FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Multi-uživatelský systém pro mixed reality
Student:	Martin Gregor
Vedoucí:	Ing. Jan Buriánek
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Počítačová grafika
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Systémy VR brýlí využívají pro určení pozice uživatele v prostoru systém „inside-out tracking" založený na speciálních kamerách integrovaných přímo v brýlích. Cílem práce je využití těchto kamer pro augmentaci virtuální 3D scény s reálnou scénou (tzv. mixed reality). Výsledný systém navíc umožní sdílet zobrazovanou 3D scénu s více lokálními uživateli najednou.

Postup:

1) Prostudujte dostupné systémy inside-out trackingu pro VR brýle.
2) Zvolte vám dostupný VR systém, který umožní využití sledovacích kamer pro mixed reality. Implementujte prototyp aplikace, která vizualizuje vybranou 3D scénu a navíc ji sdílí s více lokálními uživateli najednou.
3) Otestujte aplikaci v reálném prostředí s více uživateli najednou.
4) Práci zpracujte včetně obrazových příloh a komentovaného postupu implementace.

Elektronicky schválil/a Ing. Radek Richtr, Ph.D. dne 14. února 2022 v Praze.

Bakalářská práce

MULTI-UŽIVATELSKÝ SYSTÉM PRO MIXED REALITY

Martin Gregor

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Jan Buriánek
12. května 2022

Obsah

Poděkování vi

Prohlášení vii

Abstrakt viii

Seznam zkratek x

1 Úvod 1
- 1.1 Motivace . . . 1
- 1.2 Prospěšnost . . . 2
- 1.3 Cíle . . . 3

2 Analýza 5
- 2.1 Analýza dostupných VR systémů s inside-out trackingem . . . 5
  - 2.1.1 Přehled dostupných VR systémů s inside-out trackingem . . . 6
- 2.2 Analýza možných řešení synchronizace scény . . . 12
  - 2.2.1 Analýza problému . . . 12
  - 2.2.2 Hlavní myšlenka algoritmu . . . 15
  - 2.2.3 Řešení síťové komunikace . . . 16

3 Implementace prototypu 19
- 3.1 Volba 3D scény . . . 19
- 3.2 Implementace simulační logiky v Unity . . . 20
  - 3.2.1 MainScript . . . 22
  - 3.2.2 AnchorSpawner . . . 25
  - 3.2.3 GameStateSync . . . 27
  - 3.2.4 InputSwitchHelper . . . 30

4 Multi-uživatelské testování v reálném prostředí 33
- 4.1 Průběh testování . . . 33

5 Závěr 37

Obsah přiloženého média 41

Seznam obrázků

1.1 Monthly-connected Headsets on Steam, leden 2022 [1] . . . 2
1.2 Headset Marketshare on Steam, leden 2022 [1] . . . 2

2.1 Windows Mixed Reality - Acer [4] . . . 6
2.2 Oculus Quest 2 [5] . . . 7
2.3 Pico Neo 3 Link [7] . . . 8
2.4 HoloLens 2 [9] . . . 9
2.5 HP Reverb G2 [10] . . . 9
2.6 HTC Vive Cosmos [11] . . . 10
2.7 XTAL 3 Mixed Reality [14] . . . 10
2.8 Varjo XR-3 [15] . . . 11
2.9 Guardian [17] . . . 12
2.10 Spatial Anchor [23] . . . 15
2.11 Unity klient [28] . . . 16
2.12 Cloudová infrastruktura [29] . . . 17
2.13 Srovnání síťových řešení [30] . . . 18

3.1 Scéna v Unity . . . 19
3.2 Zapékání osvětlení do textury v grafickém editoru Blender . . . 20
3.3 Hierarchie scény v Unity . . . 21
3.4 MainScript - Lifecycle . . . 22
3.5 MainScript - Update - kontrola příchozích akcí . . . 23
3.6 MainScript - Update - kontrola příchozího offsetu scény . . . 23
3.7 MainScript - Update - kontrola příchozího offsetu HMD . . . 24
3.8 MainScript - akce . . . 25
3.9 AnchorSpawner - Awake . . . 26
3.10 AnchorSpawner - PlaceAnchor . . . 26
3.11 Spatial Anchor sample . . . 26
3.12 GameStateSync - modelové třídy . . . 27
3.13 GameStateSync - hlavní model . . . 27
3.14 GameStateSync - aktualizace lokálních proměnných . . . 28
3.15 GameStateSync - nový model . . . 29
3.16 GameStateSync - nový klient . . . 29
3.17 GameStateSync - nový host . . . 30
3.18 GameStateSync - nová akce . . . 30
3.19 InputSwitchHelper - Update . . . 31

4.1 Kotva ve scéně v levé spodní části - MR . . . 33
4.2 Volný postoj - MR . . . 34
4.3 Mávání - MR . . . 34
4.4 Viditelná hlava - MR . . . 35
4.5 Uzavírající se výtah - MR to VR . . . 35
4.6 Sledování rukou - VR . . . 36

4.7 Druhý uživatel - VR 36
4.8 Zapínaní simulace rukou - MR 36

Seznam tabulek

2.1 HMDs s inside-out trackingem - jaro 2022 7

Seznam výpisů kódu

Chtěl bych poděkovat svému vedoucímu práce Ing. Janu Buriánkovi za jeho cenné rady. Dále potom své přítelkyni, která mi asistovala během testování prototypu a dovedla mě do této finální fáze mého šestiletého bakalářského studia, které původně mělo býti tříleté.

Prohlášení

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 12. května 2022 .....

Abstrakt

Bakalářská práce řeší návrh a implementaci aplikace pro systémy virtuální reality disponující inside-out trackingem a schopností realizovat mixed reality. Aplikace umožňuje více uživatelům, kteří se nachází lokálně ve stejné místnosti, interagovat v rámci jedné 3D scény a to tak, že se uživatelé vzájemně vidí v podobě virtuálních avatarů. Pozice a orientace těchto avatarů ve virtuální 3D scéně potom odpovídají pozicím a orientacím ve světě reálném.

Práce řeší zejména způsob, jakým zajistit, aby byl počátek souřadné soustavy stejný pro všechny uživatele. K tomu byly zvoleny prostorové kotvy v podobě technologie zvané Spatial Anchors, která je součástí Oculus SDK. Tyto prostorové kotvy si jednotliví uživatelé rozmístí po místnosti na předem určená místa tak, aby se minimalizovala odchylka jednotlivých pozic a orientací těchto kotev napříč všemi systémy, jenom tak je možné zajistit věrohodné mapování scény pro všechny uživatele. Kromě offsetů uživatelů (HMDs a ovladače) od jednotlivých kotev je nutné po síti synchronizovat také stav 3D scény, k tomu byl zvolen síťový framework Normcore, který je určen pro herní engine Unity.

Zvolená technologie Spatial Anchors z Oculus SDK v kombinaci se síťovým frameworkem Normcore umožnila úspěšné vyřešení zadaného problému, tedy implementaci prototypu aplikace, která vizualizuje zvolenou 3D scénu a sdílí ji s více lokálními uživateli najednou.

Jelikož zatím neexistuje oficiální řešení lokální synchronizace více uživatelů v rámci Oculus SDK, představuje tento prototyp dobrý základ pro vývoj produkčních aplikací, které by o podobnou funkcionalitu usilovaly.

V příloze práce je k dispozici prototyp psaný v herním enginu Unity, ze kterého lze vycházet v případě realizace aplikací s podporou více lokálních uživatelů.

Klíčová slova VR, AR, MR, XR, virtuální realita, rozšířená realita, inside-out tracking, multiuživatelský systém, Oculus, Quest 2

Abstract

The bachelor thesis deals with the design and implementation of an application for virtual reality systems with inside-out tracking and mixed reality capability. The application allows multiple users, who are locally located in the same room, to interact within a single 3D scene by allowing users to see each other in the form of virtual avatars. The positions and orientations of these avatars in the virtual 3D scene then correspond to the positions and orientations in the real world.

In particular, the thesis addresses how to ensure that the origin of the coordinate system is the same for all users. To do this, spatial anchors were chosen in the form of a technology called Spatial Anchors, which is part of the Oculus SDK. These spatial anchors are deployed by individual users around the room at predetermined locations to minimize the variation in the individual positions and orientations of these anchors across all systems, only in this way can a believable scene mapping be provided for all users. In addition to the offsets of users (HMDs and

controllers) from each anchor, the 3D scene state must also be synchronized over the network, for this the Normcore network framework was chosen, which is designed for the Unity game engine.

The chosen technology, Spatial Anchors from the Oculus SDK, combined with the Normcore networking framework, enabled the successful solution of the specified problem, i.e. the implementation of a prototype application that visualizes the selected 3D scene and shares it with multiple local users simultaneously.

As there is not yet an official solution for local multi-user synchronization within the Oculus SDK, this prototype provides a good basis for the development of production applications that would strive for similar functionality.

A prototype written in the Unity game engine is provided in the appendix of this thesis, which can be used as a basis for implementing applications with support for multiple local users.

Translated with www.DeepL.com/Translator (free version)

Keywords VR, AR, MR, XR, virtual reality, mixed reality, inside-out tracking, multi-user system, Oculus, Quest 2

Seznam zkratek

VR	Virtual reality
AR	Augmented reality
MR	Mixed reality
XR	Extended reality
HMD	Head mounted display
ADB	Android Debug Bridge
API	Application programming interface
SDK	Software development kit

Kapitola 1
Úvod

Během posledních let došlo k výraznému nárůstu zájmu o systémy virtuální reality. Hlavní hnací silou byly v tomto ohledu systémy Oculus Quest a jeho následník Oculus Quest 2 společnosti Meta. Oculus Quest přišel jako první samostatně fungující HMD se systémem inside-out trackingu, který od uživatele nevyžaduje přítomnost externích kamer / sensorů pro sledování jeho polohy a zároveň ani externí počítač pro renderování scény. Tím znatelně snižuje komplexitu přípravy systému samotného pro jeho použití. Stačí si jej nasadit a kamery integrované v HMD se sami postarají o správné určení polohy uživatele.

