Università Politecnica delle Marche

Facoltà di Ingegneria,
Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Magistrale in
Ingegneria Informatica e dell'Automazione

Progettazione di esperienze immersive in sistemi CAVE

Designing immersive experiences in CAVE systems

Relatore

Prof. Zingaretti Primo

Candidato

Di Rado Simone

Anno Accademico 2022/2023

Abstract

La realtà virtuale rappresenta una tecnologia avanzata che offre esperienze immersive e interattive agli utenti, simulando ambienti tridimensionali attraverso l'uso di dispositivi digitali. Uno dei dispositivi più innovativi in quest'ambito, è il Cave Automated Virtual Environment (CAVE). Si tratta di un dispositivo che permette di proiettare immagini 3D su piani ortogonali fissi, attraverso proiettori e schermi. Il vantaggio principale rispetto ad altre tecnologie VR è l'assenza di HMD, con un conseguente incremento della semplicità d'utilizzo. Questa tecnologia presenta, però, diverse sfide nel suo utilizzo, ad esempio gli elevati costi o gli effetti di distorsione spaziale; perciò, questa tesi si pone l'obiettivo di approfondire lo studio dei sistemi CAVE, andando a comprendere le metodologie e le problematiche dietro lo sviluppo di applicazioni per tali sistemi. Il sistema di riferimento è quello installato presso l'*Università Politecnica delle Marche*.
Tra gli esperimenti presentati, il più significativo è il porting su CAVE di un'applicazione realizzata dall'*Università della Basilicata*, riguardante la navigazione di grafi 3D. In questo caso, gli strumenti utilizzati sono stati *Unity* e *TechViz*, software che presenta il vantaggio di alleggerire il carico di sviluppo su CAVE, occupandosi in maniera automatica e trasparente della gestione delle immagini da proiettare. I risultati ottenuti dalle sperimentazioni hanno dimostrato la fattibilità ed efficacia di utilizzo del CAVE per la realizzazione di applicazioni complesse, dimostrando come questa tecnologia innovativa possa integrarsi con tool e framework esistenti.

Indice

Indice **3**

1 Introduzione **4**

2 Stato dell'arte **7**

3 Il nostro sistema CAVE **23**
3.1 Componenti hardware 24
3.2 Componenti software 30

4 Sperimentazioni **38**
4.1 Installazioni software 38
4.1.1 Blender 39
4.1.2 Unity 42
4.2 VisGraph 3D 53
4.3 Altri software 63

5 Discussioni **66**

6 Conclusioni e sviluppi futuri **68**

Elenco delle figure **70**

Elenco delle tabelle **72**

Bibliografia **73**

Capitolo 1

Introduzione

Il CAVE *(Cave Automated Virtual Environment)* è un dispositivo di realtà virtuale immersiva che ha l'obiettivo di immergere l'utente all'interno di un ambiente simulato. L'utente non si sentirà completamente isolato dall'ambiente esterno, ma avrà la capacità di interagire con l'ambiente simulato, attraverso l'uso di periferiche create appositamente. Sviluppato dall'università di Illinois di Chicago, viene presentato per la prima volta ad una conferenza del SIGGRAPH nel 1992. Il nome CAVE è anche un riferimento al mito della caverna di Platone, dove il filosofo contemplava la percezione, la realtà e l'illusione.

Il sistema CAVE può giungere ad un massimo di 6 piani ortogonali fissi (con un minimo di 3) su cui vengono proiettate immagini 3D da un numero di proiettori, pari al numero di schermi disponibili. Le periferiche che l'utente può utilizzare per interagire con l'ambiente sono in numero arbitrario e rilevate da un sistema di tracking.

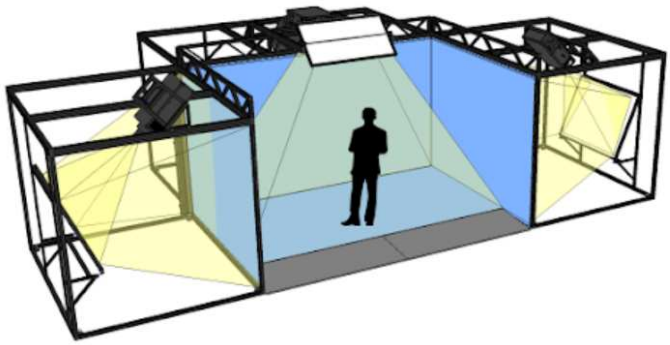

Figura 1.1: Cave Automated Virtual Environment.

Ad oggi, il cave è utilizzato sia in ambito di ricerca, ad esempio nel campo medico o nello studio del comportamento, sia in ambito aziendale come supporto allo sviluppo dei

propri prodotti. Ad esempio, per creare e testare prototipi di parti o sviluppare interfacce e simulare layout di fabbrica. I CAVE sono sempre più utilizzati anche nella progettazione collaborativa nel settore delle costruzioni. Un esempio di applicazione reale è l'indagine sui soggetti in fase di addestramento all'atterraggio di un aereo F-16.

Figura 1.2: Sistema di addestramento per aerei F16.

Il vantaggio principale è quello di creare un'esperienza realistica senza l'utilizzo di un visore HMD (*Head Mounted Display*). Di conseguenza, oltre ad essere liberi dal fastidio in termini di movimento che il visore può recare, vi è una riduzione sostanziale di motion sickness. La motion sickness è una sensazione di stordimento, disorientamento o nausea che può colpire un soggetto durante, o nelle fasi immediatamente successive, all'utilizzo di un HMD. Al fine di comprenderla meglio, puntualizziamo che tale sensazione non è limitata alla realtà virtuale, ma è la sensazione a cui ci riferiamo più comunemente con i termini di "mal d'auto" o "mal di mare".
Tuttavia, la tecnologia presenta anche degli svantaggi, in primis relativi al costo e allo spazio occorrente per installare il sistema. Inoltre, si tratta di una tecnologia nuova e non molto diffusa, motivo per il quale non si conoscono ancora bene le metodologie e gli strumenti dietro allo sviluppo di applicazioni per sistemi CAVE.

Questa tesi approfondisce le metodologie e gli strumenti a supporto dello sviluppo di esperienze immersive in sistemi CAVE.

Innanzitutto, si illustrerà lo stato dell'arte della tecnologia, in seguito, si illustreranno in maniera più dettagliata le parti che compongono un sistema CAVE, prendendo come riferimento il sistema installato presso L'*Università Politecnica delle Marche* ed, infine, verranno approfonditi gli strumenti hardware/software utilizzati per lo sviluppo di applicazioni CAVE, con relativi esempi di utilizzo e relative sperimentazioni.
I principali software di riferimento saranno Blender e Unity. Gli esperimenti avranno, invece, lo scopo di mettere alla prova il CAVE con l'utilizzo di *TechViz*, software che supporta lo sviluppo di applicazioni immersive, per capire quali sono i suoi vantaggi e svantaggi. L'esperimento più significativo è il porting di un'applicazione realizzata dall'*Università della Basilicata*, riguardante la navigazione di grafi 3D.

Capitolo 2

Stato dell'arte

Fin dalla nascita, il nostro corpo ed i nostri sensi sono abituati a interagire con l'ambiente circostante. Con l'avvento dell'informatica, successivamente, si è cominciato ad esplorare l'eventualità di generare degli ambienti virtuali in cui poter immergerci. Questo concetto iniziò a prendere vita già verso la metà degli anni '50, quando si iniziò ad esaminare la possibilità di stimolare i nostri sensi, attraverso simulazioni multisensoriali. Queste simulazioni avevano lo scopo di ricreare ambienti artificiali utilizzando stimoli che coinvolgevano più di un senso.
Il primo esempio di ciò che oggi conosciamo come realtà digitale o realtà virtuale è emerso alla fine degli anni '60 grazie agli studi che hanno portato alla creazione del primo visore di realtà virtuale, aprendo la strada a un mondo di possibilità attraverso cui le persone potessero immergersi in ambienti virtuali e interagire con essi.

Figura 2.1: Sensorama: primo visore di reltà virtuale.

Cos'è, quindi, la realtà virtuale (VR) e quali sono stati i suoi sviluppi nel tempo? La realtà virtuale è un ambiente completamente digitale creato da computer che simulano la realtà effettiva in modo non tangibile. Questo mondo virtuale viene presentato ai nostri sensi attraverso dispositivi specializzati, consentendo un'interazione in tempo reale con il mondo virtuale creato. La comunicazione tra il computer ed i nostri sensi è resa possibile da vari dispositivi come, ad esempio, periferiche di input o dispositivi per il tracciamento della nostra posizione. Seppure tutti e cinque i sensi hanno una certa importanza, ad oggi si pone maggiore enfasi sulla qualità visiva, dato che la vista è considerata il senso dominante, rendendo le esperienze virtuali più realistiche e coinvolgenti.

La Realtà Virtuale può essere suddivisa in tre categorie principali:

- **Immersiva:** l'utente viene completamente isolato dall'ambiente esterno e si immerge completamente nel mondo virtuale grazie all'utilizzo di dispositivi avanzati.
- **Non immersiva:** l'utente viene immerso all'interno di un ambiente virtuale che ha, su di lui, un impatto emotivo minore lasciandogli la consapevolezza ed il controllo dell'ambiente fisico. Un esempio potrebbe essere una console per videogiochi.
- **Semi-immersiva:** questa modalità offre all'utente un ambiente parzialmente simulato. Ci si sente, al tempo stesso, parte di una realtà diversa e connessi all'ambiente fisico circostante. Una delle attività semi-immersive più popolari è quella del virtual tour.

Figura 2.2: Virtual tour, Duomo di Milano.

In questo capitolo, si approfondiranno le tecniche e gli sviluppi della realtà virtuale immersiva, al fine di offrire al lettore un'ampia panoramica dello stato dell'arte. In particolare, ci si concentrerà sul dispositivo CAVE, protagonista di questa tesi.

Come già detto in introduzione, il primo modello di CAVE è stato sviluppato dall'università di Illinois di Chicago e presentato per la prima volta ad una conferenza del SIGGRAPH nel 1992 [1].
Nell'articolo *"The CAVE: audio visual experience automatic virtual environment"* si descrive il CAVE come una nuova interfaccia di realtà virtuale, il quale design supera molti dei problemi riscontrati da altri sistemi di realtà virtuale. Si riportano, di seguito, i principali problemi che si riscontrano nelle fasi di progettazione/utilizzo di un dispositivo di realtà virtuale immersiva [2]:

1. **Mal di mare:** Molte persone soffrono di mal di mare o di nausea quando utilizzano dispositivi VR, poiché la discrepanza tra ciò che vedono nel mondo virtuale e ciò che il loro sistema vestibolare percepisce può causare disturbi. Questo problema è noto come "mal d'auto" virtuale ed è un ostacolo importante per l'adozione della VR [3].

2. **Conforto fisico:** Indossare visori e dispositivi di realtà virtuale, per lunghi periodi, può risultare scomodo. Il peso del visore, la pressione sui punti di contatto con il viso e la temperatura all'interno del visore possono causare disagio a livello fisico.

3. **Disconnessione sociale:** L'uso della VR può isolare gli utenti dal mondo esterno, poiché sono immersi in un ambiente virtuale. Questo può portare a una mancanza di interazione sociale e alla sensazione di essere disconnessi dalla realtà.

4. **Costi elevati:** Alcuni dispositivi di realtà virtuale, come visori di alta qualità e computer potenti, possono essere costosi. Questi costi possono costituire un ostacolo all'accesso di tali dispositivi per molte persone.

5. **Problemi di salute fisica:** L'uso prolungato della VR può portare a problemi fisici, come affaticamento oculare, dolori alla testa e disturbi del sonno. È importante fare delle pause regolari durante l'uso della VR per evitare questi problemi.

6. **Limitazioni hardware:** Anche se la tecnologia VR sta avanzando rapidamente, ci sono ancora limitazioni hardware che influenzano la qualità dell'esperienza VR, come la risoluzione dei visori e la potenza di elaborazione dei computer.

7. **Contenuti limitati:** La quantità e la qualità dei contenuti disponibili per la VR possono essere limitate. Molti utenti potrebbero trovare l'offerta di giochi, esperienze e applicazioni limitata rispetto alle loro aspettative.

8. **Sicurezza fisica:** Gli utenti della VR possono ignorare il mondo fisico mentre sono immersi nell'ambiente virtuale, il che può portare a incidenti come urti contro oggetti o persone nel mondo reale.

9. **Privacy e sicurezza dei dati:** L'uso della VR può comportare la raccolta di dati personali, comprese le informazioni sul comportamento e le preferenze degli utenti. Questi dati devono essere gestiti in modo sicuro per proteggere la privacy degli utenti.

10. **Adattamento e apprendimento:** Molti utenti devono adattarsi all'uso della VR e imparare a navigare in nuovi ambienti virtuali. Questo può richiedere del tempo e può essere una barriera all'adozione di un sistema VR.

Molte di queste sfide si stanno gradualmente affrontando e superando grazie ai progressi nella tecnologia e alla continua ricerca nel settore della Computer Graphics. In particolare, il CAVE è stato progettato per affrontare alcuni dei problemi associati alla realtà virtuale tradizionale, anche se presenta alcune sfide proprie [1]. Ecco come il CAVE potrebbe risolvere almeno cinque dei problemi evidenziati:

1. **Mal di mare:** Poiché il CAVE è una stanza in cui le immagini proiettate circondano l'utente, riduce il mal di mare e la nausea, in quanto la proiezione ambientale è più coerente con i movimenti del corpo. Gli utenti hanno una visione più ampia dell'ambiente virtuale che può ridurre la sensazione di disconnessione tra ciò che vedono e ciò che percepiscono attraverso il sistema vestibolare.

2. **Conforto fisico:** Il CAVE offre un maggiore comfort fisico rispetto ai visori indossabili, poiché gli utenti non devono indossare dispositivi sulla testa o sul corpo. Tuttavia, potrebbe comunque comportare un certo affaticamento fisico dovuto alla permanenza in piedi o al movimento nell'ambiente.

3. **Disconnessione sociale:** Il CAVE consente una maggiore interazione sociale, poiché più persone possono partecipare contemporaneamente e condividere l'esperienza all'interno dello spazio virtuale. Gli utenti possono vedere e comunicare tra loro, riducendo la sensazione di isolamento.

4. **Problemi di salute fisica:** Il CAVE potrebbe presentare contenuti più ampi e dettagliati, specialmente quando utilizzato in contesti di ricerca e sviluppo. Gli utenti potrebbero avere accesso a simulazioni complesse e dettagliate, in grado di ridurre l'affaticamento della vista e altri problemi legati alla salute fisica.

Tuttavia, il CAVE presenta anche alcune limitazioni e sfide proprie:

- **Costi elevati:** L'acquisto di un CAVE richiede un investimento significativo in termini di hardware e software, rendendo il suo utilizzo costoso e limitato a istituzioni e aziende che possono permetterselo.

- **Spazio fisico:** Il CAVE richiede uno spazio fisico dedicato, che potrebbe non essere disponibile o pratico per tutti gli utenti.

- **Sicurezza:** Gli utenti all'interno del CAVE potrebbero essere a rischio di urti contro pareti o oggetti fisici nell'ambiente reale, e quindi devono essere cauti durante l'uso.

In sintesi, il CAVE può affrontare alcune delle sfide della realtà virtuale, ma introduce anche alcune sue sfide legate a costi, spazio e sicurezza. Se il CAVE è la soluzione migliore per affrontare i problemi della realtà virtuale, lo decidono le esigenze dell'utente e l'uso specifico.

Vi è, inoltre, un ulteriore problema legato al CAVE non presente tra quelli appena elencati. L'articolo *"CAVE Size Matters: Effects of Screen Distance and Parallax on Distance Estimation in Large Immersive Display Setups"* [4] ci descrive il problema della percezione spaziale. Quando si cammina all'interno di un sistema CAVE, la distanza di accomodamento, il parallasse e la risoluzione angolare variano in base alla distanza tra l'utente e le pareti di proiezione, il che può alterare la percezione spaziale. Man mano che questi sistemi diventano più grandi, c'è la necessità di valutare i principali fattori che influenzano la percezione spaziale, al fine di progettare in modo più efficace sistemi di proiezione immersivi e applicazioni di realtà virtuale.
Più avanti nella lettura, osservando più nel dettaglio i contributi nel campo della realtà virtuale da parte della letteratura scientifica, tale problematica sarà analizzata con l'introduzione di un esperimento interessante.

La realtà virtuale è un campo di ricerca multidisciplinare in costante evoluzione che coinvolge numerosi ambiti. Tra i maggiori:

1. **Intrattenimento e giochi:** L'uso più noto della VR è nell'ambito dei giochi e dell'intrattenimento. La VR offre un'esperienza di gioco immersiva in cui i giocatori possono essere trasportati in mondi virtuali, interagire con oggetti virtuali e sfidare avventure virtuali [5].

2. **Addestramento e simulazioni:** La VR è utilizzata per addestrare persone in vari settori, come medicina, aviazione, forze dell'ordine e militari. Le simulazioni virtuali consentono di praticare procedure complesse o situazioni di emergenza in un ambiente sicuro e controllato [6].

3. **Architettura e design:** Gli architetti e i designer utilizzano la VR per creare modelli virtuali di edifici e ambienti, consentendo ai clienti di esplorare e visualizzare progetti in modo dettagliato prima della costruzione fisica [7].

4. **Medicina e terapia:** La VR viene utilizzata nella riabilitazione fisica e cognitiva, aiutando i pazienti a recuperare da lesioni o disturbi neurologici. La terapia VR può anche essere utilizzata per trattare disturbi come la fobia e il disturbo da stress post-traumatico [8].

5. **Educazione:** La VR offre nuove opportunità di apprendimento immersivo. Gli studenti possono esplorare concetti complessi attraverso simulazioni educative o visite virtuali a luoghi storici o scientifici [5].

6. **Arte e creatività:** Artisti e creativi sfruttano la VR per creare opere d'arte e esperienze artistiche immersive che coinvolgono il pubblico in modo unico [9].

7. **Psicologia e neuroscienze:** La VR è utilizzata per studiare il comportamento umano, le reazioni emotive e la percezione sensoriale. Questo campo di ricerca aiuta a comprendere meglio il funzionamento della mente umana [8].

8. **Design industriale e ingegneria:** La VR è utilizzata per creare prototipi virtuali di prodotti e testarli prima della produzione fisica, risparmiando tempo e risorse [10].

9. **Realtà virtuale sociale:** La condivisione di esperienze virtuali con altre persone è un'area di crescita in cui gli utenti possono socializzare, lavorare insieme o partecipare a eventi virtuali all'interno di mondi virtuali condivisi [11].

10. **Realtà virtuale terapeutica:** La VR viene utilizzata per fornire terapie in ambienti virtuali per affrontare disturbi come l'ansia, la depressione o il dolore cronico [8].

11. **Realtà virtuale nell'industria:** L'industria utilizza la VR per la progettazione e la pianificazione dei processi, la formazione degli operatori e la simulazione di situazioni industriali complesse [10].

12. **Realtà virtuale in architettura e urbanistica:** Questo campo utilizza la VR per progettare città e spazi urbani, consentendo agli urbanisti di esplorare scenari di sviluppo e pianificazione [12].

In particolare, per il settore dell'intrattenimento, *"Developing an Interactive VR CAVE for Immersive Shared Gaming Experiences"* [13] è un recente articolo che presenta i processi di design e sviluppo per l'ambiente virtuale *MobiCave*. Nello studio, si esaminano fattori come immersione, presenza, flusso, usabilità percepita e motivazione riguardanti giocatori e spettatori. I risultati hanno mostrato conclusioni promettenti sia per scopi ludici, sia per scopi educativi, con un'esperienza giudicata altamente immersiva. Lo studio suggerisce che le configurazioni di realtà virtuale interattive per l'uso pubblico potrebbero rappresentare un'opportunità motivante per creare nuove forme di interazione sociale e collaborazione nel gaming.
L'ambiente VR *MobiCave* è, in realtà, un'implementazione semplificata di un sistema CAVE. Il sistema è, infatti, composto da tre pareti e una proiezione a pavimento, e gli utenti non indossano occhiali stereo. Il tracciamento della testa non viene utilizzato per modificare i contenuti sulle pareti e i movimenti degli utenti non influenzano i contenuti visualizzati. *MobiCave* può supportare la proiezione 3D mediante l'impiego della tecnica anaglifo. Tale tecnica coinvolge l'uso di filtri colorati per separare le immagini destre e sinistre e creare un effetto 3D. Sebbene la proiezione anaglifo potrebbe non offrire lo stesso livello di realismo di altre tecnologie di visualizzazione 3D, può comunque rappresentare uno strumento utile per visualizzare dati tridimensionali in modo economico.
Nell'articolo, si divide lo sviluppo dell'ambiente *MobiCave* in due fasi:

1. Nella prima fase, è stato definito l'ambito del progetto e sono state identificate le caratteristiche che l'ambiente virtuale avrebbe dovuto avere e come questo avrebbe dovuto migliorare l'esperienza di gioco. Le questioni considerate sono state: possibili utenti; dimensioni dell'ambiente; tipi di giochi supportati; numero di giocatori; ecc. Successivamente, è stato scelto l'hardware giusto, è stato stabilito il numero di monitor da utilizzare, le loro dimensioni ed il tipo di giochi da sviluppare e giocare. Si sono considerati fattori come costo, compatibilità e prestazioni.

