ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA
Corso di Laurea in Ingegneria e Scienze Informatiche

SVILUPPO DI APPLICAZIONI DI EXTENDED REALITY INTEROPERABILI E MULTIPIATTAFORMA

Elaborato in
SISTEMI EMBEDDED E INTERNET-OF-THINGS

Relatore
Prof. ALESSANDRO RICCI

Corelatore
Dott. Ing. ANGELO CROATTI

Presentata da
ANNA VITALI

Anno Accademico 2020 – 2021

"Everything was impossible until somebody did it"
Scott Dinsmore, writer and traveler.

Indice

Introduzione **ix**

1 Concetti di base **1**
1.1 Definizione di eXtended Reality 1
1.2 Mixed Reality e differenze rispetto ad Virtual e Augmented Reality 2
1.2.1 Perchè si chiama Mixed Reality? 2
1.2.2 Differenza fra Augmented e Mixed Reality 3
1.2.3 L'occlusione 3
1.3 Interoperabilità 4
1.4 Aspetti da considerare nella realizzazione di esperienze condivise 5
1.4.1 Qual è il metodo di condivisione? 5
1.4.2 Quali sono le dimensioni del gruppo? 6
1.4.3 Dove si trovano gli utenti? 7
1.4.4 Quando avviene la condivisione? 7
1.4.5 Quanto sono simili i loro ambienti? 8
1.4.6 Quali dispositivi verranno utilizzati? 9

2 Realizzare applicazioni di eXtended Reality interoperabili **11**
2.1 Aspetti che influenzano l'interoperabilità 11
2.1.1 Tipi di interazione 12
2.1.2 Spatial mapping e consapevolezza del mondo 13
2.2 Unity come ambiente di sviluppo 14
2.3 Mixed Reality Toolkit e AR Foundation per ottenere interazioni comuni e realizzare un'applicazione portabile 14
2.3.1 Perché utilizzarli e l'importanza di un'API comune . . . 15
2.3.2 MRTK 15
2.3.3 AR Foundation 16
2.4 Come condividere la stessa conoscenza del mondo? 17
2.4.1 Ancore spaziali 18
2.4.2 Problematiche nell'utilizzo delle ancore spaziali nelle esperienze condivise 18

2.4.3 Markers, Targets e Vuforia 20
2.4.4 Vantaggi nell'utilizzo dei targets e perché sono migliori delle ancore 22

3 OpenXR **23**
3.1 Che cos'è OpenXR 23
3.1.1 Khrons Group fondatore di OpenXR 24
3.2 Come nasce OpenXR 24
3.2.1 Problema della frammentazione 24
3.3 Obiettivi di OpenXR 25
3.4 Caratteristiche di OpenXR 27
3.4.1 Come OpenXR viene vista e implementata dai differenti dispositivi: OpenXR Runtime 27
3.4.2 Un'API estendibile grazie alle Extensions 27
3.4.3 Concetto di layered API 28
3.4.4 Threading Behaviour e Multiprocessing Behaviour 29
3.4.5 Rappresentazione degli oggetti tramite gli Handles 30
3.4.6 Return codes delle funzioni 30
3.4.7 Come avviene la comunicazione fra un runtime e l'applicazione 31
3.4.8 Sistema di coordinate utilizzato 32

4 Architettura, componenti e funzionamento di un'applicazione OpenXR **33**
4.1 Struttura e Architettura di un'applicazione OpenXR 33
4.1.1 OpenXR Loader: che cos'è e quando utilizzarlo 35
4.2 Il concetto di istanza di un'applicazione 36
4.3 Come i dispositivi sono visti dall'applicazione: System e Form Factor 37
4.4 Session e comunicazione fra runtime e applicazione 38
4.4.1 Ciclo di vita di una sessione 38
4.4.2 Gestione del frame loop 39
4.4.3 Analisi degli stati di una sessione 40
4.5 Spaces e sistemi di riferimento 41
4.5.1 Rappresentazione del mondo tramite i Reference Spaces . 41
4.5.2 Monitoraggio delle posizioni nel mondo tramite gli Action Spaces 42
4.5.3 Come individuare una posizione in uno spazio 43
4.5.4 Il concetto di tempo in OpenXR 44
4.6 Gestione degli input dei diversi dispositivi 44

4.6.1 Che cos'è e a cosa serve un Interaction Profile 44
4.6.2 Come un Interaction Profile ci consente di astrarre l'azione dall'input del dispositivo utilizzato 46
4.7 Rappresentazione delle azioni: Actions e Action Sets 47
4.7.1 Collegamento fra azione e input del dispositivo 47
4.7.2 Diversi tipi di azione che possono essere creati 48
4.7.3 Action Sets 49
4.8 Analisi del main loop di un'applicazione OpenXR 49
4.8.1 Gerarchia degli oggetti creati 51

5 Sviluppo di un'applicazione XR in Unity 53
5.1 Perché sviluppare un'applicazione in Unity 53
5.2 Iniziare a costruire l'applicazione 54
5.2.1 Come sono visti i profili di interazione e le estensioni . . 54
5.3 Configurazione dell'ambiente: XR Interaction Toolkit 55
5.3.1 Xr Rig il cuore dell'applicazione 56
5.3.2 Come sono tradotti gli Spaces di riferimento? 57
5.4 Definizione della telecamera e dei controllers all'interno dell'XR Rig 57
5.4.1 Perché la Main Camera necessita di un Tracked Pose Driver e quali funzioni svolge 57
5.4.2 Come vengono visti i controllers all'interno dell'applicazione 58
5.4.3 Definizione delle azioni 59
5.5 Interagire con un oggetto 60
5.5.1 Interactor: gli oggetti che compiono le azioni 60
5.5.2 Interactables: gli oggetti su cui le azioni vengono compiute 60
5.5.3 Considerazioni sulle interazioni in Unity e OpenXR . . . 61
5.6 Perché l'applicazione necessita di altri Plugin oltre ad OpenXR 61
5.6.1 Come cambia il progetto se inseriamo degli altri toolkit? 62
5.7 Virtual Reality avvantaggiata nello sviluppo rispetto alla Mixed e l'Augmented Reality 63
5.8 Vantaggi e svantaggi nell'utilizzo di OpenXR in Unity 64

6 Sperimentazioni di applicazioni di esperienze condivise multipiattaforma 65
6.1 Descrizione delle applicazioni 65
6.2 Individuazione degli scenari di destinazione 66

6.3 Analisi e architettura del Server 67
6.4 Analisi e architettura dei Clients 68
6.5 Sviluppo in Unity del progetto di esperienza condivisa in ambienti diversi 69
6.5.1 Quali sono le tecnologie che si è deciso di utilizzare e perché 69
6.5.2 Descrizione degli elementi nella scena 70
6.5.3 Funzionamento dell'applicazione 71
6.5.4 Gestione dello stato dell'ologramma 74
6.6 Sviluppo in Unity del progetto di esperienza condivisa nello stesso ambiente 74
6.6.1 Quali sono le tecnologie che si è deciso di utilizzare e perché 74
6.6.2 Utilizzo di un'interfaccia di interazione diversa 75
6.6.3 Descrizione degli elementi nelle diverse scene 76
6.6.4 Problemi riscontrati nel posizionamento dell'ologramma e come sono stati risolti 77
6.6.5 Funzionamento dell'applicazione 78
6.7 Considerazioni finali e possibili sviluppi futuri 80

Conclusioni **83**

Ringraziamenti **85**

Introduzione

"What make as human?" è questa la prima domanda che viene posta al lettore nel libro The Infinite Retina di Irena Croin e Robert Scoble [2]. Una prima risposta che potrebbe venire in mente a questa domanda, potrebbe essere: quello che ci rende umani è la nostra mente, il che non è del tutto sbagliato, ma nel libro viene data una risposta un po' diversa.

Quello che ci rende veramente umani e che ha consentito all'uomo di evolversi nel tempo, è l'uso sempre più sofisticato che egli ha fatto degli strumenti che li sono stati dati o che da lui stesso sono stati creati.

Se ci fermiamo a pensare per un secondo alla nostra quotidianità, ci accorgeremo che siamo circondati da oggetti e strumenti che utilizziamo tutti i giorni e che hanno migliorato la qualità della nostra vita, alcuni dei quali hanno acquisito così tanta importanza da diventare indispensabili.

Assieme a noi, anche gli strumenti si evolvono nel tempo e negli ultimi anni, hanno iniziato a prendere sempre più piede tecnologie per l'Augmented, Mixed e Virtual Reality come gli smart glasses e i visori. C'è chi dice che queste nuove tecnologie rimpiazzeranno i moderni telefoni e che entro il 2030, tutti avremo e gireremo con un proprio paio di samrt glasses in testa.

Ma che cosa rende questi dispositivi migliori dei telefoni? Innanzitutto noi viviamo in un mondo tridimensionale, mentre lo schermo del telefono ci mostra il contenuto in due dimensioni, con questi dispositivi invece siamo in grado di vedere il contenuto in tre dimensioni, quindi, per noi tutto sarebbe molto più usabile, poi i campi di applicabilità di questi dispositivi, sono veramente infiniti: medico, industriale, assistenziale ecc.

Inoltre, il fatto di avere le mani libere, durante l'utilizzo di questo strumenti, è un grande vantaggio, si pensi ad esempio ad un medico, che recatosi su un luogo di un incidente, tramite il visore è in grado di vedere quali siano i traumi che ha subito una persona e di intervenire subito, ma pensiamo anche alle persone disabili, che su un primo momento verrebbe da scartarle nell'utilizzo di questi dispositivi, soprattutto le persona non vedenti, immaginiamo invece che questi occhiali possano essere la vista di queste persone, interpretando il mondo intorno a noi: leggendo i cartelli, riconoscendo le persone, i segnali visivi ecc.

L'aumento di popolarità che questi dispositivi stanno acquisendo e acquisteranno nel tempo, farà anche sì che sempre più venditori vorranno entrare in questo mercato, sviluppando e producendo dispositivi nuovi e differenti. Già oggi il mercato presenta diversi providers, tra cui i colossi come Microsoft e Google.

Se vogliamo quindi sviluppare un'applicazione per questi dispositivi e vogliamo che funzioni per la maggior parte di quelli presenti sul mercato, almeno i principali, come possiamo fare? Tenendo conto che diversi sistemi possono presentare caratteristiche diverse, ad esempio per il mappaggio dello spazio, vi è comunque la possibilità di creare un programma che possa essere eseguito sui diversi sistemi?

Già oggi si stanno iniziando a sviluppare delle tecnologie che consentano di creare dei programmi, che possano essere eseguiti sulle diverse piattaforme, cercando anche di far in modo che questi dispositivi siano in grado di cooperare fra loro, realizzando delle esperienze condivise che coinvolgano sistemi diversi.

Infatti, maggiore sarà la popolarità che questi sistemi acquisiranno nel tempo, maggiore sarà per noi la necessita di comunicare nel migliore dei modi, attraverso di essi, arrivando a definire anche nuovi metodi per lo scambio di informazioni. Adesso magari potrebbe fare un po' sorridere ma un domani, potremmo arrivare a scambiarci non più dei documenti ma bensì degli ologrammi.

Risulta quindi essere importante, iniziare a definire un metodo per poter sviluppare delle applicazioni di eXtended Reality, cioè che siano in grado si supportare le diverse realtà: Mixed, Augmented e Virtual e che siano sostenute dalla diverse piattaforme, consentendo anche un'interoperabilità fra i diversi dispositivi.

Pertanto gli obbiettivi che questa tesi si propone di raggiungere sono di: inizialmente, definire e analizzare una serie di concetti base per comprendere appieno l'argomento principale, successivamente entrare nel merito della progettazione di esperienze condivise che coinvolgano anche differenti devices, effettuando un'analisi di quali siano le caratteristiche che influenzano l'interoperabilità e di conseguenza quali siano le tecnologie che possono essere utilizzate oggi, le quali consentono di risolvere queste differenze, descrivendo tutto questo però non con un punto di vista puramente implementativo, ma sopratutto funzionale, analizzando nel dettaglio il funzionamento di un nuovo framework OpenXR, il quale costituisce un'Application Program Interface, da poter utilizzare per la creazione di applicazioni di eXtended Reality portabili.

Infine si cercherà di mettere in pratica tutti i concetti appresi e di utilizzarli per la realizzazione di due sperimentazioni, che prevedono la creazione di un'esperienza condivisa a cui sarà possibile accedere con differenti devices.

Capitolo 1

Concetti di base

Prima di entrare nel merito della realizzazione di queste applicazioni di eXtended Reality, occorre parlare di alcuni aspetti che aiuteranno a comprendere meglio i successivi discorsi che verranno affrontati. In particolare verranno definiti i concetti di eXtended, Mixed, Augmented e Virtual Reality, cosa si intende per interoperabilità e quali siano gli aspetti da considerare nella realizzazione di esperienze condivise.

1.1 Definizione di eXtended Reality

Oggi il termine eXtended Reality non è ancora molto conosciuto, molto più spesso invece abbiamo sentito termini come Augmented o Virtual Reality. Viene quindi da chiedersi, che cosa sia e cosa introduca di diverso l'eXtended Reality.

Con il termine eXtended Reality (realtà estesa), ci si vuole riferire a tutti gli ambienti reali e virtuali combinati fra loro e a tutte le interazioni uomo-macchina, generate dalla tecnologia informatica e dai dispositivi indossabili, come ad esempio i visori. Questo termine viene comunemente utilizzato con l'abbreviazione di XR, dove la "X", rappresenta una variabile che prende il posto delle lettere: "A" di Augmented, "V" di Virtual e "M" di Mixed, mentre la lettera "R" rappresenta il termine Reality.

In conclusione possiamo riassumere il concetto di eXtended Reality come una insieme di tutte le realtà attualmente esistenti e realizzare applicazioni di eXtended Reality significa appunto, creare dei programmi che possano essere conformi alle diverse caratteristiche che queste realtà presentano.

1.2 Mixed Reality e differenze rispetto ad Virtual e Augmented Reality

La Mixed Reality, consente di avere un collegamento fra mondo fisico e mondo virtuale. In un'esperienza di Mixed Reality un utente può muoversi nel mondo fisico e nel mondo virtuale allo stesso modo e simultaneamente.

Gli oggetti virtuali riconoscono il mondo fisico e possono interagire con esso, ad esempio, possiamo posizionare un ologramma sul tavolo o fare in modo che questo rimanga sempre attaccato a una parete.

Gli ologrammi, sono visti come dei veri e propri oggetti: se ci allontaniamo li vediamo rimpicciolirsi, se ci avviciniamo si ingrandiscono e se invece ci giriamo attorno possiamo osservarli da diversi punti di vista, ma possiamo fare anche molto di più, le moderne tecnologie di Mixed Reality come ad esempio HoloLens 2, ci consentono anche di "toccare" gli ologrammi, utilizzando quella che viene definita una Natural Interface.

Il termine Natural Interface, introduce un tipo di interazione che noi siamo già predisposti ad utilizzare in quanto ci viene naturale: possiamo interagire con questi ologrammi come faremmo di solito con gli oggetti fisici del mondo reale, non abbiamo bisogno di imparare ad utilizzare quest'interfaccia siamo già in grado di farlo e tutto questo rende quindi l'interazione molto più semplice e le applicazioni più facili da utilizzare.

1.2.1 Perchè si chiama Mixed Reality?

Il termine Mixed Reality, deriva dal fatto che tale tecnologia riesce a combinare aspetti di Augmented e Virtual Reality insieme e se dovessimo tracciare un confine fra Augmented e Virtual Reality la Mixed si troverebbe proprio su questo confine in mezzo a queste realtà.

Per capire il perché di questo occorre quindi soffermarsi sulle differenze che vi sono fra Augmented e Virtual Reality.

Come riportato nel libro Augmented Reality di J.Peddie: "L'unica cosa che Augmented e Virtual Reality hanno in comune è il termine reality" [13], infatti tali tecnologie, anche se condividano simili dispositivi, offrono delle esperienze completamente diverse.

La più grande differenza che vi è fra le due è che l'Augmented Reality consente di vedere il mondo reale che ci circonda e gli ologrammi sono disposti in questo mondo fisico, riuscendo a riconoscere ad esempio il livello del pavimento; mentre nella Virtual Reality l'esperienza creata viene costruita ad hoc dai programmatori e viene progettato un mondo virtuale che non ha nulla a che vedere con quello fisico. La vista dell'utente durante l'esperienza di Virtual Reality è completamente oscurata.

1.2.2 Differenza fra Augmented e Mixed Reality

Avendo ora introdotto l'Augmented Reality, viene da chiedersi quale sia la differenza di questo tipo di realtà e la Mixed Reality, perché in entrambi i casi siamo in grado di vedere il mondo che ci circonda e gli ologrammi possono essere disposti in questo mondo.

La differenza sta proprio nella consapevolezza del mondo reale, con l'Augmented Reality ad esempio siamo in grado di riconoscere le superfici come il pavimento, ma se abbiamo un ologramma che è libero di muoversi per la nostra stanza, questo ologramma non riconoscerà i mobili o la scrivania, rimarrà sempre visibile a noi e non verrà nascosto da tali superfici, per lui sarà come se non esistessero. Nella Mixed Reality invece l'ologramma riconosce il mondo e se muovendosi finisce dietro a un mobile o sotto una scrivania, non siamo più in grado di vederlo e questo è possibile grazie a una proprietà fondamentale che nell'Augmented Reality manca che è **l'occlusione**.

1.2.3 L'occlusione

L'occlusione quindi è la proprietà che marca maggiormente la differenza fra Mixed e Augmented Reality, per poterla ottenere però bisogna mettere in campo diversi meccanismi, perché dobbiamo far si che il nostro device abbia una consapevolezza del mondo.

Per farlo quindi occorrono, non solo sensori appositi ma anche algoritmi di Intelligenza Artificiale e Semantica che consentono al dispositivo di attribuire un significato ai dati che sono stati raccolti.

Infine, i moderni dispositivi come HoloLens 2 effettuano una continua scansione della scena per tenere traccia di tutti i possibili cambiamenti che vi sono stati, ad esempio: se il dispositivo ha effettuato il mappaggio di una scrivania e successivamente viene spostato un bicchiere che vi era appoggiato, se il dispositivo non effettuasse questa scansione continua, la sagoma fantasma del bicchiere rimarrebbe salvata e riconosciuta come superficie del mondo, cosa che però sarebbe sbagliata perché ora il bicchiere non vi è più e quindi per rilevare questo cambiamento è necessario effettuare un successivo mappaggio dell'ambiente.

In conclusione, se dovessimo rappresentare con un immagine le differenze fra tutti questi tipi di realtà e la loro relazione con il mondo fisico e virtuale, arriveremmo a un risultato mostrato nella seguente figura 1.1

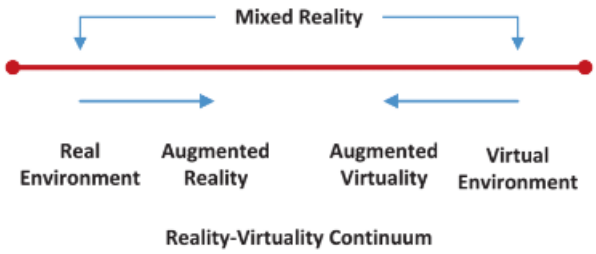


Figura 1.1: Rappresentazione semplificata delle diverse realtà

1.3 Interoperabilità

Nell'introduzione è stato detto che nel futuro, i providers di dispositivi per le differenti realtà aumenteranno sempre di più, sarà quindi importante riuscire a definire delle applicazioni, che consentano una collaborazione fra questi diversi sistemi.

Per avere un'idea maggiormente chiara su quale sarà l'obbiettivo da raggiungere, dobbiamo considerare in prima istanza, cosa intendiamo per interoperabilità.

Il termine interoperabilità vuole esprimere la possibilità di: permettere mediante procedure unificanti l'interscambio e l'interazione fra diversi dispositivi, non per forza omogenei. L'obiettivo dell'interoperabilità è quello di facilitare l'interazione fra i diversi sistemi consentendo anche un riutilizzo o rielaborazione delle informazioni.

Si vorrebbe quindi arrivare alla realizzazione di un software **portabile**, in grado di funzionare bene e senza errori su diversi devices, che offra un nuovo tipo di servizio.

Supponiamo, ad esempio, di avere a disposizione un dispositivo HoloLens 2 e di utilizzarlo durante la nostra attività fisica quotidiana e di avere anche a disposizione tutta una serie di dispositivi che monitorano il nostro battito cardiaco, l'ossigenazione del sangue, il numero di ripetizioni ecc. tutti i dati rilevati, vorremo poi che fossero visibili sul nostro visore per esserne a conoscenza, per fare questo quindi, è necessario che durante l'attività fisica, i diversi dispositivi utilizzati per il monitoraggio comunichino con HoloLens e li inviino i dati.

Possiamo però pensare anche a qualcosa di più complesso: immaginiamo di essere in un meeting aziendale, dove dobbiamo decidere il progetto per un nuovo macchinario e per farlo noi e i nostri colleghi abbiamo a disposizione un visore, che ci consente di vedere il prototipo della nostra macchina. Siccome vogliamo decidere nel dettaglio, come realizzare il nostro macchinario,

con il contributo di tutti i presenti alla riunione; vorremmo essere in grado di manipolare l'ologramma che ci troviamo davanti e che, non solo noi che effettuiamo la manipolazione, ma anche gli altri, siano in grado di vedere le nostre manipolazioni e modifiche, pur indossando visori diversi e magari anche non essendo nello stesso ambiente. Dando la possibilità ai colleghi di collegarsi da remoto e fare in modo che gli altri utenti connessi visualizzino un loro avatar, come mostrato in figura 1.2.

Figura 1.2: Esempio di shared experience

Per poter realizzare tutto questo quindi, dobbiamo riuscire per prima cosa a far interoperare i diversi dispositivi fra loro.

1.4 Aspetti da considerare nella realizzazione di esperienze condivise

Prima di implementare un'applicazione che realizzi un'esperienza condivisa dobbiamo valutare alcuni aspetti nella fase di progettazione, perché possono essere realizzate diversi tipi di esperienze condivise. Di conseguenza, la prima cosa da individuare nella realizzazione della nostra applicazione, sono gli scenari di destinazione.

Questi scenari, consentono di comprendere meglio che cosa si sta progettando e quali siano le funzionalità che volgiamo che siano presenti nella nostra esperienza.

Per aiutarci in questo, Microsoft durante lo sviluppo e l'esplorazione di applicazioni per HoloLens 2, ha creato sei domande a cui occorre darsi una risposta prima della realizzazione di esperienze condivise [11].

1.4.1 Qual è il metodo di condivisione?

Vi sono diversi modi in cui potremmo condividere dei contenuti, ad esempio: una presentazione potrebbe essere gestita da un singolo utente virtuale,

che si occupa dell'interazione con gli ologrammi, mentre gli altri utenti collaborano ma senza la possibilità di interagire, oppure come nell'esempio visto per l'interoperabilità (Sezione 1.3) potremmo avere un team di lavoro che collabora insieme per un progetto e in questo caso avremmo un maggior livello di complessità.

In definitiva possiamo dire che: la complessità di un'esperienza condivisa, aumenta in base **al livello di interazione** che un utente possiede o può possedere in uno scenario.

I modi con cui gli utenti possono condividere il contenuto, pur essendo diversi, possono comunque essere suddivisi in tre macro categorie:

- Presentation: che si ha quando lo stesso contenuto viene mostrato da un utente agli altri, che però non hanno la possibilità di interagire.
- Collaboration: quando un team o un gruppo lavora insieme per raggiungere un obiettivo comune, dove tutti sono in grado di interagire e manipolare il contenuto mostrato.
- Guidance: che invece si ha quando, una persona aiuta a risolvere un problema o spiega come svolgere un procedimento a un altra persona seguendola passo passo, ad esempio: un tecnico di un'azienda aiuta un lavoratore nell'installazione di un nuovo macchinario tramite un modello virtuale che lo guida nella spiegazione.

Proprio questo ultimo tipo di condivisione, si pensa che sarà quello che prenderà maggior piede e che le diverse imprese siano interessate ad ottenere, infatti uno dei vantaggi derivanti dall'utilizzo di questo tipo di applicazioni, che negli ultimi tempi è diventato sempre più importante è che: per gli impiegati, non ci sarebbe più la necessita di compiere lunghi viaggi di lavoro per andare ad esempio a risolvere un malfunzionamento improvviso, del macchinario installato nella ditta del cliente, producendo quindi, una riduzione degli spostamenti e un aumento della produttività, in quanto, il tempo risparmiato dal lavoratore nel non compimento del viaggio aziendale, può essere speso per compiere altre operazioni, utili all'impresa.