1.1 Motivace

Pojmem HMD (Head mounted display) rozumíme systém virtuální reality, VR brýle, či hovorově headset, tedy ten předmět, který si člověk nasadí na hlavu.

Podívejme se prvně na vývoj počtu aktivních HMDs na herní platformě Steam, konkrétně takových, které byly aktivní alespoň jednou do měsíce.

Z následujícího grafu je patrné, že přibližně od léta roku 2016 dochází k exponenciálnímu růstu počtu aktivních zařízení na této na trhu dominantní platformě.

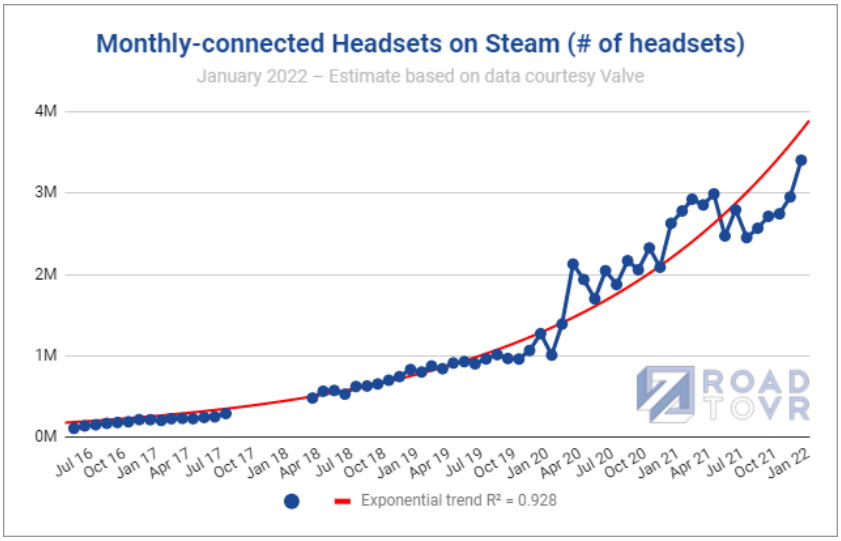
Zatímco se na konci roku 2019 jejich počet pohyboval kolem jednoho milionu, je velmi pravděpodobné, že se v současné chvíli (jaro 2022) blíží milionu již čtvrtému.

Zajímavé je také zastoupení jednotlivých HMDs. Podle posledního průzkumu z dubna roku 2022 je procentuální rozložení trhu následující: [2]

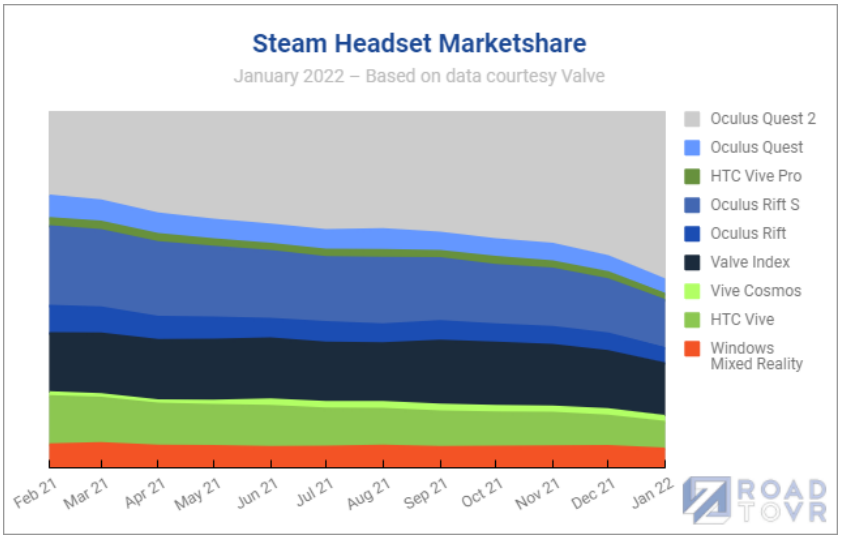
- 47.97% - Oculus Quest 2
- 15.35% - Valve Index
- 11.07% - Oculus Rift S
- 7.02% - HTC Vive
- 4.69% - Windows Mixed Reality
- 13.90% - Other

Je tedy zřejmé, že se jedná o velmi aktuální, dynamicky se rozvíjející oblast informačních technologií. Stále více lidí bude svůj čas trávit ve virtuální realitě, podobně jako jej dnes tráví hraním her na mobilních telefonech či herních konzolích.

Kromě zábavy se virtuální realita bude stále více využívat i ke kreativní tvorbě a postupně se v různých formách dostane do většiny průmyslových odvětví (zde spíše v podobě mixed či



Obrázek 1.1 Monthly-connected Headsets on Steam, leden 2022 [1]



Obrázek 1.2 Headset Marketshare on Steam, leden 2022 [1]

augmented reality), proto se podle mě jedná o perspektivní oblast a byla pro mě jasnou motivací pro volbu tématu této bakalářské práce.

1.2 Prospěšnost

Implementace multi-uživatelského systému pro mixed reality, tedy možnost sdílet stejný fyzický prostor s více uživateli rozšířený o virtuální objekty, kdy každý má svůj HMD a jejich pozice jsou korektně synchronizované, je zatím oficiálně nevyřešený problém (mezi samostatně fungujícími HMDs s inside-out trackingem neexistuje zatím žádné API či SDK, které by podobný problém řešilo), který má ale potenciálně široké uplatnění v praxi.

Mezi oblasti, kde by tento systém mohl pomoci, lze zařadit například firemní sektor - různé prezentace 3d grafů a 3d modelů objektů / schémat / plánů, které si mohou zůčastnění prohlížet

a manipulovat s nimi. Zároveň by zcela jistě mohl vést k zefektivnění pracovních procesů.

Podobně tomu jistě bude v oblastech školení zaměstnanců, kdy místo neinteraktivního poslechu prezentací budou moci zaměstnanci aktivně vyzkoušet a procvičit svoje dovednosti virtuálně s nasazeným HMD.

Již nyní podobné projekty existují, například školení zaměstnanců České spořitelny při práci s hotovostí na pobočkách. [3]

1.3 Cíle

Hlavním cílem práce je návrh a implementace multi-uživatelského systému pro mixed reality, tedy rozšíření reálné scény o virtuální 3D objekty, kterou bude možné ve stejný okamžik sdílet s více uživateli.

Teoretická část se zaměřuje na analýzu dostupných VR HMDs se systémem inside-out trackingu, dále potom zejména na analýzu možných řešení problému sdílení stejného prostoru s více uživateli, tedy na korektně a spolehlivě fungující synchronizaci polohy a rotace HMDs jednotlivých uživatelů.

Praktická část řeší volbu 3D scény a samotnou implementaci systému v Unity.

Kapitola 2
Analýza

...

2.1 Analýza dostupných VR systémů s inside-out trackingem

Rozlišujeme dva základní typy VR systémů podle způsobu pozičního trackování:

1. Inside-out tracking

   Pro určení pozice a orientace VR systému v prostoru systém využívá kamer a případně dalších senzorů, které jsou zabudované přímo v zařízení.

2. Outside-in tracking

   V prostoru kolem tohoto VR systému se nachází staticky rozmístěné kamery či jiné senzory. Typicky je určování polohy s tímto systémem přesnější, vyžaduje ale složitější přípravu a ze své podstaty je uživatel vázán na konkrétní místnost.

V případě inside-out trackingu můžeme vyčlenit dvě kategorie, sledování se značkami a bez značek (markerless).

Sledování se značkami spočívá v rozmístění pro VR systém snadno rozpoznatelných a dopředu známých tvarů na konkrétní místa v prostoru. Jedná se například o obrázky čtverců, kruhů nebo QR kódů.

Díky jejich snadné rozpoznatelnosti a znalosti jejich přesného umístění je potom snadné určit, kde se nachází HMD, který je vidí. Problém ovšem nastává v případě, kdy tyto značky zmizí ze zorného pole, tedy je zde potřeba většího plánování a přípravy prostoru, než je tento systém možné použít.

Mapování bez značek je v tomto ohledu zajímavější a proto se nadále budeme věnovat této kategorii HMDs.

VR systém s inside-out trackingem s mapováním bez značek během používání typicky mapuje okolní prostředí, ve kterém hledá význačné feature pointy (body, hrany), na základě kterých si vytváří svoji vnitřní reprezentaci scény.

Algoritmy daného systému poté z této reprezentace, respektive její dynamické změny, vyvozují pozici a orientaci HMD ve scéně.

Obecně se tato technika nazývá SLAM - Simultaneous localization and mapping.

Za první VR systém, který implementoval markerless inside-out tracking lze považovat Windows Mixed Reality Headset od společnosti Acer. Tento HMD disponuje dvěma kamerami, po-

mocí kterých vytváří mapu okolního prostředí. K tomu ovšem vyžaduje počítač se systémem Windows 10. Základní parametry HMD jsou následující:

- 1440 x 1440 LCD per eye
- 90 Hz
- 100° FOV

Obrázek 2.1 Windows Mixed Reality - Acer [4]

Za první VR systém, který inside-out tracking implementoval správně, lze potom považovat Oculus Quest společnosti Meta, který ke své funkčnosti nevyžaduje žádné další externí zařízení.