2. Nella seconda fase, è stato progettato l'allestimento di realtà virtuale in cui i giocatori si sarebbero dovuti immergere. Ciò includeva la disposizione e il design dell'ambiente, oltre a eventuali effetti speciali o elementi per migliorare l'esperienza. Il processo di progettazione prevedeva scenari con più monitor per un'esperienza di gioco VR fluida. Sono stati scelti adeguatamente la scheda grafica di sistema in grado di supportare più monitor, oltre a qualsiasi altro hardware necessario come cavi, adattatori e staffe di montaggio. Inoltre, in questa fase, si sono considerate alcune

meccaniche di gioco e funzionalità come il tracciamento del movimento all'interno dei giochi sviluppati.

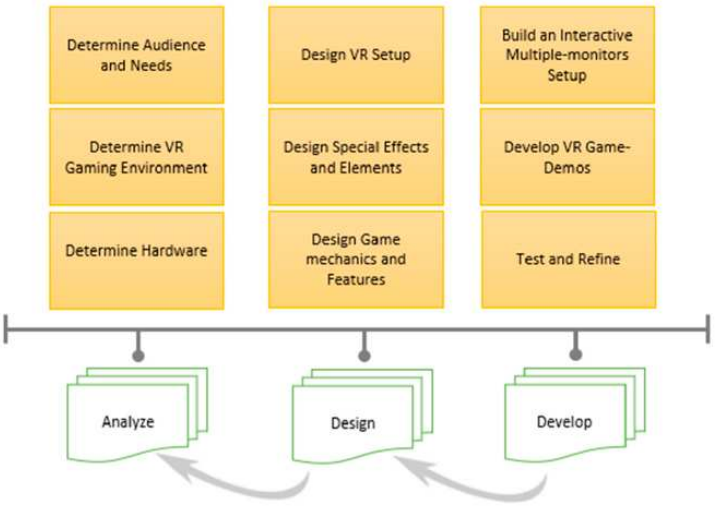

Figura 2.3: Design e sviluppo del sistema MobiCave.

MobiCave, seppure un ambiente semplificato rispetto ad un CAVE, dimostra quanto possa essere complessa la progettazione di un ambiente immersivo di questa tipologia.

La ricerca nella realtà virtuale è in continua espansione e promette di avere un impatto significativo in molti settori, migliorando le esperienze umane, l'efficienza e la sicurezza in vari contesti. Uno dei maggiori ambiti di ricerca verte verso la user experience con l'obiettivo di migliorare usabilità e accessibilità dei vari dispositivi di realtà virtuale.
Un recente lavoro [14] affronta le problematiche legate all'aggiornamento spaziale che si incontrano durante l'utilizzo di un dispositivo di realtà virtuale. In particolare, Quando ci muoviamo in un ambiente reale, le nostre rappresentazioni personali della posizione vengono aggiornate in modo automatico e senza sforzo. Tuttavia, in ambienti virtuali, questo aggiornamento spaziale è spesso disturbato o incompleto a causa della mancanza di informazioni sensoriali, specialmente quelle legate ai movimenti del corpo. Per evitare la disorientazione in ambienti di realtà virtuale, dovuta alla mancanza di informazioni sul movimento del corpo, è necessario supportare l'aggiornamento spaziale attraverso indizi sensoriali diversi, come ad esempio quelli uditivi.
Nell'esperimento descritto dall'articolo citato, i partecipanti hanno affrontato un compito di aggiornamento spaziale in un ambiente virtuale presentato in un sistema CAVE. Al termine degli esperimenti, è stato fatto compilare loro un sondaggio da cui, poi, si è dedotto il risultato dell'analisi. Il compito consisteva nel tornare ad una posizione di partenza, dopo una serie di movimenti simulati, utilizzando tre punti di riferimento lontani visibili,

come orientamento, o un unico suggerimento uditivo continuo. L'obiettivo non era quello di confrontare l'efficacia dei suggerimenti visivi rispetto a quelli uditivi, ma piuttosto di esaminare se i suggerimenti uditivi, utilizzati in combinazione con quelli visivi come riferimento, fossero validi.
Una tipica attività per indagare sull'aggiornamento spaziale è il *"Triangle-Completion Task"*. Il *"Triangle-Completion Task"* è un compito di aggiornamento spaziale nel quale il partecipante si muove lungo due lati di un percorso, prima di ritornare direttamente all'origine del percorso. Durante questo compito, dunque, i partecipanti vengono guidati lungo due lati di un triangolo e successivamente devono tornare al punto di partenza nel modo più breve possibile. Pertanto, si verifica una combinazione diretta di traslazione e rotazione per completare il triangolo.
Nella letteratura scientifica, lì dove si è già affrontato il task, sono stati osservati fenomeni generali di compressione della distanza e compressione rotazionale sia in ambienti virtuali, sia in ambienti reali. Ciò risulta in una sovrastima delle distanze brevi ed in una sottostima delle distanze lunghe. Questi fenomeni si applicano non solo ai sistemi di realtà virtuale con visori indossabili (Head Mounted Disply), ma anche e soprattutto a sistemi di proiezione come i CAVE.

Per il task di completamento del triangolo, sono stati utilizzati tre alberi, come riferimenti visivi, in modo tale che gli obiettivi fossero distinguibili dagli utenti.

Figura 2.4: Riferimenti visivi.

In Figura 2.5 si mostra la struttura dell'esperimento. I partecipanti avevano il compito di partire da una posizione iniziale (Start Position) e completare il triangolo, passando prima per il punto arancione (Firt Marker), poi per il punto rosso (Second Marker) ed, infine, tornando alla posizione iniziale. La differenza tra la posizione iniziale e quella finale dei partecipanti è stata presa in considerazione come percentuale di errore.
Gli alberi visti in Figura 2.4 sono invece rappresentati dai punti blu, mentre il riferimento uditivo è stato posto al centro dell'ambiente simulato.

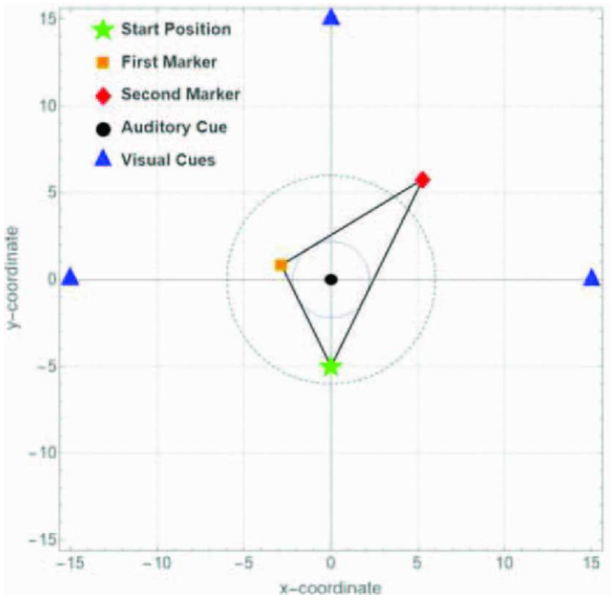

Figura 2.5: Schema dell'esperimento.

In Figura 2.9 si mostrano i risultati ottenuti dall'esperimento. Da come si osserva, media e mediana più alte sono state riscontrate negli esperimenti senza alcun riferimento (Media = 10.43m, Mediana = 8.6m). Valori più bassi si hanno, invece, negli esperimenti con riferimenti visivi/uditivi. In particolare, Media = 8.8m e Mediana = 8.53m per riferimenti visivi, Media = 8.22m e Mediana = 6.64m per iriferimenti uditivi.

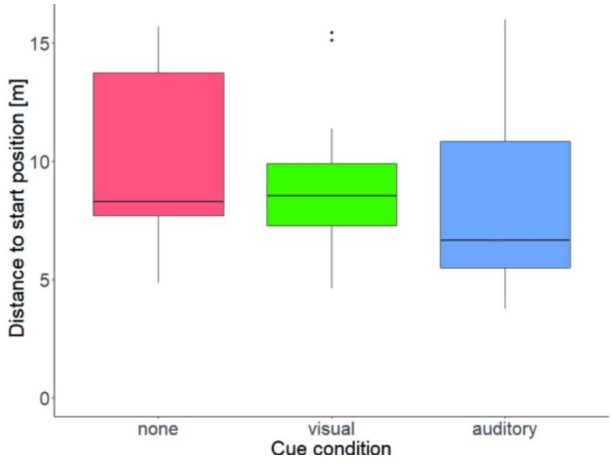

Figura 2.6: Risultati dell'esperimento.

Nel complesso, i dati hanno mostrato un miglioramento del completamento del task quando era presente un suggerimento di orientamento. I riferimenti uditivi hanno fornito un miglioramento almeno pari a quelli visivi. I risultati, ottenuti dagli esperimenti, indicano che l'uso di riferimenti uditivi in ambienti virtuali può sostenere l'aggiornamento spaziale quando mancano informazioni basate sul corpo.

All'esperimento furono in 23 a partecipare, ma tre di loro non riuscirono a completarlo per via dei problemi legati ai sintomi di cyber-sickness.
Riguardo questo problema, l'articolo *"Footstep Sound for Suppression of VR Sickness and Promotion of Sense of Agency"* [15] offre una possibile soluzione. L'articolo, infatti, considera l'utilizzo di stimoli uditivi, per potenziare la sensazione corporea, come rappresentazione indiretta di un corpo nello spazio VR. Come conclusione, il suono dei passi ha ridotto sia il mal d'auto, durante lo spostamento lungo un corridoio virtuale, quando il suono dei passi era sincronizzato con lo stimolo visivo. Non si esclude che il suono dei passi possa tornare utile anche per l'orientamento all'interno di un ambiente simulato, ma non è stato trovato alcun articolo, in letteratura, che studia questo caso.

Dall'esperimaneto visto, sono già intuibili le difficoltà che si hanno nello sviluppo di un ambiente immersivo in sistemi CAVE. Di recente, si sta cercando di trovare e studiare soluzioni atte a semplificare lo sviluppo di questi ambienti.
Uno degli articoli più recenti che prende in considerazione questa tematica è *"Dynamic Generation of Immersive CAVE Environments: Using Digital Game Mechanisms"* [16]. L'articolo presenta un generatore dinamico di ambienti collaborativi immersivi in sistemi CAVE, che utilizza la stereoscopia per proiettare immagini 3D su un CAVE simulato. L'ambiente supporta l'interazione in tempo reale con gli elementi proiettati.
Il risultato del progetto è un sistema in grado di generare dinamicamente ambienti virtuali in CAVE di dimensioni e formati variabili. In sostanza, lo strumento produce un modello virtuale di questo tipo di ambiente che mostra non solo la costruzione dell'oggetto CAVE stesso, ma simula anche le proiezioni interne. Tale strumento perende il nome di ICE-S *(Immersive CAVE Environment - Simulator)* ed è stato sviluppato utilizzando il motore di gioco Unity, con l'obiettivo di simulare un ambiente immersivo nel CAVE con dimensioni e risoluzioni dinamiche. Ciò consente la creazione di vari scenari nell'ambiente virtuale sotto forma di un modello digitale interattivo. Lo strumento è in grado di simulare l'intera struttura fisica di una CAVE e dimostrare le possibili proiezioni che possono essere realizzate all'interno dell'ambiente. Ciò può essere utilizzato per assistere nella costruzione del modello fisico e nella pianificazione dell'usabilità.
Per definire la geometria del sistema CAVE, ICE-S utilizza il raggio, che definisce il diametro del CAVE; l'altezza, che definisce l'altezza della struttura; la risoluzione, che definisce il numero di lati; e l'angolo, che stabilisce se il CAVE sarà a 360° o 180°.

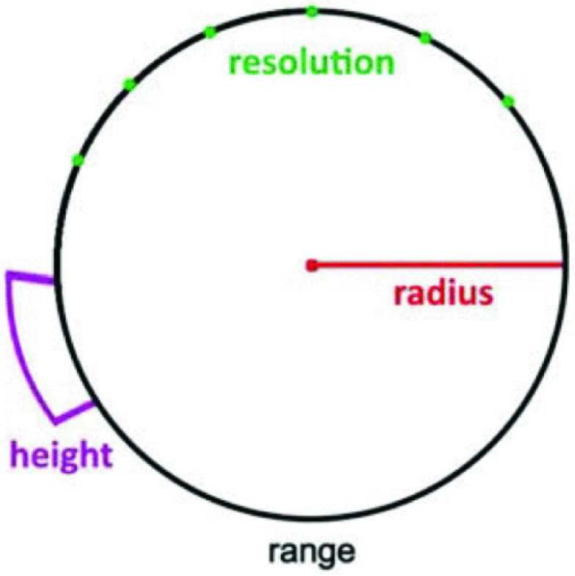

Figura 2.7: Parametri di un CAVE.

Le geometrie del CAVE che possono essere realizzate in ICE-S sono fondamentalmente di 4 tipologie. Si riporta, nella Tabella 2.1, una lista dettagliata delle possibili geometrie in cui si suppone che l'altezza sia la stessa.

Tabella 2.1: `Geometrie costruibili in ICE'S`

Tipologia	**Geometria**	**Numero di facce**	**Raggio**	**Copertura angolare**
a	Cilindrica	50	50	360
b	Mezzaluna	30	50	180
c	Cubica	4	30	360
d	Triangolare	3	40	360

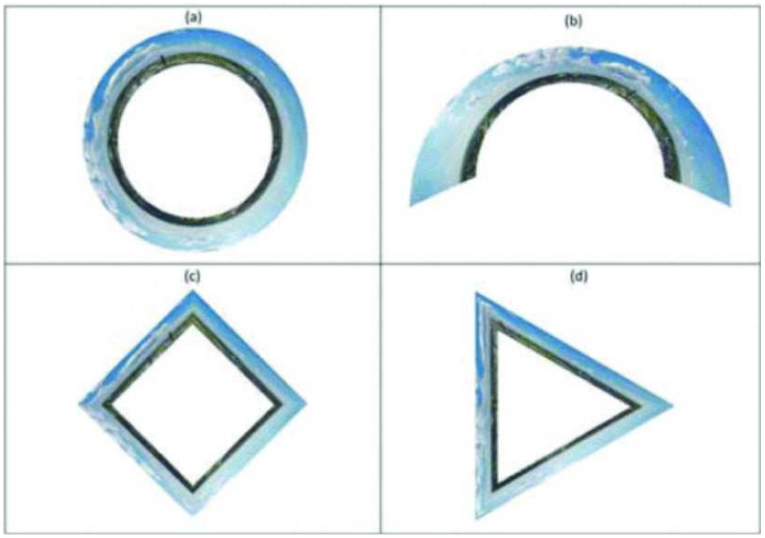

Figura 2.8: Geometrie di un CAVE realizzabili con ICE'S.

Per applicare texture bidimensionali su un oggetto tridimensionale, è necessario impostare le coordinate di texture UV. Queste definiscono come un'immagine dev'essere mappata all'interno di una geometria. Infatti, ogni coordinata risulterà associata ad un vertice della geometria.
Nelle proiezioni statiche, ICE'S genera una singola texture con tutti gli elementi che verranno resi sull'oggetto. Per i contenuti che vengono aggiornati in tempo reale, il processo è diverso. In questo caso, una texture deve essere creata dinamicamente per rappresentare lo stato più recente della proiezione, come nel caso di un video in cui la texture deve essere aggiornata per ogni frame successivo.
I passi da seguire per personalizzare la proiezione, sono i seguenti:

- Specificare l'immagine che sarà utilizzata come sfondo.
- Specificare gli elementi che saranno resi in primo piano e la posizione e le dimensioni di ciascun elemento.

La texture è creata a strati, con il primo strato che rappresenta l'immagine di sfondo e l'ultimo strato che rappresenta l'immagine visualizzata in primo piano. Le immagini possono essere archiviate nella loro dimensione originale o ridimensionate a una dimensione specifica. Per farlo, vengono salvate due istanze della texture: una con le dimensioni originali e un'altra con le dimensioni ridimensionate. Da un lato, è necessario gestire le informazioni sulle dimensioni originali della texture. Dall'altro, non è richiesto di ridimensionare la texture ogni volta che viene renderizzata. Infine, la texture generata viene renderizzata sul CAVE, utilizzando le coordinate UV, ed il CAVE regola automaticamente la proiezione nel suo spazio interno.

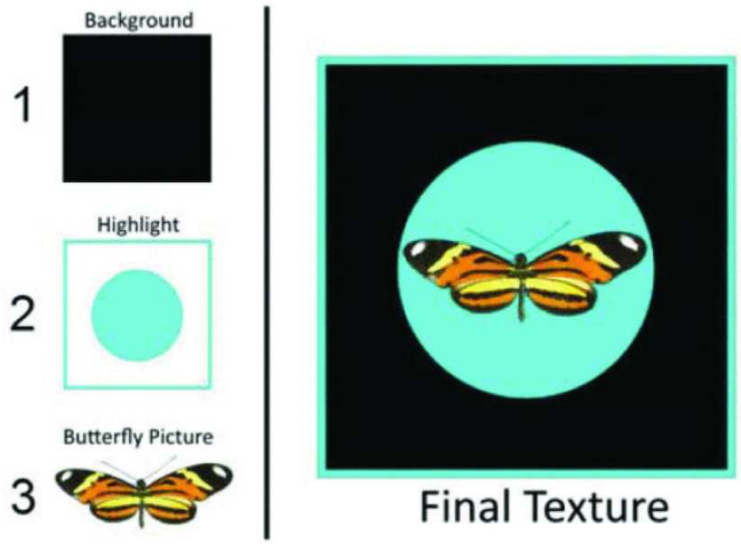

Figura 2.9: Sovrapposizione delle texture.

In un CAVE cilindrico, ad esempio, la proiezione dev'essere continua, ma i punti di inizio e fine devono incontrarsi in una punto specifico. Per garantire che la texture venga visualizzata correttamente in questo punto di intersezione, il rendering deve avvenire in maniera circolare, in modo che i pixel che si estendono oltre un'estremità siano resi sull'estremità opposta. Per ottenere ciò, la posizione xy di ogni pixel viene calcolata utilizzando delle formule specifiche di modulo, ovvero:

- $x = x \ mod \ width$: calcola il resto della divisione della coordinata x per la dimensione.
- $y = y \ mod \ height$: calcola il resto della divisione della coordinata y per l'altezza.

Dove x e y rappresentano le coordinate di ciascun pixel nella texture finale, mentre *width* e *height* corrispondono alle dimensioni complessive della texture. Queste formule assicurano che le coordinate dei pixel rimangano entro i limiti della larghezza e dell'altezza della texture, garantendo così che l'immagine possa essere visualizzata in modo corretto anche quando si attraversa il punto di intersezione.

Com'è possibile intuire, lo spostamento continuo delle texture richiede molta potenza computazionale, poiché devono essere ridisegnate in modo continuo, portando a una diminuzione del frame rate per secondo (FPS) e influendo negativamente sulla fluidità del movimento. Per ridurre questo costo, nell'articolo viene spiegata in dettaglio la strategia utilizzata per gestire il processo di rendering delle texture nella maglia triangolare. Questa strategia è composta da due strutture di programmazione orientata agli oggetti: *CaveElement* e *CaveTextureRenderer*.
In sostanza, ogni elemento da renderizzare nel CAVE è considerato un *CaveElement*, comprendente dati di texture, posizione, dimensioni e stato (modificato/non modificato, quindi valore booleano). Il *CaveTextureRenderer* crea istanze di *CaveElement* e le gestisce. Durante la compilazione del programma, il *CaveTextureRenderer* genera un insieme ordinato di istanze di *CaveElement* per ciascuna immagine nella texture, creando la texture iniziale da renderizzare. Ad ogni aggiornamento del frame, il *CaveTextureRenderer* controlla le modifiche in ciascun *CaveElement* all'interno della texture. Se ci sono elementi modificati, la texture viene rigenerata utilizzando i valori degli elementi aggiornati. Questa strategia di gestione ottiene migliori risultati nel rendering di texture in movimento, poiché elimina la necessità di ridisegnare gli elementi ad ogni frame e aggiorna gli oggetti solo quando necessario. Questa soluzione tiene conto anche degli oggetti aggiunti o rimossi dalla proiezione durante l'esecuzione.