1.4.2 Quali sono le dimensioni del gruppo?

Possiamo realizzare esperienze condivise che siano composte da un numero definito o scalabile di utenti, naturalmente, maggiore è il numero di utenti presenti, maggiori saranno le difficoltà che si dovranno gestire: dal punto di vista tecnico, per quanto riguarda la condivisione dei dati e la gestione della rete e se viene data la possibilità agli utenti di connettersi in remoto, anche dal punto

di vista sociale nella gestione ottimale dei diversi avatar che li rappresentano, in modo tale che la comunicazione possa avvenire nel migliore dei modi.

In questo caso, possiamo dire che: la complessità dell'esperienza cresce esponenzialmente **a mano a mano che si passa da un gruppo ristretto a uno sempre più grande di utenti**.

In particolare, viene definito **piccolo** un gruppo che possiede meno di sette persone e **grande** un gruppo che possiede un numero di componenti maggiore o uguale a sette.

L'aspetto della dimensione del gruppo, nella realizzazione delle applicazioni, non deve essere trascurato in quanto influenza:

- La rappresentazione delle persone nello spazio olografico.
- La scala degli oggetti.
- La scala dell'ambiente.

1.4.3 Dove si trovano gli utenti?

Le esperienze condivise, possono anche essere distinte in base al luogo in cui gli utenti che partecipano all'esperienza, si trovano.

In particolare, distinguiamo esperienze di tipo:

- Colocated: dove tutti gli utenti sono nello stesso spazio fisico.
- Remote: dove invece, gli utenti si trovano in spazi fisici differenti.
- Both (Colocated and Remote): quando possono essere presenti, utenti che si trovano nello stesso spazio fisico ma anche utenti connessi in remoto.

La risposta a questa domanda è cruciale, in quanto influenzerà: come le persone comunicheranno fra loro, quali oggetti potranno vedere, se tutti o solamente alcuni e di cosa abbiamo bisogno per adattarci ai diversi ambienti.

1.4.4 Quando avviene la condivisione?

Quando ci fermiamo a pensare ad un'esperienza multiutente, molto spesso immaginiamo quest'esperienza come sincrona, tutti siamo collegati allo stesso momento. Ma non sempre questo è vero, potremmo infatti realizzare delle applicazioni, sempre collaborative, ma che consentono agli utenti di vedere il contenuto in un secondo momento, ad esempio: supponiamo di avere un'applicazione che ci consente di lavorare su un prototipo di un edificio in costruzione

e di dover sottomettere tale prototipo al nostro capo, per l'approvazione, che però potrà visionarlo solamente in un secondo momento, di conseguenza, salviamo lo stato della nostra applicazione e aspettiamo un suo riscontro. Quando il capo accederà all'applicazione con il proprio visore, sarà in grado di vedere il lavoro che abbiamo svolto così come noi lo abbiamo lasciato e se vi sarà qualcosa da correggere, esso avrà la possibilità di lasciare delle note, con le correzioni da applicare.

In questo esempio, abbiamo un'esperienza asincrona, in quanto accediamo al contenuto condiviso in un secondo momento, mantenendo la possibilità di modificarlo, anche se non eravamo presenti al momento della creazione. Naturalmente, in questo caso la complessità aumenta rispetto a un esperienza sincrona, perché dobbiamo mantenere traccia delle modifiche effettuate da ciascun utente e gestire anche i diversi utenti che accedono all'applicazione, magari con anche diversi livelli di privacy.

Alla luce di quello che è stato detto, occorre distinguere fra esperienze di tipo:

- Synchronously: quando, la condivisione dell'esperienza olografica avviene nello stesso momento.
- Asynchronously: quando, l'esperienza olografica può essere condivisa in momenti diversi.
- Both (Synchronously, Asynchronously): quando, gli utenti a volte potranno condividere il contenuto in modo sincrono e altre invece in modo asincrono.

Anche rispondere a questa domanda è importante perché: influenza gli oggetti nell'ambiente, la loro persistenza e la prospettiva dell'utente, in quanto, a volte potremmo dover mantenere salvata l'ultima visuale dell'utente sulla scena.

1.4.5 Quanto sono simili i loro ambienti?

La condizione ideale, sarebbe quella in cui tutti gli utenti possiedono un ambiente identico in cui muoversi, purtroppo nella pratica non è mai così; ma nonostante la mancanza di ambienti identici, quello che potremmo avere è un ambiente simile, si pensi ad esempio alle sale conferenze, di solito hanno tutte una configurazione simile: pochi mobili e un grande tavolo al centro circondato da sedie.

La similarità fra le sale conferenze, è sicuramente maggiore rispetto a quella che potremmo avere se considerassimo ad esempio, un soggiorno, che per utenti

diversi può presentare oggetti diversi oppure anche oggetti simili, ma disposti nell'ambiente in modo diverso.

Quindi, possiamo distinguere gli ambienti in:

- Similar: ambienti che tendono ad avere lo stesso mobilio, luce e dimensione della stanza.
- Dissimilar: ambienti che differiscono per mobilio, luce o dimensione della stanza.

Gli aspetti che le caratteristiche dell'ambiente influenzano sono: come le persone vedranno gli oggetti, ad esempio, se l'esperienza richiede l'utilizzo di un tavolo, l'ambiente deve riuscire a soddisfare questo requisito e la scala degli oggetti, un oggetto molto grande necessita di un certo spazio, mentre un oggetto abbastanza piccolo può essere visualizzato correttamente in ogni ambiente.

1.4.6 Quali dispositivi verranno utilizzati?

Oggi siamo portati a pensare, che esperienze condivise di Mixed Reality, possano essere effettuate solamente con dispositivi immersivi, oppure con dei dispositivi olografici, però una vera esperienza condivisa dovrebbe consentire l'accesso anche ad altri e diversi dispositivi, compresi quelli 2D come telefoni o tablet.

Capire con quale tipo di dispositivi si vuole consentire l'accesso ai diversi utenti, è importante non solo dal punto di vista implementativo, ma anche per quanto riguarda l'usabilità, per riuscire a soddisfare le aspettative che un utente si crea nell'utilizzo di questo tipo di applicazioni.

Capitolo 2

Realizzare applicazioni di eXtended Reality interoperabili

Dopo aver definito i concetti base, che occorre apprendere per iniziare a sviluppare applicazioni per le diverse realtà, iniziamo a entrare nel merito di come poter ottenere l'interoperabilità fra i diversi dispositivi.

In particolare, si cercherà di individuare le differenze che i sistemi, su cui possono essere eseguite applicazioni per le differenti realtà presentano e quali siano gli strumenti e le strategie da poter utilizzare, che possono fornire un supporto ai fini dell'interoperabilità.

2.1 Aspetti che influenzano l'interoperabilità

La prima cosa da dover considerare, per poter ottenere un'applicazione interoperabile, è la comunicazione, non si può ottenere l'interoperabilità se i dispositivi non sono in grado di comunicare fra loro.

La soluzione più utilizzata, per questo primo problema, consiste nella definizione di messaggi, che potranno essere ricevuti e inviati dai dispositivi coinvolti attraverso la rete, i quali conterranno le informazioni che consentiranno ai diversi sistemi, di cooperare fra loro.

Una volta stabilito come avverrà la comunicazione, occorre analizzare le caratteristiche degli apparati che si vuole coinvolgere nell'esperienza condivisa, al fine di individuarne le differenze che possono influire nella realizzazione dell'applicazione. I differenti sistemi, infatti, presentano caratteristiche diverse, si pensi ad esempio a un visore immersivo, che consente di vedere le immagini in un formato tridimensionale e le differenze che esso presenta rispetto a uno smartphone, dotato di un display 2D, che per quanto possa essere all'avanguardia, non ci consentirà di poter vedere l'ologramma nel suo formato

tridimensionale, almeno per ora... Di conseguenza, i diversi sistemi possono presentare delle incompatibilità che occorre individuare e colmare.

2.1.1 Tipi di interazione

Una delle differenze principali, che si può notare nell'utilizzo dei diversi dispositivi è il tipo di interazione. Questa, se utilizziamo un dispositivo per la Virtual Reality probabilmente avverrà per mezzo di controllers appositi, che ci consentiranno di toccare gli oggetti del mondo virtuale, tramite la pressione di pulsanti o l'esecuzione di opportuni movimenti. Se invece utilizziamo dei visori di ultima generazione, come HoloLens 2, possiamo interagire con gli ologrammi direttamente con le nostre mani, senza l'utilizzo di controllers, in quanto, il dispositivo è dotato di un sistema di monitoraggio che con appositi sensori e algoritmi consente di individuare: non solo la posizione delle mani dell'utente, ma anche i movimenti compiuti, riconoscendo gesti specifici che consentono un'interazione con l'interfaccia presentata; come ad esempio il gesto di air tap, il quale consente di avere lo stesso effetto che si otterrebbe con un click del mouse, se stessimo utilizzando un PC. Tale gesto, si ottiene semplicemente congiungendo pollice e indice di una mano, com'è possibile vedere nella seguente figura.

(a) Controller del visore HTC vive

(b) Gesto di air tap riconosciuto da HoloLens

Figura 2.1: Differenze negli input dei dispositivi

Inoltre, se nell'esperienza condivisa vogliamo inserire anche dei dispositivi con uno schermo 2D come tablet e telefoni, avremo un altro tipo di interazione ancora, che avviene toccando lo schermo del dispositivo.

Il problema che occorre risolvere, sarà quello di realizzare un'applicazione che possieda un'apposita interfaccia che si adatti a tutti i diversi tipi di input, che possono essere registrati dai dispositivi che si intende utilizzare.

2.1.2 Spatial mapping e consapevolezza del mondo

Un'altra differenza che i diversi sistemi presentano e che influisce sul posizionamento degli ologrammi è la consapevolezza del mondo.

I dispositivi per la Mixed Reality, come HoloLens 2 sono dotati di un sistema di monitoraggio chiamato SLAM (Simultaneous localization and mapping), costituito da un insieme di telecamere e altri sensori, che li consentono di effettuare un mapping di un ambiente sconosciuto e di riconoscere la posizione del dispositivo in tale ambiente. Tale sistema, tramite raffinamenti successivi, consente di costruire una mappa sempre più dettagliata dell'ambiente in cui ci si trova, che è possibile vedere tramite una mesh, di cui un esempio è riportato nella seguente figura 2.2.

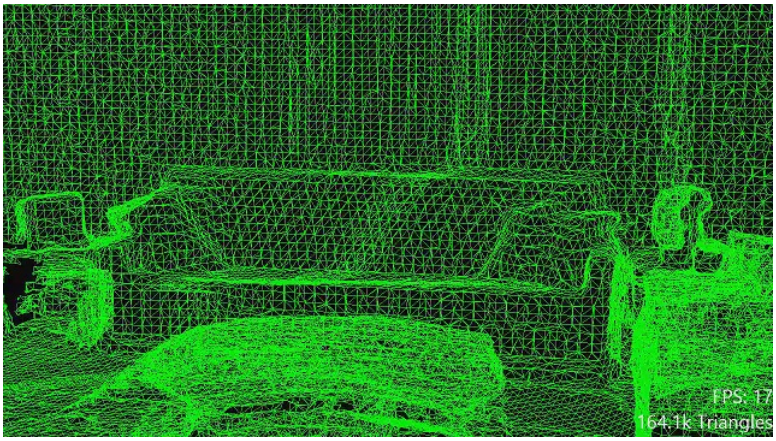


Figura 2.2: Mapping spaziale effettuato da un dispositivo HoloLens

Non tutti i dispositivi però sono come HoloLens 2 e i visori per la realtà virtuale non effettuano il mapping dell'ambiente, perché come descritto nel capitolo uno, tali sistemi non consentono la visone del mondo esterno, ma ciò che viene reso visibile all'utente è stato interamente creato dal programmatore, della specifica applicazione che si sta utilizzando. Anche gli smartphone e i tablet, seppur dotati di una telecamera, che li consente di vedere il mondo reale, non possiedono un sistema di consapevolezza spaziale.

Quindi, se consideriamo per un momento solamente esperienze di Augmented e Mixed Reality, in cui gli utenti sono in grado di vedere il mondo esterno, si vorrebbe ottenere un'applicazione, che consenta agli utenti che si trovino nello stesso ambiente, di vedere gli ologrammi nello stesso punto della stanza e che a seconda della posizione assunta dall'utente, la prospettiva che si ha sull'ologramma cambi, quindi, se ad esempio si gira attorno ad un ologramma, esso rimarrà fisso nella sua posizione e si avrà la possibilità di vederlo da diverse angolazioni.

In conclusione, il problema principale in questo caso, è fare in modo che dispositivi privi di un sistema di consapevolezza spaziale siano in grado, se si trovano nello stessa ambiente, di vedere il contenuto olografico nella stessa posizione del mondo reale.

2.2 Unity come ambiente di sviluppo

Ora che si è visto quali sono gli aspetti da considerare per ottenere l'interoperabilità e i problemi che i diversi dispositivi presentano, che influenzano sullo sviluppo delle applicazioni. Nelle successive sezioni, si entrerà nel merito di quali sono le tecnologie da poter utilizzare, per realizzare applicazioni XR a partire dall'ambiente di sviluppo Unity.

Unity è un motore grafico multi piattaforma sviluppato da Unity Technologies, inizialmente realizzato per lo sviluppo di videogiochi, ma che poi si è evoluto consentendo la realizzazione di applicazioni sempre più complesse tra cui quelle per l'eXtended Reality.

Il grande vantaggio di Unity è che,presenta una Special GUI, che può essere utilizzata per la realizzazione delle applicazioni, consentendo anche ai non programmatori di poter realizzare diverse funzionalità, anche complesse, senza dover scrivere necessariamente degli scripts. Nonostante questo, i programmatori che desiderano implementare specifiche funzionalità per il loro programma, possono aggiungere degli script `C#` al progetto, creati direttamente da loro che le implementano.

Unity però presenta anche degli svantaggi, cambia molto velocemente e vi è il rischio che un'applicazione costruita per una certa versione del programma, non funzioni in quelle successive, inoltre, il fatto che non sia di per se un ambiente di sviluppo, nato per la creazione di applicazioni XR, fa sì che tale sistema venga arricchito sempre di più con maggiori funzionalità per riuscire a sviluppare le diverse applicazioni, sovraccaricando lo sviluppatore con: pacchetti da installare, impostazioni da modificare, togliere o aggiungere ecc.

2.3 Mixed Reality Toolkit e AR Foundation per ottenere interazioni comuni e realizzare un'applicazione portabile

Uno dei problemi che influenza l'interoperabilità della applicazioni, analizzato precedentemente fu: creare un'interfaccia che si adatti ai diversi tipi di input, dei differenti devices e creare un'applicazione che sia quanto più simile per i diversi sistemi.

Per risolvere questo problema, i provider come Microsoft e Google, hanno realizzato per i diversi ambienti di sviluppo, dei toolkit che mettono a disposizione degli elementi base per la creazione di applicazioni XR e per la gestione dell'interfaccia grafica.

2.3.1 Perché utilizzarli e l'importanza di un'API comune

Il grande vantaggio nell'utilizzo di questi strumenti è che possono essere supportati da diversi dispositivi, avendo quindi la possibilità, sopratutto se i devices sono simili fra loro, di poter eseguire applicazioni identiche sui differenti sistemi senza dover cambiare nessuna linea di codice.

Viene quindi da chiedersi come tutto questo sia possibile?

Questo grande vantaggio, dato da queste tecnologie si ottiene grazie all'utilizzo di un'API comune riconosciuta dai diversi dispositivi. E' importante marcare il fatto che: **senza l'utilizzo di un'API comune, si dovrebbero creare applicazioni differenti per i diversi dispositivi, che implementano l'API specifica, da loro utilizzata** e in alcuni casi le applicazioni che si dovrebbe realizzare per far sì che il programma funzioni, possono essere davvero molto diverse fra loro.

2.3.2 MRTK

Il Mixed Reality Toolkit (MRTK), come riportato nella documentazione di Microsoft [16], è un kit di sviluppo per le applicazioni di Mixed Reality open-source e multi-piattaforma. Esso può essere utilizzato per realizzare applicazioni attraverso l'applicativo Unity o Unreal Engine, importando gli appositi packages.

Attualmente l'MRTK offre agli sviluppatori: un cross-platform input system, una serie di componenti basici per la realizzazione delle applicazioni XR e building blocks comuni per interazioni spaziali.

L'obbiettivo principale dell'MRTK, è quello di fornire un supporto per la realizzazione di programmi a più devices possibili, attualmente i devices e le piattaforme supportate sono: Microsoft HoloLens, Open VR, i dispositivi VR di Microsoft e Magic Leap, presenta inoltre una compatibilità anche con i sistemi che utilizzano AR Foundation quindi dispositivi Android e IOS.

I vantaggi che derivano dall'utilizzo del Mixed Reality Toolkit sono: la modularità, in quanto, è stato costruito in modo tale che non sia necessario includere in un progetto tutte le sue funzionalità, ma solamente quelle necessarie che risultano utili ai fini dell'applicazione, consentendo al progetto di mantenere una dimensione accettabile che lo rende più semplice da gestire e

la configurabilità, in quanto, viene data la possibilità ai programmatori, di rimpiazzare i componenti forniti dall'MRTK con dei propri, che implementino in modo diverso le funzionalità fornite.

Il Mixed Reality Toolkit, risulta essere uno strumento idoneo alla realizzazione di programmi che coinvolgono diversi dispositivi, in quanto, non solo fornisce strumenti per la creazione delle interfacce utente, ma gestisce autonomamente anche i diversi tipi di input dei differenti dispositivi che si intende utilizzare.

2.3.3 AR Foundation

AR Foundation, come riportato nella documentazione di Unity [14], consente agli sviluppatori di lavorare con la realtà aumentata, attraverso lo sviluppo di applicazioni multi-piattaforma.

Questa libreria però presenta uno svantaggio: per poter lavorare con un device target specifico, all'interno del progetto, bisogna includere l'opportuno plugin. Attualmente, i plugin disponibili, che possono essere utilizzati in Unity sono:

- ARCore XR plugin per Android.
- ARKit XR plugin per IOS.
- Magic Leap XR plugin per Magic Leap.
- Windows XR plugin per HoloLens.

AR Foundation, definisce un'API che consente ai programmatori di lavorare e utilizzare funzionalità comuni a diverse piattaforme, senza però fornire un'implementazione per tali funzionalità, a questo ci pensano i plugin.

In particolare, l'architettura di AR Foundation, è basata su quelli che vengono definiti dei `Subsystems` e la loro relativa implementazione che viene data dai `Providers`.

Un `Subsystem`, è sostanzialmente un'interfaccia definita in modo indipendente dalla specifica piattaforma che poi verrà utilizzata. L'obbiettivo di un subsystem, è quello di mettere in luce le diverse funzionalità offerte da AR Foundation, ad esempio: `XRPlaneSubsystem` è il subsystem che fornisce l'interfaccia per il riconoscimento delle superfici.

Un `Providers`, invece, rappresenta la concreta implementazione di un subsystem ad esmpio: il package `ARCore XR Plugin`, contiene l'implementazione ARCore di molti degli AR subsystem definiti da AR Foundation.

In particolare, l'elenco delle funzionalità fornite e dei diversi device supportati da AR Foundation è indicato nella seguente figura 2.3.

	ARCore	ARKit	Magic Leap	HoloLens
Device tracking	✓	✓	✓	✓
Plane tracking	✓	✓	✓	
Point clouds	✓	✓		
Anchors	✓	✓	✓	✓
Light estimation	✓	✓		
Environment probes	✓	✓		
Face tracking	✓	✓		
2D Image tracking	✓	✓	✓	
3D Object tracking		✓		
Meshing		✓	✓	✓
2D & 3D body tracking		✓		
Collaborative participants		✓		
Human segmentation		✓		
Raycast	✓	✓	✓	
Pass-through video	✓	✓		
Session management	✓	✓	✓	✓
Occlusion	✓	✓		

Figura 2.3: Elenco funzionalità e device supportati da AR Foundation

Data la sua architettura, AR Foundation presenta il vantaggio della modularità e come visto anche per l'MRTK gestisce in autonomia i diversi tipi di input derivanti dai differenti apparti, tuttavia, si tratta di una piattaforma pensata per la realizzazione di applicazioni di Augmented Reality e quindi non può essere utilizzata per lo sviluppo di applicazioni per i visori di Virtual Reality e non fornisce dei componenti per la realizzazione di interfacce utente.

2.4 Come condividere la stessa conoscenza del mondo?

Un altro problema che si è visto, negli aspetti che influenzano l'interoperabilità è: la consapevolezza del mondo e come gestire la posizione degli ologrammi nello spazio, in modo che i diversi utenti possano vederli nello stesso punto e in base alla loro prospettiva.

Per fare questo, anche in questo caso ci vengono incontro diverse tecnologie, tra cui: le ancore spaziali e i targets di cui si parlerà nelle seguenti sezioni, analizzando anche i pregi e i difetti di ognuna di queste tecnologie.

2.4.1 Ancore spaziali

Un'ancora spaziale, rappresenta un punto importante nel mondo di cui il sistema tiene traccia nel tempo. Ogni ancora presenta un proprio sistema di coordinate e esse possono essere utilizzate per mantenere la posizione degli ologrammi stabile nel tempo, ma anche per definire un origine del mondo aumentato, comune ai diversi dispositivi.

Come abbiamo visto precedentemente, i dispositivi di Mixed Reality effettuano un mappaggio dell'ambiente in cui si trovano, definendo un sistema di coordinate che nella maggior parte dei casi, ha come origine il punto in cui il dispositivo si trovava quando l'applicazione è stata lanciata. Ora se noi abbiamo diversi dispositivi, che effettuano questo tipo di mappaggio, avremmo sistemi di coordinate tutti differenti fra loro, in quanto, il punto di origine sarà diverso da dispositivo a dispositivo. Occorre trovare il modo di definire un sistema di coordinate comuni.

La prima soluzione che potrebbe venire in mente, è quella di unire i sistemi di coordinate di questi dispositivi e crearne uno unico che vada bene per tutti, tuttavia questa soluzione è molto complessa e difficile da realizzare. Quindi, quello che di solito viene fatto è utilizzare un ancora, che tutti possediamo e vediamo nello stesso punto e questo è reso possibile grazie al fatto che un'ancora, viene collegata alla mesh generata dal dispositivo e non direttamente al sistema di coordinate stesso, quindi essa è in grado di riconoscere la sua posizione nel mondo reale, ed è di conseguenza possibile utilizzare quest'ancora come origine del mondo e tutti gli oggetti del mondo aumentato verranno posizionati in funzione del sistema di riferimento da lei definito.

Tale approccio risulta essere più semplice rispetto a quello che prevede l'unione dei sistemi di coordinate, perché in questo caso è solamente necessario che i dispositivi condividano l'ancora e si trovino nello stesso ambente, per poter riuscire a vedere gli ologrammi nello stesso punto.

2.4.2 Problematiche nell'utilizzo delle ancore spaziali nelle esperienze condivise

Dalla descrizione precedente sembra che le ancore spaziali risolvano già il problema del sistema di riferimento e in parte è così, ma tuttavia, questo sistema presenta anche degli svantaggi, che è possibile individuare analizzando il progetto d'esempio, realizzato da Microsoft per la prima generazione di HoloLens, nell'ambito delle esperienze condivise [10].

In questo progetto, vi sono più utenti che attraverso il visore HoloLens 1, vogliono condividere lo stesso tipo di esperienza; per fare questo il primo utente che entra nella stanza effettua una scansione dettagliata e ripetuta del-

l'ambiente in modo tale da avere una mappatura della stanza il più precisa possibile, dopodiché è lui stesso che con il gesto di air tap posiziona l'ologramma contenente l'ancora nel mondo. Solo dopo questo momento, gli altri utenti possono aggiungersi all'esperienza condivisa, previa anche da parte loro, una scansione dettagliata dell'ambiente, altrimenti rischiano di vedere il contenuto olografico spostato rispetto alla realtà, dopodiché l'esperienza condivisa potrà avere inizio.

Da questo progetto di esempio emerge quindi che: nessun oggetto è già presente e visibile nella scena, ma tutto deve essere creato dal primo utente che accede all'applicazione, è necessario effettuare un mappaggio accurato dell'ambiente per avere una posizione precisa degli ologrammi, altrimenti si rischia di vederli spostati rispetto al punto originario, solo dopo aver creato l'ancora gli altri utenti possono unirsi all'esperienza condivisa e il primo utente si occuperà di inviare a tutti quelli che si collegano l'ancora generata.

(a) Posizionamento dell'ancora

(b) Inizio dell'esperienza condivisa

Figura 2.4: Esperienza condivisa realizzata da Microsoft per HoloLens 1

Un'ancora spaziale inoltre, è pensata per mantenere il contenuto olografico fisso in una posizione, quindi se noi vogliamo realizzare un'esperienza in cui, le persone siano in grado di muovere gli ologrammi, per consentire questo movimento dovremo distruggere l'ancora e ricrearla successivamente, altrimenti non saremmo in grado di spostare l'ologramma e consentire l'interazione.