2.1.1 Přehled dostupných VR systémů s inside-out trackingem

Na jaře roku 2022 se na trhu nachází následující zařízení:

Tabulka 2.1 HMDs s inside-out trackingem - jaro 2022

Název	Display	Cena
Oculus Quest 2	1832 x 1920 LCD per eye	128GB 399$ / 349€
Pico Neo 3 Link	1832 x 1920 LCD per eye	256GB 449€
Microsoft Hololens 2	1440 x 936 holographic per eye	3 500$
HP Reverb G2	2160 x 2160 LCD per eye	599$
HTC Vive Cosmos	1440 x 1700 LCD per eye	699$
Vrgineers XTAL 3 Mixed Reality	3840 x 2160 LCD 75 Hz 4k / 120 Hz QHD	11 500$
Varjo XR-3	1920 x 1920 OLED per eye focus area 2880 x 2720 LCD per eye peripheral	5 995€

2.1.1.1 Oculus Quest 2

Obrázek 2.2 Oculus Quest 2 [5]

Oculus Quest 2 společnosti Meta navazuje na svého úspěšného předchůdce Oculus Quest. Jedná se o samostatně fungující headset se state of the art technologií inside-out trackingu, zvanou Oculus Insight. [6]

Tato technologie ke své funkčnosti sbírá data ze třech zdrojů:

1. Obrazová data z kamer umístěných v HMD generují mapu místnosti, ve které hledá a určuje orientační body, jako jsou rohy nábytku či vzory na podlaze

2. Rychlost rotace (rotational velocity) a lineární zrychlení (linear acceleration) z IMU jednotky (inertial measurement unit) umístěné v HMD a ovladačích

3. Infračervené LED diody umístěné v ovladačích

Mimo to obsahuje algoritmus kinematické predikce - predikce pozic ovladačů / HMD na pár milisekund dopředu, trackování je tak ve většině prostředí přesné s odchylkou do jednoho milimetru.

Na rozdíl od svého předchůdce disponuje procesorem přímo určeným pro XR zařízení, tím je Qualcomm Snapdragon XR2. Jelikož je možné na tomto VR systému využívat aplikace i hry určené pro jeho předchůdce, pohybuje se tak celkový počet spustitelných titulů v oficiálním obchodu Oculus Store v řádu stovek.

Díky své nízké ceně, přesnému trackování, samostatné funkčnosti a obsáhlé databázi titulů představuje tento HMD v současné době nejpopulárnější zařízení, a to i přes nutnost permanentního přihlášení s účtem sociální sítě Facebook.

2.1.1.2 Pico Neo 3 Link

Obrázek 2.3 Pico Neo 3 Link [7]

Pico Neo 3 Link je první HMD této společnosti, který je určen pro koncové zákazníky na evropském trhu, tím se odlišuje od předchozích modelů určených pro business zákazníky na trhu asijském.

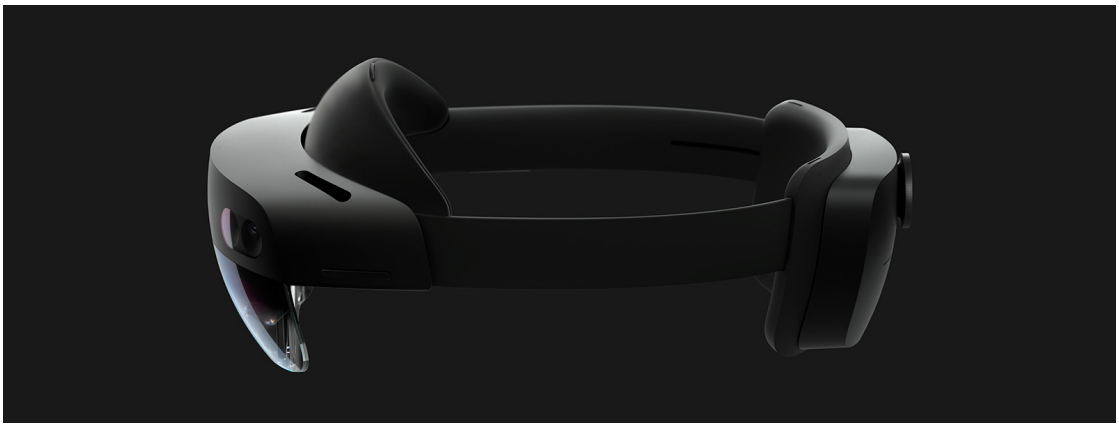
Svojí vizuální podobou se nápadně podobá Quest 2, technickými parametry se potom prakticky neliší, jediným rozdílem je limit obnovovací frekvence displeje na 90 Hz. Na rozdíl od Quest 2 nabízí pouze jeden model s 256 GB paměti a to za stejnou cenu, za kterou se prodává stejná varianta Quest 2. V čem ale Quest 2 převyšuje je kvalita příslušenství.

Přímo v základní variantě tento systém obsahuje pohodlný náhlavník s baterií umístěnou v zadní části hlavy pro rozložení váhy a tedy větší pohodlí. Quest 2 nabízí možnost pořídit si pro větší pohodlí Elite Strap za 49$. V případě komunikace s počítačem je pro Quest 2 k dispozici Link Cable za 79$, Pico Neo 3 Link jej má hned v základní výbavě, přitom se ale nejedná o USB kabel, který by obraz přenášel v komprimované formě, nýbrž o nekomprimovaný DisplayPort.

Hlavní výhodou Pico Neo 3 Link oproti Quest 2 je ale možnost využívat HMD bez nutnosti přihlášení prostřednictvím Facebooku, což byl a pro mnoho lidí stále je hlavní důvod, proč se rozhodli si Quest 2 nepořídit, Neo 3 by právě tento segment mohl přesvědčit.

Tento systém zatím není na trhu, ale očekává se, že bude uveden během několika následujících týdnů.[8]

2.1.1.3 Microsoft Hololens 2



Obrázek 2.4 HoloLens 2 [9]

Nejedná se přímo o VR headset, neboť nezobrazuje obsah na neprůhledném displeji, nýbrž jej promítá lomeným paprskem na průhledný hranol. Toto mixed reality zařízení s inside-out trackingem je určeno převážně pro firemní sektor, čemuž odpovídá také cena.

2.1.1.4 HP Reverb G2

Obrázek 2.5 HP Reverb G2 [10]

HP Reverb G2 představuje v současné sobě nejlepší řešení pro Windows Mixed Reality. Narozdíl od většiny ostatních obsahuje již v základu integrovaná sluchátka. Pro trackování polohy slouží 4 kamery.

Hlavní nevýhodou tohoto HMD je jeho nesamostatnost. Pro správnou funkčnost je potřeba permanentního připojení k počítači se systémem Windows 10.

2.1.1.5 HTC Vive Cosmos

Obrázek 2.6 HTC Vive Cosmos [11]

HTC Vive Cosmos je první HMD společnosti Vive s inside-out trackingem. Zde se implementace inside-out trackingu příliš nepovedla [12] a i kvůli tomu se společnost Vive rozhodla vydat náhradní čelní desku [13], kterou je možné umístit na přední část zařízení a která umožní sledování původní technologií, totiž outside-in tracking v podobě laserových majáčků.

2.1.1.6 Vrgineers XTAL 3 Mixed Reality

Obrázek 2.7 XTAL 3 Mixed Reality [14]

Tento HMD pochází od české společnosti Vrgineers a je určen pro profesionální použití. Pro každé oko je určen 4K displej, horizontální FOV činí 180°, vertikální potom 90°. Díky tomu se HMD náramně hodí pro různé letecké simulátory, respektive výcvik letců, k čemuž je také určen.

2.1.1.7 Varjo XR-3

Dalším vrcholným mixed reality VR systémem s podporou inside-out trackingu je Varjo XR-3.

Cena je stanovená na 5 995,00 €, v přepočtu na CZK tedy necelých 150 000 korun, není ale možné pořídit si jej jako individuální zákazník, pouze jako firma / organizace a to navíc s po-

Obrázek 2.8 Varjo XR-3 [15]

vinným předplatným v minimální době jednoho roku, tzv. Varjo Subscription za cenu 1 495,00 €, tedy minimální nutná investice činí 7 490,00 €.

Pro interní rekonstrukci a reprezentaci scény HMD využívá kombinace dat získaných ze senzoru LiDaR a RGB obrazu ze dvou kamer. Tyto kamery operují na frekvenci 90 Hz s velmi nízkou odezvou, rozlišením 12 Mpx a barevným obrazem. I díky těmto kamerám je HMD schopný vizualizovat virtuální 3D objekty ve scéně ve fotorealistické kvalitě. Sledování rukou zajišťuje senzor Ultraleap Gemini v5 s frekvencí 200 HZ.

2.2 Analýza možných řešení synchronizace scény

Kapitola se zabývá rozborem možných řešení zadaného problému a nástinem hlavní myšlenky zvoleného algoritmu.

2.2.1 Analýza problému

Na základě předchozí analýzy trhu, cenové dostupnosti a potencionálního nasazení v reálné aplikaci byl k řešení zadaného problému pro implementaci zvolen VR systém Oculus Quest 2 společnosti Meta.

Zadaný problém, tedy implementace aplikace vizualizující vybranou 3D scénu včetně sdílení s více lokálními uživateli lze rozdělit na několik podproblémů.

Našim cílem je, aby uživatelé interagovali v rámci jedné 3D scény ve stejném fyzickém prostoru. Tedy je nezbytné vyřešit způsob, jakým zajistit, aby pozice 3D avatarů reprezentujících uživatele ve virtuální 3D scéně odpovídaly pozicím uživatelů v prostoru reálném. Kromě avatarů je potřeba vyřešit také stejné zarovnání 3D scény samotné ve fyzickém prostoru napříč HMDs.