In Figura 2.10 vi è un esempio di rendering finale da parte dell'applicazione ICE'S. ICE'S si dimostra quindi un simulatore che genera ambienti di realtà virtuale immersiva in sistemi

CAVE, offrendo la possibilità di simulare l'utilizzo di risorse immersive in contesti diversi. Il simulatore consente agli sviluppatori di risparmiare tempo nello sviluppo, potendo emulare, con grande fedeltà, la resa visiva finale che si avrebbe all'interno di un sistema CAVE.

Lì dove ICE'S riesce ad ottenere un buon risultato, TechViz risulta superiore sotto certi aspetti. TechViz, uno degli argomenti principali trattati in questa tesi, sarà introdotto e approfondito nel capitolo successivo.

Figura 2.10: Rendering finale.

Un'altra tematica che si sta affrontando di recente è quella della visualizzazione dei modelli di grafo 3D.
I grafi sono strutture matematiche discrete che rivestono interesse sia per la matematica che per un'ampia gamma di campi applicativi. I grafi inoltre sono utilizzati in aree come topologia [17], teoria degli automi, funzioni speciali, geometria dei poliedri [18]. I grafi si incontrano in vari studi dell'informatica, ad esempio, per schematizzare programmi, circuiti, reti di computer, mappe di siti, ecc. Essi inoltre sono alla base di modelli di sistemi e processi studiati nell'ingegneria, nella chimica, nella biologia molecolare, nella ricerca operativa, nella organizzazione aziendale, nella geografia [19] (sistemi fluviali, reti stradali, trasporti), nella linguistica strutturale, nella storia [20] (alberi genealogici, filologia dei testi).

Qualsiasi sia il processo, ultimamente c'è una forte necessità di visualizzare tali modelli. L'interesse recente nei confronti di reti sociali, architetture software e pianificazione ha

portato all'applicazione della visualizzazione e dell'esplorazione di grafi per assistere gli analisti con indizi visivi rilevanti per comprendere la struttura intrinseca dei dati. L'articolo *"Virtual Reality Interfaces for Interacting with Three-Dimensional Graphs"* [21] nasce con lo scopo si supportare questo nuovo trend. La quantità di questi dati strutturati sta crescendo costantemente e le visualizzazioni dei grafi mirano ad aiutare nella comprensione dei grafi fornendo viste grafiche per rivelare strutture nascoste e altre interessanti features. Esistono diverse tecniche per la visualizzazione 2D, mentre la visualizzazione 3D è un nuovo campo.
L'articolo citato si pone l'obiettivo di sperimentare diverse modalità di navigazione, al fine di capire qual è la soluzione migliore. Tra le varie modalità, vi sono: mouse e tastiera, joypad, HTC Vive e Leap Motion (insieme ad un Oculus Rift, come periferica di output). Il loro studio ha comportato lo sviluppo di un'applicazione apposita per la navigazione di modelli di grafo e dimostra che gli utenti trovano particolarmente impegnative le tecnologie innovative, rispetto a quelle tradizionali.

Tra gli obiettivi che questa tesi si pone, vi è anche quello di dare un contributo allo studio appena descritto. Partendo dall'applicazione sviluppata, si è prodotto un porting sul sistema CAVE con l'obiettivo di analizzarlo come caso di studio.

Capitolo 3

Il nostro sistema CAVE

Come compreso dai capitoli precedenti, Un CAVE è tipicamente di forma cubica, con pareti che visualizzano immagini 3D, a cui l'utente può accedere indossando occhiali stereoscopici per sperimentare una percezione di profondità. La posizione e l'orientamento dell'utente vengono generalmente tracciati e utilizzati per aggiornare la visualizzazione in tempo reale, fornendo un elevato livello di interattività e immersione.
I CAVE rappresentano una sfida nella progettazione e costruzione perché i loro numerosi componenti derivano, in larga parte, da tecnologie preesistenti create originariamente per scopi diversi. L'integrazione di questi componenti e la realizzazione di particolari parti personalizzabili, come schermi e schede grafiche, hanno richiesto anni di ricerca e sviluppo.

Un CAVE è composto principalmente dai seguenti componenti:

1. **Proiettori**: Il cuore del sistema CAVE sono i proiettori che proiettano immagini 3D sulle pareti e sul pavimento della stanza. Questi proiettori possono essere più di uno e vengono sincronizzati per fornire una visione continua e fluida dell'ambiente virtuale.

2. **Schermi retroilluminati**: Le pareti e il pavimento del CAVE sono costituiti da schermi retroilluminati speciali, che riflettono le immagini proiettate dai proiettori. Questi schermi assicurano che le immagini siano chiare e visibili agli utenti che si trovano all'interno della stanza.

3. **Sistema di tracciamento**: Per monitorare la posizione e l'orientamento degli utenti all'interno del CAVE, è necessario un sistema di tracciamento. Ciò può essere realizzato utilizzando telecamere di tracciamento ottico o sensori di movimento per rilevare i movimenti e aggiornare la vista dell'ambiente virtuale in tempo reale.

4. **Computer e hardware di elaborazione**: Il sistema CAVE richiede computer potenti e hardware di elaborazione per gestire le complesse grafiche 3D ed il rendering in tempo reale delle immagini per gli schermi.

5. **Software di visualizzazione**: Il software di visualizzazione è ciò che gestisce il rendering e l'interazione con l'ambiente virtuale. Questo software elabora i dati dei modelli 3D e gestisce la visualizzazione delle immagini proiettate sui vari schermi del CAVE.

In questo capitolo si illustreranno in dettaglio gli strumenti hardware e software del sistema utilizzato dall'*Università Politecnica delle Marche*. In particolare, si presterà una maggiore attenzione al software di visualizzazione.

3.1 Componenti hardware

Il sistema a cui si farà riferimento è stato realizzato dall'azienda *More* e installato presso l'*Università Politecnica delle Marche*.

More è un'azienda di Pesaro, con sede operativa a S. Benedetto del Tronto (AP, Marche) che progetta e realizza soluzioni avanzate per la realtà virtuale 3D, che aprono nuove opportunità per i brand. Tra i principali partener dell'azienda vi sono: *ART*, *TechViz*, *Lenovo*, *nvidia*, ecc. Ed è proprio da questi partener che derivano le tecnologie di cui il sistema CAVE è composto.

1. **Proiettori**: I proiettori (Figura 3.1) sono di due tipi e possiedono caratteristiche diverse. La Tabella 3.1 mostra tali caratteristiche.

Tabella 3.1: `Caratteristiche dei proiettori`

Caratteristiche	**BenQ Videoproiettore LU960UST**	**BenQ Videoproiettore LU96OST**
Quantità	2	1
Luminosità	5200	5500
Risoluzione	1920x1200 WUXGA	1920x1200 WUXGA
Rapporto di contrasto din.	3000000:1	3000000:1
Tecnologia	DLP	DLP
Formato immagine	16:10	16:10
Dimensione	(L x A x P) 480 x 157 x 473 mm	(L x A x P) 480 x 183 x 402 mm
Larghezza	480 mm	480 mm
Altezza	157 mm	183 mm
Profondità	473 mm	402 mm
Peso	12 kg	12 kg
Potenza assorbita (attivo)	500 W	500 W

Figura 3.1: Proiettori.

2. **Schermi retroilluminati**: Struttura autoportante con 2 schermi flessibili di dimensioni 2.850 x 1.780 mm, per fronte-proiezione di circa 2.90mt, di base posti a 90°. Inoltre, un pavimento proiettabile, che copre parzialmente lo spazio totale a disposizione, di dimensioni 2.850 x 1.780 mm.

Figura 3.2: CAVE.

3. **Sistema di tracciamento**: Il sistema di tracciamento si compone a sua volta di tre componenti principali, indispensabili per l'utilizzo del CAVE:

   - ***Camere ottiche***: Le telecamere ottiche (Figura 3.3) sono in totale quattro. Le telecamere TRACKPACK/E rappresentano la seconda generazione delle telecamere di tracciamento ottico di fascia media di ART. Con le sue prestazioni di alto livello, TRACKPACK/E offre un rapporto qualità/prezzo superiore, mentre con le sue dimensioni compatte offre massima versatilità. Se ne osservano le caratteristiche in Tabella 3.2.

Tabella 3.2: `Caratteristiche delle telecamere`

Caratteristiche	**TRACKPACK/E**
Image sensor	1152 x 948 Pixel
Frame rate (sensor)	60 Hz @ full frame, up to 120 Hz with reduced field of view
Image processing	on ART Controller
IR source	high power LEDs, 850 nm, adjustable in intensity
Status indicator	2 LEDs; dimmable
Data transfer, sync and power supply	Single cable solution; data & sync: Gigabit Ethernet power: PoE (IEEE 802.3af-2003) via max. 100 m RJ45 cable
Cooling	passive, noiseless
Temperature	0 ... 38° C
Power consumption	5 W
Size (W x H x D)	70 x 59 x 97 mm
Weight	Approx. 0.5 kg

Figura 3.3: Telecamera ottica.

- ***Occhiali stereoscopici per visione 3D***: Estremamente stabili, con utilizzo a lunga distanza grazie alla frequenza radio. Per l'utilizzo di questi occhiali (Figura 3.4), non è richiesto nessuno schermo argentato e le immagini di alta qualità vengono preservate da qualsiasi posizione. La sincronizzazione con le frequenze radio evita interferenze da parte di altri dispositivi. Gli occhiali sono ricaricabili, con durata della batteria fino a 75 ore, ed il Peso è di soli 56 grammi. Riescono a rendere immagini luminose con colori naturali e possiedono una funzione di spegnimento automatico per il risparmio energetico. Grazie al nasello in gomma e le stanghette regolabili, si ha un comfort totale. Ulteriori dettaggli in Tabella 3.3.

Tabella 3.3: `Caratteristiche degli occhiali 3D`

Caratteristiche	**Occhiali 3D**
projector type	All
Screen	White - gain 1.0/1.2/...
Compatible emitter	Volfoni RF emitters
Optical transmission	38%
Stereoscopic contrast	>940:1
Field of vision (H, V)	170°-115°

Figura 3.4: Occhiali 3D.

- ***Marker***: Oltre agli occhiali, in Figura 3.4, notiamo la presenza di sei marker. ART fornisce una varietà di diversi marker e tipi di marker, in base alle esigenze di ciascuna installazione. Nel nostro caso, i marker sono passivi. I marker

passivi sono marker retro-riflettenti, ossia riflettono la radiazione infrarossa in arrivo nella direzione della luce in entrata.

- ***Flystick***: Il Flystick è un dispositivo di interazione wireless per le applicazioni di realtà virtuale con un bersaglio passivo protetto. Dispone di quattro tasti in totale. La trasmissione wireless dei dati al controller è garantita da un trasmettitore radio USB nella banda ISM. I dati del Flystick possono essere ricevuti da trackd, VRPN, VR Juggler e dalla maggior parte delle applicazioni con interfacce dirette. Per il tracciamento del dispositivo all'interno dell'ambiente virtuale, sono necessari dei marker. I marker sono uguali a quelli montati sugli occhiali e sono posizionati all'interno del controller, in modo da non essere visibili all'utente.

Tabella 3.4: `Caratteristiche del flystick`

Caratteristiche	**Flystick**
Tracking range (@ 3.5mm focal length)	approx. 4 m
Radio range	at least 7 m
Type ID	IEEE 802.15.4 Zigbee LR-WPAN
Frequency	ISM band: 2.4 GHz, 8 channels
Transmission power	4 dBm
Rechargeable batteries	standard batteries 2xAA
Battery charging duration	5 hours
Operation possible with connected charger	yes
Size	183 x 188 x 99 mm
Weight	250 g

Figura 3.5: Flystick.

4. **Computer e hardware di elaborazione**: Il nostro sistema ha un numero di schermi pari a quattro, considerando anche il monitor principale. Questo significa che i modelli che decidiamo di proiettare, devono essere renderizzati su più schermi. In termini di costi computazionali, ogni operazione ha un peso non indifferente e, per questo motivo, si è deciso di optare per componenti hardware decisamente sofisticati e con potenza di elaborazione notevole.

Tabella 3.5: `Caratteristiche del computer`

Caratteristiche	**Computer**
Processore	Processor Processore Intel Xeon Silver 4215R
RAM	6x DDR4 da 32 GB a 2.933 MHz ECC
Scheda grafica	2x NVIDIA RTX A6000 da 48GB con NVLink
Memoria di massa	1x SSD M.2 PCIe TLC Opal da 512 GB 1x SSD M.2 PCIe TLC Opal da 1.024 GB
Sistema operativo	4 Windows 10 Pro 64 for Workstation
Porte	2x port Integrated Ethernet

Oltre ad un computer molto potente, è necessario un controller per le telecamere che si occupano del tracciamento all'interno dell'ambiente CAVE. Come visto, le telecamere utilizzate sono le TRACKPACK/E, di conseguenza il controller utilizzato è l'ART CONTROLLER.
Il Controller ART può gestire e alimentare direttamente fino a 8 telecamere di tipo ARTTRACK5 & ARTTRACK6/M o TRACKPACK/E (nel nostro caso, quattro telecamere). Un maggior numero di telecamere è supportato tramite switch (ARTTRACK5 & ARTTRACK6/M) o tramite cascata (TRACKPACK/E). Il controller è disponibile in un alloggiamento montabile su rack (come nel nostro caso) o può essere posizionato anche su un tavolo, ad esempio, grazie ai piedini in gomma nella parte inferiore.

Tabella 3.6: `Caratteristiche del controller`

Caratteristiche	**ART CONTROLLER**
Voltage	100 - 240 V
Power consumption	max. 425 W
Synchronisation in	BNC: Video Signal (75 hom), TTL, LVTTL
Synchronisation out (optional)	BNC: Video Signal (75 hom)
Data in / out	2x RJ45, Gigabit Ethernet (LAN, optional Cascade)
Camera ports	8x RJ45, Gigabit Ethernet, PoE+ (IEEE 802.3at-2009)
Peripherals	6 x USB 2.0 (e.g. Radio Transceiver2/3, service access)
Temperature	0 ... 38°C
Size (W x H x D)	480 x 135 x 345 mm (3U)
Weight	approx. 9.8 kg

Figura 3.6: ART CONTROLLER.

3.2 Componenti software

Tra i componenti software principali, vi sono i seguenti:

1. **Software per il controllo dei proiettori**: Il software in questione è *DMS Local (Device Management Solution)*. Si tratta di uno strumento software per computer che permette la gestione remota e centralizzata di più proiettori digitali, attraverso una rete locale. Le capacità di gestione di DMS Local includono la possibilità di monitorare, controllare e configurare in remoto i proiettori individualmente o in gruppo. Le sue funzioni e i suoi vantaggi principali sono i seguenti:

   - Alimentazione On/Off.
   - Gestione dei compiti.
   - Identificazione.
   - Design UI amichevole per l'utente.
   - Controllo dei dispositivi cross-subnet.
   - Identificazione personalizzata del comando.
   - Monitoraggio dei dispositivi.

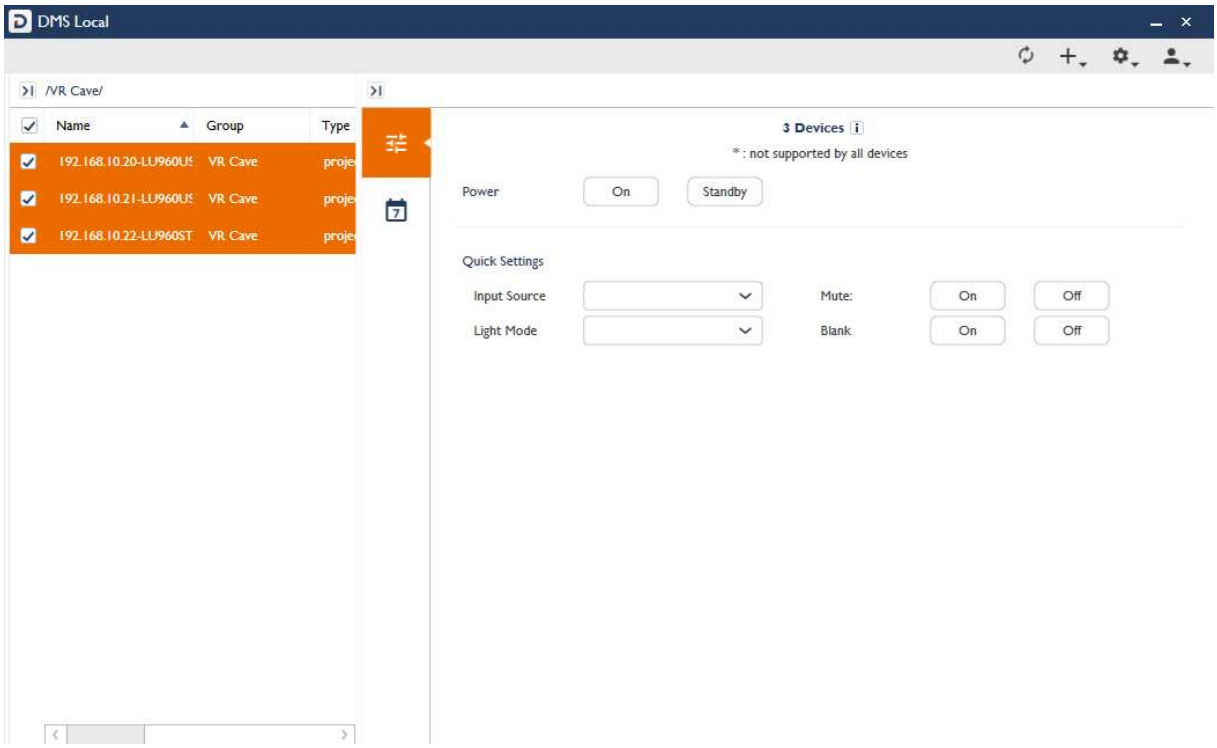

Figura 3.7: DMS Local.

2. **Software per l'allineamento dei proiettori**: *Immersive Display PRO* è il pacchetto software per la correzione geometrica e il blending soft-edge per desktop DirectX, OpenGL e Windows 7/8/10. È l'unico software che si integra perfettamente nel desktop di Windows 7/8/10 e offre correzione geometrica delle immagini e blending soft-edge utilizzando la GPU del PC. Non sono necessarie ulteriori apparecchiature per la proiezione su schermi regolari e irregolari (schermi cilindrici, cupole complete e parziali e qualsiasi altra superficie di proiezione sagomata). *Immersive Display PRO* utilizza i componenti principali di *Fly Elise-ng Immersive Display* per consentirne la configurazione.

Figura 3.8: Fly.elis-ng Immersive Display.

Un software per l'allineamento dei proiettori è necessario affinchè venga eseguito un allineamento meccanico dei proiettori, evitando, così, che le proiezioni si sovrappongano tra loro. Dunque, l'effetto di questo software è quello di avere, per

ogni proiettore, una proiezione centrata sullo schermo corrispondente. Come effetto collaterale, si avrà una disincronizzazione di uno o più schermi che è possibile correggere direttamente dal pannello di controllo *Nvidia* (3.9).

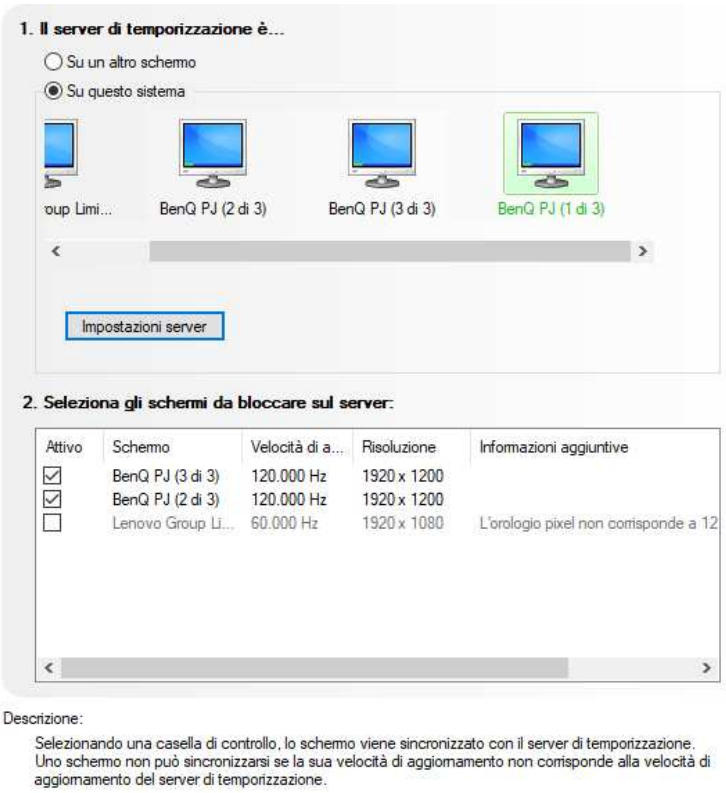

Figura 3.9: Sincronizzazione degli schermi.