Infine, l'ultimo problema che questo meccanismo presenta è la condivisone delle ancore e la loro persistenza in sessioni diverse. In particolare, i diversi provider mettono a disposizione **solo per i loro devices**, appositi anchor store in cui le ancore possono essere immagazzinate e caricate a successivi avvi dell'applicazione, tali anchor store sono impiegati anche per il caricamento delle ancore a runtime e ciascun provider mette a disposizione specifiche funzioni per poter interagire con l'anchor store del dispositivo, che naturalmente differiscono rispetto a quelle degli altri.

2.4.3 Markers, Targets e Vuforia

Sempre per fare in modo che i diversi dispositivi riescano a vedere il contenuto olografico nella stessa posizione del mondo reale, un'altra tecnologia che può essere utilizzata al posto delle ancore spaziali, sono i markers.

Un marker, come riportato nel paper Augmented Reality technologies, system and application [3], consiste tipicamente in un'immagine che può essere rilevata da una fotocamera, non solo di un dispositivo di Mixed Reality ma anche da un semplice smartphone ed è possibile rilevare, tramite opportuni programmi, la presenza e la posizione di questi marker nell'ambiente.

I markers, di solito, sono costituiti da diverse e semplici forme geometriche di colore nero su uno sfondo bianco racchiuse da un quadrato con il bordo nero, come visibile nella seguente figura 2.5.

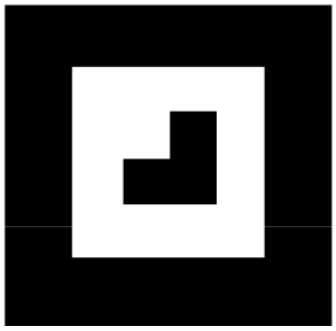


Figura 2.5: Esmpio di target

Tuttavia, non è stato definito un formato standard per la costruzione dei markers, per crearli possono essere utilizzati anche dei colori e vi è inoltre la possibilità di definire dei markers customizzati, a partire da delle vere proprie immagini che soddisfano determinati requisiti di riconoscimento, ad esempio, potremmo scegliere come marker una copertina di un libro che presenta diverse scritte e colori in quanto, tali caratteristiche, lo rendono più facilmente riconoscibile dai dispositivi.

A questi targets può essere associato un contenuto olografico e una volta che il dispositivo avrà riconosciuto il marcatore, sarà per lui possibile visionarlo ed essendo tale contenuto posizionato in relazione al marker, tutti i devices saranno in grado di vederlo nello stesso punto del mondo e da un prospettiva diversa a seconda dei movimenti che verranno effettuati. Possiamo quindi utilizzare questi marcatori come punto di accesso, per i diversi dispositivi, al mondo aumentato.

A questo punto viene da chiedersi come poter programmare un'applicazione che utilizzi questi marcatori, per poter realizzare un esperienza condivisa. Nel

corso degli anni sono state sviluppate diverse tecnologie che consentono già in fase di sviluppo delle applicazioni, di poter manipolare e interagire con questi marcatori, la tecnologia attualmente più utilizzata per questo tipo di operazioni è Vuforia [17].

Vuforia è un Software Development Kit (SDK) per l'Augmented Reality, che utilizza la computer vision per poter riconoscere e tracciare immagini e oggetti 3D nel mondo, in tempo reale.

Gli sviluppatori, possono utilizzare Vuforia direttamente in Unity importando nel progetto l'opportuno package, il quale racchiude già alcuni target di default che possono essere utilizzati, ma vi è anche la possibilità per il programmatore, tramite il developer portal di Vuforia, di poter definire dei propri database di targets con delle proprie immagini, senza dover utilizzare quelli di default.

Nell'ambiente di sviluppo Unity, vengono forniti dei componenti che rappresentano i targets che varranno utilizzati, tali targets, presentano un loro sistema di coordinate e basterà posizionare il contenuto olografico in funzione di tale sistema, per far in modo che questo risulti correttamente posizionato nell'ambiente reale.

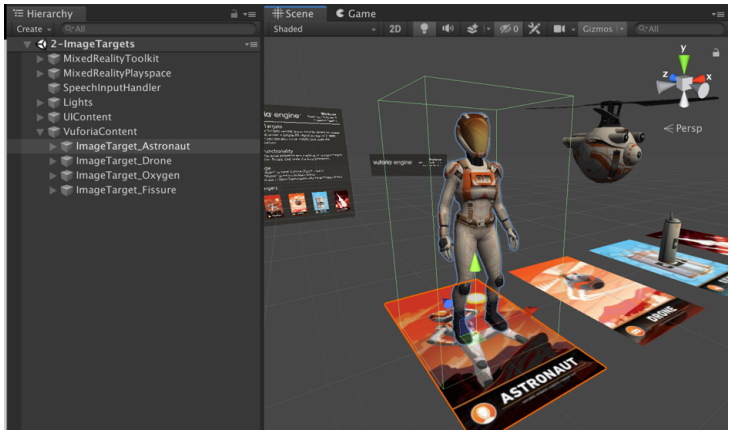


Figura 2.6: Utilizzo dei Target di Vuforia in Unity

Ma non è finita qui, perché tramite l'utilizzo combinato di Vuforia e l'MRTK abbiamo la possibilità di manipolare questi ologrammi e quindi di utilizzare questi targets, per non mostrare solamente un contenuto statico, ma degli ologrammi con cui gli utenti, che entrano a far parte dell'esperienza condivisa, possono interagire.

2.4.4 Vantaggi nell'utilizzo dei targets e perché sono migliori delle ancore

I vantaggi che derivano dall'utilizzo dei markers e di Vuforia sono che: il posizionamento corretto degli oggetti nel mondo reale si ottiene senza troppi sforzi, quasi a costo zero, non è necessario una scansione dell'ambiente per poter posizionare e vedere il contenuto olografico e gli ologrammi possono essere già visibili nella scena senza la necessita di crearli.

Risulta quindi che i markers, oltre ad essere più semplici da utilizzare rispetto alle ancore, consentono anche agli utenti di poter manipolare un oggetto senza troppe modifiche e inoltre, consentono di rappresentare meglio l'idea di un punto di accesso al mondo virtuale, in quanto fisicamente rappresentati, rispetto invece a un'ancora spaziale che non è inizialmente visibile all'utente e con cui esso non può interagire direttamente.

Ma questo sistema presenta anche uno svantaggio: il target che si decide di utilizzare deve essere **sempre inquadrato**, altrimenti si perde il riferimento principale e non è più possibile vedere gli ologrammi.

Capitolo 3

OpenXR

Nel precedente capitolo, sono state illustrate alcune tecnologie che possono essere utilizzate per creare delle applicazioni compatibili con diversi devices. In particolare, l'MRTK e AR Foundation consentono la creazione di un'applicazione, che possa essere portabile su diversi sistemi.

Viene quindi da chiedersi come facciano questi strumenti a far sì che un programma, possa essere eseguito senza troppe modifiche su devices differenti. La risposta è: grazie all'utilizzo di un'API sottostante che li consente di astrarre dal tipo di device che viene utilizzato. Sia l'MRTK che AR Foundation utilizzano l'API OpenXR.

A questo punto, quindi, viene lecito porsi delle altre domande: che cos'è OpenXR? Quali sono gli elementi che costituiscono questa API? Come occorre strutturare il software affinché possa utilizzare correttamente quest'API?

A queste e ad altre domande, si cercherà di dare una risposta in questo capitolo.

3.1 Che cos'è OpenXR

OpenXR, come riportato nelle specifiche [5], è un'Application Program Interface (API), ad alte prestazioni, priva di royality sviluppata da Khronos, la quale si occupa di fornire un accesso nativo alle piattaforme e ai dispositivi di Augmented, Mixed e Virtual Reality. Il termine XR, infatti, sta per eXtended Reality e comprende tutte le realtà elencate precedentemente.

OpenXR dal punto di vista di un programmatore, consiste in un insieme di funzioni che si interfacciano con un runtime, per eseguire operazioni comunemente richieste come ad esempio: l'accesso al controller o allo stato della periferica utilizzata, ottenere le posizioni di tracciamento dei dispositivi di input o della testa dell'utente, inviare frame renderizzati ecc.

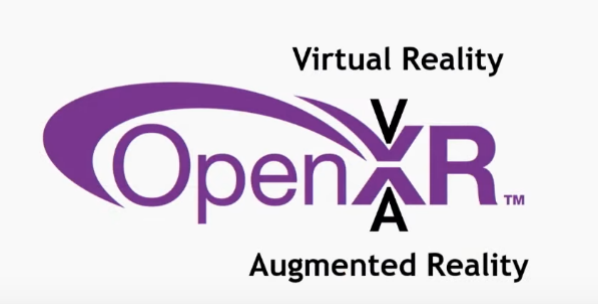


Figura 3.1: Logo di OpenXR

3.1.1 Khrons Group fondatore di OpenXR

Fondato nel 2000, Khronos Group [4] è un consorzio senza scopo di lucro, finanziato dai suoi membri, il cui obbiettivo principale è la creazione di standard aperti esenti da royality per: la grafica 3D, la realtà virtuale e aumentata, il calcolo parallelo, le reti neurali ecc.

Molti degli standard creati da Khronos come ad esmpio: OpenXR, Vulkan, OpenGL, WebGL, OpenVX ecc. sono molto utilizzati.

3.2 Come nasce OpenXR

Nel 2016 furono rilasciati i primi visori per la realtà virtuale: l'Oculus Quest e l'HTC Vive, ciascuno dei quali presentava un proprio SDK con cui interfacciarsi per la programmazione di applicazioni.

Di conseguenza gli sviluppatori per la realtà virtuale, dovevano programmare apposite applicazioni per l'uno o per l'altro device, **senza la possibilità di avere un'interfaccia comune**, nonostante il fatto che questi visori oltre che essere entrambi per la realtà virtuale, presentano funzionalità simili fra loro come: la gestione della pressione di un pulsante su un controller, il monitoraggio della posizione dei controller, della posizione della testa ecc.

3.2.1 Problema della frammentazione

Tutto questo porta in luce dei problemi nello sviluppo delle diverse applicazioni per i diversi visori, che possono essere di seguito elencati:

- Per gli sviluppatori viene richiesto un tempo maggiore di lavoro per la realizzazione di una singola applicazione, in quanto deve essere adattata alle diverse API dei diversi dispositivi, con conseguente **incremento dei costi**.

- Insieme al tempo di lavoro, aumentano anche i tempi di debug e validazione delle applicazioni, con conseguente **aumento dei costi**.

- Il tempo passato da un programmatore per costruire la stessa applicazione per diversi dispositivi, è tempo che il programmatore avrebbe potuto spendere nella realizzazione di applicazioni differenti. Quindi si ha uno **spreco di tempo e risorse**, che potrebbero essere utilizzate per svolgere altri lavori e questo porta a una **conseguente diminuzione di produttività**.

Tutti questi problemi hanno una radice comune che è **la frammentazione** delle diverse API, utilizzate per programmare i diversi sistemi.

Già nei primi sviluppi delle tecnologie di Augmented e Virtual Reality, le API utilizzate iniziavano ad essere un discreto numero, com'è possibile vedere dall'immagine sottostante.

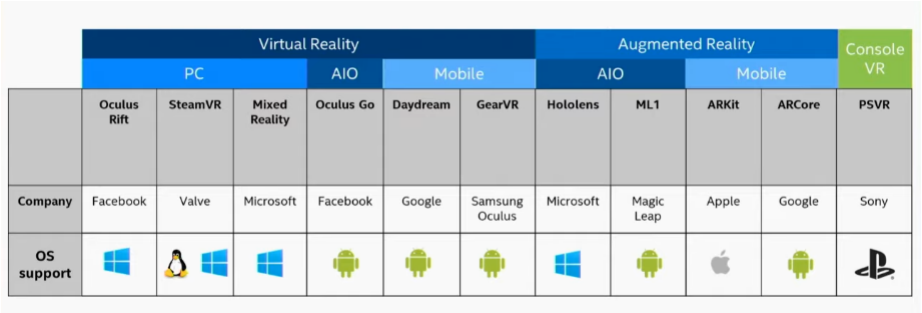


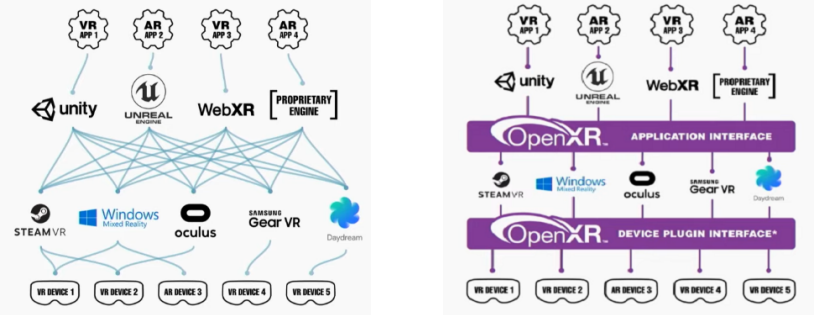
Figura 3.2: Frammentazione del mercato XR

3.3 Obiettivi di OpenXR

Il macro obbiettivo principale di OpenXR, è proprio quello di risolvere il problema della frammentazione, fornendo un'unica API per la programmazione dei diversi dispositivi per le differenti realtà; che gli sviluppatori e i provider possono utilizzare nella realizzazione delle loro applicazioni. Per poter raggiungere questo scopo OpenXR si pone come sotto-obbiettivi di:

- Realizzare un'API che abiliti all'utilizzo di applicazioni sia Virtual che Augmented Reality, unificando funzionalità comuni dei dispositivi VR e AR per consentire lo sviluppo su una vasta proprietà di prodotti e piattaforme.

- Essere a prova di futuro, vale a dire rendere l'API flessibile ed estendibile per cambiamenti futuri, in modo da supportare l'avanzamento nell'innovazione delle tecnologie negli anni avvenire, cercando però di non prevedere tali cambiamenti futuri e modificare preventivamente OpenXR, ma dando la possibilità agli sviluppatori e agli ingegneri di poter lavorare su di essa senza apportare troppe modifiche.
- OpenXR deve essere utilizzata su dei sistemi che presentano un livello critico nell'utilizzo delle performance, quindi essa deve essere diretta anche all'ottimizzazione nell'utilizzo delle risorse.



(a) Prima di OpenXR (b) Dopo OpenXR

Figura 3.3: Problema della frammentazione

OpenXR è progettata per essere un'API estendibile ed è destinata a diventare, a meno che non lo sia già, il nuovo standard per la programmazione dei dispositivi di Augmented, Virtual e Mixed Reality ed è stata già accolta da numerosi providers, alcuni dei quali facenti parte del gruppo di lavoro di Khronos che hanno contribuito a realizzarla.

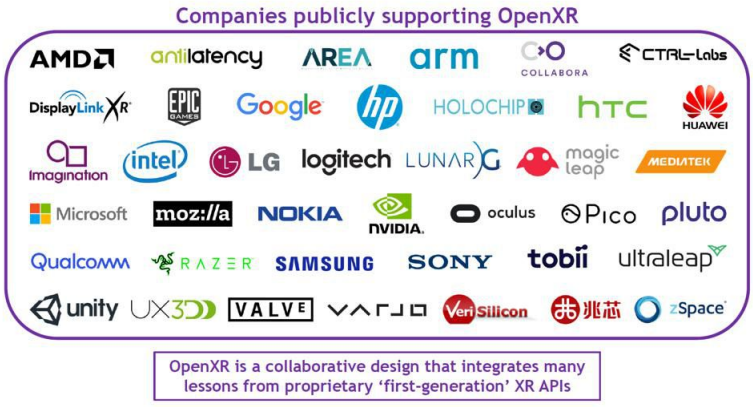


Figura 3.4: Provider che aderiscono a OpenXR

3.4 Caratteristiche di OpenXR

Prima di andare ad analizzare gli elementi di OpenXR e come essi possano essere utilizzati per la creazione di un'applicazione di eXtended Reality, occorre analizzare alcune sue caratteristiche che consentono di meglio descrivere e comprendere questa tecnologia.

3.4.1 Come OpenXR viene vista e implementata dai differenti dispositivi: OpenXR Runtime

Un OpenXR runtime è un software che implementa l'API OpenXR. Possono esserci più runtimes installati in un sistema ma solo uno di essi può essere attivo in un dato momento.

Un runtime è importante perché, l'applicazione di eXtended Reality parlerà direttamente con questo sistema, per gestire le diverse funzionalità dell'applicazione attraverso una coda di eventi. Inoltre, una funzione chiave, svolta dal runtime, che consente ad un'applicazione di poter essere eseguita su diversi devices, consiste nella gestione dei collegamenti fra il dispositivo di input effettivo e le azioni che possono essere svolte dall'utente.

In ogni caso, sono i diversi provider di dispositivi XR, che devono fornire e equipaggiare il dispositivo utilizzato con tale sistema software.

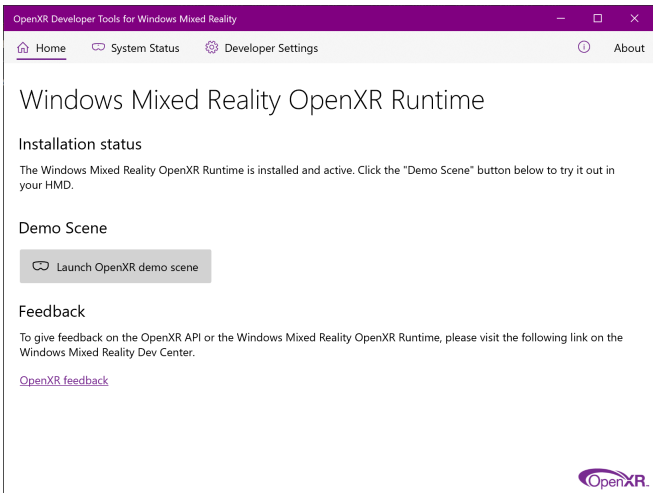


Figura 3.5: Esempio runtime di Microsoft

3.4.2 Un'API estendibile grazie alle Extensions

Nelle precedenti sessioni si è detto che OpenXR è un'API estendibile, a cui possono essere aggiunte nuove funzionalità. Tali funzionalità aggiuntive,

vengono implementate attraverso delle estensioni che espongono nuovi funzioni di OpenXR o modificano il comportamento di quelle precedenti. Esistono diversi tipi di estensioni, che è possibile vedere nella figura 3.6 e che possono essere di seguito approfondite:

- Il Core Standard: contiene le estensioni fondamentali utili a tutte le diverse applicazioni per il corretto funzionamento.
- KHR Extensions: sono delle estensioni sviluppate direttamente da Khronos, che definiscono funzionalità che ci si aspetta siano supportate dalla maggior parte dei runtimes.
- le EXT Extensions: sono estensioni sviluppate da diversi provider di dispositivi, che hanno collaborato per coprire tramite OpenXR funzionalità comuni ai loro devices.
- Vendor Extesnion: sono estensioni definite da un solo venditore per ricoprire una funzionalità specifica del device offerto.

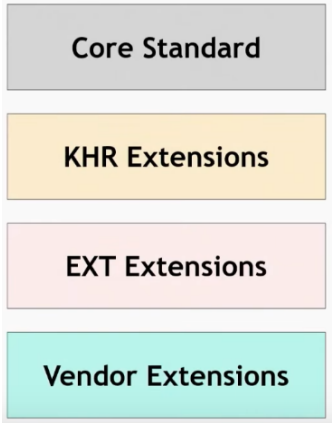


Figura 3.6: Estensioni di OpenXR

3.4.3 Concetto di layered API

OpenXR è progettata per essere una layered API, il che significa che gli utenti possono inserire degli API layers fra l'applicazione ed il runtime. Tali livelli aggiuntivi, servono per aggiungere nuove funzionalità che altrimenti non si avrebbero o che dovrebbero essere eseguite senza il layer specifico.

Nel caso più semplice un layer chiama funzioni del layer sottostante passandogli gli stessi argomenti, in altri casi invece, già il layer che viene chiamato implementa la funzionalità richiesta e non c'è bisogno di rivolgersi ad altri.

Tale meccanismo, consente l'implementazione di una funzionalità di "function shimming" [1] o "intercept", che è destinata a sostituire metodi più informali di "hooking" [2] delle chiamate a API.

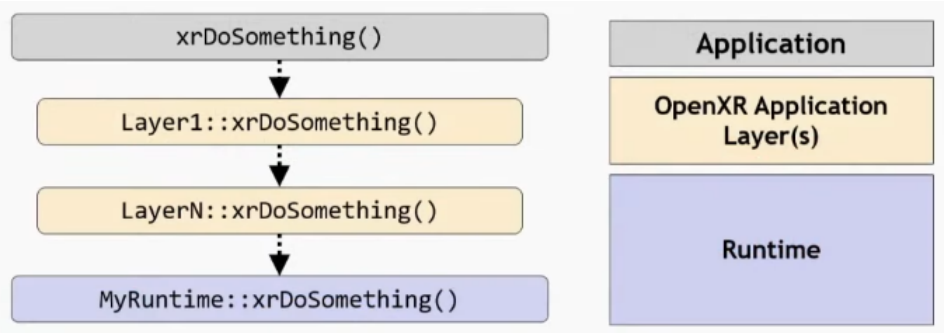


Figura 3.7: Funzionamento degli API layers

Gli API leyers, implementano diverse funzioni di OpenXR e le funzionalità che offrono vengono esposte attraverso le estensioni.

Quindi, per esempio, supponiamo che un'API layer per il monitoraggio delle mani voglia esporre una nuova funzionalità, che consenta a un'applicazione di effettuare il tracking completo di entrambe le mani, con anche le articolazioni delle dita. Siccome le nuove funzionalità devono essere esposte tramite un'estensione, il venditore, che ha pensato di implementare questa feature per il suo device, avrà creato un'estensione chiamata `XR_ACME_full_art_hand_tracking` che espone nell'API layer e implementa la funzionalità richiesta.

L'applicazione a questo punto, che desidera ottenere questo tipo di monitoraggio delle mani, può aggiungere l'API layer a quelli già presenti e richiedere tale funzionalità attraverso l'estensione.

3.4.4 Threading Behaviour e Multiprocessing Behaviour

L'API OpenXR, ha lo scopo di fornire prestazioni scalabili se utilizzata su più thread host. Per questo motivo, tutte le funzioni dell'API, supportano il fatto di poter essere chiamate con-correntemente da più threads.

Tuttavia certi parametri o componenti di parametri utilizzati dalle funzioni stesse, possono essere definiti in modo da essere sincronizzati dall'esterno, il che significa che sarà compito del chiamante garantire che non più di un thread possa utilizzare quel parametro in un determinato momento.

[1] shim = aggiungere spessore
[2] hook = agganciare

Infine, l'API OpenXR, non riconosce ne richiede il supporto esplicito di più processi che utilizzino il runtime contemporaneamente, ma non impedisce a un runtime di definire tale supporto.

3.4.5 Rappresentazione degli oggetti tramite gli Handles

Gli oggetti allocati dal runtime per conto dell'applicazione sono rappresentati da degli handles.

Gli handles, altro non sono che degli identificatori opachi per gli oggetti, la cui vita è controllata dall'applicazione tramite dei metodi di creazione e distruzione. Se non è altrimenti specificato una funzione di creazione di un handle, restituisce un handle nuovo e univoco, inoltre, sempre se non è specificato diversamente, gli handle sono implicitamente distrutti quando il loro parent viene distrutto a sua volta.

Gli handles risultano quindi, essere organizzati in una struttura gerarchica ad albero, dove la distruzione di un nodo padre comporta anche la distruzione di tutti i suoi nodi figli. Ad esempio, nella seguente figura 3.8 possiamo vedere l'organizzazione gerarchica di tre tipi di handles di OpenXR: Instance, Session e ActionSet, ora se noi distruggessimo l'handle Istance verrebbero distrutti anche ActionSet e Session, se invece cancellassimo solamente l'handle Session, Instance e ActionSet non verrebbero distrutti.

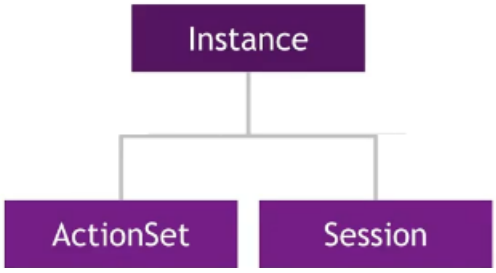


Figura 3.8: Organizzazione gerarchica degli Handles

3.4.6 Return codes delle funzioni

Il Core dell'API, non è progettato per catturare degli utilizzi incorretti e per essere in grado di sapere se una funzione è andata a buon fine oppure no, vengono utilizzati dei codici di ritorno.