Jelikož má každý HMD jiné mapování místnosti, tzv. Guardian [16] (jedná se o oblast v místnosti, kde je bezpečné se pohybovat, její určení je první fází po nasazení HMD), má každý HMD i jiný počátek scény, který je navíc možné resetovat na aktuální fyzickou pozici uživatele podržením tlačítka Oculus Home.

Obrázek 2.9 Guardian [17]

Tím vyvstává první podproblém, totiž musíme být schopni určit, kde se HMDs nachází relativně vůči sobě. K tomu bychom ideálně potřebovali stejný počátek scény. Jelikož je ale počátek scény určen mapováním Guardianu, respektive místem, ve kterém proběhl reset tlačítkem Oculus Home, není možné přenášet pozice a orientace vztažené k tomuto počátku a tvářit se, že se jedná o stejný počátek ve všech zúčastněných zařízeních.

Jako první se nabízí možnost pokusit se o stejné mapování Guardianu na všech HMDs. Body,

které Guardian definují jsou přístupné ze třídy OVRBoundary, tedy bychom tyto body mohli mezi zařízeními synchronizovat a pokusit se namapovat jednotlivé Guardiany přes sebe. Zbavili bychom se tak problému rozdílného počátku, totiž v každém okamžiku bychom byli schopni určit pozici HMD relativně od Guardianu a tu mezi zařízeními synchronizovat. [18]

Je ovšem velmi nepravděpodobné, že bychom byli schopni namapovat Guardiany na všech zařízeních dostatečně přesně na to, aby výsledně určená virtuální pozice měla dostatečně malou odchylku od pozice reálné, stejně tak jako je velmi nepraktické spoléhat se pro synchronizaci na celý Guardian, sebemenší odchylka bude mít vliv na výslednou pozici. Pokusme se nalézt lepší způsob.

Abychom byli schopni určit relativní pozice HMDs vůči sobě a jelikož se nemůžeme spolehnout na výchozí počátky scén, musíme si sami zajistit nějaký bod ve fyzickém prostoru, vůči kterému budeme pozice a orientace vztahovat.

Mezi řešení, které již byly vyzkoušené a které do jisté míry fungují lze zařadit synchronizaci 1v1. Lze ji provést například tak, že se k sobě uživatelé postaví s nasazenými HMDs ve stejné výšce čelem k sobě a potvrdí stiskem tlačítka. Jeden z HMDs či původní pozice jednoho z nich pak bude sloužit jako kotva, vůči které se budou pozice počítat. [19]

Tento typ synchronizace je ale nepraktický hned z několika důvodů. Zaprvé vyžaduje dva uživatele, kteří provedou synchronizaci současně. Dále ačkoliv je v případě těsného kontaktu HMDs známá vzdálenost jejich středů, tak není vůbec zaručena stejná výška od země. Každý člověk je jinak vysoký a bylo by nepraktické snažit se údaj o rozdílu výšek do řešení kombinovat.

Jako lepší řešení se jeví umístění bodu (kotvy) do scény na všem uživatelům známé místo. [20] K tomu lze pro větší pohodlí využít ovladače (alternativa, kdy se uživatel snaží HMD, tedy hlavou, trefit na předem známé místo se nejeví jako příliš přesná, ani praktická).

Od verze operačního systému v34 lze na zařízeních Oculus Quest a Oculus Quest 2 aktivovat Passthrough [21] (průhled do reálného prostoru skrze kamery umístěné v HMD) v aplikačním režimu a efektivně tak umožnit realizaci rozšířené reality. Zejména díky tomu je možné znát s poměrně vysokou přesností fyzické umístění HMD (uživatele), ale hlavně ovladačů bez nutnosti sejmout HMD, což je pro umístění kotev klíčová vlastnost.

Jako kotva by mohl sloužit jednoduchý bod, který by měl svoji pozici a rotaci ve scéně a který by uživatelé umístili za pomoci ovladačů na stejné, dopředu smluvené místo. Díky technologii SLAM a dalším proprietárním algoritmům společnosti Oculus řešícím mapování si můžeme být jisti, že se virtuální bod bude v průběhu simulace nacházet na přibližně stále stejném místě. Problém ovšem nastane v případě, kdy se uživatel rozhodne pro reset počátku scény tlačítkem Oculus Home. V okamžiku umístění bodu (kotvy) na předem určené fyzické místo v prostoru se pozice a rotace bodu zafixují jako vzdálenost a orientace vůči počátku scény a zůstanou tak napevno po zbytek simulace. To se ovšem neslučuje s resetem fyzického umístění počátku, který nezbytně způsobí přesun námi umístěné kotvy na jiné fyzické místo (relativní vzdálenost a orientace vůči virtuálnímu počátku zůstane zachovaná). Jelikož není možné zajistit reset počátku na stejné místo i na ostatních zařízeních, nelze takto umístěný bod (kotvu) považovat za stabilní a pro potřeby našeho řešení za dostatečnou. Zároveň by bylo nutné tento bod (kotvu) umisťovat při každém spuštění simulace, což by bylo značnou překážkou při reálném nasazení aplikace do produkčního prostředí.

2.2.1.1 Přehled verzí Oculus SDK pro Unity Engine

Následuje přehled verzí a klíčových funkcí Oculus SDK, z nichž zvýrazněné tučným písmem představují klíčové funkce pro řešení našeho problému. [22]

- v12 - 23.12.2019

  Hand Tracking - preview

- v13 - 12.02.2020

  Vulkan API support

- v19 - 05.08.2020

  System Keyboard in-app overlay

- v23 - 21.12.2020

  OpenXR - experimental plugin

- v28 - 30.04.2021

  High frequency hand tracking

  120 Hz display - experimental support

- v31 - 13.08.2021

  Passthrough API - experimental support

- v34 - 11.11.2021

  Passthrough API - production release

  Application SpaceWarp

- v35 - 07.12.2021

  Spatial Anchors - experimental release

  Hand Tracking with OpenXR backend

- v37 - 02.02.2022

  Interaction SDK - experimental release

- v39 - 28.04.2022

  Hand Tracking v2 - fast movement, light hand-hand interaction, partial occlusion

Výše uvedené funkce zvýrazněné tučným písmem jsou klíčové pro realizaci našeho zadání.

2.2.1.2 Spatial Anchors

Problémy popsané v analýze problému řeší technologie zvaná Spatial Anchors. API, které tuto technologii zprostředkovává bylo do Oculus SDK přidáno ve verzi v35 v prosinci roku 2021, tedy teprve nedávno. I kvůli tomu se stále jedná o experimentální verzi a pro možnost jejího využití je nutné uvést zařízení, na kterém bude aplikace instalována, do experimentálního režimu.

Pro uvedení do experimentálního režimu stačí s připojeným zařízením spustit přes příkazovou řádku následující příkaz: *adb shell setprop debug.oculus.experimentalEnabled 1*. (adb - Android Debug Bridge, nástroj příkazové řádky pro komunikaci s Android zařízeními připojenými přes USB či WiFi)

Spatial Anchors umožňují umístit bod (kotvu) s pevnou pozicí a orientací vztaženou vůči reálnému prostoru. To znamená, že pokud dojde k resetu počátku tlačítkem Oculus Home, objekt reprezentující kotvu ve scéně zůstane na stále stejném fyzickém místě. Tedy se změní jeho relativní pozice a orientace vůči počátku scény.

Díky této vlastnosti jsme schopni zajistit pevný bod ve scéně v průběhu celé simulace.

Spatial Anchors ale umožňují více, totiž umístěnou kotvu je možné označit za perzistentní. Perzistentní kotvy mají tu vlastnost, že přetrvají nejen po vypnutí simulace (aplikace), ale také samotného systému. V důsledku toho je možné provést umístění kotvy při prvním spustění simulace a v následujících bězích při startu pouze načítat kotvy již v minulosti umístěné. Je možné, že dojde k nerozpoznání dříve umístěné kotvy, mělo by se tak stát ale pouze v případě, pokud uživatel provedl nové mapování Guardianu, ke kterému standardně nemá důvod (pokud často nestěhuje nábytek).

Tato vlastnost zase ulehčuje potenciální nasazení do reálné aplikace.

Operace podporované Spatial Anchors API jsou následující:

Obrázek 2.10 Spatial Anchor [23]

- Create - *SpatialEntityCreateSpatialAnchor*
- Locate - *LocateSpace*
- Destroy - *DestroySpace*
- Save - *SpatialEntitySaveSpatialEntity*
- Erase - *SpatialEntityEraseSpatialEntity*
- Query - *SpatialEntityQuerySpatialEntity*

Pro účely zadání není potřeba toto API detailně rozebírat, pro zvídavé čtenáře je k dispozici odkaz do dokumentace. [24]

2.2.2 Hlavní myšlenka algoritmu

1. Umístění kotvy do scény na předem určené místo
2. Výpočet pozice a rotace HMD vztažené k této kotvě
   - Tedy nestačí vzít pozici a rotaci HMD tak jak je, neboť ta je ve světových souřadnicích
   - My ji ovšem potřebujeme znát v souřadnicích udávaných kotvou
   - Pozici získáme voláním funkce *InverseTransformPoint(...)*
   - Rotaci potom *Quaternion.Inverse(...)*
3. Vytvoření po síti synchronizovaného objektu ve scéně reprezentující offset tohoto HMD a nastavení jeho pozice a rotace na hodnoty získané v předchozím kroku
4. Ostatní zařízení ve stejné serverové místnosti tento objekt detekují
5. Na základě čehož aktivují Avatara, který reprezentuje hráče, kterému tento offset patří
6. Pozici a rotaci Avatara nastaví jako pozici a rotaci offsetu, ovšem vztaženou opět vůči vlastní kotvě
7. K tomu se využije volání funkce *TransformPoint*

2.2.3 Řešení síťové komunikace

Původní nativní řešení pro síťovou komunikaci v Unity UNet je již nějakou dobu ve stavu deprecated a jeho následník v podobě Netcode je zatím stále ve fázi preview, tedy byla prozkoumána alternativní řešení síťové komunikace. [25][26]

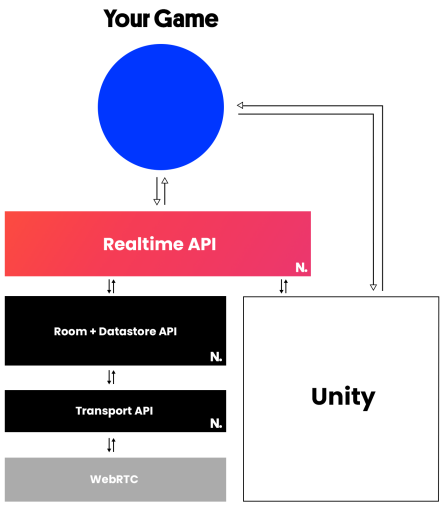
Jako dobře navržený a pro implementaci přímočarý se ukázal framework Normcore, jehož implementace do Unity je zprostředkována v podobě plug-inu. [27] Normcore představuje kompletní řešení pro síťovou komunikaci a je založen na bázi modelu klient-server.