3. **Software per la gestione dell'ART CONTROLLER**: Tutti i sistemi ART sono stati progettati attorno a un'architettura integrata e adattabile che offre un tracciamento del movimento affidabile e preciso per sistemi di proiezione immersiva.

Come si è visto, il sistema hardware di solito è composto da quattro parti principali:

- ***Controller***: fornisce l'interfaccia fisica tra le telecamere e il computer host, direttamente o tramite una rete.
- ***Array di tracciamento infrarosso***: Può essere una combinazione di telecamere TRACKPACK, ARTTRACK e SMARTTRACK.
- ***Dispositivi di navigazione e interazione precisi ed ergonomici***: Flystick o Fingertracking.
- ***Vasta gamma di target ottici personalizzati ART***: Utili per occhiali stereoscopici, HMD, motion capture e molti altri usi.

Il software di controllo DTRACK è il cuore di ogni sistema ART. Ha lo scopo di supportare la configurazione rapida, facile e accurata delle prorpie telecamere e di fornire una certa stabilità durante l'operazione. I dati 3DOF e 6DOF vengono catturati dalle telecamere e inviati tramite il Controller a DTRACK, dove le orientazioni e le posizioni in tempo reale vengono calcolate, risolte e inviate direttamente al software 3D.

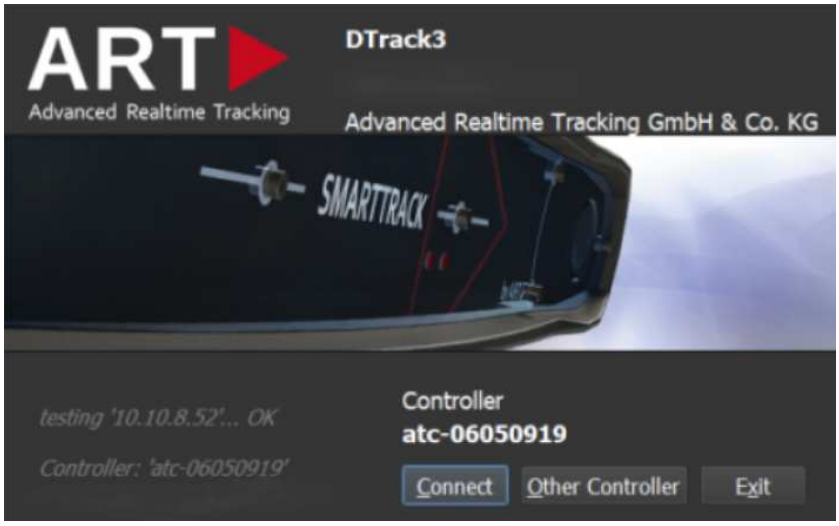

Figura 3.10: DTRACK.

DTRACK è progettato con un'interfaccia utente semplice e amichevole che può essere utilizzata direttamente connessa a un computer host del sistema. Consente di controllare il sistema da qualsiasi punto attraverso una rete informatica. I dati di tracciamento vengono scambiati tra il Controller ART e l'interfaccia utente DTrack tramite una connessione TCP/IP, offrendo flessibilità e stabilità durante l'operazione.

Durante la configurazione del software saranno rilevati in automatico i punti di luce che devono essere compensati durante la gestione delle proiezioni. Un risultato migliore si ottiene in un ambiente più scuro possibile, senza fonti di luce esterne che potrebbero disturbare. In ogni caso, saranno sempre presenti disturbi dovuti alla luce riflessa, da parte di ogni singolo schermo, sugli altri schermi dell'ambiente di simulazione. Questi disturbi vengono segnalati dal software attraverso quadratini grigi o punti rossi, visibili in Figura 3.11.

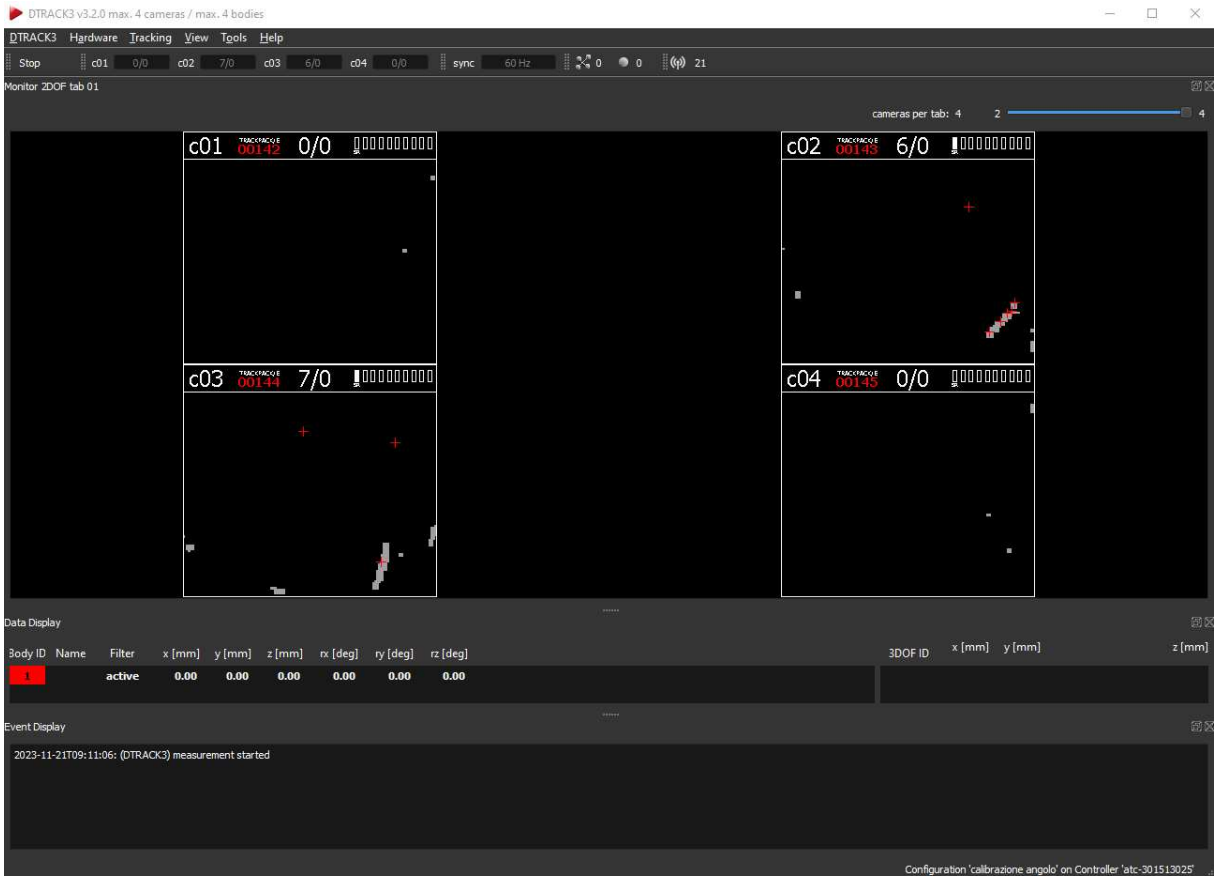

Figura 3.11: Configurazione dell'ART controller.

4. **Software per la gestione delle proiezioni**: In ultimo, ma in assoluto il più importante, il software che gestisce le proiezioni sugli schermi del CAVE. In Figura 3.12 se ne presenta l'architettura.
   Da come si osserva, *TechViz* rappresenta ciò che in informatica si definisce "demone". Un demone, in informatica e più in generale nei sistemi operativi multitasking, è un programma eseguito in background, cioè senza che sia sotto il controllo diretto dell'utente, tipicamente fornendo un servizio all'utente. Principalmente viene utilizzato sui server ma anche su normali PC.
   TechViz è un software che si interpone fra l'applicazione grafica (alto livello) e la scheda grafica (basso livello). L'obiettivo è quello di raccogliere le informazioni, che dall'applicazione vengono inviate alla scheda grafica, sfruttandole per gestire correttamente il rendering sugli schermi del CAVE. Oltre a raccogliere i segnali dall'applicazione grafica, vi è anche un controller che legge le informazioni di tracking in input e le invia ai proiettori.

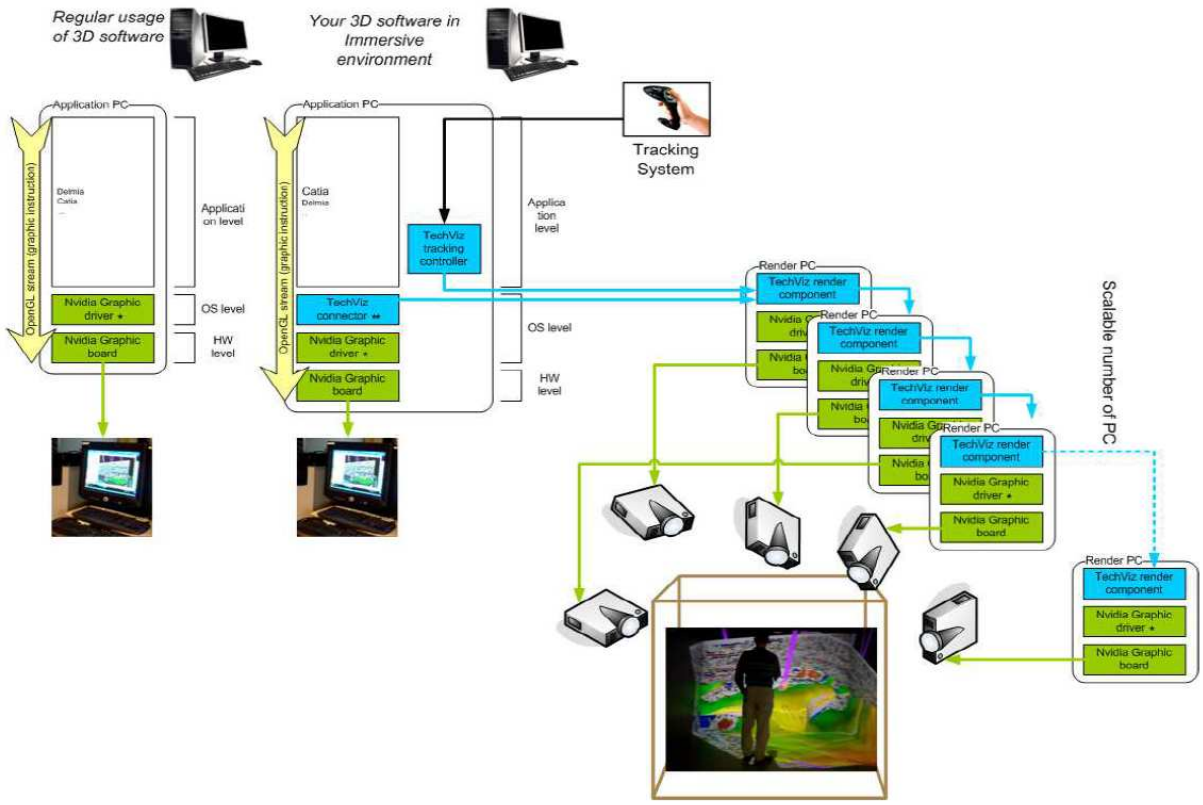

Figura 3.12: Architettura di TechViz.

Le funzionalità di cui dispone *TechViz* sono molteplici. Infatti, è possibile:

- Creare interazioni personalizzate.
- Creare dei driver.
- Personalizzare diverse funzioni (mapping dei tasti, menù, variabili, ecc.).
- Catturare immagini/video di modelli 2D/3D.
- Utilizzare software perfettamente integrati, come Unity o Blender.
- Sfruttare le informazioni di tracking (completamente indipendenti dal software).

Più precisamente, i passi del funzionamento di TechViz (Figura 3.13) sono i seguenti:

- Il *TechViz Connector* intercetta le istruzioni OpenGL dall'applicazione 3D e le invia al *TechViz Render Component*.
- Il *TechViz Tracking controller* connette lo stream di informazioni dei dispositivi di tracking con il *TechViz Render Component* sul cluster di rendering.
- *TechViz API* permette agli utenti di interagire con il modello.

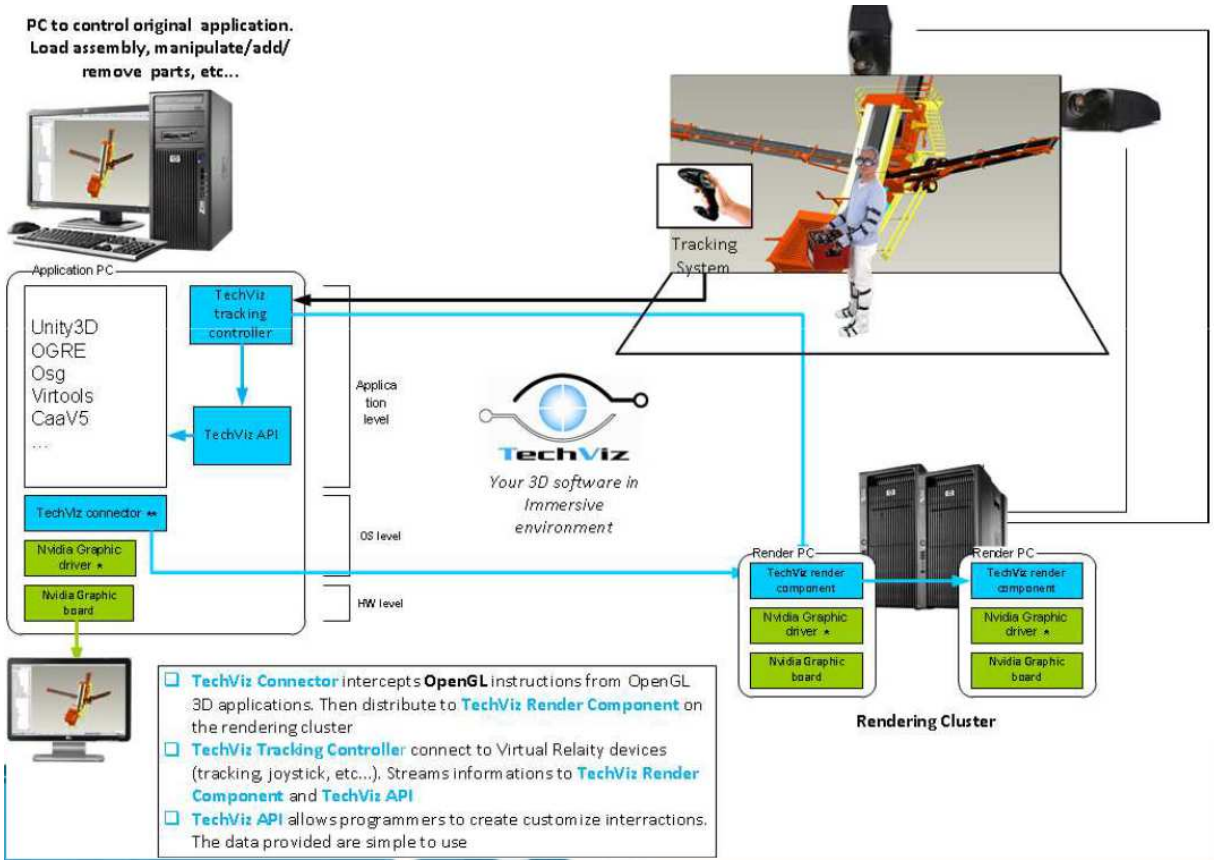

Figura 3.13: Componenti di TechViz.

In Figura 3.15 osserviamo uno schema esplicativo su come avvengono le interazioni all'interno dell'ambiente virtuale.
I dati dell'applicazione grafica vengono letti da *TechViz* con lo scopo di effettuare il rendering all'interno dell'ambiente virtuale. Ne consegue che ogni interazione che l'utente avrà con il modello verrà letta da *TechViz* ed i cambiamenti saranno visibili, in tempo reale, anche all'interno dell'ambiente virtuale.
TVZLib è il nome dell'API di *TechViz* che permette la comunicazione tra i dispositivi di tracking e l'applicazione grafica realizzata in Unity. Grazie a TVZlib, si possono ricevere informazioni sullo stato dell'utente all'interno dell'ambiente. Utilizzando queste informazioni come input nell'applicazione nativa 3D, è possibile:

- Gestire gli input da parte dell'utente e le possibili interazioni con l'ambiente.
- Sviluppare funzioni in fase di programmazione dell'applicazione grafica.
- Osservare nell'ambiente VR, grazie alla connessione in tempo reale con l'applicazione nativa, tutte le modifiche visive apportate al modello 3D.

TVZLib consente, dunque, di sviluppare un numero infinito di scenari nell'ambiente VR, come ad esempio cambiare il materiale/dimensioni di un oggetto, avviare un'a-

nimazione nell'applicazione nativa (vista in tempo reale nell'ambiente VR), aprire porte, interagire con oggetti in una scena, ecc.

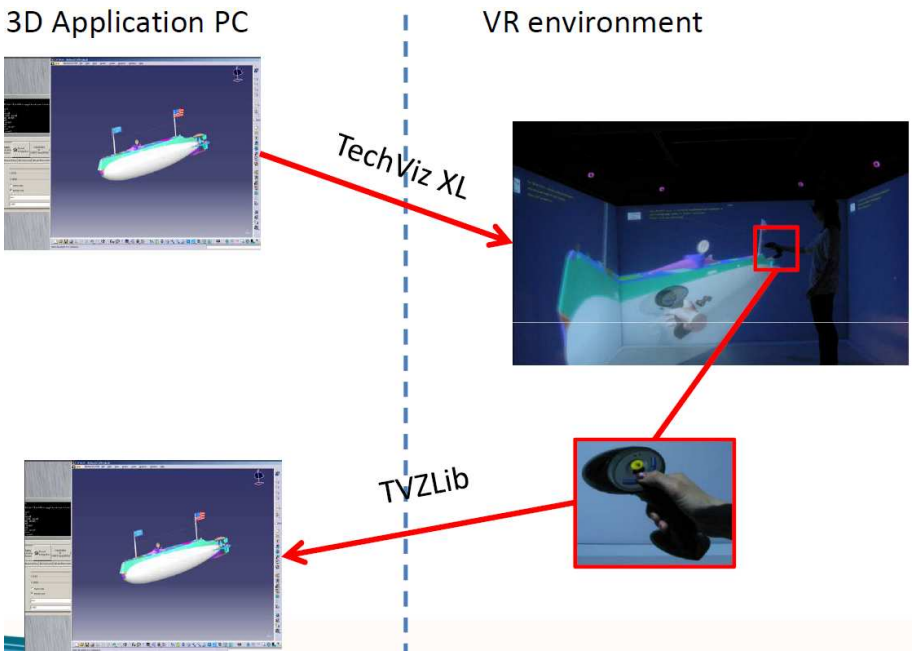

Figura 3.14: Funzionamento di TechViz.

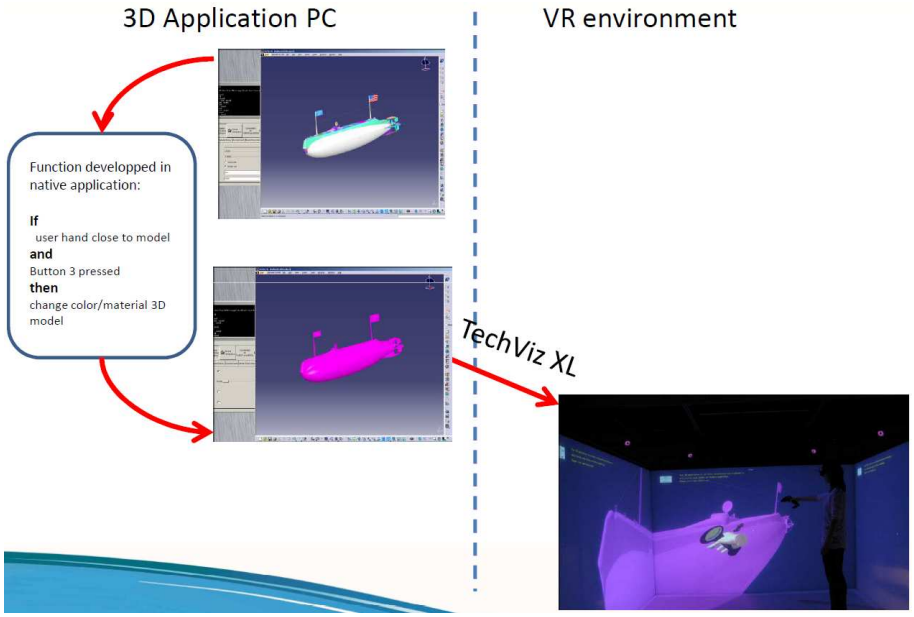

Figura 3.15: TVZLib.