I codici, sono definiti all'interno di una enum chiamata `XrResult`, la cui definizione è contenuta nelle specifiche di OpenXR alla voce Return Codes [5].

Tutte le funzioni restituiscono un valore contenuto in questa enum per indicare il il loro stato, il loro successo o il loro fallimento.

I return codes, infatti, si distinguono in due categorie:

- Successful completation codes: sono i codici restituiti quando una funzione, ha la necessità di comunicare un successo o informazioni di stato. Tutti i successful completation code, sono identificati da dei valori non negativi.
- Runtime error codes: sono i codici restituiti quando una funzione ha la necessità di comunicare un fallimento, il quale può essere rilevato solamente a run-time, tutti questi codici sono rappresentati da dei valori negativi.

3.4.7 Come avviene la comunicazione fra un runtime e l'applicazione

L'applicazione e il runtime comunicano attraverso lo scambio di eventi. Tali eventi, sono inseriti in una coda che l'applicazione si occupa di leggere con regolarità attraverso il meccanismo di "polling". In particolare, il runtime riempie la coda con gli eventi generati e l'applicazione si occupa di richiamare con una certa frequenza, la funzione `xrPollEvent`, che nel caso vi sia un evento disponibile, lo restituisce rimuovendolo dalla coda.

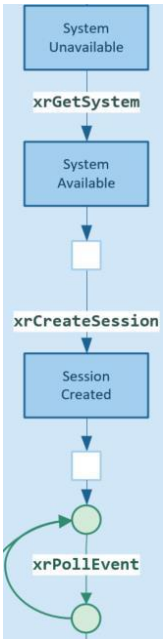


Figura 3.9: Chiamata alla funzione xrPollEvent nel ciclo di vita di una sessione

3.4.8 Sistema di coordinate utilizzato

L'API utilizza un sistema di riferimento cartesiano right-handed, il che significa che: l'asse X cresce verso destra, l'asse Y verso l'altro, mentre l'asse Z punta verso l'interno, come mostrato in figura 3.10.

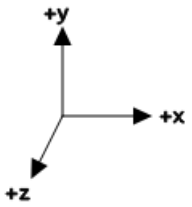


Figura 3.10: Sistema di coordinate right-handed

OpenXR utilizza vettori floating point 2D, 3D e 4D per descrivere punti e direzioni in uno spazio, in particolare si ha che, i tipi di dato: `XrVector2f`, `XrVector3f` e `XrVector4f` vengono utilizzati per esprimere delle distanze in metri attraverso l'utilizzo di 2, 3, o 4 variabili. La rotazione di un oggetto, invece, viene espressa con un `xrQuaternionf`.

Infine è presente anche un quarto tipo di dato `XrPosef`, che consente di definire una posa costituita da una rotazione e una posizione espressa con 3 coordinate, la cui definizione è possibile trovare nelle specifiche alla voce Coordinate System [5].

Capitolo 4

Architettura, componenti e funzionamento di un'applicazione OpenXR

Passiamo ora all'analisi di un applicazione OpenXR, descrivendo quella che è la sua architettura e gli elementi base che vengono utilizzati per la sua realizzazione, consentendone il suo funzionamento.

Verrà quindi chiarito il concetto di: Istance, Session, System, Form Factor e Action di un applicazione OpenXR; andando poi a concludere con l'analisi del main loop di un'applicazione nativa, analizzando i diversi passaggi principali che il sistema esegue.

La maggior parte dei concetti esposti in questo capitolo, hanno come riferimento: le specifiche di OpenXR [5], la conferenza SIGGRAPH del 2019, tenuta da Khronos per presentare OpenXR [6], la presentazione di Collabora, tenuta dal Principal Software Engineer di OpenXR Rayan A. Pavlik [1] e la presentazione di OpenXR e del suo utilizzo, tenuta da Oculus nel 2019 [12].

4.1 Struttura e Architettura di un'applicazione OpenXR

Lo schema in figura 4.1, mostra l'architettura di un'applicazione OpenXR e come il flusso di informazioni venga gestito all'interno del sistema.

In particolare, l'applicazione XR comunica direttamente con l'API, la quale come detto nel capitolo precedente, potrebbe essere costituita da più livelli e si occupa di intrattenere i rapporti sia con il runtime del venditore che con l'applicazione stessa, consentendo il dialogo fra questi due. Dei livelli presentati ve ne è uno opzionale, indicato con il nome: "OpenXR Device plugin Exsten-

sion". Questo livello, serve quando un venditore vuole esporre determinate funzionalità del suo device, attraverso la realizzazione di un plugin che supporti OpenXR. A questo punto, i programmatori interessati, possono decidere di utilizzare tale plugin in quanto contenente funzionalità utili al loro progetto e pertanto decidono di includerlo.

Infine è presente un livello "Vendor-supplied Device Driver", che consiste appunto in un driver messo a disposizione dal venditore, la cui funzione principale è quella di dialogare con il device utilizzato e se presente anche con l' OpenXR Device Plugin, altrimenti esso si rivolge direttamente al runtime del sistema.

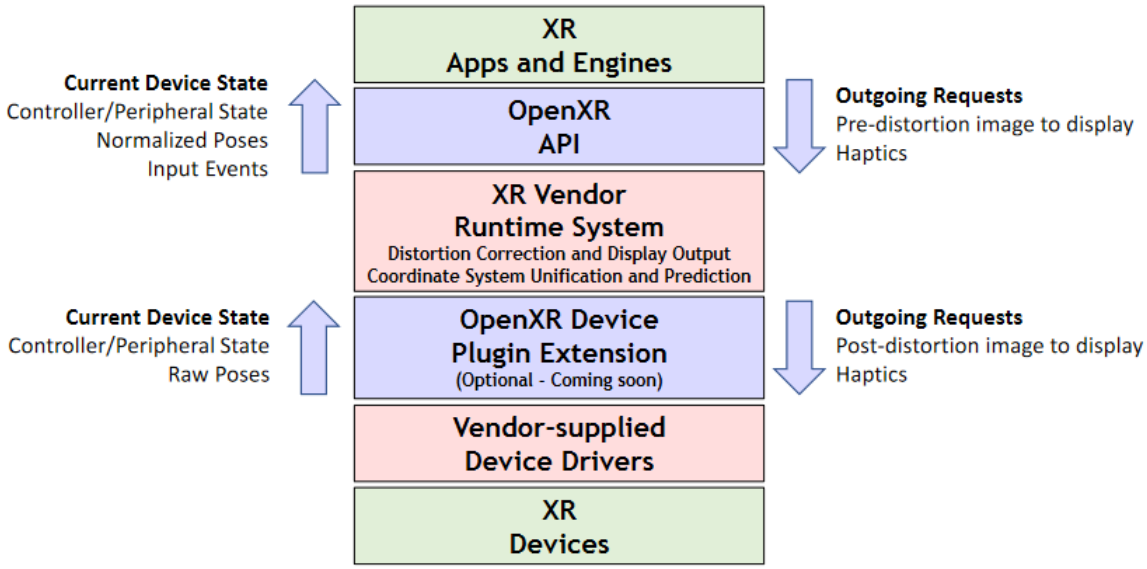


Figura 4.1: Architettura di un'applicazione XR

Tramite l'OpenXR API, il dispositivo invia all'applicazione informazioni come: il suo stato, lo stato dei controller, il monitoraggio delle posizioni degli oggetti, gli eventi di input ecc.

Queste informazioni vengono ricevute e elaborate dall'applicazione, la quale in cambio, restituisce l'immagine che deve essere processata dall'XR Device, per far si che essa sia resa visibile all'utente.

L'API quindi, si interpone fra l'applicazione e il dispositivo utilizzato, consentendo il dialogo fra queste due entità.

Al contrario di quello che si potrebbe pensare, OpenXR non rimpiazza i Runtime Systems, forniti dai diversi venditori, ma consente a ogni providers di esporre delle API per accedere alle loro funzionalità.

Dal discorso precedente, è emerso che si possono avere due tipi diversi di strutture di un'applicazione OpenXR: quelle che prevedono l'utilizzo di un

OpenXR plugin Extension e quelle che invece non lo prevedono. La possibilità per i runtimes di supportare specifici plugin, per i diversi devices, fu effettuata principalmente per due ragioni:

- La sicurezza, in quanto, alcune piattaforme presentano delle notevoli restrizioni dal punto di vista della sicurezza, che rendono difficile il supporto arbitrario di dispositivi hardware.

- Semplicità di implementazione delle applicazioni che supportano un solo tipo di hardware, in quanto, in questi casi non ha senso includere tutte le diverse funzionalità, ma solamente quelle dell'hardware specifico.

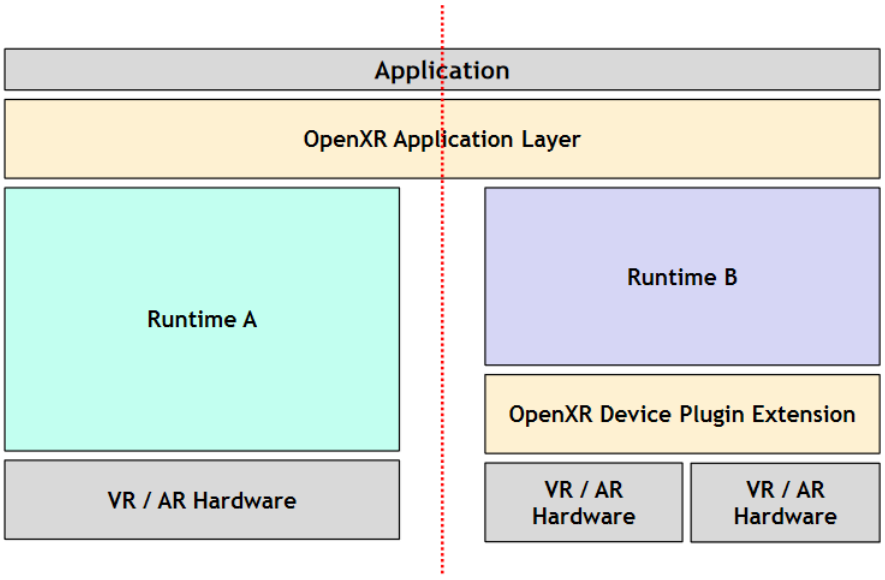


Figura 4.2: Struttura di un'applicazione XR

4.1.1 OpenXR Loader: che cos'è e quando utilizzarlo

Parlando dell'architettura di un'appliczione OpenXR, bisogna introdurre anche il concetto di OpenXR Loader. L'OpenXR Loader è un componente separato, in grado di supportare runtimes multipli in un unico sistema. Il Loader è il meccanismo mediante il quale un'applicazione sarà in grado di parlare a uno specifico runtime, attraverso apposite funzioni e meccanismi, che consentono all'XR app di determinare con quale dei runtimes comunicare.

Va precisato che sistemi che possiedono un unico runtime non hanno la necessita di utilizzare un OpenXR Loader.

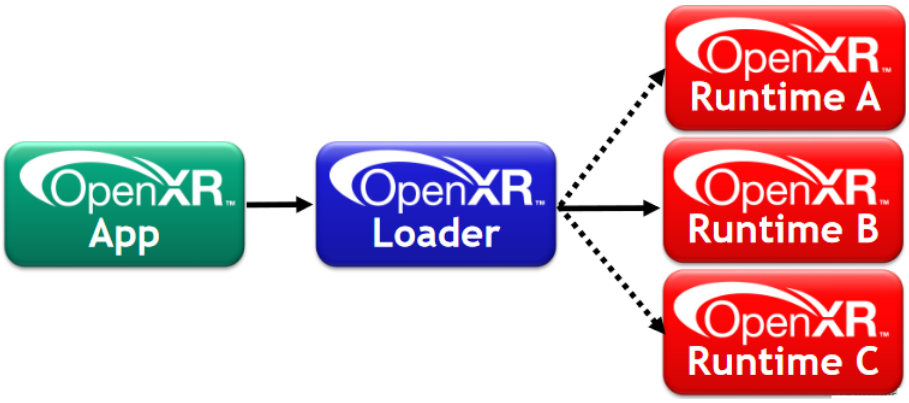


Figura 4.3: Funzionamente di un OpenXR Loader

4.2 Il concetto di istanza di un'applicazione

Un `XrInstance` è un oggetto che consente a un'applicazione OpenXR, di comunicare con un runtime. L'istanza di un'applicazione OpenXR, consiste sostanzialmente, nella rappresentazione del runtime sottostante ed individua il modo con cui l'applicazione dialoga con il runtime.

E' possibile creare un'istanza attraverso la chiamata alla funzione `xrCreateInstance`, la quale restituisce un handle per l'oggetto `XrInstance` creato, che si occupa di salvare e tenere traccia dello stato dell'applicazione.

L'insieme degli API layers e delle estensioni da abilitare, deve essere specificato al momento della creazione dell'istanza, se una specifica API layer non può essere trovata, l'`XrInstance` non potrà essere creata e verrà restituito un codice di errore.

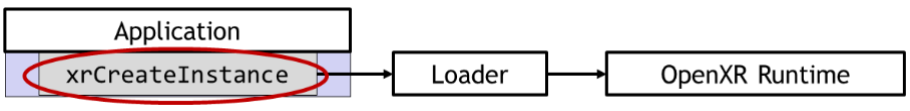


Figura 4.4: Creazione di un'istanza

Un'istanza, dopo essere stata creata può essere distrutta, attraverso la chiamata alla funzione `xrDestroyInstance` e si ricorda che in questo modo, non solo verrà distrutta l'istanza ma anche tutti i suoi figli, come spiegato nella sezione 3.4.5.

4.3 Come i dispositivi sono visti dall'applicazione: System e Form Factor

Dopo la creazione di un'istanza, si può passare alla definizione di un `xrSystem`.

Un sistema altro non è che la rappresentazione del dispositivo o dei dispositivi su cui l'applicazione verrà eseguita.

Il primo passo che un'applicazione deve fare per selezionare un sistema, è specificare il Form Factor desiderato.

Un Form Factor, descrive come i diversi display si muoveranno nell'ambiente, a seconda che siano posizionati sulla testa dell'utente o si tratti di dispositivi mobile e come l'utente interagirà con l'esperienza XR.

Attualmente, sono supportati solamente due tipi diversi di Form Factor: gli head monted display, vale a dire i visori per le diverse realtà, che vengono posizionati sulla testa dell'utente e i dispositivi mobile.

Figura 4.5: Tipi di Form Factor

E' importante notare che: sebbene un'applicazione XR possa effettuare un rendering che li consente di passare da un Form Factor ad un altro, la sua interfaccia utente probabilmente non è in grado di farlo, essa verrà sempre scritta in modo da soddisfare uno specifico Form Factor (un dispositivo mobile con uno schermo 2D o un visore che consente una visualizzazione 3D) quindi, per poter essere utilizzata con differenti Form Factor, richiederà specifici adattamenti.

Tornando ora al concetto di System, quello che effettua l'API è separare quello che è il sistema fisico, costituito dagli xrDevices, rispetto a quelli che sono gli oggetti logici che rappresentano gli xrDevices e con cui l'applicazione dialoga direttamente.

In sostanza un sistema, rappresenta una collezione di devices correlati fra loro nel runtime, tipicamente costituiti da diversi elementi hardware che collaborano insieme per realizzare una singola esperienza XR.

Per ottenere il riferimento al sistema, l'applicazione richiama la funzione `xrGerSystem`, la quale restituisce un `xrSystemId`, il quale a sua volta, rappresenta i diversi dispositivi che il runtime utilizzerà per supportare il Form Factor specificato.

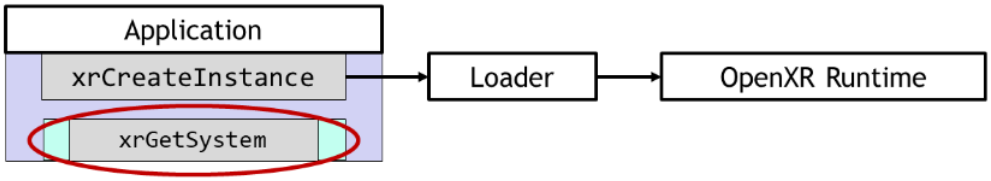


Figura 4.6: Chiamata alla funzione xrGetSystem

4.4 Session e comunicazione fra runtime e applicazione

Una sessione rappresenta l'intento da parte di un'applicazione di voler mostrare il contenuto XR all'utente; essa possiede un proprio stato e un proprio ciclo di vita, in particolare, un'applicazione crea una sessione tramite il metodo `xrCreateSession` e può distruggerla tramite il metodo `xrDestroySession`.

4.4.1 Ciclo di vita di una sessione

Quando una sessione viene creata, essa entra in uno stato di idle ed è pronta per essere attivata quando il runtime lo consente.

I cambiamenti di stato della sessione, verranno notificati dal runtime e l'applicazione si dovrà occupare di gestirli, essa infatti, dopo aver creato la sessione, imposterà un **xrPoolEvent loop**, che li consentirà di ricevere gli eventi generati dal runtime.

Quando un runtime vuole notificare all'applicazione che è pronto per poter mostrare il contenuto XR, cambia lo stato della sessione da idle a ready, a questo punto l'applicazione che ha ricevuto l'evento di notifica, si occupa di: per prima cosa, far partire la sessione attraverso la chiamata del metodo `xrBeginSession`, che richiede come parametro la sessione precedentemente creata e come seconda cosa, di far partire il **frame loop** per la generazione delle immagini.

Una volta che il runtime risulta essere sincronizzato con il frame loop dell'applicazione, ed è pronto per mostrare i frame che li vengono inviati, la sessione può passare allo stato synchronized e fin tanto che una sessione si trova in questo stato, il frame loop continuerà la sua esecuzione.

Dopo che una sessione è entrata nello stato synchronized, un'applicazione può decidere di terminarla, notificando tale volontà al runtime attraverso il metodo `xrRequestExitSession`. Il runtime a questo punto si occuperà di stoppare le sessione, passando il suo stato a quello di stop e notificando tale cambiamento all'applicazione, la quale potrà richiamare il metodo `xrEndSession`, che terminerà definitivamente la sessione, facendola passare prima allo stato di idle e poi allo stato di exit.

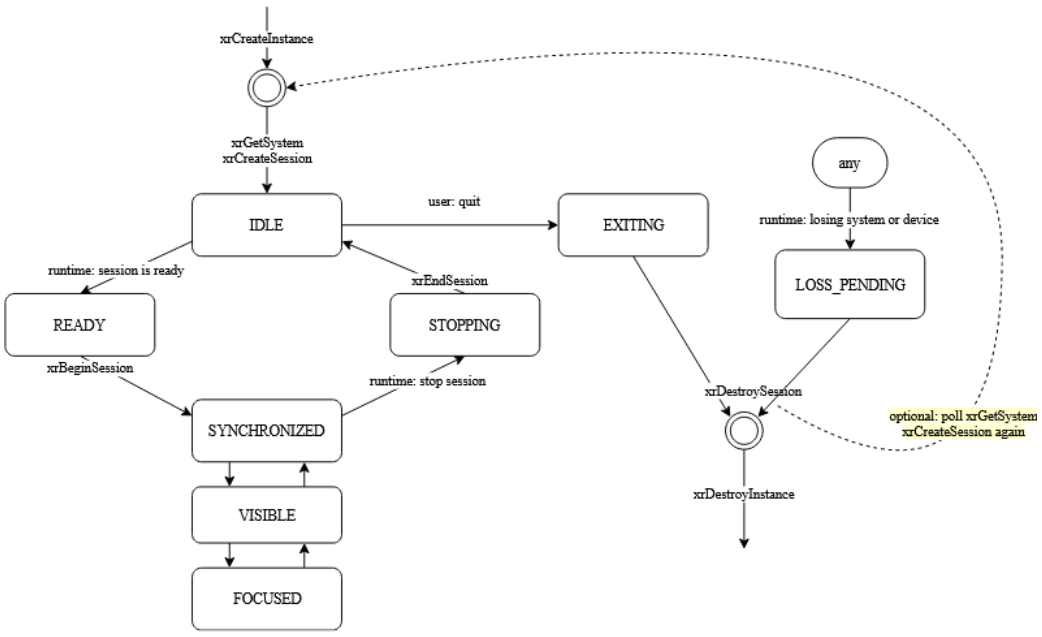


Figura 4.7: Diagramma degli stati di una sessione

Una cosa importante da sottolineare è che: **solamente le sessioni in esecuzione, possono ottenere il focus per ricevere XR input**, quindi, una sessione terminata non sarà in grado di ricevere gli input dell'utente.

4.4.2 Gestione del frame loop

Precedentemente abbiamo detto che, mentre una sessione è in esecuzione, l'applicazione si deve occupare di eseguire il frame loop, il quale altro non è che un ciclo, in cui vengono chiamate continuamente e in sequenza le funzioni:

- `xrWaitFrame`, che si occupa di stoppare il frame loop, in modo da sincronizzare l'invio dei frame da parte dell'applicazione, con la ricezione di questi da parte del display.
- `xrBeginFrame`, che si occupa di iniziare il rendering del frame.
- `xrEndFrame` che invece, si occupa di finire il rendering del frame.

4.4.3 Analisi degli stati di una sessione

Tutti gli stati che possono essere assunti da una sessione è possibile visualizzarli alla figura 4.7. Tali stati sono definiti nell'API attraverso una enum `XrSessionState`, i cui principali sono:

- `XR_SESSION_STATE_IDLE`: indica che secondo il runtime la sessione si trova nello stato di idle. L'applicazione quando la sessione si trova in questo stato, dovrebbe minimizzare il consumo di risorse, ma continuare a chiamare la funzione `xrPoolEvent`, per verificare se vi sono dei nuovi eventi.
- `XR_SESSION_STATE_READY`: questo stato indica all'applicazione di preparare le risorse per il rendering, iniziare la sessione e sincronizzare il frame loop con il runtime.
- `XR_SESSION_STATE_SYNCHRONIZED`: indica che l'applicazione ha sincronizzato il suo frame loop con il runtime, ma che i frame non sono ancora visibili all'utente.
- `XR_SESSION_STATE_VISIBLE`: indica che l'applicazione ha sincronizzato il suo frame loop con il runtime e che i frame risultano ora essere visibili all'utente, tuttavia la sessione, che si torva in questo stato, non è ancora pronta a ricevere XR input.
- `XR_SESSION_STATE_FOCUSED`: indica che un'applicazione ha sincronizzato il proprio frame loop con il runtime, che i frame sono visibili all'utente e che è pronta a ricevere gli input. Nel caso vi siano più sessioni, il runtime dovrebbe concedere il focus, solo a una sessione per volta e per un tempo predefinito.
- `XR_SESSION_STATE_STOPPING`: indica che il runtime ha ricevuto la richiesta dell'applicazione di terminare la sessione. Una volta in questo stato la funzione dovrebbe chiamare la funzione `xrEndSession`.
- `XR_SESSION_STATE_EXITING`: indica all'applicazione che la sessione è terminata.

4.5 Spaces e sistemi di riferimento

Sia per quanto riguarda i dispositivi di Augmented Reality che per i dispositivi di Virtual Reality, le applicazioni hanno la necessità di mappare e di conoscere la posizione degli oggetti virtuali in relazione al mondo reale, in cui essi sono stati renderizzati.

Gli Spaces, rappresentati dagli handles `XrSpace`, consentono a un'applicazione di creare e specificare esplicitamente i frame of reference, rispetto ai quali i devices decideranno di tracciare il mondo reale e di determinare come questi sistemi di riferimento, si sposteranno l'uno rispetto all'altro.

Ogni qual volta un'applicazione richiama una funzione che restituisce delle coordinate, essa fornisce un `XrSpace`, per specificare il sistema di riferimento nel quale queste coordinate verranno espresse. Similmente, quando vengono fornite delle coordinate a una funzione, l'applicazione specifica quale `XrSpace`, deve essere utilizzato per interpretare queste coordinate.

4.5.1 Rappresentazione del mondo tramite i Reference Spaces

OpenXR definisce tre tipi diversi di reference spaces, che possono essere utilizzati: VIEW, LOCAL e STAGE. Ogni reference spaces, ha un significato ben chiaro e definito, che stabilisce dove verrà posizionata l'origine e come gli assi saranno orientati.

I diversi tipi di `Spaces` sono mantenuti all'interno dell'API da una enum `XrReferenceSpaceType`, la cui definizione è possibile trovare nelle specifiche, alla sezione Spaces [5].

- VIEW: si tratta di uno space che posiziona l'origine degli assi, in base alla prima posizione della vista dell'utente, tenuta nell'utilizzo dell'applicazione. In particolare, una volta definita l'origine, si avrà che: l'asse X crescerà verso destra, l'asse Y verso l'altro e l'asse Z sarà negativo difronte all'utente e positivo dietro di lui.