Na rozdíl od tradičních síťových řešení, která typicky využívají RPC zprávy k synchronizaci stavu napříč zařízeními, je zde veškerý stav uchováván v jedné real-time databázi. Tím se předejde chybám, ke kterým často dochází během synchronizace stavu napříč klienty.

Zároveň neplatí, že by server odesílal veškeré informace o stavu do všech klientů, nýbrž posílá pouze informace o změnách stavu (delta updates), čímž se minimalizuje množství dat, které je nutné doručit, zároveň se tak minimalizuje latence.

Normcore nabízí v základu dva cenové tarify, bezplatný Prototype a placený Pro za 49$/měsíc. Jelikož tarif Prototype umožňuje až 30 souběžně aktivních uživatelů a pro účely prototypu potřebujeme synchronizovat uživatele 2, ukazuje se tato bezplatná varianta jako více než dostatečná.

2.2.3.1 Unity klient



Obrázek 2.11 Unity klient [28]

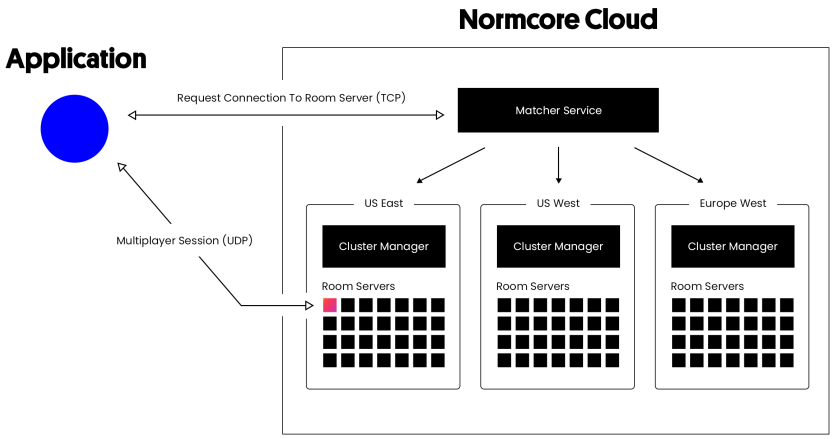
Normcore plug-in sestává z několika vrstev, přičemž každá vrstva (kromě WebRTC) staví na vrstvě předchozí, čímž poskytuje vyšší míru abstrakce.

1. Realtime API - Přímo navázáno na objekty ve scéně, k řešení našeho problému je zapotřebí manipulovat pouze s touto vrstvou
2. Room + Datastore API - Spravuje připojení k místnosti na serveru a řeší synchronizaci surového stavu, tj. nemá přehled o scéně, ani objektech v ní přítomných.
   - Room - Normcore zavádí koncept místnosti. Stav simulace se synchronizuje se všemi klienty v dané místnosti.

- Datastore - Řeší synchronizaci stavu, uchovává si svůj lokální stav, přijímá změny ze serveru a zároveň změny na server odesílá.

3. Transport - Má na starosti posílání zpráv mezi body A a B.

2.2.3.2 Cloudová infrastruktura



Obrázek 2.12 Cloudová infrastruktura [29]

Normcore Cloud backend sestává ze třech částí:

1. Matcher - Při pokusu o připojení do místnosti prvně Matcher určí, který Cluster bude pro danou místnost vyhrazen, typicky v závislosti na rychlosti odezvy.
2. Clusters - Reprezentují shluk místností na daném serveru. Má na starost zapínání a vypínaní místností, stejně tak jako ukládání a načítaní perzistentního stavu.
3. Rooms - Reprezentují jednotlivé místnosti, do kterých se mohou klienti připojovat.

2.2.3.3 Transportní vrstva

Transportní vrstva je tvořená protokolem WebRTC. WebRTC kromě jiného podporuje komunikaci s prohlížečem, tedy v případě potřeby by neměl být problém rozšířit scénu o možnost dalších uživatelů, kteří by místo HMDs interagovali se scénou přímo z prohlížeče.

Jako síťové protokoly využívá Normcore díky WebRTC jak UDP, tak TCP. Proměnné objektů, u kterých se předpokládá velmi častá změna, jako je například pozice s rotací, je možné využít UDP protokol pro minimalizaci latence. Je zde předpoklad, že v nejbližší chvíli dorazí informace o aktuální, nové poloze a tedy není problém, pokud dojde k výpadku v důsledku nespolehlivosti tohoto protokolu.

Naproti tomu například informace o stisku tlačítka musí být spolehlivě a za každou cenu doručená, tedy zde se naopak uplatní TCP protokol.

	WebRTC (Normcore)	WebSockets (Croquet)	eNet (Photon)	Telepathy (Mirror)	kcp2k (Mirror)
UDP	✅	❌	✅	❌	✅
TCP Fallback	✅	✅	❌	✅	❌
Reliable messages	✅	✅	✅	❌	✅
Unreliable messages	✅	❌	✅	✅	✅
TLS/DTLS encryption	✅	✅	❌	❌	❌
Congestion/Flow control	✅	✅	❌	✅	✅
Video/Audio Streaming	✅	❌	❌	❌	❌
Browser compatibility	✅	✅	❌	❌	❌

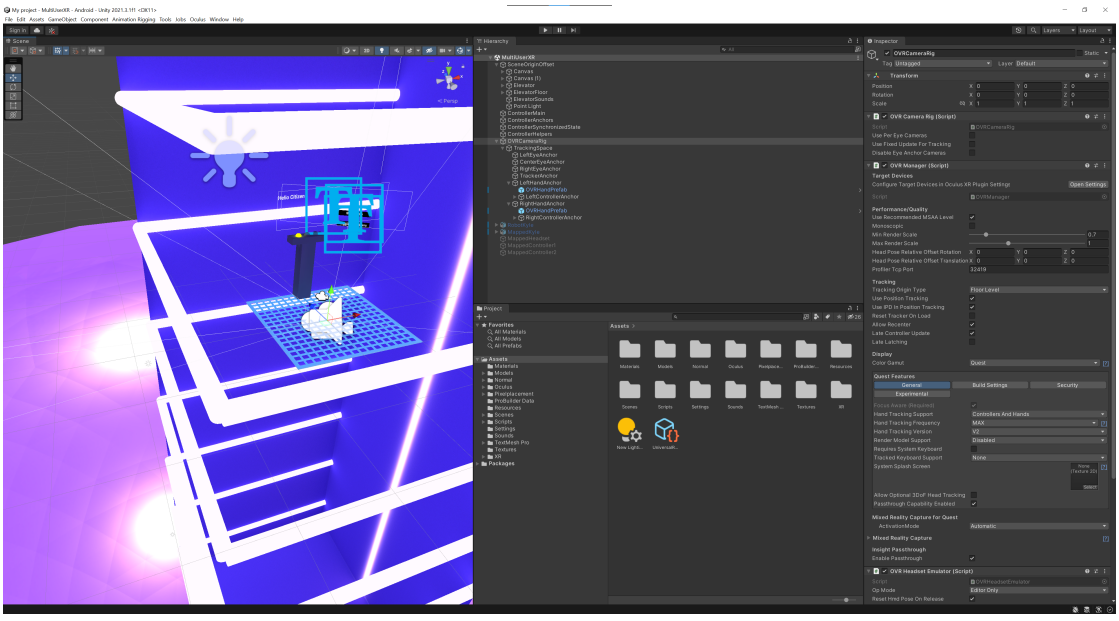
Obrázek 2.13 Srovnání síťových řešení [30]

Kapitola 3
Implementace prototypu

Kapitola se zabývá rozborem konkrétní implementace prototypu v Unity.

3.1 Volba 3D scény

Pro simulaci byla zvolena jednoduchá 3D scéna výtahu, který se při přechodu z mixed reality do virtual reality ze všech čtyř stran uzavře naškálováním a translací stěn směrem ke středu scény, zároveň dojde k vysunutí plošiny uprostřed výtahu a to obdobným způsobem.



