e conservati tra diverse esecuzioni.

La valutazione ha inoltre permesso di individuare con maggiore precisione il comportamento della pipeline. I tempi misurati mostrano che la durata complessiva dell'operazione è determinata quasi interamente dalla generazione del modello, mentre il caricamento del file GLB nel client incide in misura marginale. Nelle prove effettuate, non sono stati osservati fallimenti tecnici nei casi analizzati.

Anche la valutazione svolta presenta alcuni limiti. Gli scenari sono circoscritti a dieci tipologie di oggetti e non permettono quindi di generalizzare i risultati a ogni possibile richiesta. La qualità geometrica è stata inoltre analizzata in modo descrittivo, senza adottare una metrica quantitativa capace di confrontare sistematicamente i diversi output. Infine, non è stato condotto uno studio con utenti, che avrebbe permesso di valutare alcuni aspetti quali: la facilità d'uso, l'efficacia dell'interazione immersiva e l'accettabilità dei tempi d'attesa.

A partire da questi limiti possono essere individuate diverse direzioni di sviluppo. L'evoluzione dei modelli generativi tridimensionali rende di particolare interesse la possibilità di sostituire Shap-E con soluzioni open source più recenti, qualora risultino compatibili con le risorse disponibili e con i requisiti di integrazione della pipeline. Sul versante dell'interazione, una componente di manipolazione progettata specificatamente per la VR potrebbe migliorare precisione e naturalezza rispetto all'adattamento del sistema attuale. La valutazione potrebbe inoltre essere estesa attraverso un insieme più ampio di scenari, di metriche strutturate per la qualità dei modelli e di misure che comprendano anche la parte di riconoscimento vocale. Infine, uno studio con utenti permetterebbe di affiancare alle prestazioni tecniche una valutazione dell'esperienza d'uso del sistema. Tali sviluppi consentirebbero di proseguire il lavoro svolto trasformando il prototipo attuale in una soluzione più completa sia dal punto di vista della qualità dei contenuti sia da quello dell'interazione in ambiente immersivo.

# Bibliografia


[1] H. Jun, A. Nichol. (2023). *Shap-E: Generating Conditional 3D Implicit Functions*. arXiv:2305.02463.

[2] OpenAI. (2023). *Shap-E: Generate 3D Objects Conditioned on Text or Images*. Repository GitHub. https://github.com/openai/shap-e.

[3] M. Behravan, D. Gračanin. (2025). *From Voices to Worlds: Developing an AI-Powered Framework for 3D Object Generation in Augmented Reality*. arXiv:2503.16474.

[4] S. Weng, Y. Chiou, E. Do. (2024). *Dream Mesh: A Speech-to-3D Model Generative Pipeline in Mixed Reality*. 2024 IEEE International Conference on Artificial Intelligence and eXtended and Virtual Reality. DOI: 10.1109/AIxVR59861.2024.00059.

[5] Y. Kim, M. Kang, S. Kim, Y. Jung. (2026). *Voice-to-Mesh: A Mixed Reality Pipeline for 3D Content Generation Using Text-to-3D Model*. Journal of Digital Media & Culture Technology. DOI: 10.29056/jdmct.2026.06.07.

[6] E. Santarnecchi, E. Balloni, M. Paolanti, E. Frontoni, L. Stacchio, P. Zingaretti, R. Pierdicca. (2025). *MineVRA: Exploring the Role of Generative AI-Driven Content Development in XR Environments through a Context-Aware Approach*. IEEE Transactions on Visualization and Computer Graphics. DOI: 10.1109/TVCG.2025.3549160.

[7] Y. Tong, Y. Qiu, R. Li, S. Qiu, P. Heng. (2025). *MS2Mesh-XR: Multi-modal Sketch-to-Mesh Generation in XR Environments*. 2025 IEEE International Conference on Artificial Intelligence and eXtended and Virtual Reality. DOI: 10.1109/AIxVR63409.2025.00052.

[8] J. Xiang, Z. Lv, S. Xu, Y. Deng, R. Wang, B. Zhang, D. Chen, X. Tong, J. Yang. (2025). *Structured 3D Latents for Scalable and Versatile 3D Generation*. Conference on Computer Vision and Pattern Recognition. DOI: 10.1109/CVPR52734.2025.02000.

[9] A. Radford, J. Kim, T. Xu, G. Brockman, C. McLeavey, I. Sutskever. (2023). *Robust Speech Recognition via Large-Scale Weak Supervision*. ICML 2023.

[10] Khronos Group. (2021). *glTF 2.0 Specification*.

[11] OpenAI. (2022). *Whisper: Robust Speech Recognition via Large-Scale Weak Supervision*. Repository GitHub. https://github.com/openai/whisper.

[12] G. Hinton, L. Deng, D. Yu, G. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, B. Kingsbury. (2012). *Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups*. IEEE Signal Processing Magazine, 29(6).