  Lo space di tipo VIEW, è molto utile per effettuare un raycast, quindi per proiettare dei raggi utili all'applicazione a partire dalla vista dell'utente o per mostrare del contenuto che si vuole mantenere nella vista dell'utente stesso. Il contenuto renderizzato in questo space, rimarrà in una posizione fissa.

- LOCAL: questo space, stabilisce un'origine fissa, allineata con la gravità e il sistema di riferimento è orientato come nello space di tipo VIEW. Il sistema, risulta essere fisso sia nell'origine che nel suo orientamento e

il runtime può decidere la posizione iniziale al lancio dell'applicazione, tipicamente viene scelta la posizione della testa dell'utente.

Uno space di tipo LOCAL è utile quando, un runtime deve mostrare del contenuto seated-scale, quindi secondo una scala che prevede l'utente in una posizione seduta, che non necessita di un riferimento in base al pavimento.

- STAGE: uno STAGE refernce space, specifica uno spazio piatto e rettangolare sul quale è possibile camminare. L'origine di questo space si trova nel centro del rettangolo, con l'asse Y che cresce verso l'alto mentre l'asse X e l'asse Z, sono allineati con gli angoli del rettangolo.

  Questo space, è utile quando occorre definire delle applicazioni, che necessitano di una standing scale, quindi una scala che preveda l'utente in piedi durante l'esperienza.

Nei capitoli precedenti, quando si è parlato di mapping spaziale, si è anche detto che esistono dispositivi in grado di ampliare e raffinare la loro conoscenza del mondo, a mano a mano che il dispositivo rimane in funzione e si muove nell'ambiente.

Questi sistemi, che hanno una conoscenza dinamica del mondo, possono tenere traccia degli spaces in modo indipendente fra loro, questo comporta, che anche se un LOCAL space e uno STAGE space mappano la loro origine in una posizione statica del mondo, un runtime con un sistema di monitoraggio continuo come HoloLens, potrebbe introdurre durante il suo mappaggio, degli aggiustamenti ai punti di origine di questi spaces, in modo da continuare a mantenerli in un punto specifico del mondo reale. A tal proposito, la funzione `XrEventDataReferenceSpaceChangePending` prevede che quando uno space, modifica la sua origine, venga inviato un evento all'applicazione che la avverte di questo cambiamento.

4.5.2 Monitoraggio delle posizioni nel mondo tramite gli Action Spaces

Oltre ai reference spaces, che ci consentono di definire dei sistemi di riferimento per l'ambiente circostante, OpenXR mette anche a disposizione dei runtimes degli altri indipendently-tracked spaces, definiti con il termine di Action Space e indicati dal tipo di dato `XrActionSpace`.

Questi space, ci consentono di monitorare delle posizioni di oggetti in modo indipendente dagli altri sistemi di riferimento, essi di solito vengono utilizzati per monitorare la posizione dei controller o delle mani dell'utente.

Sostanzialmente a ciascun controller viene attribuito un `XrActionSpace` e per individuare la posizione di tale controller rispetto nell'ambiente circostante, si va a riprendere la posizione attuale dell'Action Space associato a tale controller, rispetto a quella di un altro space, che rappresenta il mondo intorno all'utente.

4.5.3 Come individuare una posizione in uno spazio

La cosa importante da notare, nel discorso precedente è che: **non riusciremo mai a trovare la posa di uno space, quello che invece andremo a cercare e utilizzare, sarà sempre la posizione di uno space rispetto ad un altro**.

Per mantenere traccia della posizione di uno space rispetto a quella di un altro, si utilizza la funzione `XrLocateSpace`, la quale prende in ingresso: il tempo passato in cui si vuole sapere la posizione dello spazio, lo space di riferimento, lo space di cui si vuole sapere la posizione rispetto a quello di riferimento e un puntatore dove poter salvare tale posizione.

Questa funzione permette quindi, di sapere nel momento specificato come parametro, la posizione dello space rispetto all'altro. Per capire meglio questo concetto, procediamo con un esempio: supponiamo di aver assegnato un `Action Space` alla mano destra dell'utente e di aver utilizzato uno space di tipo STAGE per descrivere la scena, ora vogliamo far si che al gesto di air tap dell'utente venga posizionato un ologramma nel punto in cui tale gesto è avvenuto. Quindi per fare questo, salviamo il tempo in cui il gesto di air tap è avvenuto, dopodiché passiamo: questo tempo, l'`XrActionSpace` della mano dell'utente e l'`XrSpace` che descrive l'ambiente, alla funzione `XrLocateSpace`.

A questo punto la funzione ci restituirà, nel puntatore passato come parametro, la posizione della mano quando è avvenuto l'air tap dell'utente, rispetto al sistema di riferimento specificato. Infine, utilizziamo tale posizione per posizionare il nostro ologramma.

La particolarità di OpenXR nel tracciamento è proprio questa, in altre XR API di solito, la posizione di un oggetto, viene riportata in base a un presunto space global sottostante, in quest'API invece, si è prestato attenzione a non definire un global space, in quanto **esso non può essere applicato a tutti i sistemi**. Alcuni sistemi non supportano uno STAGE space, altri non supportano un LOCAL space e altri ancora, come abbiamo visto, possono addirittura cambiare l'origine degli assi dinamicamente. Per questo motivo, **le pose degli oggetti vengono definite sempre, come la relazione fra due spazi**.

4.5.4 Il concetto di tempo in OpenXR

Dalla precedente analisi, emerge che il concetto di tempo in OpenXR è fondamentale, per riuscire a riconoscere una posizione di uno spazio in un dato istante.

Questo tempo all'interno dell'API è rappresentato da un intero a 64 bit, ed è espresso in nanosecondi, esso aumenta sempre con una velocità costante e non è influenzato dai cambiamenti del clock, dai fusi orari, dalla zona ecc.

`XrTime` è la variabile che rappresenta il tempo in OpenXR, indica il tempo trascorso, in nanosecondi, da un epoca scelta dal runtime. `XrTime` non rappresenta quindi una durata, ma bensì un tempo trascorso, rispetto a una specifica epoca.

4.6 Gestione degli input dei diversi dispositivi

Vi sono diversi modi per gestire l'input dei dispositivi, il primo che potrebbe venire in mente consiste nell'associare ad ogni specifico pulsante del controller o gesto delle mani un'azione specifica, ma questo sarebbe incredibilmente difficile da gestire, soprattutto se si vuole realizzare qualcosa che sia conforme ai diversi dispositivi.

Per questo motivo, OpenXR, ha fatto sì che l'input di un dispositivo attraversi un layer di astrazione, costruito attorno alla operazioni di input, questo consente ai programmatori, di definire delle operazioni di input basate sul risultato che si vuole ottenere come: grab, jamp, teleport ecc. dando quindi la possibilità, di separare il pulsante che viene premuto o il gesto che viene compiuto, dall'azione che si vuole compiere.

Riassumendo, in OpenXR non verranno definiti i pulsanti o i gesti da compiere, ma le azioni che verranno compiute, tuttavia, per poter effettivamente compiere queste azioni occorre che esse vengano collegate a pulsanti e gesti effettivi, come accade questo in OpenXR? Lo sviluppatore avrà il compito di collegare queste azioni differenti, a pulsanti o gesti tramite un interaction profile.

4.6.1 Che cos'è e a cosa serve un Interaction Profile

Un interaction profile, è sostanzialmente una descrizione di un controller fisico, il runtime avrà il compito di mappare i bottoni fisici alle azioni, in base ai suggerimenti che sono stati dati dai programmatori proprio attraverso gli interaction profiles. Il runtime però, non è forzato ne obbligato, a seguire i collegamenti che i programmatori hanno specificato, perché ad esempio, potrebbe essere data la possibilità anche agli utenti finali, di decidere a quali pulsanti o

gesti far corrispondere certe azioni, aumentano così anche l'accessibilità delle applicazioni.

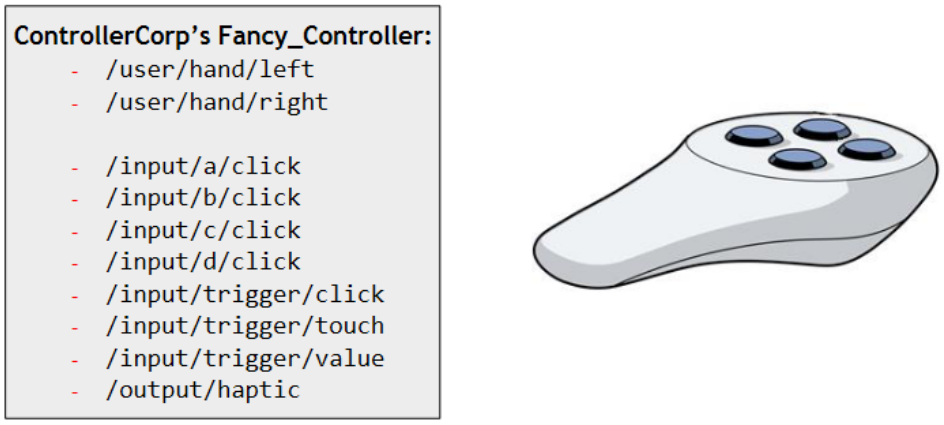


Figura 4.8: esmpio di un porfilo di interazione

Khronos ha definito un Simple Controller Interaction Profile, che fornisce una serie di strumenti di base come: pose, bottoni, supporto degli haptic ecc. ad applicazioni che ne hanno bisogno, non vi è nessun hardware particolare associato a questo profilo, in quanto esso è pensato per riferirsi a un controller generico; sarà quindi compito del runtime effettuare i collegamenti opportuni con il dispositivo utilizzato.

Anche i provider dei diversi dispositivi, possono fornire un profilo di interazione specifico per i loro devices, attualmente i profili disponibili sono:

- Google Daydream Controller Profile.
- HTC Vive Controller Profile.
- HTC Vive Pro Profile.
- Microsoft Mixed Reality Motion Controller Profile.
- Microsoft Xbox Controller Profile.
- Oculus Go Controller Profile.
- Oculus Touch Controller Profile.
- Valve Index Controller Profile.

4.6.2 Come un Interaction Profile ci consente di astrarre l'azione dall'input del dispositivo utilizzato

Ora quindi, la domanda sorge spontanea: come funziona un interaction profile? Per capirlo procediamo con un esempio: supponiamo di avere a disposizione un controller con diversi pulsanti e di voler far sì che, quando l'utente preme il pulsante A, si attiva l'azione di teletrasporto, per fare questo, il pulsante che può essere premuto descriverà un profilo di interazione simile a un path URL: `/user/hand/left/input/a/click` e a tale path il programmatore provvederà ad associare il collegamento all'azione Teleport. A questo punto, quando l'utente premerà il pulsante A, il runtime andrà ad effettuare una ricerca nella tabella dei suggested bindings, dove ad ogni path corrisponde un'azione specifica, verificando che per il pulsante A, l'azione consiste nel teletrasporto e procederà ad eseguirla.

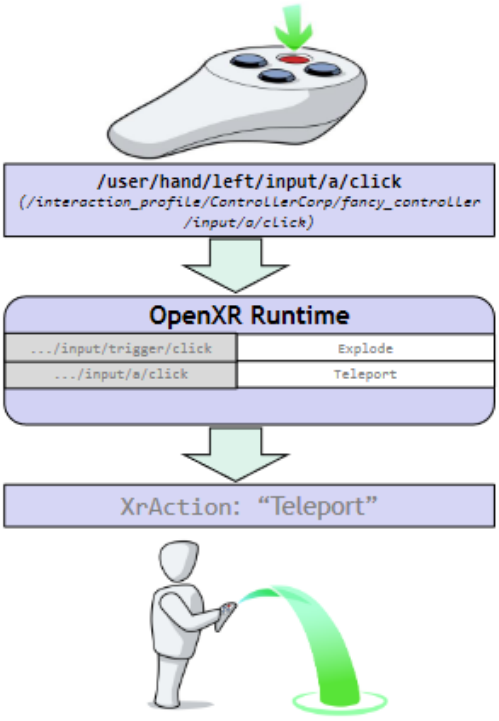


Figura 4.9: Utilizzo di un profilo di interazione

Dal discorso precedente emerge quindi che: **sebbene l'applicazione possa suggerire le associazioni consigliate, spetta al runtime associare le sorgenti di input alle azioni come meglio crede**.

Se viene chiesto a uno degli sviluppatori di OpenXR il perché di questa scelta, la risposta che è stata data al SIGGRAPH del 2019 è stata: "Dev teams

are ephemeral, games last forever!" [6], ovvero i team di sviluppo sono effimeri, sono destinati a non durare nel tempo, mentre i video games dureranno in eterno, che cosa significa quindi? Significa che, noi possiamo avere il nostro gioco preferito costruito 10 anni fa, con dei controller che ormai non esistono più, ma a noi piaceva davvero tanto quel gioco quindi vorremmo rigiocarci utilizzando i nostri controller di ultima generazione, di conseguenza, se vi è la possibilità per il runtime di capire quali azioni si vuole compiere e di associare tali azioni ai pulsanti dei controller più recenti, potremmo ancora giocare al nostro videogioco preferito.

4.7 Rappresentazione delle azioni: Actions e Action Sets

Nella sessione precedente, abbiamo visto come possono essere definiti i profili di interazione, passiamo ora a vedere come possono essere definite, le azioni da associare a tali profili d'interazione.

Le applicazioni OpenXR, comunicano con i dispositivi di input attraverso delle `XrActions`, esse vengono create in fase di inizializzazione e vengono utilizzate in fase di esecuzione per: richiedere lo stato dei devices, creare action spaces o controllare eventi di haptics. Sostanzialmente, gli handles della azioni di input, rappresentano le "azioni", di cui l'applicazione è interessata ad ottenere lo stato e non l'input diretto del dispositivo hardware, quindi ad esempio l'applicazione, anziché richiedere lo stato del pulsante A durante l'interazione con un menu, creerà un'azione `menu_select` in fase di inizializzazione e dopodiché durante l'interazione richiederà ad OpenXR lo stato dell'azione.

4.7.1 Collegamento fra azione e input del dispositivo

E' possibile creare un'azione attraverso la chiamata alla funzione `xrCreateAction` definita in questo modo:

```
XrResult xrCreateAction(
    XrActionSet                                 actionSet,
    const XrActionCreateInfo*                   createInfo,
    XrAction*                                   action);
```

Listato 4.1: funzione xrCreateAction

In particolare, la funzione come si può vedere, prende in ingresso il campo `XrActionCreateInfo`, tale oggetto contiene tutti gli attributi necessari alla descrizione e identificazione dell'azione, nonché il path che descrive il profilo di

interazione, di cui si è parlato nella precedente sessione, indicato con il nome di `subactionPaths`

```
typedef struct XrActionCreateInfo {
    XrStructureType  type;
    const void*      next;
    char             actionName[XR_MAX_ACTION_NAME_SIZE];
    XrActionType     actionType;
    uint32_t         countSubactionPaths;
    const XrPath*    subactionPaths;
    char
        localizedActionName[XR_MAX_LOCALIZED_ACTION_NAME_SIZE];
} XrActionCreateInfo;
```

Listato 4.2: oggetto XrActionCreateInfo

I paths, sono un meccanismo attraverso il quale le applicazioni, possono utilizzare gli stessi nomi di azioni e handles su dispositivi diversi, essi consentono di differenziare i dati provenienti dai diversi dispositivi e danno la possibilità al runtime di raggruppare azioni equivalenti, in differenti User Interface.

Infine, le applicazioni di solito, hanno bisogno di fornire dei default bindings per le azioni che possono essere svolte al runtime, in modo che i dati di input, possano essere mappati appropriatamente rispetto alle azioni. Tutto questo è possibile farlo tramite la funzione `xrSuggestInteractinoProfileBindings`, che consente di definire per ogni profilo di interazione il collegamento suggerito.

4.7.2 Diversi tipi di azione che possono essere creati

Come è possibile vedere dal listato 4.2 a un'azione viene associato un tipo, in particolare, le azioni possono essere distinte in: azioni di input e azioni di output e ve ne possono essere di diversi tipi, specificatamente tutti i tipi diversi di azioni sia di input che di output, sono racchiusi nella enum `XrActionType` e sono:

- `XR_ACTION_TYPE_BOOLEAN_INPUT`: da utilizzare quando l'azione può essere espressa con un booleano, quindi ad esempio un'azione di click di un pulsante, se esso risulta essere premuto lo stato è true altrimenti risulta essere false.
- `XR_ACTION_TYPE_FLOAT_INPUT`: è utile utilizzare un'azione di tipo float per un input analogico, il suo stato quindi ritornerà il valore rilevato.
- `XR_ACTION_TYPE_VECTOR2F_INPUT`: il suo stato restituisce un vettore di 2 coordinate espresse con un tipo di dato float.

- `XR_ACTION_TYPE_POSE_INPUT`: è un tipo particolare di azione, che restituisce una posa e può essere utile per tracciare la posizione di un oggetto.
- `XR_ACTION_TYPE_VIBRATION_OUTPUT`: è un'azione di output che consente di produrre un effetto di vibrazione sul dispositivo dell'utente.

4.7.3 Action Sets

Gli action sets, sono delle collezioni di azioni definite dall'applicazione, è possibile creare e distruggere un `XrActoinSet` rispettivamente con le funzioni `xrCreateActionSet` e `xrDestroyActionSet`, ricordando che se si deciderà di distruggere un action set, verranno automaticamente distrutte anche tutte le azioni contenute in esso.

Gli action sets, sono collegati a una specifica sessione tramite la funzione `xrAttachSessionActionSets`. Tali insiemi di azioni, possono essere abilitati o disabilitati dall'applicazione, tramite la funzione `xrSyncActions` a seconda del contesto. Ad esempio, un gioco potrebbe avere due set di azioni, uno per controllare le azioni del personaggio e un altro per poter navigare nel menu, questi action sets, possono essere attivati o disattivati, a seconda di quali siano le operazioni che l'utente vuole compiere, tramite la semplice chiamata alla funzione `xrSyncActions`.

4.8 Analisi del main loop di un'applicazione OpenXR

Passiamo ora ad analizzare, un esempio di main loop di una possibile applicazione OpenXR, in particolare il codice a cui si farà riferimento, è stato realizzato e reso pubblico da Microsoft, il cui team di sviluppo, si è impegnato nella creazione di alcuni progetti di esempio che utilizzano OpenXR, tra cui un'applicazione XR nativa [9].

Il main loop di seguito riportato listato: 4.3, ci mostra quali sono i passi che occorre compiere per realizzare un'applicazione OpenXR funzionante, che possono essere di seguito elencati:

1. Creazione di un'istanza.
2. Scelta delle estensioni da utilizzare, degli API layer e dell'API grafica.
3. Definizione delle azioni e dei bindings.
4. Creazione di una sessione.

5. Creazione degli Spaces.

6. Rendering delle immagini ed esecuzione del frame loop.

```
void Run() override {
          CreateInstance();
          CreateActions();

          bool requestRestart = false;
          do {
              InitializeSystem();
              InitializeSession();

              while (true) {
                  bool exitRenderLoop = false;
                  ProcessEvents(&exitRenderLoop, &requestRestart);
                  if (exitRenderLoop) {
                      break;
                  }

                  if (m_sessionRunning) {
                      PollActions();
                      RenderFrame();
                  } else {
                      // Throttle loop since xrWaitFrame won't be
                          called.
                      using namespace std::chrono_literals;
                      std::this_thread::sleep_for(250ms);
                  }
              }

              if (requestRestart) {
                  PrepareSessionRestart();
              }
          } while (requestRestart);
      }
```

Listato 4.3: esmpio di main loop di un applicazione OpenXR

Il primo passo da fare consiste nella creazione di un'istanza, per fare questo nel main loop viene richiamato il metodo `CreateInstance()`. All'interno di questo metodo, vengono selezionate e abilitate le estensioni che si è deciso di utilizzare per l'applicazione e si procede alla creazione dell'istanza con la chiamata al metodo `xrCreateInstance`.

Il secondo passo da compiere consiste nella definizione delle azioni, in particolare, tramite il metodo`CreateAction()` verrà definito un `ActionSet`, che conterrà tutte le azioni che potranno essere compiute. Una volta create le azioni, associandoli loro un nome e un tipo, sarà necessario definire i collegamenti con le interazioni dei controllers utilizzati, per fare questo va scelto e utilizzato un profilo di interazione come ad esempio `/khr/simple_controller`.

Il terzo passo da compiere, consiste nella creazione di una sessione, ma per poterlo fare occorre avere il riferimento al sistema utilizzato, ottenendo l'id che lo rappresenta, queste operazioni sono svolte all'interno del main loop dalla funzione `InitializeSystem()`, la quale esegue un loop infinito richiamando la funzione `xrGetSystem`, fino a quando non ottiene un risultato di successo.

Una volta fatto questo possiamo tornare alla creazione della sessione, attraverso il metodo `InitializeSession()`, che al suo interno richiama la funzione `xrCreateSession`, collega le azioni alla sessione, crea gli spaces per il sistema di riferimento e anche per il monitoraggio delle mani ed infine, entra nel frame loop dove gestisce il rendering delle immagini che saranno mostrate all'utente e che verranno renderizzate termite opportuni Swapchain.

4.8.1 Gerarchia degli oggetti creati

Alla fine di tutto quindi, la gerarchia degli handls e degli oggetti creati risulta essere rappresentata dalla seguente figura.

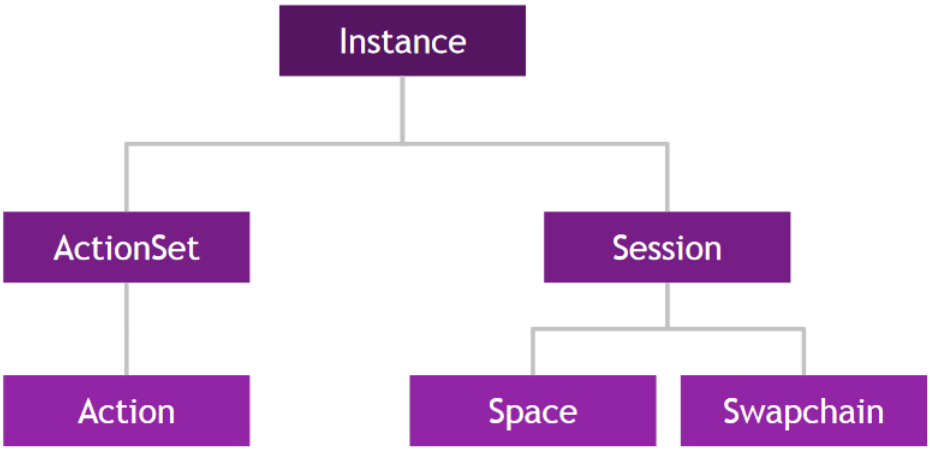


Figura 4.10: Gerarchia degli handles creati

Alla radice dell'albero troviamo l'istanza della nostra applicazione, dopodiché abbiamo ActionSet e Session che si trovano allo stesso livello, naturalmente essendo un'ActionSet un insieme di azioni, sotto di lui troveremo le

Action che lo costituiscono. Mentre dopo la creazione di una sessione, possiamo passare alla creazione degli Spaces e la gestione del frame loop tramite gli Swapchain, che si trovano entrambi sotto Session ma allo stesso livello gerarchico.

Capitolo 5

Sviluppo di un'applicazione XR in Unity

Nel precedente capitolo abbiamo analizzato, quali sono gli elementi che compongono un'applicazione OpenXR, come funzionano e come questi possano essere utilizzati dai programmatori nello sviluppo dei loro programmi.

Nella pratica però, la maggior parte degli sviluppatori di applicazioni di eXtended Reality, non programmano le applicazioni in linguaggio nativo, ma utilizzano appositi strumenti di sviluppo, che rendono più semplice sia la progettazione che la programmazione.

Per questo motivo, in questo capitolo, andremo a vedere come gli elementi analizzati prima, per OpenXR, si traducono in un ambiente di sviluppo come Unity e come poter definire un'applicazione di eXtended Reality, che sfrutti OpenXR e i suoi elementi.

5.1 Perché sviluppare un'applicazione in Unity

Gli applicativi per lo sviluppo di applicazioni con grafica 3D, come Unity, danno la possibilità ai programmatori di vedere il contenuto della scena in un ambiente simulato, inizialmente vuoto e di poter definire degli oggetti che potranno essere disposti in questo ambiente, con cui l'utente potrà interagire o che semplicemente costituiscono la scena in cui l'utente si potrà muovere.

Il grande vantaggio, che deriva dall'utilizzo di questi programmi per lo sviluppo delle applicazioni, è che gran parte delle funzionalità, anche complesse, che richiede un sistema di questo tipo, vengono gestite interamente da Unity, senza il bisogno che lo sviluppatore debba intervenire sul codice. Questo

fa sì quindi, che anche una persona non esperta possa costruirsi la propria applicazione di Mixed Reality.