Obrázek 3.1 Scéna v Unity

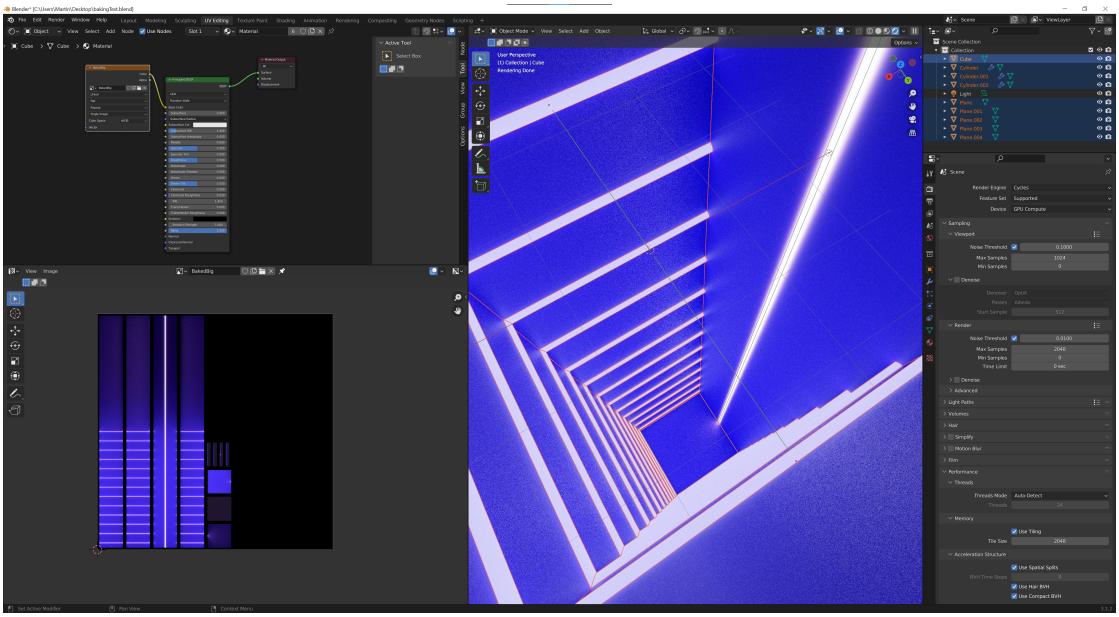
Původní záměr byl vymodelovat celou scénu v Unity a nechat *GPU Progressive Lightmapper*, v Unity nativní path-tracing based lightmapper vygenerovat a zapéct osvětlení do textur, aby bylo možné nasimulovat realistické osvětlení za cenu co nejnižších systémových požadavků.

Tento záměr se nakonec nepodařilo naplnit, protože ačkoliv by u světel v Unity mělo být možné pomocí přepínače *Culling Mask* specifikovat vrstvy a tedy objekty, kterých by se měl lightmapping týkat, tak se tak kvůli chybě v Unity bohužel neděje a každé světlo tak osvětluje

veškeré objekty ve scéně. I přes zákryt v podobě poklopu pod výtahem docházelo k light bleedingu a světlo z kopule se tak negativně projevovalo na osvětlení výtahu.

Tato chyba je v Unity přítomna od roku 2015, tedy již sedm let. [31]

V důsledku toho bylo rozhodnuto část s výtahem vymodelovat a zapéct v Blenderu. Část s kopulí zůstala zapečená z Unity, model byl též vytvořený v Blenderu.



Obrázek 3.2 Zapékání osvětlení do textury v grafickém editoru Blender

3.2 Implementace simulační logiky v Unity

Pro implementaci prototypu byl zvolen herní engine Unity. Alternativa v podobě Unreal Engine nebyla v době psaní práce možná, protože Spatial Anchors API nebylo zatím v Oculus SDK pro Unreal Engine dostupné. Další možností by byla implementace nativní aplikace v prostředí OpenGL / Vulkan, pro které sice podpora Spatial Anchors API již k dispozici byla, ale svým rozsahem by daleko přesahovala záměr této práce, proto byl zvolen herní engine Unity.

První obrázek popisuje hierarchii scény.

- SceneOriginOffset

  Obsahuje veškeré objekty scény, které se mají všem uživatelům zobrazovat na stejném místě ve fyzickém prostoru. Změnou pozice a rotace tohoto objektu lze docílit korektního zarovnání scény napříč zařízeními.

- ControllerMain

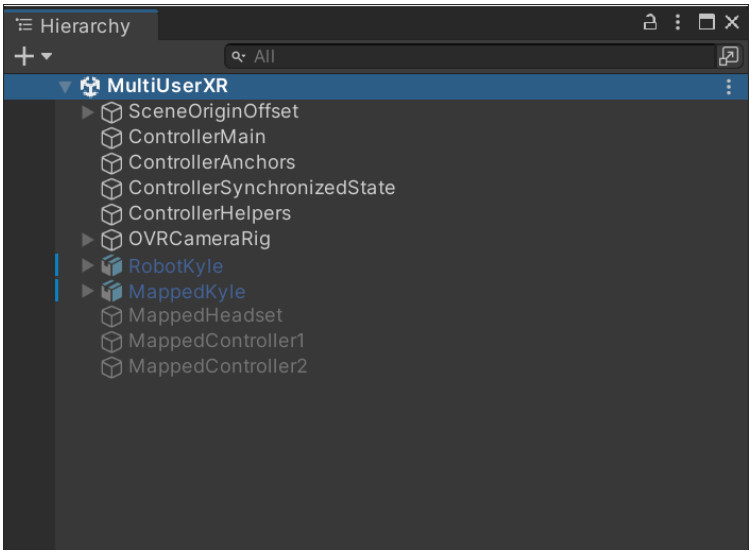
  Představuje hlavní ovladač scény a obsahuje dva scripty.

  - MainScript

    Řeší hlavní věci ve scéně, jako je mapování offsetů vůči kotvám z / na server, interakci uživatelů ve scéně, zpracování příchozích požadavků na akci ze serveru.

  - Realtime

    Obecné nastavení k síťové komunikaci se servery Normcore.



Obrázek 3.3 Hierarchie scény v Unity

- App Key - string - Unikátní klíč k síťové komunikaci. Je nutné jej vygenerovat ze strany vývojáře založením účtu na stránkách Normcore. Zamezuje využívání serverů nad rámec zvoleného tarifu.
- Join Room On Start - bool - Určuje, zda se má aplikace po zapnutí scény okamžitě snažit o navázání spojení se serverem. Zvoleno true.
- Room Name - string - Název místnosti, do které se aplikace pokusí po zapnutí scény připojit. Pro účely prototypu napevno zvolen název Test Room.

- ControllerAnchors

  Představuje hlavní ovladač kotev ve scéně a obsahuje dva scripty.

  - AnchorSpawner

    Řeší umístění kotvy na konkrétní místo ve scéně. Po probuzení objektu na sebe naváže SpatialAnchorSession script, který má na starost životní cyklus kotev ve scéně.

  - AnchorUIManager

    Script, který má na starost komunikaci s UI pro přidávání / odebírání kotev do scény.

- ControllerSynchronizedState

  Představuje hlavní ovladač synchronizace scény a obsahuje opět dva scripty.

  - RealtimeView

    Každý objekt ve scéně, který má být synchronizován po síti, musí obsahovat tuto komponentu. Každé RealtimeView má svoje unikátní UUID.

  - GameStateSync

    Stará se o synchronizaci aktuální scény.

- ControllerHelpers

  Řeší vypínaní a zapínaní objektů ve scéně v závislosti na aktuální metodě interakce - ovladače / sledování rukou.

- OVRCameraRig

  Proprietární objekt z Oculus SDK. Obsahuje několik scriptů a dalších podobjektů, které umožňují konfiguraci scény pro virtuální realitu.

- RobotKyle

  Avatar reprezentující aktuálního uživatele. Aktivní je pouze v případě, kdy uživatel k interakci se scénou využívá sledování rukou.

- MappedKyle

  Avatar reprezentující druhého uživatele, se kterým probíhá interakce ve scéně. Data o poloze a orientaci jsou nastavená podle následujících třech objektů:

- MappedHeadset, MappedController1, MappedController2

  Představují HMD a ovladače druhého uživatele, jejich poloha a orientace je přijatá po síti.

3.2.1 MainScript

Pojmem *Transform* či *transform* uvažujeme komponentu v Unity, která je přítomná na každém objektu ve scéně a která určuje pozici, rotaci a škálování tohoto objektu.

Start zajistí navýšení obnovovací frekvence displeje z 90Hz na 120Hz v případě, kdy je aplikace nainstalována na systém virtuální reality Oculus Quest 2. Dále registruje callback *DidConnectToRoom*, který se zavolá v okamžiku navázání spojení se serverem a připojením do místnosti.

OnDisable odregistruje výše zmíněný callback, aby v důsledku asynchronního běhu kódu nedocházelo k pokusům o registraci klienta po vypnutí / pozastavení aplikace.

DidConnectToRoom v okamžiku navázání spojení s místností přidá současného klienta do seznamu klientů, pokud v něm již není. Dále se pokusí nastavit současného klienta jako hosta, stane se tak pouze pokud v místnosti zatím žádný host není.

```
1    private void Start()
2    {
3        var headsetType = OVRPlugin.GetSystemHeadsetType();
4        if (headsetType == OVRPlugin.SystemHeadset.Oculus_Quest_2) OVRPlugin.systemDisplayFrequency = 120.0f;
5
6        realtime.didConnectToRoom += DidConnectToRoom;
7    }
8
9    private void OnDisable()
10   {
11       realtime.didConnectToRoom -= DidConnectToRoom;
12   }
13
14   private void DidConnectToRoom(Realtime realtime)
15   {
16       gameStateSync.AddClient(realtime.clientID);
17       gameStateSync.SetHostClientID(realtime.clientID);
18   }
```

Obrázek 3.4 MainScript - Lifecycle

Update se v každém cyklu simulace stará o tři věci:

1. Kontrola příchozích akcí ze serveru
2. Kontrola a aplikace offsetu příchozího počátku scény od hosta, pokud klient není host

3. Kontrola a aplikace offsetů HMD a ovladačů druhého hráče

```
1    private void Update()
2    {
3        if (gameStateSync.Actions.ContainsKey((uint)realtime.clientID))
4        {
5            var action = gameStateSync.Actions[(uint)realtime.clientID].action;
6            gameStateSync.Actions.Remove((uint)realtime.clientID);
7            switch (action)
8            {
9                case 1:
10                   GetReady(true);
11                   break;
12               case 2:
13                   StartElevator(true);
14                   break;
15           }
16       }
17
18       ...
```

Obrázek 3.5 MainScript - Update - kontrola příchozích akcí

Kontrola příchozích akcí nejprve ověřuje, zda globálně synchronizovaný objekt Actions (jedná se o dictionary) obsahuje záznam s klíčem odpovídajícím lokálnímu clientID, tedy zda je pro lokálního klienta k dispozici nějaká akce, kterou by měl vykonat.