[13] J. Lear, S. Scarle, R. McClatchey. (2019). *Asset Pipeline Patterns: Patterns in Interactive Real-Time Visualization Workflow*. EuroPLoP 2019. DOI: 10.1145/3361149.3361155.

[14] A. Graves, S. Fernández, F. Gomez, J. Schmidhuber. (2006). *Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks*. ICML 2006.

[15] B. Poole, A. Jain, J. Barron, B. Mildenhall. (2023). *DreamFusion: Text-to-3D Using 2D Diffusion*. ICLR 2023.

[16] B. Mildenhall, P. Srinivasan, M. Tancik, J. Barron, R. Ramamoorthi, R. Ng. (2020). *NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis*. ECCV 2020.

[17] J. Ho, A. Jain, P. Abbeel. (2020). *Denoising Diffusion Probabilistic Models*. NeurIPS.

[18] A. Nichol, H. Jun, P. Dhariwal, P. Mishkin, M. Chen. (2022). *Point-E: A System for Generating 3D Point Clouds from Complex Prompts*. arXiv:2212.08751.

[19] B. Kerbl, G. Kopanas, T. Leimkühler, G. Drettakis. (2023). *3D Gaussian Splatting for Real-Time Radiance Field Rendering*. arXiv:2308.04079.

[20] I. Hossain, I. Shen, T. Igarashi, O. van Kaick. (2023). *Data-guided Authoring of Procedural Models of Shapes*. Computer Graphics Forum. DOI: 10.1111/cgf.14935.

[21] R. Smelik, T. Tutenel, R. Bidarra, B. Benes. (2014). *A Survey on Procedural Modelling for Virtual Worlds*. Computer Graphics Forum. DOI: 10.1111/cgf.12276.

[22] F. Bernardini, H. Rushmeier. (2002). *The 3D Model Acquisition Pipeline. Computer Graphics Forum*. DOI: 10.1111/1467-8659.00574.

[23] F. Remondino, S. El-Hakim. (2006). *Image-based 3D Modelling: A Review*. DOI: 10.1111/j.1477-9730.2006.00383.x.

[24] P. Prusinkiewicz, A. Lindenmayer. (1990). *The Algorithmic Beauty of Plants*. DOI: 10.1007/978-1-4613-8476-2.

[25] K. Perlin. (1985). *An Image Synthesizer*. DOI: 10.1145/325334.325247.

[26] A. Lagae, S. Lefebvre, R. Cook, T. DeRose, G. Drettakis, D.S. Ebert, J.P. Lewis, K. Perlin, M. Zwicker. (2010). *A Survey of Procedural Noise Functions*. DOI: 10.1111/j.1467-8659.2010.01827.x.

[27] K. Fukaya, D. Daylamani-Zad, H. Agius. (2025). *Intelligent Generation of Graphical Game Assets: A Conceptual Framework and Systematic Review of the State of the Art*. DOI: 10.1145/3708499.

[28] M. Foxman. (2019). *United We Stand: Platforms, Tools and Innovation With the Unity Game Engine*. DOI: 10.1177/2056305119880177.

[29] H. Lee, M. Savva, A. X. Chang. (2024) *Text-to-3D Shape Generation*. DOI: 10.1111/cgf.15061.

[30] J. Liu, X. Huang, T. Huang, L. Chen, Y. Hou, S. Tang, Z. Liu, W. Ouyang, W. Zuo, J. Jiang, X. Liu. (2024). *A Comprehensive Survey on 3D Content Generation*. arXiv:2402.01166.

[31] F. Rahimi, A. Sadeghi-Niaraki, S. M. Choi. (2025). *Generative AI Meets Virtual Reality: A Comprehensive Survey on Applications, Challenges, and Future Direction*. DOI: 10.1109/ACCESS.2025.3574779.

[32] Open Source Initiative. (2007). *The Open Source Definition.* Version 1.9

[33] R. Prabhavalkar, T. Hori, T.N. Sainath, R. Schlüter, S. Watanabe (2024). *End-to-End Speech Recognition: A Survey*. DOI: 10.1109/TASLP.2023.3328283.

[34] SideFX. (s.d.). *Networks and parameters*. Houdini Documentation.

[35] Meshy. (s.d.). *Meshy API Documentation*.

[36] Tripo AI. (s.d.). *Tripo API Documentation*.

[37] Hyper3D. (s.d.). *Rodin API Documentation*.

[38] Unity Technologies. (s.d.). *Unity Asset Store*.

[39] Sketchfab. (s.d.). *Sketchfab for Developers*.

[40] TurboSquid. (s.d.). *TurboSquid – 3D Models Marketplace*.

[41] Blender Foundation. (s.d.). *Blender Manual – About Blender*.

[42] Autodesk. (s.d.-a). *Autodesk Maya*.

[43] Autodesk. (s.d.-b). *Autodesk 3ds Max*