Capitolo 4

Sperimentazioni

Nel precedenta capitolo, si è visto come il software *TechViz* è utile a gestire la visualizzazione sul CAVE e quali sono le sue funzioni principali.
TechViz supporta un'ampia gamma di software per la realizzazione di applicazioni grafiche, purchè tali software siano opportunamente configurati. Il processo di configurazione è abbastanza complesso, ma è richiesto una sola volta per la maggior parte delle applicazioni e questo rende *TechViz* un software con un livello di usabilità molto elevato. Un discorso a parte, invece, è da fare per l'utilizzo di *Unity*. *Unity* è un software che permette lo sviluppo di applicazioni grafiche con la possibilità di implementare interazioni in tempo reale. La possibile presenza di interazioni rende la sua configurazione per *TechViz* molto più complessa rispetto agli altri software. In questo caso, come già fatto notare nel capitolo precedente, diventa indispendabile *TVZLib*, l'API di *TechViz*.

Il cuore di questo capitolo saranno le esperienze avute con *TechViz* durante lo svolgimento del tirocinio, dalle fasi di configurazione dei principali software utilizzati (*Blender* e *Unity*) alla realizzazione di esperienze immersive vere e proprie, insieme alle relative impressioni e ai risultati ottenuti.

Vista la mancanza di un'adeguata strumentazione, una premessa importante da fare è che le immagini di questo capitolo non sempre risultaranno molto nitide e non saranno in grado di offrire al lettore la sensazione tridimensionale del CAVE.

4.1 Installazioni software

Come premesso, prima che *TechViz* possa connettersi all'ampia gamma di software supportati, questi devono essere opportunamente configurati. In primo luogo, bisogna for-

zarli ad utilizzare l'unica libreria grafica supportata da *TechViz*, *OpenGL*. A seconda del software, poi, è necessario considerari alcuni aspetti importanti.

4.1.1 Blender

Blender è un software open source e multipiattaforma di modellazione, rigging, animazione, compositing e rendering di immagini tridimensionali. Dispone inoltre di funzionalità per mappature UV, simulazioni di fluidi, di rivestimenti, di particelle, altre simulazioni non lineari. È disponibile per vari sistemi operativi: Microsoft Windows, macOS, GNU/-Linux, FreeBSD, assieme a porting non ufficiali per BeOS, SkyOS, AmigaOS, MorphOS e Pocket PC. Blender è dotato di un robusto insieme di funzionalità paragonabili, per caratteristiche e complessità, ad altri noti programmi per la modellazione 3D come *3D Studio Max* e *Maya*. Tra le funzionalità di Blender vi è anche l'utilizzo di raytracing e di script in Python.
Per il suo utilizzo su CAVE, sono stati necessari degli step di configurazione del software e di *TechViz*. *TechViz* possiede un file di configurazione che dev'essere opportunamente modificato andando ad aggiungere l'eseguibile del software 3D interessato (in questo caso, Blender). Nella cartella di Blender dev'essere aggiunto, invece, il *.dll* di *OpenGL*, affinchè l'applicazione venga forzata ad avviarsi con la libreria grafica di *OpenGL* e non *DirectX*, dal momento che il sistema operativo utilizzato è *Windows 10*.
Trattandosi di un'operazione complessa, è stata svolta direttamente da un tecnico di *More srl*, pertanto non si approfondirà oltre quest'argomento.

Terminata la fase di configurazione di *TechViz*, vi sono alcuni accorgimenti da tenere in considerazione prima di avviare la simulazione su CAVE. All'avvio, *Blender* si presenta con una determinata interfaccia (Figura 4.1). In quest'interfaccia sono attive di default la visibilità della griglia e degli strumenti di *Blender*. Disattivare la visibilità di griglia e strumenti è un'operazione consigliata, dal momento che potrebbero causare disturbo e interferenze durante la visualizzazione del modello 3D su CAVE.

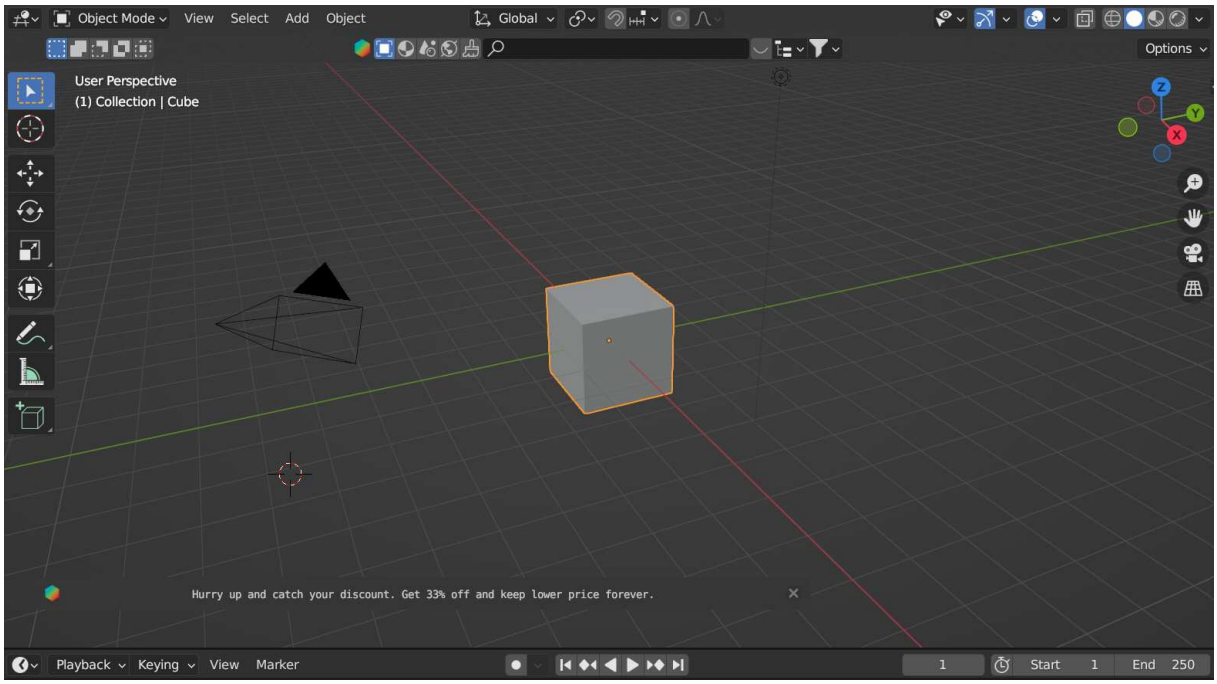

Figura 4.1: Interfaccia di Blender all'avvio dell'applicazione.

Prima dell'avvio di Blender o di qualsiasi altro software di grafica 3D, è importante verificare che il *TechViz Server* ed il *TechViz Controller* siano attivi ed in ascolto del software. In Figura 4.2 si osserva come il *TechViz Controller* si presenta all'avvio.

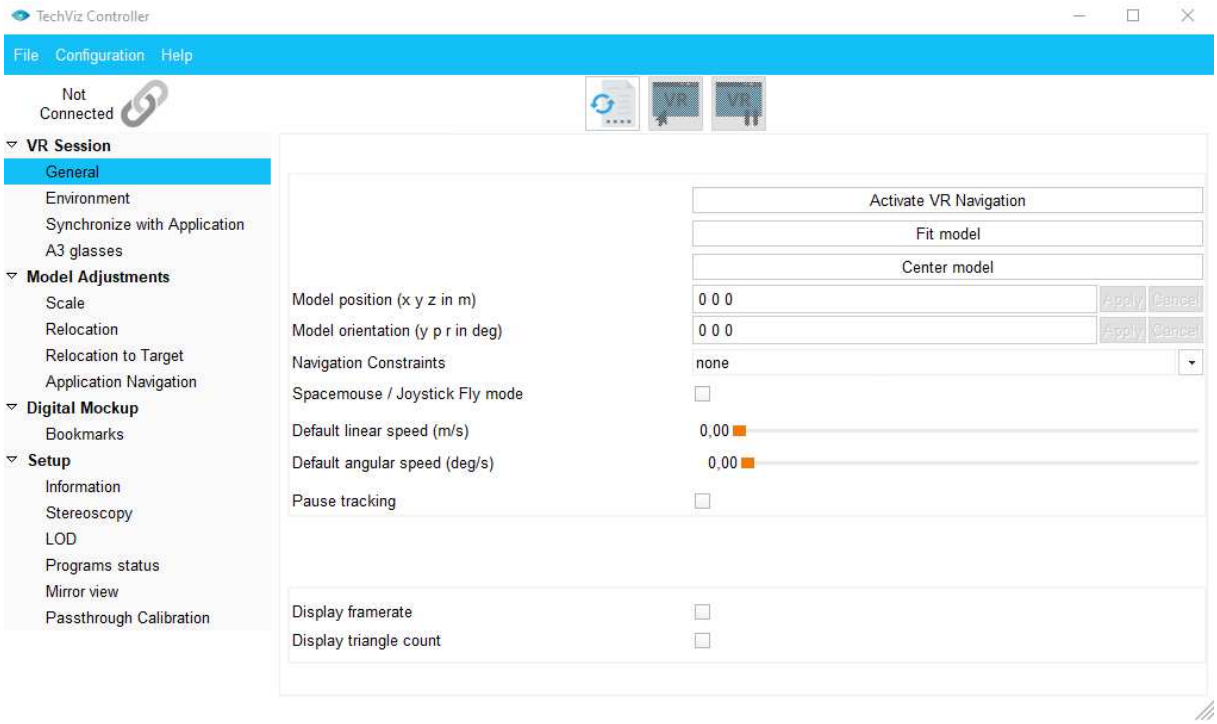

Figura 4.2: TechViz Controller.

La Figura 4.2 è stata catturata in un momento in cui il controller non era connesso con nessuna applicazione ("Not Connected"). Dopo l'avvio di *Blender* il controller passerà in stato di "Connected [Eseguibile App.]".

Il controller presenta diverse funzionalità, tra cui:

- Avvio/interruzione della visualizzazione su CAVE.
- Settaggio di posizione/orientazione del modello
- Settaggio di velocità lineare o angolare per la navigazione all'interno del CAVE, mediante Flystick.
- Visualizzazione delle performance (numero di fps e triangoli del modello)
- Attivazione/disattivazione della visione 3D e del livello di dettaglio del modello renderizzato su CAVE.

A questo punto, per avviare la visualizzazione su CAVE, basterà cliccare sul tasto di avvio, presente sul controller, e successivamente cliccare sulla finestra di *Blender*.
Il primo modello, in Blender, testato su CAVE è stato il modello 3D di una classroom. Il test è stato effettuato in modalità *Solid Mode* ed in modalità *Material Mode*. La modalità *Render Mode* non è ancora supportata da *TechViz* e, infatti, non è possibile avviare la visualizzazione su CAVE quando selezionata.
In Figura 4.4, una foto scattata di un modello in Blender, durante l'utilizzo del CAVE.

Figura 4.3: Modello realizzato in Blender su CAVE.

In termini di performance, *Blender* non soffre di particolari problemi. Su un modello di circa 500.000 poligoni, infatti, si riesce ad ottenere un framerate per second pari a circa 60/70. Le prestazioni sono piuttosto soddisfacenti, considerando che un'applicazione si può ritenere fluida con il raggiungimento di almeno 30 fps. La fluidità diventa un aspetto molto importante specialmente in ambienti VR, poichè un alto frame rate contribuisce ad una riduzione della motion sickness.
Rispetto a *Blender*, *TechViz* si comporta molto bene anche con la simulazione di fluidi, o particellari vari, e con il rendering di modelli nella scala di milioni di poligoni. Ovviamente, con l'aumentare del numero di poligoni, si riducono il numero di fps, ma non sono mai stati osservati cali inferiori a 30 fps, neanche su modelli con più di un milione di poligoni.

4.1.2 Unity

Mentre per *Blender* e altri software di grafica 3D sono necessari pochi accorgimenti per il loro utilizzo con *TechViz*, per *Unity* il discorso è molto più complesso.

Unity è una piattaforma di sviluppo di videogiochi e di creazione di esperienze interattive in tempo reale. È un motore di gioco multi-piattaforma che consente agli sviluppatori di creare giochi, simulazioni, applicazioni e altre esperienze interattive in modo relativamente facile e efficiente. Unity è ampiamente utilizzato nell'industria dei videogiochi, ma trova anche applicazioni in settori come la realtà virtuale (VR), la realtà aumentata (AR), l'architettura, la simulazione e l'addestramento virtuale.
Le caratteristiche principali di Unity includono:

- **Multi-piattaforma**: Consente di sviluppare giochi e applicazioni per una vasta gamma di piattaforme, tra cui PC, console, dispositivi mobili, VR/AR e altro ancora.

- **Grafica avanzata**: Supporta grafica 2D e 3D di alta qualità, con un rendering avanzato, illuminazione globale, effetti speciali e altro.

- **Fisica**: Include un motore fisico che consente di simulare il comportamento realistico degli oggetti in gioco.

- **Animazioni**: Permette di creare animazioni complesse per personaggi e oggetti.

- **Intelligenza artificiale**: Offre strumenti per implementare comportamenti intelligenti per personaggi non giocanti (NPC) e avversari.

- **Unity Asset Store**: Una vasta libreria online di risorse, asset, script e plugin che possono essere utilizzati per accelerare lo sviluppo del gioco.

- **Linguaggio di scripting**: Utilizza principalmente il linguaggio di scripting C#, che è accessibile anche a sviluppatori meno esperti.

Quando si sviluppa un'applicazione in Unity compatibile con *TechViz*, bisogna ricorrere all'utilizzo dell'API di *TechViz*, *TVZLib*. L'API *TVZLib* consente di sviluppare applicazioni OpenGL 3D completamente immersive, in modo rapido e facile, simili ad applicazioni 3D normali controllate da mouse e tastiera. Inoltre è importante la versione utilizzata. Infatti, l'ultima versione momentaneamente supportata da *techViz* è la 2020.3.47. I passaggi necessari per sviluppare questa tipologia di applicazioni sono:

1. Sviluppare la propria applicazione *OpenGL* 3D in *Unity*.
2. Collegare la propria applicazione al modulo API *TVZLib*.
3. Aggiungere le funzioni di tracciamento 6DOF supportate dall'API *TVZLib*, rimappando eventuali input sul Flystick.

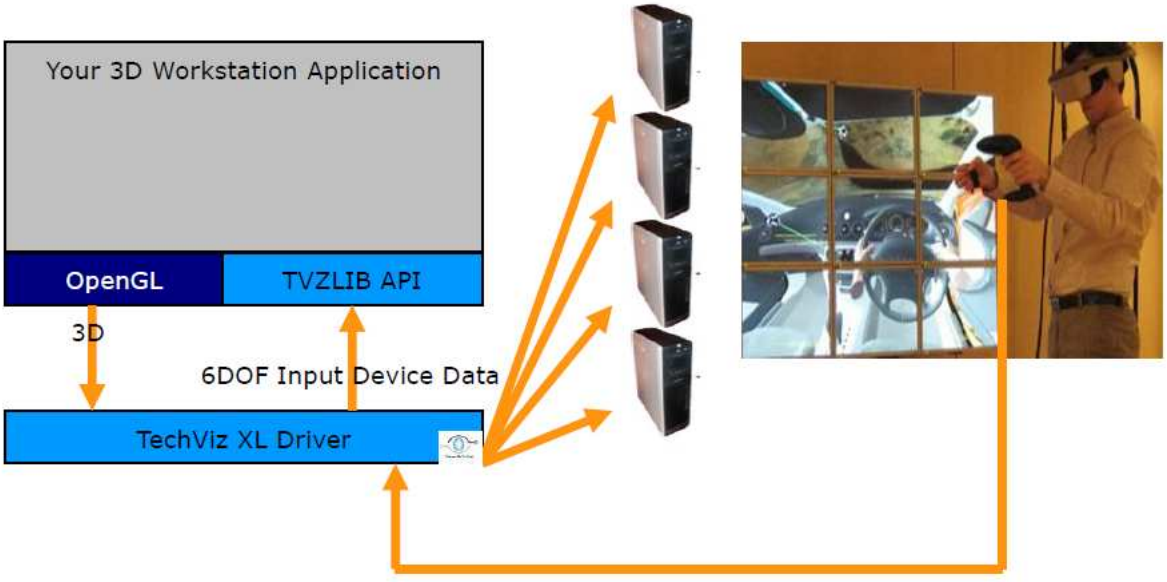

Figura 4.4: Schema di funzionamento di una soluzione globale per Unity

L'API *TVZLib* consente all'utente di ricevere i dati 6DOF nel sistema di coordinate delle applicazioni della workstation. Elimina la necessità che l'applicazione debba tenere conto della navigazione dell'utente, eseguita da TechViz nel sistema immersivo. Se necessario, è anche possibile ottenere le informazioni nel sistema di coordinate dell'ambiente VR e tutte le informazioni di navigazione. In Figura 4.5, si osserva lo schema di funzionamento di un'applicazione in Unity che si interfaccia con *TVZLib*.

Il modello di programmazione dell'API *TVZLIB* non richiede di occuparsi della questione della sincronizzazione. Poiché il programma è scritto come un'applicazione OpenGL sequenziale normale per workstation, non è necessario scrivere un programma parallelo. TechViz XL si occupa autonomamente di distribuire l'esecuzione dell'applicazione nell'ambiente virtuale. Non è richiesta alcuna modifica al proprio programma per gestire la visualizzazione su CAVE (Figura 4.6).

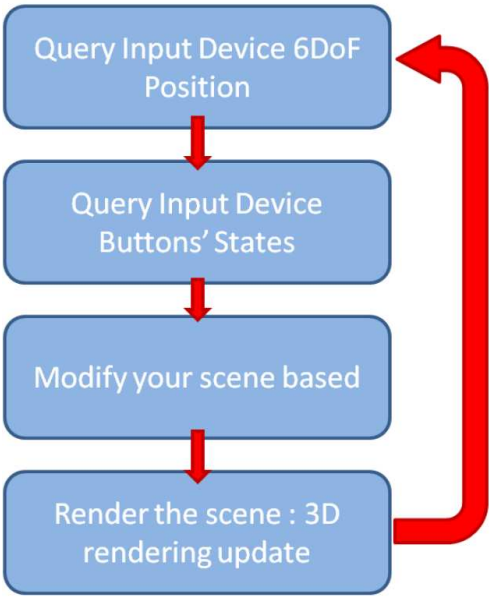

Figura 4.5: Architettura di una soluzione globale.

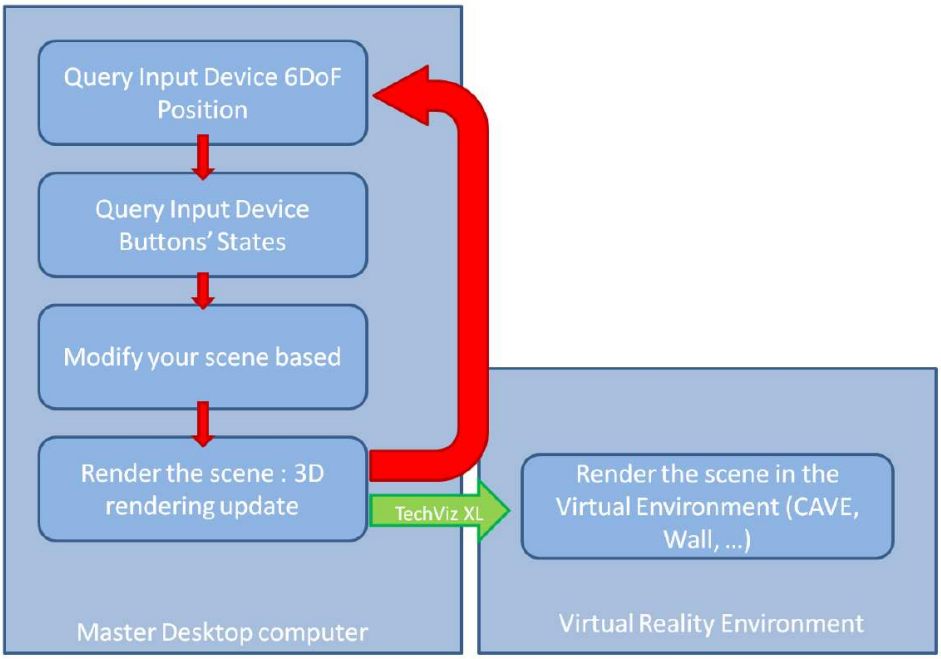

Figura 4.6: Rendering sull'ambiente virtuale.