Pokud je akce k dispozici, na řádku číslo 5 si ji uloží a hned na následujícím ji z globálně synchronizovaného objektu akcí smaže (lokálně se změny projeví okamžitě, v synchronizovaných zařízeních hned jak informace o změně dorazí ze serveru). Nakonec se vykoná požadovaná akce podle přiděleného ID.

```
1        ...
2
3        if (gameStateSync.hostClientID != -1 && gameStateSync.hostClientID != realtime.clientID)
4        {
5            var offsetSceneOrigin = GameObject.FindWithTag("OffsetSceneOrigin");
6            if (offsetSceneOrigin != null)
7            {
8                var sceneOriginPosition = _anchor.transform.TransformPoint(offsetSceneOrigin.transform.position);
9                var sceneOriginRotation = _anchor.transform.rotation * offsetSceneOrigin.transform.rotation;
10               sceneOriginOffset.transform.position = sceneOriginPosition;
11               sceneOriginOffset.transform.rotation = sceneOriginRotation;
12           }
13       }
14
15       ...
```

Obrázek 3.6 MainScript - Update - kontrola příchozího offsetu scény

Ve druhé části se kontroluje příchozí offset scény. Pokud má místnost přiděleného hosta a pokud tento host není zároveň stávajícím klientem, potřebujeme zjistit offset od počátku scény, aby ji bylo možné umístit na stejnou fyzickou pozici, jako ji vidí host.

Povinností hosta je vytvořit a pravidelně aktualizovat globálně synchronizovaný objekt se značkou (*FindWithTag*) OffsetSceneOrigin. Pokud se nám tento objekt podaří ve scéně najít, můžeme si být jisti, že se jedná o offset počátku scény od kotvy umístěné hostem.

Abychom scénu dostali do stejné pozice a orientace jako ji má host, musíme změnit atributy *position* a *rotation* komponenty *transform* objektu sceneOriginOffset, jehož potomkem jsou všechny ostatní objekty ve scéně.

Jelikož pozice objektu offsetSceneOrigin udává relativní pozici vůči kotvě, ne absolutní pozici ve scéně, musíme ji prvně transformovat voláním funkce *TransformPoint*. Konkrétně volání *_anchor.transform.TransformPoint(offsetSceneOrigin.transform.position)* vrátí jako návratovou hodnotu offsetSceneOrigin vztažený vůči kotvě _anchor jako absolutní pozici offsetu scény.

Obdobně musíme získat absolutní rotaci scény, zde stačí pronásobit rotaci kotvy a offsetu samotného.

```
1        ...
2
3        var offsetsHeadset = GameObject.FindGameObjectsWithTag("OffsetHeadset");
4        foreach (var offset in offsetsHeadset)
5        {
6            var realtimeTransform = offset.GetComponent<RealtimeTransform>();
7            if (realtimeTransform.ownerIDSelf != realtime.clientID)
8            {
9                mappedKyle.SetActive(true);
10               mappedHeadset.SetActive(true);
11               var headsetPosition = _anchor.transform.TransformPoint(offset.transform.position);
12               var headsetRotation = _anchor.transform.rotation * offset.transform.rotation;
13               mappedHeadset.transform.position = headsetPosition;
14               mappedHeadset.transform.rotation = headsetRotation;
15           }
16       }
17
18       ...
```

Obrázek 3.7 MainScript - Update - kontrola příchozího offsetu HMD

Třetí část kontroluje offset HMD a ovladačů (rukou v případě, že uživatel ovladače odložil). Jelikož pro účely prototypu předpokládáme maximální počet dvou uživatelů, je kontrola offsetů implementovaná následujícím způsobem.

Prvně získáme veškeré objekty označené OffsetHeadset. Zajímají nás pouze ty, které nepatří stávajícímu klientovi, tedy provedeme kontrolu na řádku číslo 7. Pokud patří jinému klientovi, tj. druhému uživateli, aktivujeme Avatara *mappedKyle*, stejně jako *mappedHeadset*, který bude udávat absolutní pozici HMD ve scéně.

Avatar si v každém cyklu scény sám zjišťuje pozici a orientaci objektu *mappedHeadset*, tedy není nutné ji zde jakkoliv měnit.

Transformace pozice a rotace probíhá stejným způsobem jako v případě offsetu počátku scény ve druhé části.

Obdobně se kontrola a nastavení transformací provede i pro offsety ovladačů / rukou.

Tím končí metoda *Update*.

```
1    public void GetReady(bool fromServer = false)
2    {
3        if (!fromServer)
4        {
5            gameStateSync.AddNewAction(1);
6        }
7
8        ...
9    }
10
11   public void StartElevator(bool fromServer = false)
12   {
13       if (!fromServer)
14       {
15           gameStateSync.AddNewAction(2);
16       }
17
18       ...
19   }
```

Obrázek 3.8 MainScript - akce

Poslední část scriptu obsahuje dvě metody, GetReady a StartElevator pro interakci se scénou. Každá z nich nejdříve kontroluje původ svého volání. Pokud byla zavolána v důsledku příchozí akce ze serveru, pouze se vykoná kód v těle metody, pokud se ovšem jedná o lokálně provedenou akci, je nutné informovat ostatní klienty, aby došlo k synchronizaci stavu. O synchronizaci se postará ControllerSynchronizedState s navázaným scriptem GameStateSync, kterému voláním metody AddNewAction předáme informaci o akci, kterou má synchronizovat do ostatních klientů.

První zmíněná metoda způsobí přechod z mixed reality do virtual reality, konkrétně dojde k animaci stěn výtahu, které se prvně naškálují z 0 na 1 ve směru všech os a následně dojde k jejich translaci směrem ke společnému počátku scény, čímž se vytvoří klec / kabina výtahu.

Druhá zmíněná metoda poté spustí výtah, který se v závislosti na aktuálním stavu buď rozjede, nebo zastaví. Pokud dorazil do cíle a byl opět spuštěn, vydá se opačným směrem.

3.2.2 AnchorSpawner

AnchorSpawner je modifikovaná verze původního scriptu AnchorSpawner z ukázkového projektu pro implementaci Spatial Anchors z Oculus SDK.

mainScript představuje referenci na MainScript z objektu ControllerMain. Reference předána v editoru.

Řádky 5-12 zajišťují, že se jedná o singleton. Podobný kód lze nalézt i v ostatních scriptech pocházejících z ukázky.

Řádek 13 přidá komponentu / script SpatialAnchorSession. Jedná se o vzorovou implementaci Spatial Anchor management systému, která představuje doporučený základ pro modifikaci ve vlastních aplikacích. [32]

Řádek 14 předá referenci na mainScript scriptu SpatialAnchorSession. SpatialAnchorSession mimo jiné aktualizuje pozice a rotace kotev ve scéně, po provedení této aktualizace informuje o nových pozicích a rotacích MainScript za účelem aktualizace pozic offsetů. Z toho důvodu potřebuje znát referenci na MainScript.

```
1    [SerializeField] private MainScript mainScript;
2
3    private void Awake()
4    {
5        if (Instance == null)
6        {
7            Instance = this;
8        }
9        else
10       {
11           Destroy(this);
12       }
13       var anchorSession = gameObject.AddComponent<SpatialAnchorSession>();
14       anchorSession.mainScript = mainScript;
15   }
```

Obrázek 3.9 AnchorSpawner - Awake

PlaceAnchor po stisku tlačítka na ovladači umístí kotvu na konkrétní *transform*. Než se tak ale stane, dojde k modifikaci rotace. Konkrétně je na řádku 5 vynulována y souřadnice dopředného vektory tohoto *transform* a na řádku 6 vypočtená nová rotace kotvy jako *Quaternion.LookRotation(forward, Vector3.up)*.

V důsledku toho bude kotva vždy na všech HMDs umístěna vodorovně. Pokud by nebyla, tak by se vzrůstající vzdálenosti od kotvy vzrůstala chyba ve vertikálním směru (mapovaný Avatar by začal ujíždět buď příliš vysoko, nebo příliš nízko).

```
1    public void PlaceAnchorAtTransform(Transform transform)
2    {
3        var oldRotation = transform.rotation;
4        var forward = transform.forward;
5        forward.y = 0.0f;
6        transform.rotation = Quaternion.LookRotation(forward, Vector3.up);
7
8        ...
```

Obrázek 3.10 AnchorSpawner - PlaceAnchor

Obrázek 3.11 Spatial Anchor sample

3.2.3 GameStateSync

```
1   [RealtimeModel]
2   public partial class ClientIDModel
3   {
4       [RealtimeProperty(1, true, true)] private int _clientID;
5   }
6
7   [RealtimeModel]
8   public partial class ActionModel
9   {
10      [RealtimeProperty(1, true, true)] private int _action;
11  }
```

Obrázek 3.12 GameStateSync - modelové třídy

```
1   [RealtimeModel]
2   public partial class GameStateSyncModel
3   {
4       [RealtimeProperty(1, true, true)] private int _hostClientID;
5       [RealtimeProperty(2, true, true)] private RealtimeSet<ClientIDModel> _clients;
6       [RealtimeProperty(3, true, true)] private RealtimeDictionary<ActionModel> _actions;
7   }
```

Obrázek 3.13 GameStateSync - hlavní model

Data, která mají být synchronizována mezi klienty, musí být specifikována v modelové třídě, která musí být označena atributem *[RealtimeModel]*. Samotná data potom atributem *[RealtimeProperty]*.