Tuttavia il fatto che un programmatore, non abbia la possibilità di mettere le mani su tutti gli elementi dell'applicazione, anche quelli di più basso livello, potrebbe essere uno svantaggio se si desidera implementare funzionalità particolari, cosa che invece in un linguaggio nativo non si avrebbe, perché verrebbe gestito tutto interamente dal codice realizzato.

Nel capitolo 2 abbiamo detto che: Unity inizialmente era stata progettata per lo sviluppo di Videogame e che solo in un secondo momento è stata adattata per lo sviluppo di applicazioni di eXtended Reality, come mai questo? Vuoi per il fatto che i provider di Videogame, siano gli stessi che forniscono i dispositivi per la Virtual o la Mixed Reality e quindi sarebbe tornato utile utilizzare uno strumento conosciuto, oppure vuoi per il fatto che Unity presenta già elementi compatibili e simili a quelli richiesti, per la creazione di applicazioni per le differenti realtà... Fatto sta che Unity e uno degli applicativi più utilizzati, per lo sviluppo di questo tipo di programmi.

5.2 Iniziare a costruire l'applicazione

Per iniziare a costruire la nostra applicazione e riempirla con i primi elementi, la prima cosa che dobbiamo fare, dopo aver creato il progetto, è importare il plugin OpenXR che è stato definito appositamente per poter utilizzare OpenXR in Unity. Questo passaggio è quindi essenziale per far sì che la nostra applicazione funzioni e possa utilizzare l'API.

Una volta fatto questo passo e configurato opportunamente le impostazioni, come riportato nella documentazione [15], possiamo notare che alcune configurazioni del nostro progetto, mettono già in luce elementi di OpenXR, di cui abbiamo parlato nel precedente capitolo.

5.2.1 Come sono visti i profili di interazione e le estensioni

In particolare, se andiamo nelle impostazioni del progetto (Project Settings) e ci spostiamo nella sezione `XR Plugin Management` alla voce `OpenXR`, possiamo trovare la definizione dei profili di interazione e delle funzionalità dei diversi devices, offerti dai diversi providers, per i plugin che abbiamo attualmente installato.

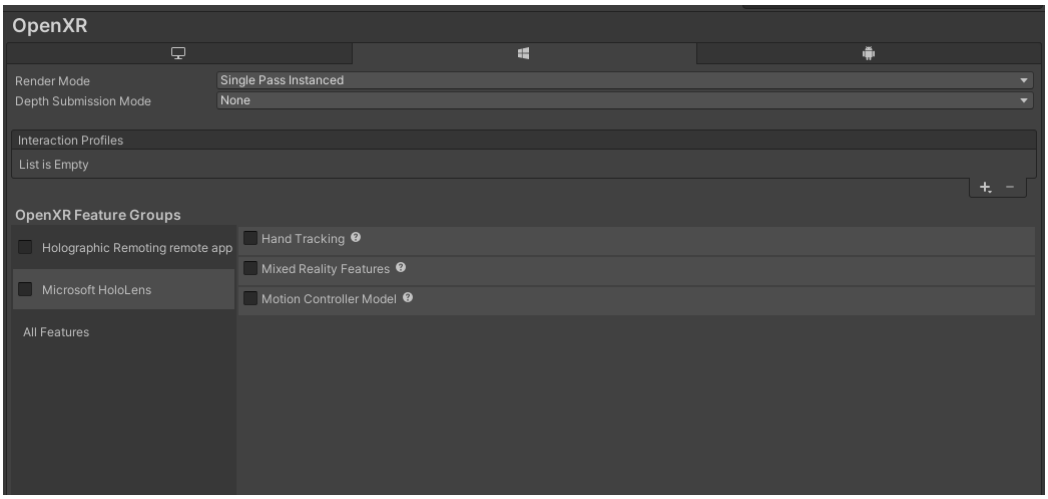


Figura 5.1: Profili di interazione e features dei devices

Ricordando che: le funzionalità dei diversi dispositivi, vengono implementate dai providers di questi, in opportuni API layers e esposte tramite delle estensioni, possiamo dire che: tali funzionalità vengono mostrate in Unity attraverso delle impostazioni e possiamo decidere se utilizzarle oppure no tramite un'operazione di selezione. Più pacchetti di OpenXR dei diversi devices installiamo nel nostro progetto, maggiori saranno le funzionalità a disposizione.

In questo caso nella figura 5.1, abbiamo a disposizione un unico plugin per HoloLens 2 e ci vengono messe a disposizione le funzionalità specifiche di: Hand Tracking, Motion Controller e altre funzionalità di Mixed Reality.

In questa sezione, delle impostazioni del progetto, possiamo anche indicare quali sono i profili di interazione che andremo a utilizzare a seconda del dispositivo di destinazione, possiamo aggiungere ovviamente più di un profilo se desideriamo avere più devices di destinazione.

5.3 Configurazione dell'ambiente: XR Interaction Toolkit

Per poter definire un sistema di riferimento all'interno della nostra scena e iniziare a lavorare sulle interazioni, occorre importare un ulteriore pacchetto che è: XR Interaction Toolkit.

Ora, dopo che l'installazione è avvenuta correttamente, se ci spostiamo nella scena principale del nostro progetto e clicchiamo con il testo destro sull'elemento Hierarchy, potremo notare una voce XR come mostrato nella figura sottostante e se la selezioniamo, possiamo trovare i diversi elementi messi a disposizione dal package XR Interaction Toolkit.

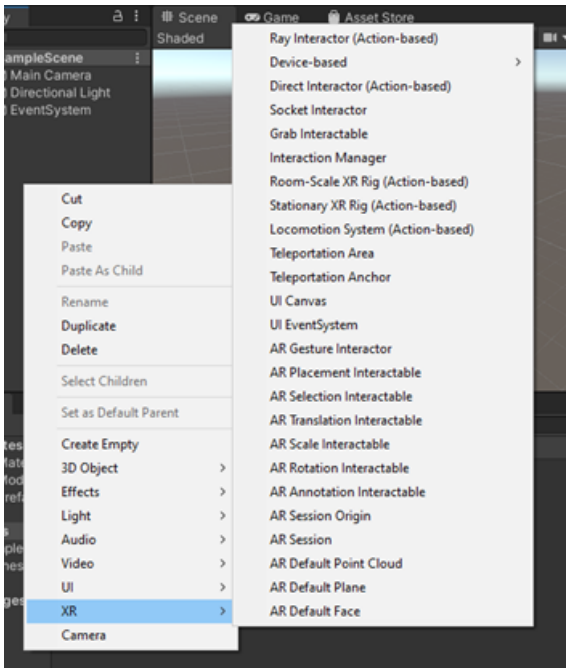


Figura 5.2: Elementi di XR

5.3.1 Xr Rig il cuore dell'applicazione

Per poter definire lo spazio, in cui l'utente si muoverà e compirà le sue azioni, dobbiamo inserire nella nostra Hierarchy un Xr Rig.

Un Xr Rig rappresenta gli occhi, le orecchie e le mani del nostro utente nel mondo aumentato. L'XR Interaction Toolkit definisce due tipi di XR Rig:

- Stationary XR Rig: utilizza il device dell'utente per definire l'origine del mondo, consentendogli di muoversi secondo tre gradi di libertà, è utile utilizzarlo per progetti in cui l'utente deve rimanere seduto, ad esempio, la simulazione di una montagna russa.

- Room-Scale XR Rig: consente allo sviluppatore di specificare le modalità di tracciamento dell'origine degli assi, specificatamente le opzioni sono: unknown, device, floor, tracking reference e unbounded. Questo tipo di Xr Rig da la possibilità all'utente di muoversi secondo sei gradi di libertà, in modo da consentirgli l'esplorazione del mondo aumentato.

Nel nostro caso, siccome vogliamo realizzare un'applicazione in cui l'utente sarà in grado di muoversi, sceglieremo il Room-Scale XR Rig.

5.3.2 Come sono tradotti gli Spaces di riferimento?

Alla luce di quanto detto nel capitolo precedente, per quanto riguarda gli spaces, possiamo intuire che: se decidiamo di utilizzare, uno Stationary Xr Rig, lo spazio che verrà utilizzato per rappresentare il mondo verrà dato da uno space di tipo LOCAL, in quanto costruito e pensato appositamente per una seted-scale experience, mentre se scegliamo un Room-Scale XR Rig, il mondo verrà tracciato con uno space di tipo STAGE, il quale consente all'utente di muoversi nell'ambiente.

5.4 Definizione della telecamera e dei controllers all'interno dell'XR Rig

Se espandiamo l'oggetto Xr Rig della nostra Hierarchy, possiamo notare che esso contiene, l'oggetto `Camera Offset`, che a sua volta contiene: la telecamera principale e la definizione dei controllers per la mano destra e sinistra.

Analizzando l'oggetto telecamera, possiamo notare che fra i suoi componenti ve ne è uno denominato: `Tracked Pose Driver`, questo elemento deve sempre essere applicato alla camera principale, se vogliamo realizzare un'applicazione di eXtended Reality, tramite OpenXR.

5.4.1 Perché la Main Camera necessita di un Tracked Pose Driver e quali funzioni svolge

Come mai c'è bisogno che la Main Camera possieda un componente `Tracked Pose Driver`, per far sì che la nostra applicazione funzioni? Questo perché, la telecamera in Unity, si occupa di mostrare il contenuto dell'applicazione all'utente.

Il modo con cui si lavora con la telecamera, nelle applicazioni di eXtended Reality però, è diverso rispetto alle applicazioni di videogiochi 3D. In un videogioco, l'utente vede il mondo tramite lo schermo di un computer, quindi se l'utente muove il computer o lo ruota, il contenuto dello schermo non cambia, se egli vuole muoversi per il mondo del videogioco dovrà per forza utilizzare il mouse o la tastiera.

Le applicazioni di eXtended Reality lavorano diversamente, in quanto il contenuto viene mostrato tramite un visore e l'utente può osservare il mondo attorno a lui tramite dei movimenti della testa, questo di conseguenza, farà ruotare e muovere la telecamera del mondo aumentato. Per mantenere traccia della posizione della testa dell'utente, a cui associamo la visuale della telecamera, utilizziamo un `Tracked pose driver`.

La funzione principale che svolge un `Tracked pose driver` quindi, è quella di mantenere tracciata la posizione di un dispositivo nel tempo e di applicare tale posizione all'oggetto a cui l'abbiamo assegnato.

Concludendo, se abbiamo necessita di sapere nel nostro progetto, la posizione di un sistema hardware nel tempo, quello che dobbiamo fare è: rappresentare tale sistema con un oggetto nella nostra Hierarchy e applicarli il componente `Tracked Pose Driver`.

5.4.2 Come vengono visti i controllers all'interno dell'applicazione

Quando creiamo un Xr Rig, automaticamente vengono definiti, i controllers per la mano destra e sinistra.

Di default, i controllers che vengono creati sono degli Action-based controller, e questo è possibile notarlo, grazie al componente `XR Controller (Action-based)`, che è stato attribuito agli oggetti.

Possiamo intuire che esistono diversi tipi di controllers, in particolare, sono disponibili due tipi, che sono:

- Action-based controller: utilizza le Actions, per leggere indirettamente l'input da uno o più controller.
- Device-based controller: utilizza l'input del device specifico, per comprendere quale sia l'azione da compiere.

Selezionando uno degli oggetti che rappresenta i controllers nella scena, possiamo notare che ad essi vengono attribuiti altri tre componenti che sono:`XR Ray Interactor`, `Line Render` e `XR Interactor Line Visual`. Questi elementi danno la possibilità all'utente, quando l'applicazione è in esecuzione, di vedere i controllers, tramite appunto una linea diretta che parte dal device di input o dalle mani, tale linea costituisce anche il raggio con cui si potrà interagire con gli oggetti presenti nella scena.

A questi controllers possiamo associare degli oggetti figli, in modo da avere una visualizzazione un po' più chiara delle nostre mani o del dispositivo che stiamo utilizzando come input.

In conclusione possiamo dire che: in Unity i controllers sono visti come degli oggetti, a cui vengono associati degli specifici script che consentono di definirli e alla luce di quanto detto precedentemente, siccome l'utente utilizzerà questi oggetti per compiere delle azioni, ai controllers verranno assegnati degli Action Space, che consentiranno all'applicazione di poterli gestire e collocare nello spazio.

5.4.3 Definizione delle azioni

Se scegliamo di utilizzare un Action-based controller, sarà necessario definire tutte le diverse azioni che sarà possibile compiere e i corrispettivi collegamenti, essendo questa un'operazione complicata, Unity viene in contro al programmatore definendo un set di azioni di base, che è possibile scaricare e importare nel nostro progetto, andando nella finestra del Package Manager e selezionando il pacchetto Xr Interaction Toolkit, alla voce Samples, importando l'elemento Default Input Actions.

Se ci spostiamo nell'opportuna cartella importata, potremo accedere all'apposito assets per la definizione delle azioni.

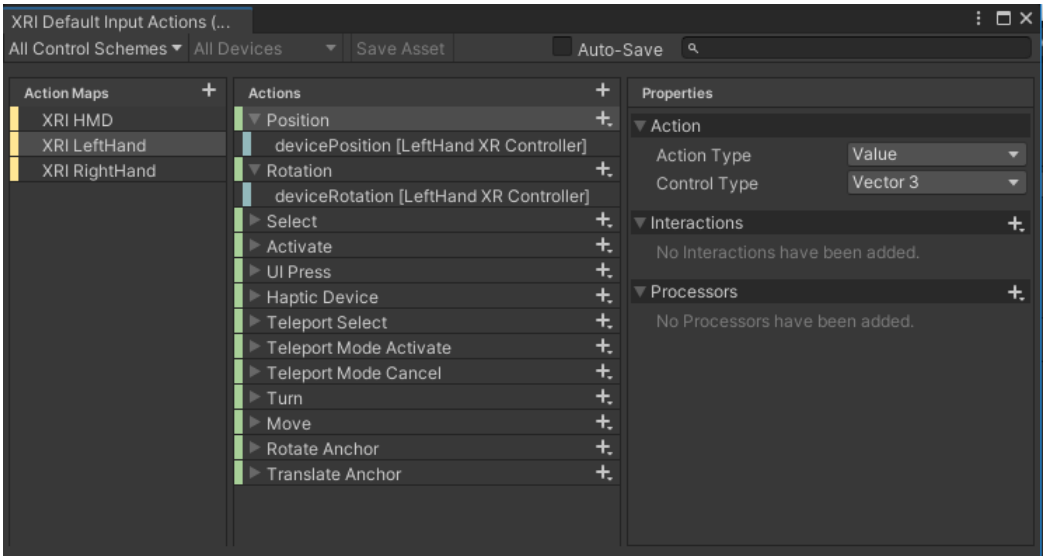


Figura 5.3: Schermata per la definizione delle azioni

Come possiamo notare dalla figura 5.3, le azioni vengono divise in quelle compiute con la mano destra e quelle compiute invece con la mano sinistra, vengono già definite delle azioni di base, come ad esempio: Position, Rotation e Select a cui viene associato l'apposito bindings. Ad un'azione possiamo aggiungere più di un bindings, ad esempio se occorre gestire diversi dispositivi di input.

I bindings che possiamo assegnare alle azioni, dipendono dai profili di interazione che abbiamo deciso di utilizzare all'inizio.

Una volta definite tutte le nostre azioni e i rispettivi bindings, possiamo assegnarle al nostro controller, aggiungendo gli opportuni componenti.

5.5 Interagire con un oggetto

Dopo aver definito le azioni e aver impostato correttamente gli oggetti che li rappresentano, la prossima cosa che rimane da fare, è consentire l'interazione con un ologramma, che sarà rappresentato da un opportuno oggetto in Unity.

Quando abbiamo creato un Xr Rig, automaticamente nella nostra Hierarchy è stato aggiunto anche un ulteriore oggetto: `Xr Interaction Manager`, questo componente si occupa di gestire i diversi oggetti Interactors e Interactables all'interno della scena, senza questo elemento le interazioni non potrebbero avvenire.

5.5.1 Interactor: gli oggetti che compiono le azioni

Gli interactor forniscono all'utente un modo per: selezionare, prendere o porre il focus su un oggetto. Il componente interactor, assegnato ad un elemento lavora con l'Interaction Manager, per la gestione degli eventi e delle interazioni.

Esistono diversi tipi di interactor, uno dei quali abbiamo già potuto vedere in precedenza per i controllers, che è il `Ray Interactor`, ad ogni modo le opzioni disponibili sono:

- Direct interactor: questo tipo di interactor, consente all'utente di interagire direttamente con gli elementi del mondo aumentato, senza la necessità che questi vengano prima selezionati.
- Ray Interactor: tale interactor presenta un raggio che li consente di selezionare gli oggetti con cui interagire, la visualizzazione di questo raggio è resa possibile tramite opportuni componenti, ad esso automaticamente associati, che generano una linea in corrispondenza del raggio, la quale cambia colore se un oggetto è selezionabile.
- Socket Interctor: è un interactor che si occupa di mantenere un riferimento ad un oggetto interactable e di generare un evento quando questo viene inserito o rimosso dal Socket.

5.5.2 Interactables: gli oggetti su cui le azioni vengono compiute

Gli oggetti che presentano un componente di tipo interactables, invece, rappresentano gli elementi su cui l'utente potrà compiere determinate azioni come: grab, throw, moving, dropping ecc. Attualmente gli interactables che vengono forniti sono:

- Simple Interactable: è la versione più semplice di un interactable, esso fornisce una semplice implementazione della classe. `XRBaseInteractable`. E' pensato per essere utilizzato in modo da rispondere a operazioni di Hover o Select.

- Grab Interactable: consente la funzionalità base di grab di un oggetto, avendo la possibilità di attaccare direttamente l'oggetto all'interactor che lo ha selezionato, seguendolo e obbedendo anche ai principi della fisica.

5.5.3 Considerazioni sulle interazioni in Unity e OpenXR

Quando abbiamo parlato di OpenXR, non abbiamo mai definito degli oggetti di tipo interactor e interactable, questi oggetti infatti li troviamo solamente all'interno dell'ambiente di Unity, grazie all'XR Interaction Toolkit.

Ciononostante, per quanto detto fin ora, possiamo desumere che: agli oggetti di tipo interactables, siccome sono elementi su cui potranno essere compiute delle azioni, molto probabilmente il sistema gli assegnerà un'action space, per rappresentarli e reagire quando l'azione si verifica. Anche per gli oggetti di tipo interactor, essendo coloro che compiono le azioni, il sistema provvederà ad attribuirli un opportuno Action Set, che conterrà le operazioni che possono essere compiute e uno space di tipo action, sia per individuare la loro posizione, che per capire quando l'azione verrà scaturita.

5.6 Perché l'applicazione necessita di altri Plugin oltre ad OpenXR

Da tutta questa discussione emerge che OpenXR, non presenta di per se una definizione dei componenti che utilizzeremo normalmente in Unity, questo perché quest'API è più diretta ad uniformare i devices, fornendo un layer di astrazione fra le azioni che possono essere compiute e gli input fisici da cui esse vengono scaturite.

Il suo scopo principale quindi, non è quello di definire quali sono le interazioni che possono essere effettuate sugli ologrammi o una rappresentazione dei controllers. **Quest'API di per se, è pensata per fornirci tutti gli strumenti utili a creare un'applicazione, che concorrono al raggiungimento del suo scopo**, vale a dire: risolvere il problema della frammentazione, consentendo a un programma di poter essere eseguito su differenti devices, senza che lo sviluppatore per farlo abbia la necessità di scrivere, due applicazioni completamente diverse.

Dobbiamo vedere quindi OpenXR come un meccanismo che lavora sotto alla nostra applicazione, con cui noi non interagiamo direttamente nella programmazione Unity, ma che vediamo attraverso determinati elementi.

Gli elementi che ci vengono dati dal OpenXR plugin e l'XR interaction Toolkit, sono degli elementi basilari e se vogliamo realizzare qualcosa di più complesso, elaborato o più sofisticato abbiamo due alternative: o ci costruiamo noi gli elementi di cui abbiamo bisogno, oppure chiediamo aiuto ad altri toolkit realizzati per la creazione di applicazioni per la Mixed la Virtual o l'Augmented Reality, che a loro volta supportano OpenXR.

Alcuni di questi toolkit li abbiamo analizzati nel capitolo 2, ed è possibile utilizzarne più di uno all'interno del nostro progetto, purché i diversi elementi non presentino delle incompatibilità o dipendenze particolari fra loro. La scelta che di solito viene effettuata infatti è quella di utilizzare OpenXR, insieme ad AR Foundation o l'MRTK.

5.6.1 Come cambia il progetto se inseriamo degli altri toolkit?

I toolkit di cui abbiamo parlato, al contrario di XR, mettono a disposizione moltissimi elementi per la nostra applicazione, alcuni dei quali possono andare a sovrascrivere quali analizzati fin ora.

Se infatti, ad esempio, importiamo nel nostro progetto l'MRTK esso presenta già una propria configurazione della telecamera e attiva automaticamente il monitoraggio delle nostre mani. Quindi tutto quello che abbiamo detto e fatto fin ad ora, non serve più a niente? Ovviamente no, se vogliamo partire già ad utilizzare l'MRTK, per la nostra applicazione, nessuno ci vieta di farlo, ma se vogliamo partire dal plugin OpenXR, con l'aggiunta di altre funzionalità date dall'MRTK, possiamo fare anche questo.

Dobbiamo comprendere che l'obbiettivo di questi toolkit, è quello di fornire a noi sviluppatori, un modo più semplice per programmare la nostra applicazione: è molto più semplice importare l'MRTK nella scena che configura in automatico camera e monitoraggio delle mani, invece che dover creare l'XR Rig, settare i controller, le azioni ecc...

Al suo interno però, questi toolkit che supportano OpenXR, sfruttano proprio i suoi elementi che abbiamo visto precedentemente e li utilizzano per sviluppare funzionalità sempre più complesse e metterle a disposizione dell'utente. Risulta quindi esser importante capire e apprendere il funzionamento dell'API e dei suoi elementi.

5.7 Virtual Reality avvantaggiata nello sviluppo rispetto alla Mixed e l'Augmented Reality

Un'ultima considerazione che emerge, nell'utilizzo di OpenXR in Unity per lo sviluppo dei diversi programmi è che, la realizzazione di applicazioni per la Virtual Reality, risulta essere avvantaggiata rispetto a quelle per l'Augmented Reality e soprattutto la Mixed Reality.

Questo perché quando definiamo l'Xr Rig e i controllers, automaticamente questi risultano essere correttamente configurati per un'applicazione di Virtual Reality, dove i dispositivi utilizzati per l'interazione sono fisici ed essa per tanto è resa più semplice grazie all'utilizzo di un ray interactor, che parte dal device fino alla lunghezza della linea.

Nell'Augmented Reality però, questi controllers fisici non ci sono e nella Mixed Reality possono esserci, ma ormai grazie agli sviluppi della tecnologia, non sono più necessari e l'utente può interagire con gli ologrammi, direttamente con le proprie mani. Quello che servirebbe all'utilizzatore dell'applicazione quindi, sarebbe una mesh opportuna che li consenta di riconoscere effettivamente che l'applicazione sta provvedendo al tracciamento delle mani.

Questo significa quindi, che le impostazioni di base di OpenXR, per lavorare con Augmented e Mixed Reaity, devono essere modificate.

Per capire meglio questo concetto, possiamo mettere in luce un aspetto analizzato prima: il Grab interactable, messo a disposizione dall'XR Interaction Toolkit, consente a un utente tramite un controller, di afferrare un oggetto e quando quest'oggetto viene preso, tramite l'opportuna azione, esso rimane attaccato al controller utilizzato e se si rilascia la presa questi viene automaticamente lanciato, reagendo ai principi della fisica.

Questo comportamento, se ci si ferma a riflettere un attimo è ottima per la Virtual Reality, dove non si possiede il monitoraggio e l'articolazione delle mani e quindi è possibile solo afferrare un oggetto senza manipolarlo.

Molto spesso, nella Virtual Reality, dopo aver afferrato un oggetto o lo si utilizza, svolgendo opportune operazioni o appunto lo si lancia, pensiamo ad esempio alla possibilità di afferrare una spada, una volta che l'abbiamo presa la teniamo in mano e muoviamo il controller per utilizzarla e sconfiggere i nemici.

Nella Mixed Reality però, avendo a disposizione nella maggior parte dei casi, il riconoscimento delle mani, l'operazione più comune e anche più bella che si ha la possibilità di fare, è quella della manipolazione degli ologrammi, quindi, quando utilizzo l'applicazione e interagisco con un oggetto, vorrei poter prendere l'ologramma e manipolarlo a mio piacimento, ma se utilizziamo il

Grab Interactable, messo a disposizione dall'XR interaction Toolkit, questo non possiamo farlo.