Non è necessario occuparsi della fotocamera del sistema di tracciamento quando si scrive l'applicazione. Si possono utilizzare la tastiera o il mouse per controllare il punto di vista nella propria applicazione. *TechViz XL* si occuperà del calcolo della fotocamera tracciata e del frustum asimmetrico quando l'immagine viene visualizzata nel sistema immersivo.
La *TVZLib* è stata progettata come una libreria asincrona. Le frequenze di fotogrammi degli elementi del setup, del programma di rendering in 3D o del sistema di tracciamento possono essere diverse. Inoltre, l'applicazione nativa che crea il contenuto 3D deve eseguire il rendering il più velocemente possibile. Infatti, un fattore importante della realtà virtuale è il rendering 3D veloce. Pertanto, per non bloccare il processo di rendering, tutte le chiamate fatte alla *TVZLib*, così come la connessione, sono non bloccanti. Per tener conto di questo fattore, l'applicazione 3D sviluppata deve utilizzare, per la connessione con la TVZLib:

TVZerror TVZSetConnectionHost(const char* host, int port);

Dove "host" è il nome o il numero IP dell'host in cui è in esecuzione il programma TechViz Otrack (di solito il computer principale in cui vengono eseguiti sia l'applicazione nativa 3D, sia Otrack, e quindi "localhost" è il valore di questo parametro), e il numero di porta che è definito su TechViz di default a 1337:

TVZSetConnectionHost ("localhost", 1337);

Se per qualche motivo la connessione dovesse perdersi, allora verrà ristabilita automaticamente il prima possibile. L'API *TVZLib* offre la possibilità di restituire per ogni funzione un valore *TVZerror*. I valori possibili possono essere essenzialmente divisi in 2 gruppi:

- **OK**: restituisce il valore *kNoError*.
- **KO**: restituisce un errore. Se la connessione non viene stabilita, allora si restituisce il valore *kCantConnect* value is sent back.

```
1 #define PLANE_NAME_SHOOTER "TVZLib"
2     void EventsLoop()
3     {
4     TVZerror request_error ;
5     char plane_name[64] ;
6     request_error = TVZGetCurrentPlane(kWand, plane_name);
7     if(request_error != kNoError)
```

```
8         {
9         // Error while getting the current plane name,
10        //while I needed this value to continue in this function
    EventsLoop
11        return ;
12        }
13     else
14        {
15        // kNoError was sent back, I can continue and work with the
    updated data, here the plane name
16        if(strcmp(plane_name, PLANE_NAME_SHOOTER))
17            {
18            //Do what you want in that part since the user has selected
    the plane named TVZLib
19            }
20        else
21            {
22            // Not the good plane name: do nothing
23            }
24        }
25     }
```

Un primo esempio di sperimentazione in *Unity* è la modifica di un'applicazione realizzata come demo per *TechViz*. La demo in questione è un'applicazione nata con lo scopo di navigare all'interno di un appartamento e realizzata per l'utilizzo da mouse e tastiera. A partire da quest'applicazione, si illustreranno le modifiche apportate per il suo utilizzo su CAVE.

Figura 4.7: Applicazione realizzata in Unity.

All'interno dell'appartamento (Figura 4.8) sono possibili due azioni principali:

- Trascinare gli oggetti, utilizzando il mouse.

- Spegnere le luci con un semplice click.

L'obiettivo finale, quindi, è quello di passare dalle periferiche mouse/tastiera alla periferica Flystick per effettuare le stesse identiche azioni all'interno del CAVE.

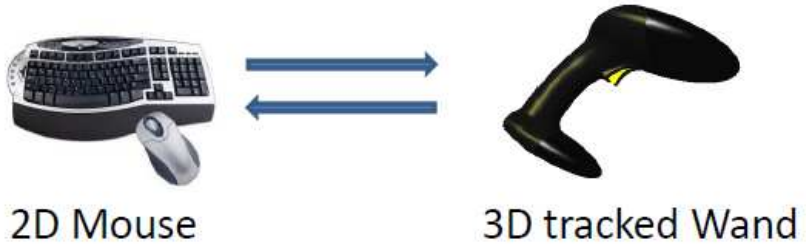

Figura 4.8: Obiettivo finale.

Il primo passo però, è far sì che il modello venga renderizzato nella sua interezza all'interno del CAVE, senza che vi siano parti tagliate. *TechViz* renderizza sul CAVE tutto ciò che viene inquadrato dalla camera di *Unity*. Questo significa che la camera di *Unity* dev'essere posta in modo tale da inquadrare l'intero modello (in questo caso l'intero appartamento). Si osserva l'operazione descritta in Figura 4.9.

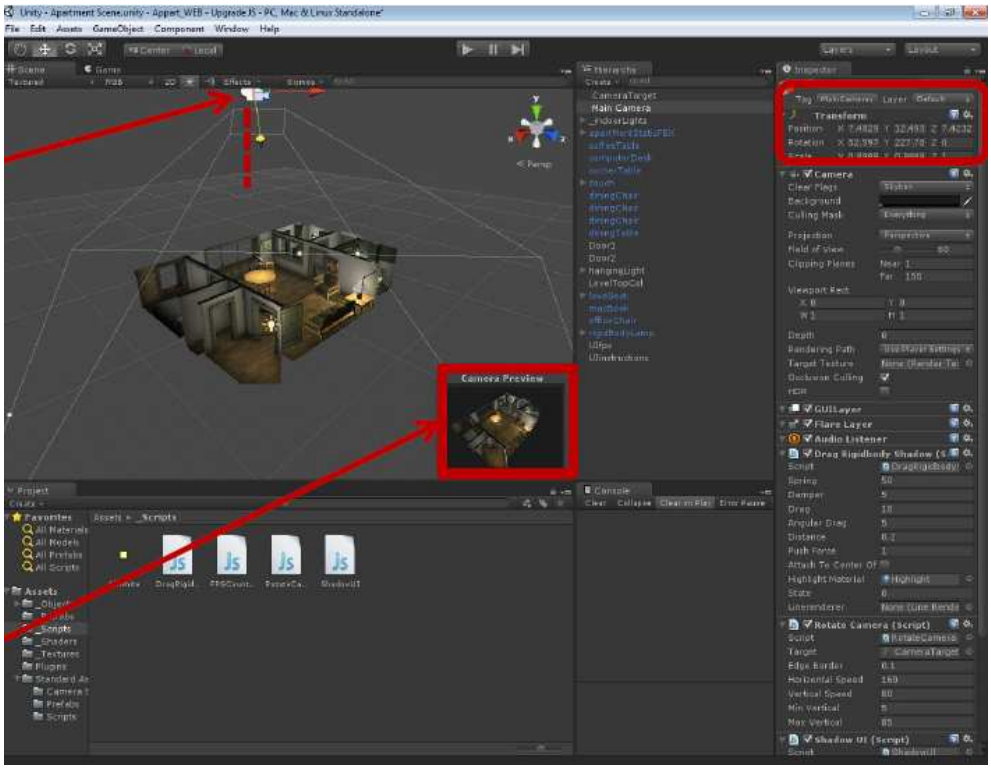

Figura 4.9: Obiettivo finale.

Per il mapping degli input da mouse/tastiera a Flystick, invece, è necessario aggiungere tre file al progetto originale:

- Il file *TVZLib.dll*
- Il file *TVZLib.cs*, principale wrapper tra *TVZLib.dll* ed un programma realizzato in C#.
- Il file *TVZUnity.cs*, una classe che fornisce interfacce di alto livello per l'utilizzo di funzioni di basso livello in *TVZLib.dll*, scritte in C.

Nelle Tabelle 4.1 e 4.2 vi sono alcuni esempi di funzioni.

Tabella 4.1: `Alcune funzioni di TVZLib.cs`

Funzioni	Descrizione
TVZSetConnectionHost	To set the parameters for establishing a connection with the TVZLib
GetDevicePosInUnitySpace	To get the 6DoF position of the interactive device regarding the Unity coordinate system
TVZGetMappedButtonState	To get the status (On 1/Of 0) of one of the button of the interactive device, through its logical name
TVZGetCurrentPlane	To get the current active plane of TechViz*

Tabella 4.2: `Alcune interfacce di TVZUnity.cs`

Funzioni	Descrizione
TVZUnityConnect()	TVZSetConnectionHost ("localhost", 1337);
GetWand0Pos()	TVZGetDevicePos (TVZTypeDevice.kWand, 0, ...)
GetHead0Pos()	TVZGetDevicePos (TVZTypeDevice.kHead, 0, ...)
TVZGetCurrentStringPlane()	TVZGetCurrentPlane (TVZTypeDevice.kWand, plane)

Come esempio di utilizzo delle funzioni sopra elencate, Si mostra di seguito lo script per il trascinamento degli oggetti.

```
1 using UnityEngine;
2 using System;
3
4 public class Interactions : MonoBehaviour {
5
6     public Transform wand6dof;
7
8     void Start() {
9         // Connessione con TechViz
10        Application.runInBackground = true;
11          TVZUnity.TVZUnityConnect();
12    }
13
```

```
14     void Update() {
15           // Questo metodo imposta la variabile checkError a false e va
     richiamata all'inizio della funzione Update solo se utilizziamo
     TVZUnity.cs
16           TVZUnity.ResetCheckError();
17
18           // Ad ogni frame si reimposta la posizione del cubo
     aggiornandola con quella del device
19           TVZTrans3D myTransform = TVZUnity.TVZGetDeviceInUnitySpace(
     TVZTypeDevice.kWand, 0);
20          wand6dof.position = new Vector3(myTransform.pos.x, myTransform.
     pos.y, myTransform.pos.z);
21          wand6dof.rotation = new Quaternion(myTransform.orient.x,
     myTransform.orient.y, myTransform.orient.z, myTransform.orient.w);
22
23          if (wand6dof.GetComponent<FixedJoint>()!=null && wand6dof.
     GetComponent<FixedJoint>().connectedBody != null && (
     GetActionButtonState()==0))
24          {
25               // We unlink the object and we do not highlight anymore.
26               wand6dof.GetComponent<FixedJoint>().connectedBody = null;
27          }
28     }
29
30     private float GetActionButtonState() {
31           float result = -1;
32
33           // On Flystick 2 check for "action1" button
34           result = TVZUnity.TVZReturnButtonState("action1");
35           if (TVZUnity.LastError == 0) return result;
36
37           // On Flystick 3 check for "defaultaction" button
38           result = TVZUnity.TVZReturnButtonState("defaultaction");
39           if (TVZUnity.LastError == 0) return result;
40
41           // Should not happen !
42           result = -1;
43           return result;
44     }
45
46     // "collider" rappresenta l'oggetto con cui il cubo che segue la
     posizione del device sta collidendo
47     void OnTriggerStay(Collider other){
48          // Se l'oggetto rappresenta un rigidbody
49          if (other.attachedRigidbody != null){
```

```
50              // Se siamo nel piano "TVZLib" ed il tasto "action1" viene
    premuto
51              if (TVZUnity.TVZGetCurrentStringPlane().Equals("Tvzlib",
    StringComparison.CurrentCultureIgnoreCase) && GetActionButtonState()
    ==1)
52                  {
53                      // Mi connetto al rigidbody dell'oggetto
54                      GetComponent<FixedJoint>().connectedBody = other.
    GetComponent<Rigidbody>();
55                  }
56              }
57      }
58 }
```

L'obiettivo di questo script è quello di aggiornare un oggetto (invisibile all'interno della scena) con la posizione del flystick. Ogni volta che l'oggetto collide con un altro oggetto della scena e si preme un tasto specifico, allora i corpi si connettono, dando la sensazione di presa. Da notare che la funzione è attiva solo all'interno del piano *"TVZLib"* di *TechViz*.

Nella demo non erano previste collisioni. In normali applicazioni, realizzate con *Unity*, in genere le collisioni vengono gestite attraverso la camera, aggiungendo un componente *collider* (componente che serve a definirne una forma ed una geometria della zona occupata nello spazio da un particolare gameobject) a quest'ultima. Il problema di questa soluzione è che risulta incompleta, inafatti, affinchè abbia effetto all'interno del CAVE, serve un'accortezza in più. Quando ci si muove all'interno del CAVE, la camera utilizzata non è quella di *Unity*, che rimane fissa ad inquadrare l'intera scena da renderizzare, bensì quella di *TechViz*. Dal momento che non è possibile aggiungere direttamente un componente *collider* alla camera di *TechViz*, si è deciso di optare per un'altra soluzione.
La funzione **GetDevicePosInUnitySpace()**, oltre alla posizione del Flystick, consente di restituire in output anche la posizione della testa dell'utente, data dai marker montati sugli occhiali. Si è pensato, dunque, di aggiornare la posizione della camera di *Unity* con quella della testa dell'utente. Alla camera di *Unity* è stato aggiunto un componente *collider* per permettere le collisioni. Di seguito, l'implementazione in linguaggio C#.

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class FollowTechViz : MonoBehaviour
6 {
7     public Transform unityCamera;
8     private TVZTrans3D _techvizCamera;
```

```
9
10      // Start is called before the first frame update
11      void Start()
12      {
13          // Connessione con TechViz
14          Application.runInBackground = true;
15          TVZUnity.TVZUnityConnect();
16      }
17
18      // Update is called once per frame
19      void Update()
20      {
21          // Questo metodo imposta la variabile checkError a false e va
    richiamata all'inizio della funzione Update solo se utilizziamo
    TVZUnity.cs
22          TVZUnity.ResetCheckError();
23
24          // Ad ogni frame si reimposta la posizione della camera di Unity
     aggiornandola con quella di TechViz
25          TVZTrans3D myTransform = TVZUnity.TVZGetDeviceInUnitySpace(
    TVZTypeDevice.kHead, 0);
26          unityCamera.position = new Vector3(myTransform.pos.x,
    myTransform.pos.y, myTransform.pos.z);
27          unityCamera.rotation = new Quaternion(myTransform.orient.x,
    myTransform.orient.y, myTransform.orient.z, myTransform.orient.w);
28      }
29  }
```

Terminati il mapping e la scrittura degli script, prima di procedere con la build dell'applicazione, sono indispensabili altri due script che vanno aggiunti come componenti della camera, **ForceGenericShaders.cs** e **DisableFrustumCulling.cs**.

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class ForceGenericShaders : MonoBehaviour
6 {
7     void Start()
8     {
9           Matrix4x4 mat = GetComponent<Camera>().projectionMatrix;
10          GetComponent<Camera>().projectionMatrix = mat;
11    }
12 }
```

Se l'applicazione utilizza sempre il frustum simmetrico predefinito per la camera, il motore ottimizza gli shader utilizzati. Questo comportamento è problematico per *TechViz*, quindi si sta forzando il motore a utilizzare shader generici. Utilizzare una copia della matrice di proiezione della camera iniziale è sufficiente e non cambia nulla nell'applicazione.

```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4 public class DisableFrustrumCulling : MonoBehaviour
5 {
6     private Camera _camera;
7     public float fov = 90;
8     public float aspect = 1.778f;
9     public float zNear = -999;
10    public float zFar = 999;
11
12    void Start()
13    {
14            _camera = this.GetComponent<Camera>();
15    }
16    void OnPreCull()
17    {
18            _camera.cullingMatrix = Matrix4x4.Perspective(fov, aspect,
      zNear, zFar) * Matrix4x4.Translate(Vector3.forward * -99999 / 2f) *
      _camera.worldToCameraMatrix;
19    }
20    void OnDisable()
21    {
22            _camera.ResetCullingMatrix();
23    }
24 }
```

Qui si sta disabilitando il culling del frustum della camera in *Unity*. Il culling del frustum è una tecnica utilizzata per determinare quali oggetti sono attualmente visibili nella vista della camera e quali possono essere omessi dal rendering per migliorare le prestazioni. Tuttavia, per la visualizzazione su CAVE, si desidera disabilitare questa funzionalità.

A questo punto mancano solo le impostazioni di build. Per la build dell'applicazione è assolutamente necessario disattivare l'utilizzo della libreria *DirectX* e attivare l'utilizzo di *OpenGL*, libreria supportata da *TechViz*. Effettuata la build, vanno copiati due file all'interno della cartella della nostra applicazione: **OpenGL.dll** e **SceneLoaderUnity.cmd**. La prima è la libreria che fornisce le implementazioni delle funzioni di basso

livello di *OpenGL*, chiamate dalle applicazioni grafiche. Il secondo file è un comando che forza l'avvio dell'applicazione con le funzioni della libreria *OpenGL*.

```
1 @echo off
2 set TVZ_APPLICATION_ALIAS=%~n1.exe:unity_2019
3 start /b "TechViz" "%~1" -force-opengl
```

Ora è possibile avviare la visualizzazione su CAVE trascinando l'eseguibile dell'applicazione sullo *SceneLoaderUnity.cmd* e seguendo gli stessi passi visti con *Blender* sul *TechViz Controller*. In Figura 4.10, si osserva la visualizzazione dell'appartamento sul sistema CAVE.

Figura 4.10: visualizzazione dell'appartamento su CAVE.

4.2 VisGraph 3D

L'esperimento più importante è stato sicuramente quello riguardante l'applicazione *VisGraph*, in collaborazione con l'*Università della Basilicata*. Come già spiegato, l'applicazione ha come obiettivo quello di mettere a confronto più sistemi, con lo scopo di misurare il livello di usabilità di ognuno in riferimento alla navigazione di grafi tridimensionali.
L'esperimento in questione consiste in un porting della loro applicazione sul nostro dispositivo CAVE. Da parte dell'*Università della Basilicata* c'è interesse ad avere un dispositivo in più da comparare con gli altri dispositivi interessati. Per noi, *Università Politecnica delle Marche* si è rivelata un'occasione importante per valutare l'efficacia di *TechViz*, in riferimento alla sua facilità di effettuare porting su CAVE.

VisGraph è un'applicazione realizzata in *Unity* e utilizzabile su più periferiche: mouse e tastiera, HTC Vive, leap motion.

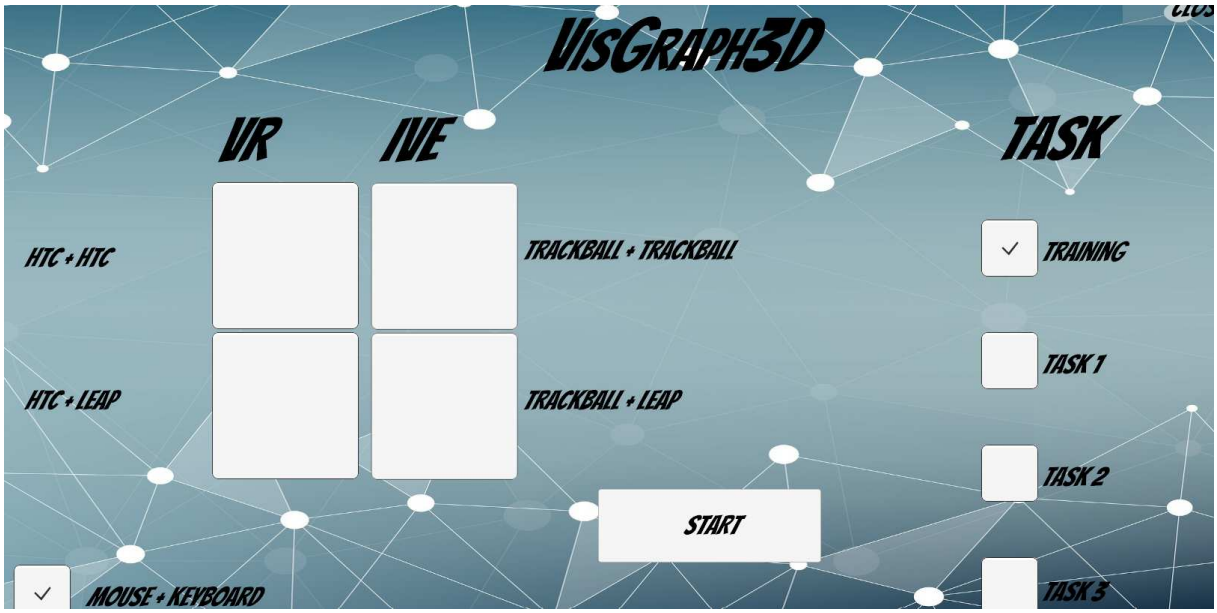

Figura 4.11: Menù iniziale dell'applicazione VisGraph3D.

Dunque, lo scopo dell'esperimento è stato quello di aggiungere il CAVE alla lista delle periferiche supportate. Per farlo, è stata creata una nuova scena, dedicata esclusivamente ai componenti per CAVE. Lo sviluppo è stato diviso in due passi: in un primo passo ci si è occupati dell'implementazione della navigazione all'interno del CAVE, in un secondo passo ci si è occupati del mapping degli input sul Flystick.

1. Per quanto riguarda la navigazione, all'interno della scena CAVE, sono stati inseriti gli script *ForceGenericShaders.cs* e *DisableFrustumCulling.cs*, come componenti della camera.

   Una prima difficoltà è giunta con il posizionamento della camera. La camera dovrebbe essere posizionata in modo da avere l'intero modello all'interno del suo campo di vista, ma questa soluzione è stata pressochè impossibile da attuare. Infatti, data l'immensità dei grafi, non si è riusciti a far rientrare gli interi modelli all'interno del campo di vista. Motivo per cui, come componente della camera, si è inserito uno script che ha il compito di aggiornare la posizione della camera con la testa di *TechViz* (lo script è lo stesso utilizzato nell'applicazione dell'appartamento per implementare le collisioni). In questo modo si è sicuri che la camera di *Unity* e, dopo un tempo impercettibile, anche la camera di *TechViz* renderizzino in tempo

reale l'intero modello.

Una seconda difficoltà sorta è stata quella relativa alle prestazioni. Essendo i modelli molto grandi, l'applicazione originale utilizza una tecnica detta *Level Of Detail* (LOD). Tale tecnica permette di impostare un certo dettaglio dei modelli, in base alla distanza dalla camera. La LOD originale prevede tre livelli in totale:

- **Livello 1**: Modello di n poligoni che corrisponde al massimo livello di dettaglio.
- **Livello 2**: Modello di n poligoni che corrisponde ad un livello medio di dettaglio.
- **Livello 2**: Modello di n poligoni che corrisponde livello basso di dettaglio.
- **Culling**: Rappresenta una distanza oltre la quale gli oggetti non sono più visibili alla camera.

Questa LOD si applica ai soli nodi del grafo. Ciò significa che gli archi saranno sempre visibili durante il funzionamento dell'applicazione. Si osservi, in Figura 4.12, com'era stata impostata la LOD sull'applicazione originale.

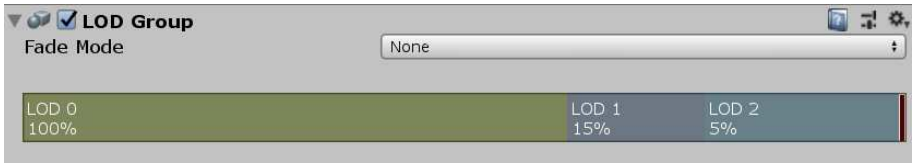

Figura 4.12: LOD originale.

Con una LOD così impostata, non si ottengono buone prestazioni su CAVE. Al fine di ottenere prestazioni soddisfacenti, la LOD è stata modificata, aumentando il secondo livello e aggiungendo un ulteriore livello di *culling*. Si osservi la nuova LOD in Figura 4.13.

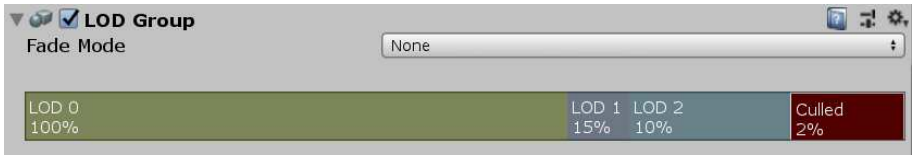

Figura 4.13: LOD impostata per la visualizzazione su CAVE.

2. Nella seconda fase ci si è occupati del mapping dei comandi su Flystick. All'interno dell'applicazione originale, sono presenti cinque principali azioni:

- **Selezione dei nodi**: I nodi del grafo possono essere selezionati alla pressione di un tasto. La selezione comporta un cambio di colore del nodo e l'aggiunta del nodo ad una lista, visibile nella sezione dettagli del menù. I nodi selezionati possono anche essere spostati all'interno della scena.

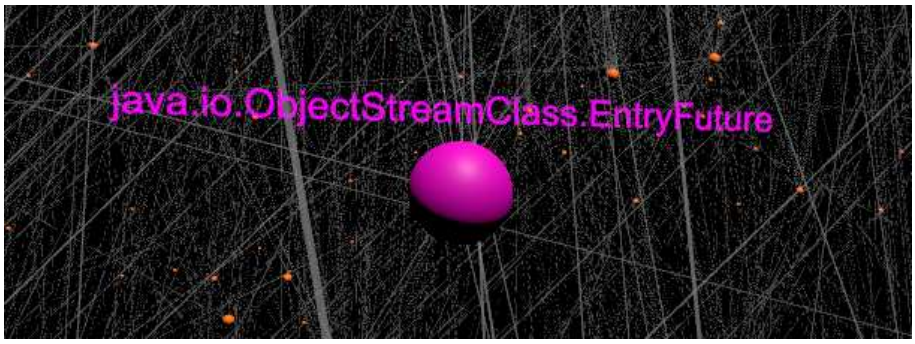

Figura 4.14: Esempio di nodo selezionato.

- **Deselezione dei nodi**: I nodi selezionati possono essere deselezionati, uno alla volta o tutti contemporaneamente, alla pressione di un tasto. La deselezione comporta che il nodo venga tolto dalla lista dettagli del menù.
- **Clustering**: I nodi selezionati possono essere clusterizzati (uniti) in un unico nodo rosso. Questo nodo rosso avrà un grado pari alla somma dei gradi dei nodi selezionati per il clustering.

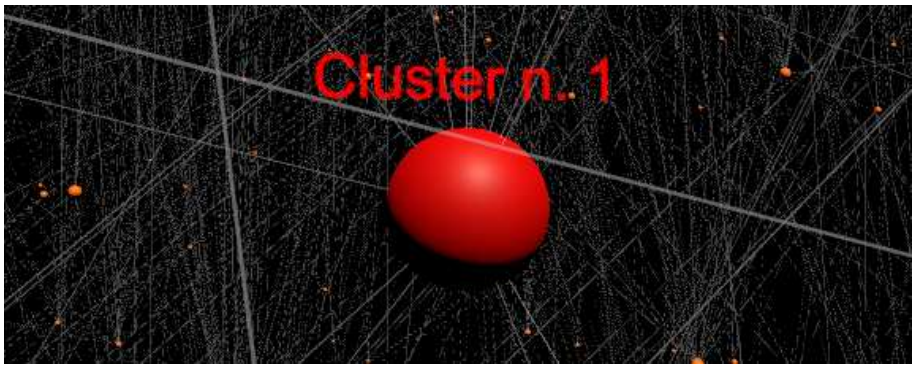

Figura 4.15: Esempio di cluster di due nodi.

- **Unclustering**: Quest'azione comporta la decomposizione di un nodo rosso, restituendo i nodi originali.
- **Apertura del menù**: Si può accedere al menù per visualizzare i dettagli dei nodi selezionati ed effettuare altre azioni, come ad esempio tornare al menù principale.

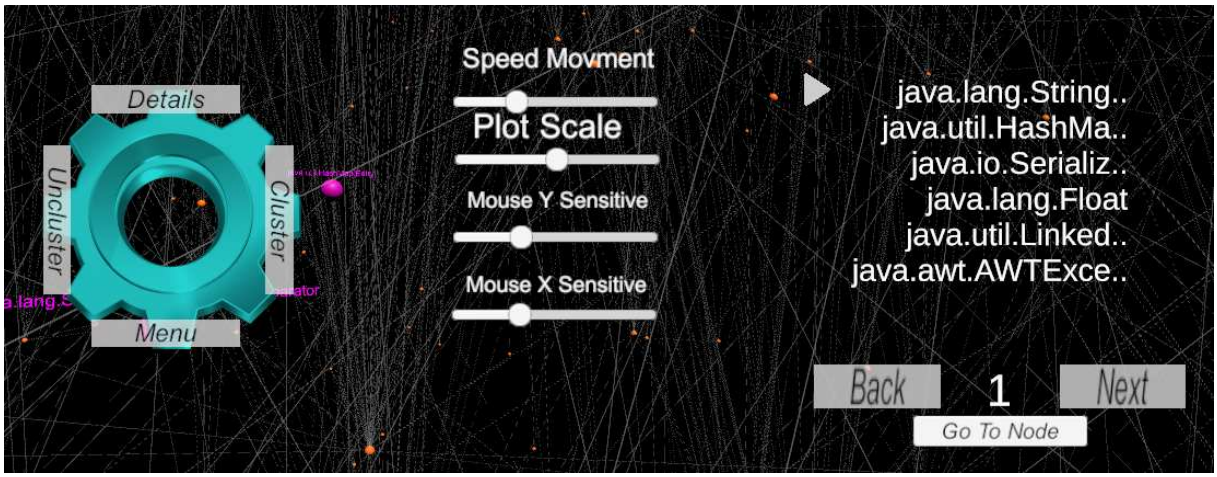

Figura 4.16: Menù.

Riguardo l'utilizzo dell'applicazione su CAVE, per risparmiare in termini di tempo di sviluppo, si è deciso di non implementare il menù, dal momento che non è risultato fondamentale. All'interno del progetto era già presente uno script che implementa le azioni sopra descritte. Si è dunque partiti da questo script, come base, modificandolo in maniera adeguata. Di seguito, si riportano solo le modifiche.

```
1 using System.Collections.Generic;
2 using UnityEngine;
3 using System;
4 using UnityEngine.SceneManagement;
5
6 public class SelectionCave : MonoBehaviour {
7     public GameObject newParent;
8     public GameObject newParentCluster;
9
10    public RaycastHit hit;
11    private GameObject TemporalParent;
12    private GameObject pointHolder;
13    private GameObject hitObj = null;
14    private Camera cam;
15    private bool MoveOn = false;
16    private bool move = false;
17    private Color colorTemp;
18
19    [Range(1.0f, 200.0f)]
20    public float longShot;
21
22    private Vector3 startPosition;
23
```

```
24      public Transform wand6dof;
25
26      private LineRenderer laserLine;
27      private SpriteRenderer crossFire;
28      public Vector3 rayOrigin;
29
30      private Dictionary<string, Color> ColorNode = new Dictionary<
    string, Color>();
31
32      private DataPlotter dataplotter;
33      private float plotSCale;
34
35      private GameObject cluster;
36      private Dictionary<string, GameObject> actualCluster = new
    Dictionary<string, GameObject>();
37      private Dictionary<string, Cluster> multyCluster = new
    Dictionary<string, Cluster>();
38      private Cluster cl;
39      private GameObject clusterDestroy;
40      private bool multiCluster = false;
41
42      private Vector3 temporalHit;
43
44      // Use this for initialization
45      void Start () {
46
47          // Connessione con TechViz
48          Application.runInBackground = true;
49          TVZUnity.TVZUnityConnect();
50
51          crossFire = transform.GetComponentInChildren<SpriteRenderer
    >();
52          crossFire.enabled = true;
53
54          laserLine = GetComponent<LineRenderer>();
55          laserLine.startColor = Color.cyan;
56          laserLine.endColor = Color.cyan;
57
58          dataplotter = Archive.DataPlotter();
59          plotSCale = dataplotter.plotScale;
60          TemporalParent = Instantiate(newParent);
61          pointHolder = GameObject.FindGameObjectWithTag("
    PointContainer");
62          TemporalParent.transform.parent = pointHolder.transform.
    parent;
```

```
63          cam = GameObject.FindGameObjectWithTag("MainCamera").
    GetComponent<Camera>();
64      }
65
66      // Update is called once per frame
67      void Update()
68      {
69          // Questo metodo imposta la variabile checkError a false e
    va richiamata all'inizio della funzione Update solo se
    utilizziamo TVZUnity.cs
70          TVZUnity.ResetCheckError();
71
72          // Ad ogni frame si reimposta la posizione del cubo
    aggiornandola con quella del device
73          TVZTrans3D myTransform = TVZUnity.TVZGetDeviceInUnitySpace(
    TVZTypeDevice.kWand, 0);
74          wand6dof.position = new Vector3(myTransform.pos.x,
    myTransform.pos.y, myTransform.pos.z);
75          wand6dof.rotation = new Quaternion(myTransform.orient.x,
    myTransform.orient.y, myTransform.orient.z, myTransform.orient.w
    );
76
77          rayOrigin = cam.ViewportToWorldPoint(new Vector3(0.5f, 0.5f
    , 0));
78          crossFire.transform.position = rayOrigin + (wand6dof.
    forward * (longShot/2));
79          laserLine.SetPosition(0, wand6dof.position);
80          laserLine.SetPosition(1, crossFire.transform.position);
81
82          if (TVZUnity.TVZGetCurrentStringPlane().Equals("Tvzlib",
    StringComparison.CurrentCultureIgnoreCase) &&
    GetActionButtonState1()==1)
83          {
84              if (Physics.Raycast(rayOrigin, wand6dof.forward, out
    hit) && Archive.SelectedNode().Contains(hit.collider.gameObject)
    )
85              {
86                  laserLine.SetPosition(1, hit.point);
87                  laserLine.startColor = Color.yellow;
88                  laserLine.endColor = Color.yellow;
89
90                  foreach (GameObject o in Archive.SelectedNode())
91                  {
92                      o.transform.SetParent(pointHolder.transform);
93                  }
```

```
94                  TemporalParent.transform.position = hit.point;
95                  foreach (GameObject o in Archive.SelectedNode())
96                  {
97                      o.transform.SetParent(TemporalParent.transform)
    ;
98                  }
99                  MoveOn = true;
100                 temporalHit = hit.point;
101             }
102             else
103             {
104                 laserLine.SetPosition(1, crossFire.transform.
    forward);
105                 laserLine.startColor = Color.cyan;
106                 laserLine.endColor = Color.cyan;
107             }
108             laserLine.enabled = true;
109         }
110         if (TVZUnity.TVZGetCurrentStringPlane().Equals("Tvzlib",
    StringComparison.CurrentCultureIgnoreCase) &&
    GetActionButtonState1()==1)
111         {
112             if( MoveOn && !move)
113             {
114                 TemporalParent.transform.position = Vector3.
    MoveTowards(TemporalParent.transform.position,wand6dof.position
    + (wand6dof.forward * Vector3.Distance(temporalHit,wand6dof.
    position)), 500.0f);
115                 laserLine.SetPosition(1, crossFire.transform.
    position);
116                 dataplotter.moveArch();
117             } else
118             {
119                 Selection();
120             }
121
122         }
123
124         if ((TVZUnity.TVZGetCurrentStringPlane().Equals("Tvzlib",
    StringComparison.CurrentCultureIgnoreCase) &&
    GetActionButtonState1() == 0))
125         {
126             if (MoveOn || move)
127             {
128                 move = false;
```

```
129                 MoveOn = false;
130                 dataplotter.RepositionArch(Archive.SelectedNode());
131             }
132             laserLine.enabled = false;
133             /*
134             dataplotter.RepositionArch(Archive.SelectedNode());
135             MoveOn = false;
136             */
137         }
138 
139         //Action for deselection
140         if ((TVZUnity.TVZGetCurrentStringPlane().Equals("Tvzlib",
    StringComparison.CurrentCultureIgnoreCase) &&
    GetActionButtonState2() == 1))
141         {
142             Deselection();
143         }
144 
145         ...
146 
147         if (Input.GetMouseButtonDown(0))
148         {
149             Clustered();
150         }
151 
152         if (Input.GetMouseButtonUp(0))
153         {
154             dataplotter.callCluster();
155         }
156 
157         if (Input.GetMouseButtonDown(1))
158         {
159             Unclustered();
160         }
161 
162         if (Input.GetMouseButtonUp(1))
163         {
164             dataplotter.callUncluster();
165         }
166 
167         if (Input.GetMouseButtonDown(2))
168         {
169             Archive.ClearAll();
170             SceneManager.LoadScene("SceneMenu");
171         }
```

```
172     }
173
174     ...
175
176     private float GetActionButtonState1()
177     {
178         float result = -1;
179
180         // On Flystick 2 check for "action1" button
181         result = TVZUnity.TVZReturnButtonState("action1");
182         if (TVZUnity.LastError == 0) return result;
183
184         // On Flystick 3 check for "defaultaction" button
185         result = TVZUnity.TVZReturnButtonState("defaultaction");
186         if (TVZUnity.LastError == 0) return result;
187
188         // Should not happen !
189         result = -1;
190         return result;
191     }
192
193     private float GetActionButtonState2()
194     {
195         float result = -1;
196
197         // On Flystick 2 check for "action1" button
198         result = TVZUnity.TVZReturnButtonState("action2");
199         if (TVZUnity.LastError == 0) return result;
200
201         // On Flystick 3 check for "defaultaction" button
202         result = TVZUnity.TVZReturnButtonState("defaultaction");
203         if (TVZUnity.LastError == 0) return result;
204
205         // Should not happen !
206         result = -1;
207         return result;
208     }
209     ...
210 }
```

In sostanza, sono state aggiunte le funzioni di lettura degli input dalla periferica Flystick, **GetActionButtonState1()** e **GetActionButtonState2()**, e si sono modificate le condizioni delle strutture *IF/ELSE*, in modo da poter individuare la pressione dei tasti del Flystick. Si è poi implementato l'utilizzo di un raggio, come

puntatore del Flystick, che cambia colore a seconda dell'azione effettuata. Si noti come lo script utilizza funzioni del wrapper *TVZLib.cs* e della classe *TVZUnity.cs* che sono stati aggiunti in una cartella apposita del progetto, insieme al file *TVZLib.dll*

Inizialmente si è riscontrato un problema sull'azione di trascinamento dei nodi selezionati. Questi, infatti, venivano teletrasportati in un punto della scena talmente lontano da non essere più renderizzati dalla camera di *Unity* e, di conseguenza, non erano più visibili all'interno del CAVE. La soluzione è stata quella di ancorare i nodi selezionati all'estremo del raggio puntatore del Flystick, calcolato nel momento della collisione con il nodo da trascinare.

Terminate le due fasi di sviluppo, è stata modificata la scena iniziale per dare la possibilità all'utente di selezionare il CAVE come periferica. Infine, sono stati eseguiti i passi di configurazione dell'applicazione, affinchè la build finale fosse utilizzabile con *TechViz*. In Figura 4.17, l'applicazione renderizzata su CAVE.

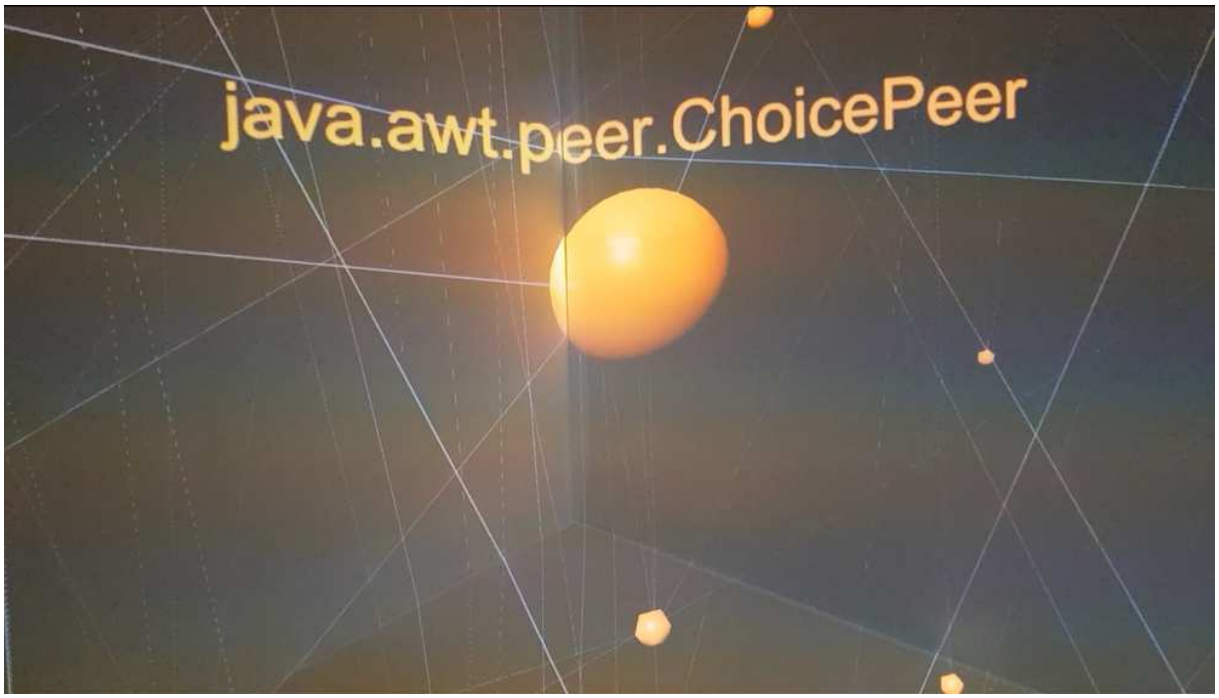

Figura 4.17: applicazione VisGraph 3D su CAVE.

4.3 Altri software

Blender e *Unity* sono stati i software su cui ci si è concentrati di più, poichè sono quelli più utilizzati dalla nostra università. Ulteriori prove, però, sono state effettuate anche

con altri software: *ProductView Express*, *JT2GO*, *3D Studio Max* e *MeshLab*.
Questa breve sezione servirà ad offrire al lettore una panoramica più completa del funzionamento di *TechViz* con altri software grafici.