Di conseguenza, le funzionalità offerte dalla Mixed Reality, risultano essere più complesse rispetto a quelle di Virtual e Augmented, in questo caso dobbiamo riconoscere i gesti e la posizione delle mani dell'utente e gli elementi che abbiamo di OpenXR in Unity, pur supportando queste funzionalità, non ci forniscono effettivamente una loro implementazione, vale a dire che, possiamo creare il nostro oggetto che grazie ai profili di interazione, reagisce al gesto di air tap dell'utente, ma tuttavia non abbiamo elementi che ci consentono di individuare le mani dell'utente con un'opportuna mesh, che metta in luce anche le articolazioni cioè che non sia solamente statica, ma che si muova in funzione del movimento delle dita.

In conclusione, risulta essere necessario ancora una volta utilizzare appositi toolkit che ci forniscono le funzionalità di cui abbiamo bisogno per sviluppare l'applicazione di Mixed Relaity, mentre al contrario le applicazioni di Virtual Reality, siccome non hanno bisogno di svolgere questo tipo di operazioni, possono utilizzare direttamente gli elementi che li vengono forniti, senza dover cambiare le impostazioni.

5.8 Vantaggi e svantaggi nell'utilizzo di OpenXR in Unity

In questo capitolo abbiamo analizzato come costruire un'applicazione che sfrutti l'API in Unity e come i suoi elementi vengano tradotti e abbiamo introdotto quali siano vantaggi e svantaggi che possono derivare dal suo utilizzo.

Riassumendo possiamo dire che: OpenXR ci da la possibilità in Unity di costruire applicazioni che possano essere eseguite sui differenti devices, sgravando il compito di gestione di alcune funzionalità complesse direttamente all'ambiente di sviluppo, tuttavia le applicazioni create risultano essere compatibili **fino a un certo punto**. Essendo le esperienze offerte dalle differenti realtà diverse fra loro, per poter integrare nel progetto funzionalità specifiche per l'una o per l'altra Reality di destinazione, i soli elementi di XR che abbiamo in Unity non bastano.

L'auspicio degli sviluppatori, che si sono scontrati con lo sviluppo delle applicazioni di eXtended Reality in Unity che sfruttino l'API OpenXR, è molto probabilmente quello che in futuro, **si arrivi a definire un unico strumento da poter utilizzare in Unity, che fornisca funzionalità per sviluppare applicazioni per tutti i tipi di realtà, senza la necessita di doverne introdurre degli altri**.

Capitolo 6

Sperimentazioni di applicazioni di esperienze condivise multipiattaforma

Ora che abbiamo visto tutti gli elementi che occorrono per creare le applicazioni per le diverse realtà e il loro funzionamento, in questo capitolo verranno descritti e analizzati due programmi che realizzano un esperienza condivisa di realtà mista, con differenti dispositivi, attraverso l'utilizzo di buona parte degli strumenti visti in precedenza.

6.1 Descrizione delle applicazioni

Si è deciso di realizzare due applicazioni: la prima consente la realizzazione di un'esperienza condivisa, tramite una partecipazione da remoto, senza che gli utenti debbano condividere lo stesso ambiente e la seconda invece, consente ai partecipanti di prendere parte all'esperienza condivisa, trovandosi nello stesso luogo. In entrambi i casi, quello che si propone di fare il programma è: consentire a più utenti di poter vedere e manipolare uno stesso ologramma, facendo in modo che, quando un partecipante effettua una manipolazione, tutti siano in grado di vederla.

Inoltre all'esperienza condivisa, saranno in grado di partecipare più utenti con differenti devices, in particolare, il programma è stato sviluppato in modo da poter essere eseguito su un sistema Android e un visore HoloLens 2 per la Mixed Reality.

Un partecipante all'esperienza, tramite l'applicazione, si collegherà ad un server per la condivisione delle informazioni con gli altri partecipanti, dopodiché egli sarà in grado di vedere il contenuto olografico condiviso e le mani-

polazioni di questo ad opera degli altri utenti collegati, avendo la possibilità di stare in ambienti differenti in un caso e in un ambiente uguale invece nell'altro.

Per individuare meglio i partecipanti all'esperienza, ad ognuno di essi è stato associato un colore e quando un utente effettua una modifica all'ologramma, gli altri utenti potranno visualizzare il suo colore, tramite una bounding box che racchiuderà l'oggetto condiviso.

6.2 Individuazione degli scenari di destinazione

Nel capitolo 2 è stata dedicata un'intera sessione, che descrive quali sono gli aspetti che occorre considerare nella realizzazione delle esperienze condivise e quali siano, di conseguenza, le domande che occorre porsi, prima di poterne realizzare una.

Prima di passare all'effettiva implementazione delle due applicazioni si è cercato quindi, di dare una risposta alle domande presentate in precedenza.

- Qual è il metodo di condivisione? In entrambi i casi la modalità di condivisione del contenuto è di tipo collaborativo, ciascun utente è in grado di vedere e manipolare il contenuto olografico, partecipando attivamente all'esperienza.

- Quali sono le dimensioni del gruppo? Anche in questo caso in tutti e due i progetti il gruppo di utenti che può prendere parte all'esperienza è ridotto ad un massimo di sette persone, se però un utente decide di uscire dell'applicazione un altro può prendere il suo posto, quindi il gruppo di utenti può crescere e diminuire dinamicamente. In conclusione, per quanto detto nel capitolo due, il gruppo di persone è identificato come piccolo, perché minore o uguale a sette.

- Dove si trovano gli utenti? Nella prima applicazione gli utenti si trovano in ambienti fisici diversi, mentre nella seconda essi condividono lo stesso ambiente e sono in grado di vedere il contenuto del mondo olografico, nello stesso punto.

- Quando avviene la condivisione? In entrambi i casi l'esperienza condivisa è sincrona, gli utenti sono quindi in grado di vedere i cambiamenti quando questi avvengono e lo stato dell'applicazione non viene mantenuto ad avvi successivi

- Quanto sono simili i loro ambienti? Nel primo caso gli ambienti dei rispettivi partecipanti sono diversi fra loro, non è necessario che siano

simili, nel secondo caso invece, tutti i partecipanti si trovano nello stesso luogo e quindi presentano un ambiente identico.

- Quali dispositivi verranno utilizzati? Gli utenti che decidono di partecipare, possono farlo o indossando un dispositivo HoloLens 2, oppure tramite il loro tablet o telefono che abbia un sistema operativo Android.

6.3 Analisi e architettura del Server

In entrambi i progetti il server si occupa di gestire lo scambio di informazioni fra i diversi clients, in modo tale che tutti gli utenti possano vedere l'oggetto muoversi, scalare o ruotare in sincronia.

Per evitare conflitti, dovuti al fatto che più utenti possano modificare l'oggetto contemporaneamente, si è fatto sì che quando un partecipante, inizia la manipolazione esso invii una richiesta al server, richiedendoli il controllo dell'ologramma, se questa viene accettata, il server si occupa di inviare a tutti gli altri clients connessi, una richiesta di disabilitazione dei controlli che consentono la manipolazione dell'oggetto, i quali rimarranno inattivi, fino a quando l'utente che detiene il controllo non lo rilascia.

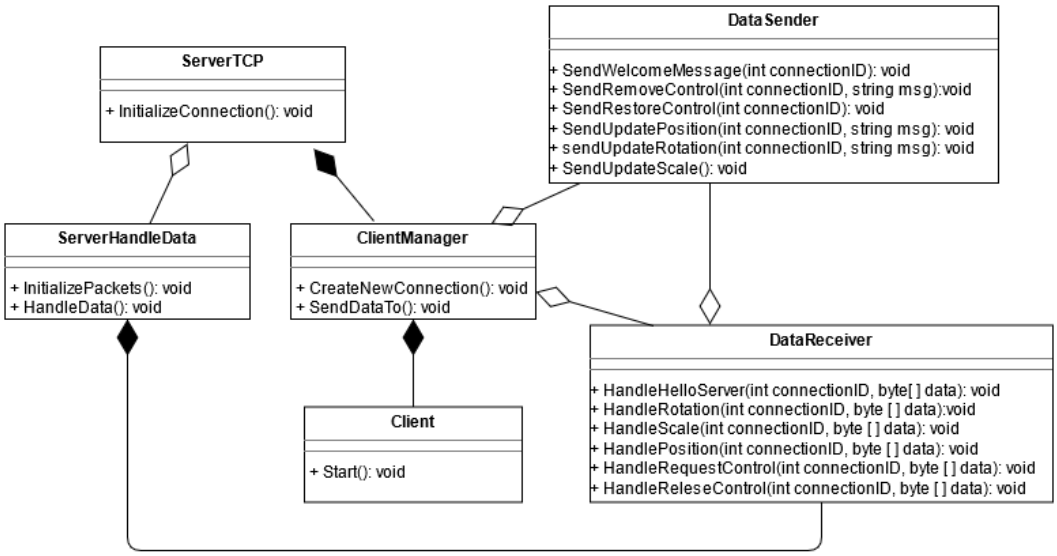


Figura 6.1: Architettura del server

Com'è possibile vedere dalla figura 6.1, che rappresenta l'architettura del server, esso si compone di diversi elementi. Innanzitutto, per poter gestire e inviare i messaggi ai diversi clients, occorre una classe che li rappresenti

e questo è proprio quello che si propone di fare, la classe `Client`, poi per mantenere salvati i diversi utenti che si connettono al server è stata realizzata la classe `ClientManager`, che quando si verifica una nuova connessione si occupa di salvare i `Client` in un opportuno `Dictionary`, dove ad ognuno di essi viene associato un connection ID univoco.

I pacchetti che il server può inviare sono contenuti in una enum all'interno della classe `DataSender`, i cui metodi sono richiamati quando un opportuno pacchetto, tra quelli indicati, deve essere inviato. Anche i pacchetti che il server può ricevere sono contenuti all'interno di una enum nella classe `DataReciver`, i metodi di questa classe sono chiamati dall'elemento `ServerHandleData`, che si occupa di ricevere i pacchetti e di smistarli in base a un opportuno codice identificativo, dato appunto dalla enum di `DataReciver`, che identifica il tipo di pacchetto.

Infine la classe `ServerTCP`, si occupa di inizializzare i pacchetti che possono essere ricevuti e i metodi necessari per la loro corretta elaborazione e di impostare ed eseguire la Socket, con cui si metterà in ascolto di connessioni su una determinata porta, demandando poi l'aggiunta del nuovo client, a connessione avvenuta, al `ClientManager`.

6.4 Analisi e architettura dei Clients

In Entrambi i due diversi tipi di esperienza, i clients, si connettono al server per ricevere gli aggiornamenti sull'ologramma o inviare agli altri partecipanti i propri dati, derivanti dalla manipolazione. Gli script dei clients, sono stati inseriti ed utilizzati direttamente nell'ambiente di sviluppo Unity.

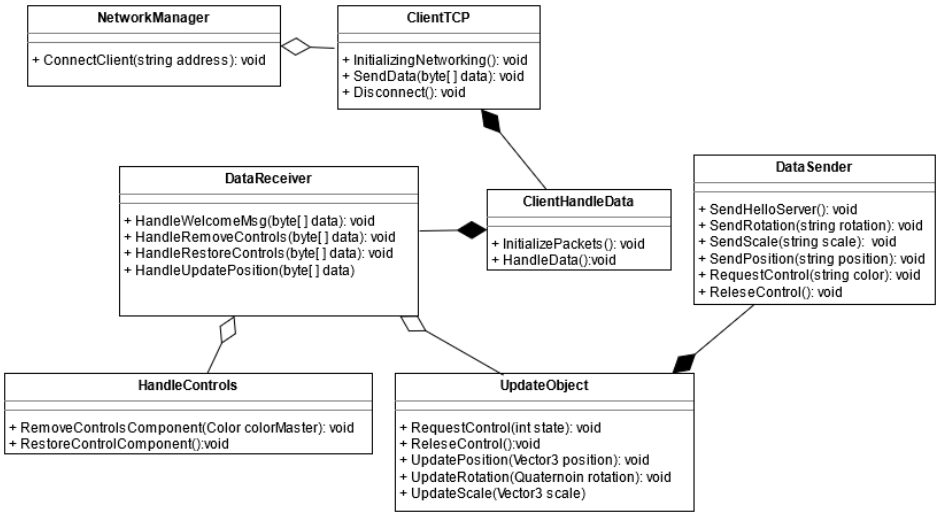


Figura 6.2: Architettura del client

Come si può vedere dalla figura 6.2, un client è rappresentato dalla classe `ClientTCP`, la quale si occupa di istanziare la Socket attraverso la quale ci si connetterà al server e ci si metterà in ascolto su una determinata porta, dei pacchetti in arrivo, la cui gestione viene demandata alla classe `ClientHandleData`. Tale classe, si occupa di smistare i pacchetti ricevuti, in base a un codice che essi presentano dato da una enum dichiarata in `DataReciver`, associando quindi ad ognuno dei pacchetti che possono essere ricevuti, un opportuno metodo per l'elaborazione.

L'elaborazione effettiva del pacchetto avviene ad opera della classe `DataReciver`, la quale a sua volta interagisce con altri oggetti che sono `HandleControls` e `UpdateObject`, che contengono i riferimenti agli elementi nella scena di Unity e `DataReciver` gli utilizza come tramiti, per aggiornare tali elementi nella scena.

Infine, quando un utente detiene il controllo dell'oggetto, ci si occupa di inviare gli aggiornamenti relativi al parametro modificato che può essere: la posizione, la rotazione o la scala, attraverso la classe `DataSender`.

6.5 Sviluppo in Unity del progetto di esperienza condivisa in ambienti diversi

In questa sezione e nelle successive, si entrerà nel merito dello sviluppo del primo progetto, che prevede la realizzazione di un'esperienza condivisa, a cui è possibile partecipare tramite un dispositivo HoloLens 2 o uno smartphone o tablet che presenta un sistema operativo Android, stando in ambienti differenti e vedendo le manipolazioni effettuate dagli altri utenti in tempo reale.

6.5.1 Quali sono le tecnologie che si è deciso di utilizzare e perché

Si è deciso di sviluppare il progetto tramite la piattaforma Unity, tuttavia, come descritto precedentemente essa cambia molto velocemente nel tempo e questo può portare a una serie di problemi, come malfunzionamenti o incompatibilità nelle diverse versioni con i packages utilizzati.

Per il progetto quindi, si è deciso di utilizzare l'MRTK con il supporto ad OpenXR, per poter avere lo stesso tipo di interazione, sia lato HoloLens che lato dispositivo Android, tuttavia il supporto dell'MRTK ai dispositivi utilizzati comunemente per l'Augmented Reality, come i telefoni, tramite AR Foundation, viene fornito fino alla versione 2019 di Unity, mentre nella versione 2020, questo supporto è ancora in fase sperimentale.

Per questo motivo, il progetto per Android è stato sviluppato tramite la versione LTS di Unity 2019 come riportato nella documentazione di Microsoft, relativa al supporto dell'MRTK ai sistemi Android e IOS [7], mentre per HoloLens, siccome questa versione ha presentato problematiche per la consapevolezza spaziale, durante le prove del suo utilizzo, si è deciso di utilizzare quella LTS del 2020, come richiesto e consigliato nella documentazione di Microsoft, per lo sviluppo di applicazioni per Hololens 2 [8].

Ad ogni modo, la creazione di due progetti Unity diversi per la realizzazione dell'applicazione sarebbe stata comunque una scelta consigliata, in quanto realizzare tutto in un unico progetto, anche se con due scene diverse, avrebbe come svantaggio il fatto che: quando si prosegue a fare il deployment sui diversi devices, occorre modificare le impostazioni di build e nel caso di Android, viene effettuata una conversione degli script dell'MRTK, in modo che essi risultino essere compatibili con AR foundation, dopodiché questi script dovrebbero essere nuovamente riconvertiti per poter essere eseguiti su HoloLens e tutte queste procedure di conversione e riconversione, pur avvenendo automaticamente ma in modo visibile al programmatore, avrebbero potuto generare degli errori nel progetto, provocandone il malfunzionamento.

Per questo motivi si preferisce tenere separate le applicazioni per i diversi dispositivi, rendendo anche più agevole il deployment.

Per entrambi i progetti, la versione dell'MRTK utilizzata è la 2.6.1, mentre per il progetto Android sono state installate le versioni 2.1.8 di AR Foundation e 2.1.11 dell'ARCore XR Plugin, i quali consentono e prevedono l'utilizzo di OpenXR e risultano essere necessari affinché l'MRTK possa funzionare sul dispositivo, come indicato nella rispettiva documentazione [7].

6.5.2 Descrizione degli elementi nella scena

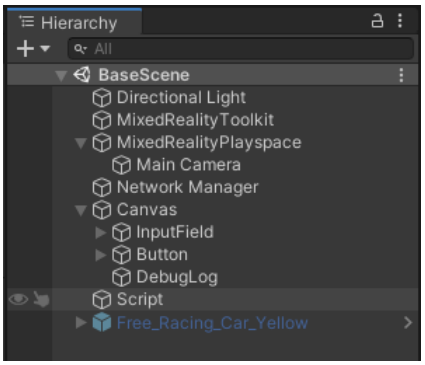


Figura 6.3: Elementi della scena

Com'è possibile vedere dalla figura 6.3, all'interno della scena sono presenti:

- Gli elementi del MRTK: il `MixedRealityToolkit`, che contiene la definizione delle funzionalità dell'MRTK e la configurazione dei profili che si è deciso di utilizzare e il `MixedRealityPlayspace`, il quale contiene l'oggetto `MainCamera` che rappresenta la vista dell'utente sulla scena che si sta costruendo.

- L'oggetto `Network Manager`: che si occupa di gestire la connessione e gli script dei clients analizzati in precedenza nella sezione 6.4 e sostanzialmente, la funzione principale che esso svolge, è richiamare i metodi della classe `ClientTCP` per instaurare una nuova connessione.

- L'elemento `Canvas`: che contiene al suo interno, gli oggetti che costituiscono la GUI, con cui l'utente interagirà all'avvio dell'applicazione, per impostare la connessione con il server.

- L'oggetto `Script`: il quale presenta come componenti, tutti gli script che si è deciso di utilizzare all'interno della scena, che consentono la gestione dei suoi elementi. Tale oggetto, verrà utilizzato dagli altri, in modo da poter richiamare i metodi dei suoi componenti, al verificarsi di determinati eventi.

- L'ologramma oggetto della manipolazione: che all'inizio non risulta essere visibile all'utente, ma verrà attivato solo dopo aver stabilito una connessione con il server.

6.5.3 Funzionamento dell'applicazione

All'avvio dell'applicazione, all'utente viene presentata una Input Field, in cui poter inserire l'indirizzo IP del server e un bottone per poter confermare tale indirizzo. Quando avviene il click sul pulsante "Enter", l'IP immesso viene controllato e se risulta essere valido, viene utilizzato per stabilire una nuova connessione con il server e infine viene mostrato l'ologramma. Altrimenti, nel caso in cui l'indirizzo non risulti essere corretto, viene restituito un messaggio di errore.

A questo punto l'utente, se la connessione è andata a buon fine, è in grado di visualizzare l'oggetto condiviso e di poter interagire con esso.

In particolare, affinché l'interazione possa avvenire, all'oggetto Unity che rappresenta l'ologramma, sono stati associati tre script forniti dall'MRTK: `NearInteractionGrabbable`, `ObjectManipulator` e `BoundsControl`.

Questi script, precisamente, oltre che consentire la manipolazione effettiva dell'ologramma, presentano una serie di metodi che consentono di individuare se la posizione, la scala o la rotazione dell'ologramma stanno cambiando; a tali metodi e possibile associare degli eventi, facendo si che quando si verifica una modifica, possa essere eseguita un determinata funzione. Nella figura sottostante 6.4, infatti, possiamo vedere come viene utilizzato l'oggetto `Script` dal componente `BoundsControl`, per richiamare i metodi del componente `UpdateObject` assegnato a tale elemento, al verificarsi di un evento che modifica: la scala o la rotazione dell'oggetto, mentre l'evento di spostamento dell'oggetto viene gestito tramite il componente `ObjectManipulator`.

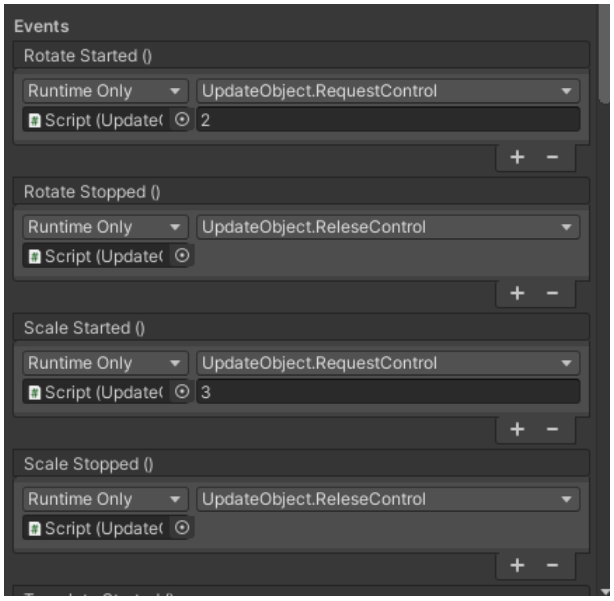


Figura 6.4: Metodi e eventi per l'aggiornamento dell'oggetto nel componente Bounds Controls

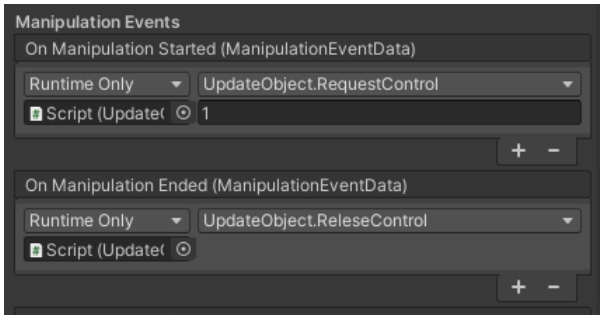


Figura 6.5: Metodi e eventi per l'aggiornamento dell'oggetto nel componente Object Manipulator

Un'altra funzionalità che lo script `BoundsControl` offre all'utente, è la possibilità di visualizzare con maggior chiarezza, le manipolazioni che è possibile effettuare, racchiudendo l'ologramma in dei confini che formano una box, visibile nella figura 6.6. Questa box presenta specifici elementi grafici, che consentono di intuire il tipo di modifica che l'utente può apportare, ad esempio: se si seleziona il quadratino degli angoli è possibile modificare la scala dell'oggetto, se invece si sceglie il puntino a metà del bordo è possibile ruotare l'oggetto.

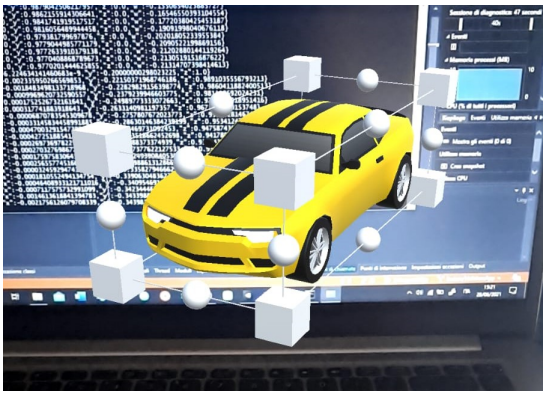


Figura 6.6: Interfaccia per la manipolazione

Se all'utente viene mostrato l'ologramma all'interno della bounding box come mostrato in figura 6.6, significa che attualmente nessun altro utente sta effettuando operazioni su di esso e per questo motivo il partecipante ha la possibilità di richiedere e prendere il controllo sull'oggetto. Altrimenti se l'oggetto viene mostrato come nella figura 6.7 , significa che un altro utente, identificato dal colore della bounding box, sta effettuando la manipolazione e detiene il controllo dell'oggetto. Per tanto fino a quando egli non lo rilascia, non sarà possibile agli altri partecipanti manipolare l'ologramma.

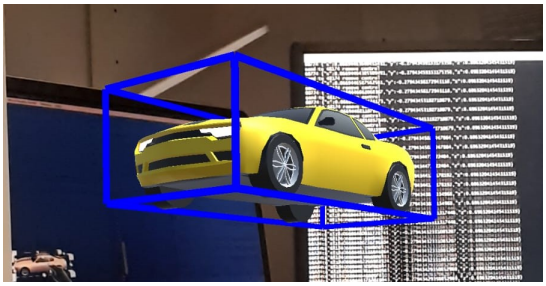


Figura 6.7: Interfaccia durante la manipolazione di un altro utente

6.5.4 Gestione dello stato dell'ologramma

Lo stato dell'ologramma viene mantenuto all'interno dello script `Update Object` ed è definito da una enum contenuta all'interno di questa classe, in particolare, gli stati presenti sono:

- Mooving: indica che la posizione dell'oggetto è in cambiamento.
- Rootating: indica che la rotazione dell'oggetto sta cambiando.
- Scailing: indica che la dimensione dell'oggetto sta cambiando.
- None: è lo stato che l'oggetto assume quando nessuna manipolazione è in atto

Se si guardano le figure 6.4 e 6.5, si può notare che gli eventi associati ai metodi dei componenti `BoundsControl` e `ObjectManipulator`, hanno come scopo quello di modificare lo stato dell'ologramma. Ad esempio, quando inizia l'operazione di modifica della rotazione, essa viene notificata alla classe `UpdateObject`, la quale si occupa di inviare al server, ad istanti regolari di tempo, aggiornamenti sulla rotazione, fino a quando l'operazione non termina e viene rilasciato il controllo, grazie alla chiamata al metodo `ReleseControl()`.

Infine, quando l'operazione di modifica dell'ologramma inizia e viene notificato il cambiamento di stato, la classe `UpdateObject` richiamando i metodi della classe `DataSender`, si occupa di inviare la richiesta di controllo sull'oggetto, la quale contiene il colore che è stato assegnato all'utente all'instaurazione della connessione, che verrà notificato agli altri clients, durante l'invio da parte del server, del messaggio di rimozione dei controlli per la manipolazione.

6.6 Sviluppo in Unity del progetto di esperienza condivisa nello stesso ambiente

Dopo aver visto il progetto che realizza un'esperienza condivisa in cui gli utenti possono partecipare stando in ambienti diversi, vediamo adesso, la realizzazione di un esperienza in cui invece, gli utenti si trovano nello stesso luogo e sono in grado di vedere l'oggetto condiviso nello stesso punto e con una prospettiva diversa in base alla loro posizione nell'ambiente.

6.6.1 Quali sono le tecnologie che si è deciso di utilizzare e perché

Anche in questo caso si è deciso di utilizzare versioni di Unity diverse per i diversi porgetti, questo perché al fine di riuscire a realizzare, l'obbiettivo

di poter vedere un ologramma nello stesso punto dell'ambiente si è deciso di impiegare dei targets, come descritto nel capitolo due. In particolare, sono stati utilizzati i targets messi a disposizione da Vuforia, il cui package è importabile in Unity.

Tuttavia questa tecnologia, ha presentato dei problemi di funzionamento per Android nella versione LTS 2020 di Unity, generando un problema con il deployment sul dispositivo relativo alla fotocamera, la quale non risulta venire attivata durante l'utilizzo dell'applicazione e questo fa sì che, invece del mondo esterno, venga mostrato all'utente un "black-screen".

Questo problema è risultato essere un bug conosciuto nell'utilizzo di Vuforia in Unity 2020 e nonostante si sia cercato di risolverlo, dopo diversi tentativi, basandosi anche su discussioni di forum ufficiali della libreria, che però non sono andati a buon fine, si è deciso di passare alla versione LTS 2019 di Unity, dove questo problema non sussiste.

Per HoloLens invece, si è deciso di continuare nell'utilizzo della versione Unity 2020.3 LTS, consigliata da Microsoft, per i motivi illustrarti nella sezione 6.5.1.

Anche in questo caso quindi, sono stati realizzati due progetti separati, uno per lo sviluppo in Android e uno invece per HoloLens, ad ogni modo pure in questo scenario è consigliabile utilizzare progetti diversi, per rendere più agevole il deployment, altrimenti ogni volta che si effettua la build dell'applicazione è necessario riconfigurare tutte le impostazioni di Unity, previste per l'una o per l'altra piattaforma.

Sia per questo progetto che per quello precedente emerge uno svantaggio nell'utilizzo di Unity, dovuto appunto alle diverse configurazioni del sistema e alla struttura che questo ambiente di sviluppo presenta, il quale non rende semplice il deployment su dispositivi diversi attraverso l'utilizzo di un unico progetto.

6.6.2 Utilizzo di un'interfaccia di interazione diversa

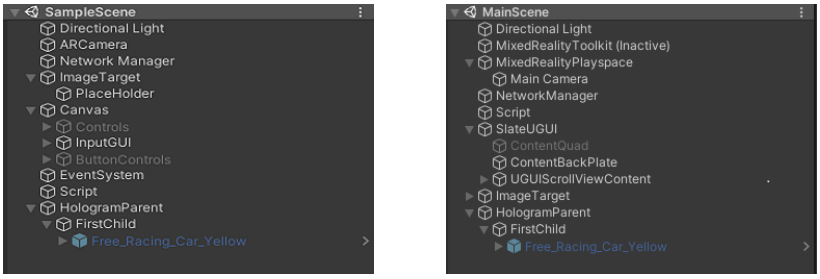
Per lo sviluppo in HoloLens, come nel caso precedente, si è deciso di utilizzare l'MRTK, il quale mette a disposizione tutti gli elementi visti in precedenza, utili a creare un'applicazione di Mixed Reality.

Per Android però, in questo caso, non si ha avuto la possibilità di utilizzare tale tecnologia, in quanto l'utilizzo combinato di Vuforia e il Mixed Reality Toolkit, sui dispositivi di Augmented Reality, che sfruttano AR Foundation, genera dei conflitti nella telecamera, questo perché l'MRTK per poter funzionare in Android, effettua degli opportuni settaggi per la fotocamera che vanno in contrasto con quelli, invece, richiesti da Vuforia per poter funzionare correttamente.

Quindi, non avendo la possibilità di avere un'interfaccia comune, per l'interazione lato Android, si è deciso di utilizzare una GUI specifica costruita tramite gli elementi di Unity, che consenta la manipolazione dell'ologramma.

In particolar modo, si è deciso di utilizzare degli sliders che rappresentino e modifichino, la rotazione sull'asse: x, y o z e la scala dell'oggetto. Tali sliders inoltre, vengono aggiornati in modo invisibile all'utente, durante la manipolazione ad opera di un altro partecipante all'esperienza, in modo tale che come per HoloLens, l'utente che partecipa attraverso un dispositivo Android, abbia la possibilità di riprendere la modifica dell'oggetto dal punto in cui colui, che precedentemente aveva il controllo, l'ha lasciata.

6.6.3 Descrizione degli elementi nelle diverse scene



(a) Scena progetto Android (b) Scena progetto HoloLens

Figura 6.8: Scene applicazione

Come si può vedere dalle figure riportate, gli elementi presenti nelle due diverse scene differiscono leggermente fra loro, in HoloLens abbiamo l'aggiunta dell'MRTK mentre in Android viene utilizzata la `ARCamera` di Vuforia, la quale presenta appositi script forniti dalla libreria stessa, che a runtime consentono l'individuazione dei targets.

I entrambi i casi, gli elementi comuni e principali alle due scene sono:

- `NetworkManager`: il quale si occupa di, gestire la rete lato Unity e di consentire la connessione di un nuovo client con il server.
- `ImageTarget`: rappresenta il target fisico che verrà utilizzato nel mondo reale, per visualizzare l'ologramma in una certa posizione.
- `Canvas` per Android e `SlateGUI` per HoloLens: i quali contengono gli oggetti, che costituiscono la GUI che verrà mostrata all'utente durante la fase iniziale e nel caso di Android anche per controllare l'ologramma.

- `Script`: detiene gli script principali, che sono stati realizzati in Unity per la gestione dei diversi elementi .

6.6.4 Problemi riscontrati nel posizionamento dell'ologramma e come sono stati risolti

L'utilizzo di Vuforia in Unity, consente di inserire nel ambiente di sviluppo, degli oggetti che rappresentino i targets, nel nostro caso si è deciso di utilizzare delle immagini che svolgessero questa funzione e tali oggetti sono rappresentati nella scena dall'elemento `ImageTarget`, fornito direttamente da Vuforia. Per far sì che un ologramma appaia quando l'apposito marcatore viene rilevato, è sufficiente porre l'oggetto che lo rappresenta, come figlio di `ImageTarget`.

Seguendo queste istruzioni però si è notato che a runtime, la posizione dell'ologramma cambia nel tempo, in quanto la fotocamera aggiusta continuamente la posizione del target durante il tracciamento, così facendo però l'immagine dell'ologramma associato risulta essere traballante e la sua posizione non stabile. Di conseguenza, si è deciso di rimuovere l'oggetto come figlio del target e sostituirlo con un altro elemento "Placholder", dotato di un apposito script il quale fa si che, quando il target viene rilavato l'oggetto che non è più figlio diretto venga mostrato.

Questo però non risolve tutti i nostri problemi, in quanto quando viene mostrato l'ologramma, vorremmo che esso sia orientato rispetto al target in un certo modo e che quando questi si sposti, anche l'ologramma segua lo spostamento del marcatore, come se fosse ancorato a lui, pur non essendolo direttamente.

Per seguire questi spostamenti però non possiamo muovere direttamente il nostro ologramma, perché esso rappresenta l'oggetto condiviso e modificare direttamente la sua posizione non sarebbe corretto, in quanto influenzerebbe anche quella degli altri utenti collegati, di conseguenza esso è stato posto come oggetto figlio di un elemento `HologramParent` la cui posizione sarà aggiornata in relazione a quella del target, tramite un opportuno script. In questo modo il nostro ologramma seguirà il target.

Tuttavia, il nostro oggetto condiviso è ancora soggetto agli aggiustamenti di posizione del target, perché le sue coordinate vengono espresse in relazione a quelle del padre, le quali vengono prese dal marcatore e un loro cambiamento fa si che anche il figlio venga spostato senza che però esse cambino effettivamente, cioè ad esempio: se poniamo un oggetto padre alle coordinate (1,1,1) del mondo e creiamo un oggetto figlio di questi che presenta coordinate (0,0,0), queste non sono in relazione a quelle del mondo ma a quelle del padre, se ora spostiamo il padre in posizione (2,1,1) esso cambierà posizione rispetto al mondo e il figlio

lo seguirà, ma le sue coordinate non cambieranno, perché esso si troverà ancora nel punto (0,0,0), rispetto al padre.

Abbiamo quindi bisogno di un padre, per il nostro ologramma, che mantenga fisse le coordinate, ma che allo stesso tempo ci consenta di seguire la posizione del marcatore. Per fare questo, è stato creato l'oggetto `FirstChild` come primo figlio di `HologramParent`, il quale per quanto detto prima seguirà la posizione e la rotazione del padre, senza che però le sue coordinate cambino effettivamente e al suo interno come figlio diretto possiamo mettere il nostro ologramma, che non avendo più un genitore che modifica la sua posizione, non è più soggetto a continui aggiustamenti del marcatore.

Così facendo, abbiamo fatto sia in modo che il nostro ologramma segua la posizione e la rotazione del target, senza che le sue vengano effettivamente modificate e che l'immagine dell'ologramma risulti essere stabile e non traballante.

6.6.5 Funzionamento dell'applicazione

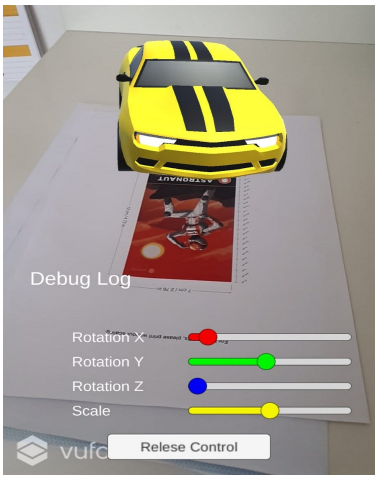
All'avvio dell'applicazione, viene mostrata all'utente un'interfaccia in cui poter immettere l'indirizzo IP del server, a cui potersi connettere, per poter inviare e ricevere le informazioni. Una volta che la connessione è andata a buon fine l'utente potrà visualizzare l'ologramma oggetto della condivisione, attraverso l'opportuno inquadramento del target, che deve essere presente e ben visibile nell'ambiente.

In questo caso si è data la possibilità ai partecipanti di modificare la rotazione e la scala dell'oggetto, attraverso le differenti interfacce grafiche.

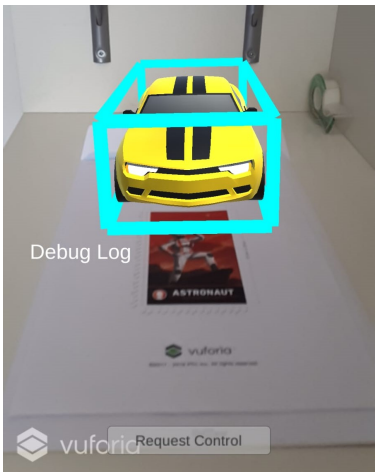
Lato HoloLens è stato utilizzato, come nel caso precedente lo script `BoundsControl`, che consente la modifica dei valori dell'ologramma attraverso l'interazione con una bounding box, come mostrato nella figura 6.6 e metodi che consentono di individuare quando avvengono le operazioni di modifica della rotazione o della scala dell'elemento a cui esso è associato, facendo si, anche in questo caso, che quando un utente effettua una manipolazione gli altri possano visualizzare il suo colore, attraverso una bounding box che circonda l'ologramma.

Lato Android invece, è stata realizzata un'opportuna GUI, tramite gli elementi di Unity, che presenta degli sliders di colori diversi, con cui l'utente può interagire per poter modificare la rotazione su un asse: x, y e z o la scala dell'oggetto, com'è possibile vedere dalla figura sottostante.

(a) schermata richiesta di controllo



(b) schermata gestione ologramma



(c) schermata controllo da parte di un altro utente

Figura 6.9: Gestione dell'interazione lato Android

Anche in questo caso, si vuole evitare che più utenti modifichino contemporaneamente l'ologramma, per fare questo, in Android è stato inserito un pulsante "Request Control" che può essere premuto dall'utente, quando nessun altro sta manipolando l'oggetto, per averne il controllo, altrimenti esso risulta essere disattivato e come per HoloLens l'ologramma viene mostrato all'interno di una box, del colore dell'utente che lo sta modificando come è possibile vedere dalla figura 6.9c, quando invece l'utente detiene il controllo dell'oggetto, gli verranno mostrati gli opportuni elementi dell'interfaccia grafica che li consentiranno di modificarlo e il pulsante per poter rilasciare il controllo e far si

che un altro utente possa interagire con l'ologramma.

Per quanto riguarda invece HoloLens 2, l'interfaccia presentata è la stessa mostrata nelle figure 6.7 e 6.6, con l'unica differenza che, in questo progetto la posizione dell'ologramma sarà relativa al target presente nell'ambiente e la richiesta di controllo della manipolazione viene gestita come visto per il progetto precedente, tramite il componente `BoundsControl`, attraverso la chiamata di metodi che modificano lo stato dell'ologramma, a seconda dell'evento che viene generato.

6.7 Considerazioni finali e possibili sviluppi futuri

Al termine dell'analisi e descrizione dei porgetti che sono stati realizzatati, possiamo quindi notare che, sebbene **OpenXR ci dia la possibilità di creare un software che sia portabile sui diversi dispositivi, il quale da un certo punto di vista avvantaggia l'interoperabilità di questi, essa non ci mette ancora a disposizione degli elementi veri e propri che ci cosentino di realizzarla solamente tramite il suo utilizzo, senza dover integrare ulteriori tecnologie**.

Sarebbe molto utile infatti, se i diversi providers si mettessero d'accordo per creare delle estensioni dell'API, che cosentino un collegamento fra i sistemi forniti, ma attualmente queste funzionalità non sono ancora presenti.

Tramite questi porgetti, abbiamo comunque visto come poter realizzare esperienza condivise con gli strumenti che sono disponibili oggi e il fatto di aver connesso fra loro in un'unica esperienza un dispositivo Android, dotato di uno schermo 2D e un visore HoloLens, dotato invece di un sistema di consapevolezza spaziale e un visualizzazione 3D del contenuto olografico, ha reso l'impresa un po' più complessa di quello che si avrebbe avuto, se invece avessimo deciso di sviluppare un'applicazione in cui possono partecipare solamente utenti dotati di visori per la Mixed Reality, come ad esempio Magic Leap e HoloLens 2. Avendo tali sistemi, un maggior numero di elementi in comune, l'applicazione che si sarebbe dovuta realizzare sarebbe stata abbastanza simile, soprattutto quella relativa a un'esperienza condivisa nello stesso ambiente, che nel nostro caso invece, ha richiesto l'utilizzo di due interfacce differenti per l'interazione.

Ad ogni modo, queste applicazioni che sono state realizzate, possono costituire un punto di partenza per la costruzione di qualcosa di più complesso, possibili sviluppi possono prevedere: l'integrazione di altri dispositivi, l'aggiunta a runtime di altri ologrammi da poter manipolare, la possibilità di realizzare un'esperienza asincrona e realizzare altri tipi di contenuti condivisi, che pos-

sono essere associati agli ologrammi, ecc. le possibilità sono davvero molte e i vantaggi che possono nascere dall'utilizzo di applicazioni di esperienze collaborative sono davvero infinite, tramite questi nuovi strumenti che grazie alle loro potenzialità, prenderanno sempre più piede nel futuro.

Conclusioni

Alla fine di tutto, sebbene questi nuovi strumenti per la Mixed, Virtual e Augmented Reality, prenderanno sempre più piede nel nostro futuro, le tecnologie che ci consentono di sviluppare oggi delle applicazioni interoperabili e multipiattaforma, risultano essere ancora acerbe, esse ci forniscono sì un aiuto, ma che richiede comunque opportuni adattamenti per poter realizzare la nostra idea di applicazione.

OpenXR per quanto sia un'API abbastanza potente, presenta delle difficoltà di utilizzo nello sviluppo dei programmi tramite degli ambienti come Unity, risulta essere invece più utile lato venditori, per sviluppare i sistemi stessi per le diverse realtà. Inoltre, come accennato nei precedenti capitoli che analizzavano questa tecnologia, attualmente essa è più diretta ad uniformare i devices che utilizzano le applicazioni, basandosi molto sull'astrazione degli input dei dispositivi e la azioni che vengono compiute. Attualmente essa ci da la possibilità si di creare un software che possa essere eseguito su differenti piattaforme, ma per ottenere effettivamente un'interoperabilità tra i diversi sistemi, durante l'utilizzo dell'applicazione, dobbiamo utilizzare anche altri strumenti, quindi OpenXR da sola non basta.

Anche Unity stesso, come ambiente di sviluppo, presenta degli svantaggi nella realizzazione di questi tipi di programmi, che sono emersi diverse volte durante le precedenti spiegazioni.

Quello che sembra ancora mancare oggi, che dovrebbe costituire il punto di partenza, per la realizzazione di applicazioni valide, che possano essere portabili e interconnettere i dispositivi fra loro, è un idoneo ambiente di sviluppo, che presenti funzionalità e impostazioni interamente dedicate alla Mixed, Augmented o Virtual Reality e che allo stesso tempo risulti essere semplice da utilizzare, senza l'aggiunta di troppe funzionalità come invece accade proprio in Unity e creare questo ambiente con già il supporto a tutte le tecnologie che ci servono, fra cui appunto OpenXR.

Durante lo sviluppo dei diversi progetti e lo studio per questa tesi, si è arrivati alla conclusione che la necessita di un idoneo ambiente di sviluppo crescerà sempre di più nel tempo, ma a mano a mano che si procrastina questa scelta nel futuro, sempre più complesso e difficile diventerà l'implementazione

lato Unity, con sempre più plugin da aggiungere, impostazioni da settare, versioni da installare o modificare ecc.

Durante invece, l'utilizzo di Hololens 2 per i diversi progetti e sempre per lo studio di questa tesi, si è capito quanto sia potente uno strumento di questo genere e tutti i benefici e gli aiuti che potrebbero derivare del suo utilizzo e dallo sfruttamento al meglio delle sue possibilità. Quasi tutti i libri per le diverse realtà, quando iniziano a descrivere questi strumenti, parlano sempre di una rivoluzione e all'inizio una persona che non gli ha mai visti o provati, potrebbe essere titubante, ma avendo avuto la possibilità di provarli e di toccarli con mano, posso confermare quello che viene scritto su questi testi: siamo di fronte a una rivoluzione, un cambiamento che ci porterà ancora una volta ad evolverci come esseri umani.

Se le predizioni sono vere e entro il 2030 gireremo tutti con il nostro paio di smartglasses in testa, allora è tempo di iniziare a pensare e a lavorare su nuovi tipi di programmi che ci diano la possibilità di interagire fra noi in un nuovo modo, tramite questi nuovi strumenti, che però ci consentiranno di sfruttare un tipo di interazione naturale che siamo già predisposti ad utilizzare, toccando gli oggetti del mondo olografico come se fossero reali, ma per far si che tutto funzioni e che ci sia effettivamente una rivoluzione, le applicazioni che utilizzeremo dovranno supportare le diverse piattaforme e consentire una forma di interoperabilità fra i diversi dispositivi.

Ringraziamenti

La prima persona che devo ringraziare, che è stata veramente un punto di riferimento per lo sviluppo di questa tesi e che mi ha seguito durante tutto il mio percorso di tirocinio, di studio e di realizzazione delle sperimentazioni è il professor Angelo Croatti.

La seconda persona che devo ringraziare, i cui spunti di riflessione sono risultati essere determinanti durante tutto il percorso, che mi ha consentito l'accesso all'utilizzo di HoloLens 2 e lo sviluppo di una tesi che riguardasse quest'argomento, nonché durante le lezioni mi ha appassionato alla materia "Sistemi Embedded e Internet-Of-Things" è il professor Alessandro Ricci.

Inoltre un ringraziamento particolare, va rivolto anche al collega di tirocinio Filippo Vissani, con cui ho esplorato una parte delle funzionalità di HoloLens 2 e di Unity per il supporto alla Mixed Reality.

Infine un ultimo ringraziamento speciale, va alla mia famiglia per avermi sostenuto durante il periodo degli studi, alla mia collega Elena Yan compagna di studi e di progetti, che si è occupata di recensire questa tesi e per ultimi, ma non per importanza, tutti gli amici e compagni di università che hanno seguito il mio percorso durante questi anni.

Bibliografia

[1] Collabora. Unifying reality: Building experiences with openxr. https://www.youtube.com/watch?v=F6jZCwko1Qs, 2020.

[2] Robert Scoble Irena Cronin. *The Infinite Retina*. Packt, 2020.

[3] Julie Carmignani, Borko Furht, Marco Anisetti, Paolo Ceravolo, Ernesto Damiani, Misa Ivkovic. Project glass: An extension of the self. *Springer Science + Buisness Media*, 51:341–377, 2013.

[4] Khronos. The khronos group. https://www.khronos.org/. Accessed June 2021.

[5] Khronos. The openxr specification. https://www.khronos.org/registry/OpenXR/specs/1.0/html/xrspec.html. Accessed June 2021.

[6] Khronos. A look at openxr - siggraph 2019. https://www.youtube.com/watch?v=cnNvR-Tp5xg&t=925s, 2019.

[7] Microsoft. Building and deploying to android and ios via ar foundation. https://docs.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/supported-devices/using-ar-foundation?view=mrtkunity-2021-05. Accessed June 2021.

[8] Microsoft. Choosing a unity version and xr plugin. https://docs.microsoft.com/en-us/windows/mixed-reality/develop/unity/choosing-unity-version. Accessed June 2021.

[9] Microsoft. Open xr samples for mixed reality developers. https://github.com/microsoft/OpenXR-MixedReality#openxr-preview-extensions. Accessed June 2021.

[10] Microsoft. Hololens (1st gen) sharing 240: Multiple hololens devices. https://docs.microsoft.com/en-us/windows/mixed-reality/develop/unity/tutorials/holograms-240, 2019.

[11] Microsoft. Shared experiences in mixed reality. https://docs.microsoft.com/en-us/windows/mixed-reality/develop/platform-capabilities-and-apis/shared-experiences-in-mixed-reality, 2019.

[12] Oculus. No more hacks: Building cross-device ue4 apps with openxr. https://www.youtube.com/watch?v=F6jZCwko1Qs, 2019.

[13] Jon Peddie. *Augmented Reality - Where We Will All Live*. SPRINGER, 2017.

[14] Unity. About ar foundation. https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@4.1/manual/index.html. Accessed June 2021.

[15] Unity. Open xr plugin. https://docs.unity3d.com/Packages/com.unity.xr.openxr@0.1/manual/index.html. Accessed June 2021.

[16] Unity. Introducing mrtk. https://docs.microsoft.com/en-us/windows/mixed-reality/develop/unity/mrtk-getting-started, 2019.

[17] Vuforia. Vuforia developer portal. https://developer.vuforia.com/. Accessed June 2021.