Fatta eccezione per *Unity*, con tutti gli altri software il processo di installazione è identico a quello di *Blender*. Bisogna quindi forzarli all'utilizzo della libreria *OpenGL*, modificando la cartella di installazione. Inoltre, è necessario aggiungere gli eseguibili all'interno del file di configurazione di *TechViz*. Non è consigliabile modificare il file di configurazione, ma l'aggiunta degli eseguibili è un processo semplice e non incide in alcun modo sul funzionamento delle applicazioni già presenti. Parlando invece dei risultati ottenuti, per quanto riguarda *ProductView Express* e *JT2GO* si raggiungono prestazioni al pari, se non superiori, a quelli di *Blender*. In Tabella 4.3, si osservino i risultati di alcuni test.

Tabella 4.3: `Test`

Software	**N. Poligoni**	**FPS**
JT2GO	16.41 e6	70 circa
	347.71 e3	121 circa
	19.17 e6	41 circa
ProductView Express	5.52 e6	120 circa

In Figura 4.18, un modello 3D di una navicella renderizzata su CAVE, tramite connessione con *ProductView Express*.

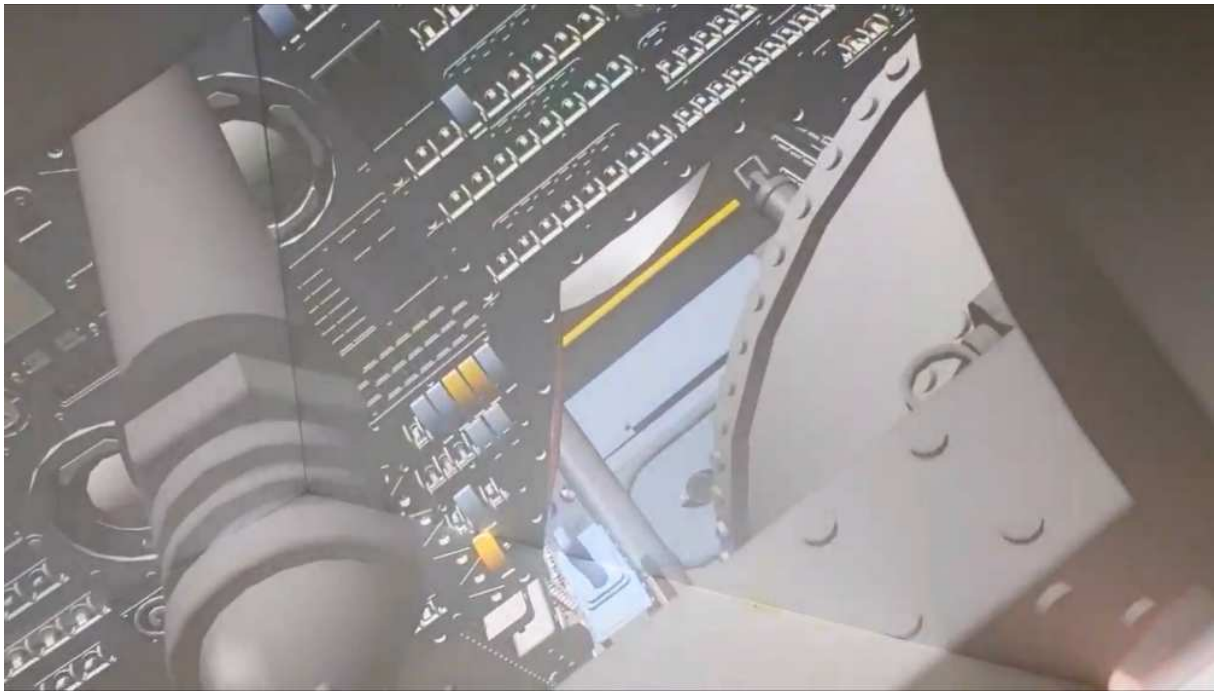

Figura 4.18: Interno di una navicella.

In riferimento ai risultati ottenuti in Tabella 4.3, si può notare come *TechViz* si comporti molto bene con modelli grandi (nell'ordine di milioni di poligoni). Con modelli nell'ordine di migliaia di poligoni, il numero di frame per secondo è notevole. Man mano che si sale, la fluidità delle applicazioni si va perdendo, fino a raggiungere un frame rate di 41 fps con un numero di poligoni pari a 19.17 e6. Con questo, non si vuole intendere che 41 fps non siano indice di fluidità, ma si vogliono semplicemente evidenziare i limiti di *TechViz*.

Con 3D Studio Max e MeshLab, invece, si sono fatti dei tentativi, ma senza ottenere dei risultati soddisfacenti.

- **3D Studio Max**: è stato modificato il file di configurazione di *TechViz*, aggiungendo l'eseguibile dell'applicativo. In seguito, è stato aggiunto il file *OpenGL.dll* all'interno dell'applicativo. All'avvio della visualizzazione di un modello sul CAVE, i proiettori non restituiscono nulla in output e gli schermi appaiono tutti e tre neri.

- **MeshLab**: il processo di configurazione è stato esattamente lo stesso. In questo caso *TechViz* non riesce a connettersi con l'applicativo, quindi non è possibile avviare alcuna visualizzazione sul CAVE.

Il perchè di queste problematiche non è ancora chiaro. Un motivo potrebbe essere che *TechViz* non supporti ancora tali applicazioni o comunque le loro versioni più recenti.

Capitolo 5

Discussioni

Dagli esperimenti del capitolo precedente, si evince come *TechViz* sia un software più che valido per lo sviluppo di applicazioni per CAVE. Grazie ad esso, infatti, è possibile sviluppare un'applicazione da zero, senza preoccuparsi di gestire il rendering sul CAVE, oppure effettuare il porting di applicazioni già realizzate. Esistono, poi, diversi software supportati da *TechViz*, tra cui *Unity* e *Blender*. *TechViz* riesce ad ottenere risultati migliori con software che non prevedono le interazioni. Questo lo si intuisce dal fatto che, a parità di numero di poligoni, la visualizzazione di modelli realizzati con *Blender* (o anche *ProductView Express* e *JT2GO*) sono molto più fluide, rispetto a *Unity*. L'unico grande svantaggio di *TechViz* è che purtroppo appesantisce la memoria della scheda grafica, al punto che l'hardware a disposizione risulterà meno performante in alcuni casi.

A volte è capitato che *TechViz* smettesse di funzionare, causando dei crash della scheda grafica talmente pesanti da comportare il riavvio del computer. La maggior parte dei crash si sono verificati all'avvio della visualizzazione di applicazioni grafiche realizzate in *Unity*. Da precisare, però, che non è ancora chiaro quale sia la causa del problema. Questo perchè nei mesi successivi all'acquisto del CAVE, ci sono stati problemi hardware, a causa dei quali il computer è stato spedito in assistenza per essere riparato. Si suppone quindi che i continui crash della macchina possano essere stati causati da un difetto del computer e non è detto, quindi, che siano stati causati da *TechViz*.

Con *Blender*, *Techviz* potrebbe risultare un po' fastidioso da utilizzare. Appena avviata la visualizzazione, infatti, non è detto che appaia sugli schermi in maniera corretta. Il modello potrebbe risultare bloccato o doppiato. Per sistemare l'errore bisogna ridurre *Blender* in modalità finestra e trovare la sua dimensione corretta, finchè la visualizzazione non si sistemerà automaticamente.

Una cosa che accomuna tutti i software è il disturbo causato dal puntatore del mouse quando lo si sposta sulla finestra del software 3D in questione. Per questo motivo, bisogna stare attenti a non muovere il mouse sulla finestra dell'applicazione, nel mentre che è attiva la visualizzazione sul CAVE. Inoltre, è consigliabile non utilizzare in alcun modo il computer, poichè, quando le applicazioni 3D passano in secondo piano rispetto ad altre applicazioni, le performance della visualizzazione su CAVE calano drasticamente.

I sistemi CAVE possono essere un'ottima alternativa agli Head Mounted Display. Rispetto allo sviluppo di normali applicazioni 3D, quando si progetta un'applicazione per HMD, vi sono alcuni accorgimenti in più e differenze di cui tener conto. Per la progettazione di esperienze immersive su CAVE, qualora si utilizzi *TechViz*, questo non accade ed i tempi di sviluppo non variano in maniera considerevole rispetto a quelli di una normale applicazione 3D.

Il fatto che *TechViz* non sia open source è da ostacolo per lo studio del software. In fase di training all'utilizzo del software, sono state poste domande a cui i tecnici dell'azienda non hanno potuto rispondere. In futuro si potrebbe provare ad approfondire lo studio di *TechViz*, attraverso dei test più mirati sulla valutazione delle prestazioni che si possono raggiungere con il CAVE.

Tra i vari esperimenti, il più importante è stato il porting dell'applicazione *VisGraph 3D*, sviluppata dall'*Università della Basilicata*. L'esperimento ha dimostrato che il porting di applicazioni realizzate in *Unity* è possibile e non richiede tempi di sviluppo considerevoli. In futuro, ci sarà sicuramente la possibilità di portare avanti la collaborazione insieme all'*Università della Basilicata*. Il prossimo passo sarà quello di raccogliere un gruppo di persone, al fine di valutare l'usabilità del CAVE e confrontarla con quella degli altri dispositivi supportati dall'applicazione.

Capitolo 6

Conclusioni e sviluppi futuri

In questa tesi non ci si è soffermati su tutte le funzionalità di *TechViz*. Oltre a ciò che si è visto, esistono funzionalità che permettono, ad esempio, di scattare delle istantanee del modello, di scrivere delle note su alcuni punti del modello o di cambiare modalità di navigazione. C'è, però, una funzionalità interessante, che sicuramente verrà sperimentata in futuro, la simulazione collaborativa. *TechViz* permette, infatti, di stabilire una connesione a distanza con un altro dispositivo CAVE al fine di condividere un'esperienza immersiva.
La simulazione collaborativa apre nuove strade verso la progettazione di prodotti aziendali e nel campo della ricerca, dando la possibilità a più università o più aziende di cooperare più facilmente. Potrebbe essere l'occasione di valutare come *TechViz* si comporta in questi casi.

Si è visto come una delle tematiche più importanti è l'usabilità. Non tutti gli utenti, che hanno sperimentato l'utilizzo del CAVE per la prima volta, sono stati subito in grado di muoversi con facilità. Un possibile sviluppo futuro potrebbe essere quello di trovare una soluzione agevole per questa tipologia di utenti, passando magari il controllo ad una persona più esperta (in applicazioni in cui non sono previste interazioni) o migliorando la mobilità all'interno del CAVE con il supporto di un secondo controller.
Il CAVE riduce gli effetti della motion sickness, ma purtroppo non risolve il problema. Per esperienza personale, l'effetto di disorientamento e perdita dell'equilibrio dipende molto da come si utilizza il CAVE. Infatti, tale effetto aumenta maggiormente quando si è sopra lo schermo posto sul pavimento. Ci sono molti studi scientifici che propongono soluzioni interessanti per HMD, ad esempio tramite l'implementazione di effetti sonori. Un possibile sviluppo futuro potrebbe essere quello di sperimentare queste soluzioni su CAVE.

In futuro, potrebbe essere importante effettuare dei test mirati allo studio delle per-

formance di *TechViz*. Come si è visto, le prestazioni variano di molto al crescere del numero di poligoni di un modello. Potrebbe essere molto più interessante scoprire come ottimizzare un'applicazione, cercando di capire, ad esempio, con quali modelli *TechViz* si comporta meglio sulla base di texture o altre carattesistiche importanti.
Per le applicazione realizzate in *Unity* esiste una variabile, *unity big model*, di default impostata al valore 2 e commentata. Se attivata, consente di ottenere prestazioni di granlunga superiori, a discapito di possibili errori durante la visualizzazione su CAVE. Non è ancora chiaro cosa indichi precisamente questa variabile, ma si potrebbe indagare, chiedendo all'azienda techViz maggiori spiegazioni.

TechViz inoltre dispone di un emulatore per CAVE che potrebbe semplificare ancora di più la fase di sviluppo delle applicazioni. Si potrebbe, in futuro, richiederne la licenza e testarne l'effettiva utilità.

Elenco delle figure

1.1 Cave Automated Virtual Environment. 4
1.2 Sistema di addestramento per aerei F16. 5

2.1 Sensorama: primo visore di reltà virtuale. 7
2.2 Virtual tour, Duomo di Milano. 8
2.3 Design e sviluppo del sistema MobiCave. 14
2.4 Riferimenti visivi. 15
2.5 Schema dell'esperimento. 16
2.6 Risultati dell'esperimento. 16
2.7 Parametri di un CAVE. 18
2.8 Geometrie di un CAVE realizzabili con ICE'S. 18
2.9 Sovrapposizione delle texture. 19
2.10 Rendering finale. 21

3.1 Proiettori. 25
3.2 CAVE. 25
3.3 Telecamera ottica. 26
3.4 Occhiali 3D. 27
3.5 Flystick. 28
3.6 ART CONTROLLER. 30
3.7 DMS Local. 31
3.8 Fly.elis-ng Immersive Display. 31
3.9 Sincronizzazione degli schermi. 32
3.10 DTRACK. 33
3.11 Configurazione dell'ART controller. 34
3.12 Architettura di TechViz. 35
3.13 Componenti di TechViz. 36
3.14 Funzionamento di TechViz. 37
3.15 TVZLib. 37

4.1 Interfaccia di Blender all'avvio dell'applicazione. 40
4.2 TechViz Controller. 40
4.3 Modello realizzato in Blender su CAVE. 41
4.4 Schema di funzionamento di una soluzione globale per Unity 43
4.5 Architettura di una soluzione globale. 44
4.6 Rendering sull'ambiente virtuale. 44
4.7 Applicazione realizzata in Unity. 46
4.8 Obiettivo finale. 47
4.9 Obiettivo finale. 47
4.10 visualizzazione dell'appartamento su CAVE. 53
4.11 Menù iniziale dell'applicazione VisGraph3D. 54
4.12 LOD originale. 55
4.13 LOD impostata per la visualizzazione su CAVE. 55
4.14 Esempio di nodo selezionato. 56
4.15 Esempio di cluster di due nodi. 56
4.16 Menù. 57
4.17 applicazione VisGraph 3D su CAVE. 63
4.18 Interno di una navicella. 64

Elenco delle tabelle

2.1 Geometrie costruibili in ICE'S 18

3.1 Caratteristiche dei proiettori 24
3.2 Caratteristiche delle telecamere 26
3.3 Caratteristiche degli occhiali 3D 27
3.4 Caratteristiche del flystick 28
3.5 Caratteristiche del computer 29
3.6 Caratteristiche del controller 29

4.1 Alcune funzioni di TVZLib.cs 48
4.2 Alcune interfacce di TVZUnity.cs 48
4.3 Test 64

Bibliografia

[1] Carolina Cruz-Neira, Daniel J. Sandin, Thomas A. DeFanti, Robert V. Kenyon e John C. Hart. «The CAVE: Audio Visual Experience Automatic Virtual Environment». In: *Commun. ACM* 35.6 (giu. 1992), pp. 64–72. ISSN: 0001-0782. DOI: 10.1145/129888.129892. URL: https://doi.org/10.1145/129888.129892.

[2] Dipam S. Patel Ronak Dipakkumar Gandhi. «Virtual Reality – Opportunities and Challenges». In: (2018). DOI: 5.01:2714-2724.10.1080/10447318.2018.1429061.

[3] Ahmed Jamah Ahmed Alnagrat, Abdulaziz E Salem Shagluf, Rajaa. F. Mahmoud Sulieman, Abubaker Almintisir Abubaker Akeel, Rizalafande Che Ismail e Syed Zulkarnain Syed Idrus. «Assessing Virtual Reality Sickness in Highly Immersive Virtual Laboratory Environments : Simulator Sickness Questionnaire and Mitigation Strategies». In: (2023), pp. 376–381. DOI: 10.1109/MI-STA57575.2023.10169158.

[4] Gerd Bruder, Ferran Argelaguet, Anne-Héléne Olivier e Anatole Lécuyer. «CAVE Size Matters: Effects of Screen Distance and Parallax on Distance Estimation in Large Immersive Display Setups». In: *Presence* 25.1 (2016), pp. 1–16. DOI: 10.1162/PRES_a_00241.

[5] Bustillo A. Checa D. «A review of immersive virtual reality serious games to enhance learning and training». In: (2019). DOI: 10.1007/s11042-019-08348-9.

[6] Guoxiang Hou Xinxiong Liu Jing Zhang. «Virtual Reality and Its Application in Military». In: (2018). DOI: 10.1088/1755-1315/170/3/032155.

[7] M.E. Portman, A. Natapov e D. Fisher-Gewirtzman. «To go where no man has gone before: Virtual reality in architecture, landscape architecture and environmental planning». In: *Computers, Environment and Urban Systems* 54 (2015), pp. 376–384. ISSN: 0198-9715. DOI: https://doi.org/10.1016/j.compenvurbsys.2015.05.001.

[8] Bratosin Ioan Alexandru, Pavaloiu Ionel Bujorel, Goga Nicolae, Luca Andreea Iuliana e Podina Ioana. «Virtual Reality Application for Acute Pain Therapy - User Experience». In: (2022), pp. 1–6. DOI: 10.1109/ECAI54874.2022.9847474.

[9] Laura Raya, José Jesús García-Rueda, Daniel López-Fernández e Jesús Mayor. «Virtual Reality Application for Fostering Interest in Art». In: *IEEE Computer Graphics and Applications* 41.2 (2021), pp. 106–113. DOI: 10.1109/MCG.2021.3055685.

[10] Vance J.M. Berg L.P. «Industry use of virtual reality in product design and manufacturing: a survey». In: (2017). DOI: 10.1007/s10055-016-0293-9.

[11] P. Curtis e D.A. Nichols. «MUDs grow up: social virtual reality in the real world». In: *Proceedings of COMPCON '94*. 1994, pp. 193–200. DOI: 10.1109/CMPCON.1994.282924.

[12] Jos P. van Leeuwen, Klaske Hermans, Antti Jylhä, Arnold Jan Quanjer e Hanke Nijman. «Effectiveness of Virtual Reality in Participatory Urban Planning: A Case Study». In: *Proceedings of the 4th Media Architecture Biennale Conference*. MAB18. Beijing, China: Association for Computing Machinery, 2018, pp. 128–136. ISBN: 9781450364782. DOI: 10.1145/3284389.3284491. URL: https://doi.org/10.1145/3284389.3284491.

[13] Anastasios Theodoropoulos, Dimitra Stavropoulou, Panagiotis Papadopoulos, Nikos Platis e George Lepouras. «Developing an Interactive VR CAVE for Immersive Shared Gaming Experiences». In: *Virtual Worlds* 2.2 (2023), pp. 162–181. ISSN: 2813-2084. DOI: 10.3390/virtualworlds2020010. URL: https://www.mdpi.com/2813-2084/2/2/10.

[14] Christiane Breitkreutz, Jennifer Brade, Sven Winkler, Alexandra Bendixen, Philipp Klimant e Georg Jahn. «Spatial Updating in Virtual Reality – Auditory and Visual Cues in a Cave Automatic Virtual Environment». In: (2022), pp. 719–727. DOI: 10.1109/VR51125.2022.00093.

[15] Reon Nashiki, Vibol Yem, Tomohiro Amemiya e Yasushi Ikei. «Footstep Sound for Suppression of VR Sickness and Promotion of Sense of Agency». In: (2019). A cura di Yasuaki Kakehi e Atsushi Hiyama. DOI: 10.2312/egve.20191293.

[16] Marina De Lara, Edson José Rodrigues Justino e Edson Emilio Scalabrin. «Dynamic Generation of Immersive CAVE Environments: Using Digital Game Mechanisms». In: (2023), pp. 923–928. DOI: 10.1109/CSCWD57460.2023.10152798.

[17] K.L. Calvert, M.B. Doar e E.W. Zegura. «Modeling Internet topology». In: *IEEE Communications Magazine* 35.6 (1997), pp. 160–163. DOI: 10.1109/35.587723.

[18] Jeff Erickson. «Local Polyhedra and Geometric Graphs». In: SCG '03 (2003), pp. 171–180. DOI: 10.1145/777792.777820. URL: https://doi.org/10.1145/777792.777820.

[19] Paulo Morgado e Nuno Costa. «Graph-based model to transport networks analysis through GIS». In: (2011), pp. 2–5.

[20] Patricia Ferreira-Lopes e Francisco Pinto-Puerto. «GIS and graph models for social, temporal and spatial digital analysis in heritage: The case-study of ancient Kingdom of Seville Late Gothic production». In: *Digital Applications in Archaeology and Cultural Heritage* 9 (2018), e00074.

[21] Luca Pepe Ugo Erra Delfina Malandrino. «Virtual Reality Interfaces for Interacting with Three-Dimensional». In: (2019), pp. 75–88. DOI: 10.1080/10447318.2018.1429061.