Každá *[RealtimeProperty]* má tři argumenty:

1. PropertyID - uint - Unikátní ID této proměnné. Pokud by v budoucnu bylo potřeba změnit například datový typ této proměnné, tak je potřeba vytvořit novou proměnnou opět s unikátním ID. Kdyby došlo pouze ke změně datového typu, tak by nastala vzájemná nekompatibilita starší verze s novou, což by mohl být v produkčním prostředí problém. Není totiž možné zaručit, že všichni uživatelé budou mít pokaždé k dispozici nejnovější verzi.
2. Reliable / unreliable - bool - Určuje, zda je potřeba tuto proměnnou synchronizovat spolehlivě či nikoliv. Nespolehlivá může být například hodnota animující se barvy či poloha pohybujícího se tělesa, lze totiž předpokládat, že hned v následujícím herním cyklu dojde ke změně a tedy výpad jedné hodnoty bude nepostřehnutelný. Naopak informace o změně stavu scény, jako je spuštění nějaké akce, musí zaručeně dorazit všem klientům. Lze na tuto možnost pohlížet také jako na volbu mezi TCP a UDP protokolem.
3. Change Event - bool - Určuje, jestli se pro tuto proměnnou má vygenerovat událost informující o změně hodnoty, či zda není zapotřebí.

V našem případě chceme synchronizovat následující data:

- _hostClientID - int - Jedná se o ClientID hostujícího klienta. Hostující klient má tu vlastnost, že určuje počátek scény. Tedy pokud hostující klient změní svůj počátek scény (tlačítkem

Oculus Home), změní se i všem ostatním klientům. Výchozí hodnota, kdy klient ještě není nastaven, je -1.

- _clients - RealtimeSet<ClientIDModel> - Množina ClientID modelů, každý ClientIDModel uchovává jeden int reprezentující konkrétního klienta. Představuje všechny klienty připojené do místnosti.

- _actions - RealtimeDictionary<ActionModel> - Slovník Action modelů, každý ActionModel uchovává jeden int reprezentující konkrétní akci. Klíče jsou automaticky typu uint a představují ClientID klienta. Pokud je v tomto slovníku záznam, znamená to, že má klient s daným klíčem ClientID vykonat danou akci. Po jejím vykonání daný klient záznam ze slovníku odebere.

```
1   public class GameStateSync : RealtimeComponent<GameStateSyncModel>
2   {
3       public int hostClientID;
4       public RealtimeSet<ClientIDModel> Clients;
5       public RealtimeDictionary<ActionModel> Actions;
6
7       private void UpdateHostClientID()
8       {
9           hostClientID = model.hostClientID;
10      }
11
12      private void UpdateClients()
13      {
14          Clients = model.clients;
15      }
16
17      private void UpdateActions()
18      {
19          Actions = model.actions;
20      }
```

Obrázek 3.14 GameStateSync - aktualizace lokálních proměnných

GameStateSync jako třída dědící od *RealtimeComponent<GameStateSyncModel>* řeší synchronizaci dat ze synchronizovaného modelu do lokálních veřejných proměnných.

Lokální varianta proměnných odpovídá struktuře z *GameStateSyncModel*.

UpdateHostClientID, *UpdateClients* a *UpdateActions* jsou privátní metody volané z této třídy pro aktualizaci lokálních proměnných z modelu.

```
protected override void OnRealtimeModelReplaced(GameStateSyncModel previousModel, GameStateSyncModel
                                                                                    currentModel)
{
    if (previousModel != null)
    {
        previousModel.hostClientIDDidChange -= HostClientIDDidChange;
    }

    if (currentModel != null)
    {
        if (currentModel.isFreshModel) hostClientID = realtime.clientID;

        UpdateHostClientID();
        UpdateClients();
        UpdateActions();

        currentModel.hostClientIDDidChange += HostClientIDDidChange;
    }
}

private void HostClientIDDidChange(GameStateSyncModel model, int value)
{
    UpdateHostClientID();
}
```

Obrázek 3.15 GameStateSync - nový model

Metoda *OnRealtimeModelReplaced* je volána po připojení na server s aktuálními daty ze serveru. Pokud se uživatel v rámci serveru přepojuje do jiné místnosti, parametr *previousModel* bude neprázdný. V našem prototypu bude uživatel vždy ve stejné místnosti. tedy *previousModel* bude vždy null.

```
public void AddClient(int newHostClientID)
{
    var clientIDModel = new ClientIDModel
    {
        clientID = newHostClientID
    };
    var present = Clients.Any(idModel => idModel.clientID == newHostClientID);
    if (!present)
    {
        Clients.Add(clientIDModel);
    }
}
```

Obrázek 3.16 GameStateSync - nový klient

Metoda *AddClient* přidá nově připojeného klienta do množiny klientů, a to pouze pokud se v ní dosud nevyskytuje.

```
1    public void SetHostClientID(int newHostClientID)
2    {
3        if (hostClientID == -1)
4        {
5            model.hostClientID = newHostClientID;
6        }
7    }
```

Obrázek 3.17 GameStateSync - nový host

Metoda *SetHostClientID* nastaví klienta jako hosta, pouze pokud již host není nastaven.

```
1    public void AddNewAction(int action)
2    {
3        foreach (var clientIDModel in Clients)
4        {
5            if (clientIDModel.clientID != realtime.clientID)
6            {
7                var actionModel = new ActionModel
8                {
9                    action = action
10               };
11               Actions.Add((uint) clientIDModel.clientID, actionModel);
12           }
13       }
14   }
```

Obrázek 3.18 GameStateSync - nová akce

Metoda *AddNewAction* přidá do slovníku akcí novou akci pro každého klienta vyjma toho, který tuto metodu volal.

3.2.4 InputSwitchHelper

Jedná se o jednoduchý script s jedinou metodou pro změnu módu interakce.

Pokud uživatel používá k interakci ruce, modely ovladačů a menu pro umístění kotev se vypnou, avatar představující hráče se zapne.

Pokud místo rukou používá uživatel ovladače, avatar je deaktivován, modely reprezentující ovladače jsou zapnuty a menu pro umístění kotev je také aktivováno.

```
1     [SerializeField] private GameObject controller1;
2     [SerializeField] private GameObject controller2;
3     [SerializeField] private GameObject controllerMenu;
4     [SerializeField] private GameObject robot;
5
6     private bool _alreadyDeactivated;
7     private bool _alreadyActivated;
8
9     private void Update()
10    {
11        if (OVRPlugin.GetHandTrackingEnabled())
12        {
13            if (_alreadyDeactivated) return;
14            _alreadyDeactivated = true;
15            _alreadyActivated = false;
16
17            controllerMenu.SetActive(false);
18            controller1.SetActive(false);
19            controller2.SetActive(false);
20            robot.SetActive(true);
21        }
22        else
23        {
24            if (_alreadyActivated) return;
25            _alreadyActivated = true;
26            _alreadyDeactivated = false;
27
28            robot.SetActive(false);
29            controller1.SetActive(true);
30            controller2.SetActive(true);
31            controllerMenu.SetActive(true);
32        }
33    }
```

Obrázek 3.19 InputSwitchHelper - Update

Kapitola 4

Multi-uživatelské testování v reálném prostředí

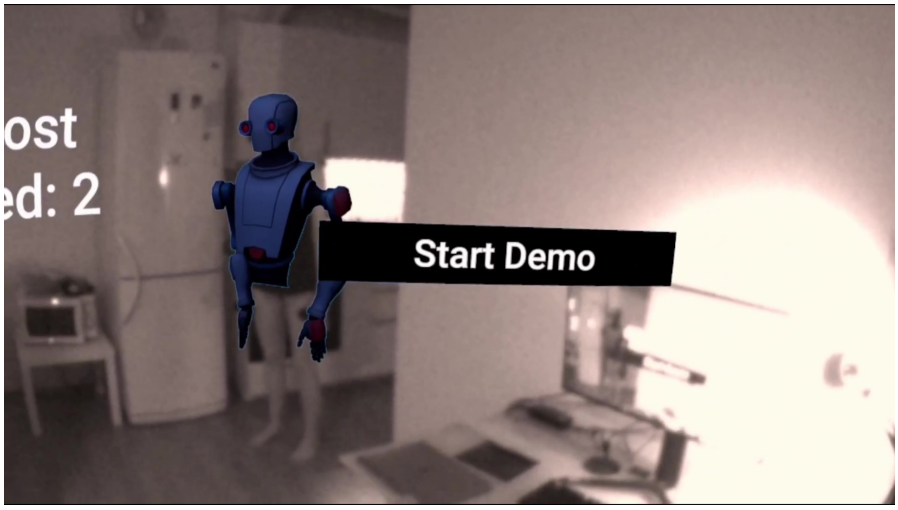
Kapitola pojednává o průběhu testování a dokládá fotodokumentací.

4.1 Průběh testování

V první fázi každý z uživatelů po zapnutí aplikaci umístil kotvu na předem smluvené místo ve scéně. V našem případě se jednalo o prostředek hrany stolu, jak je vidět na následujícím obrázku.

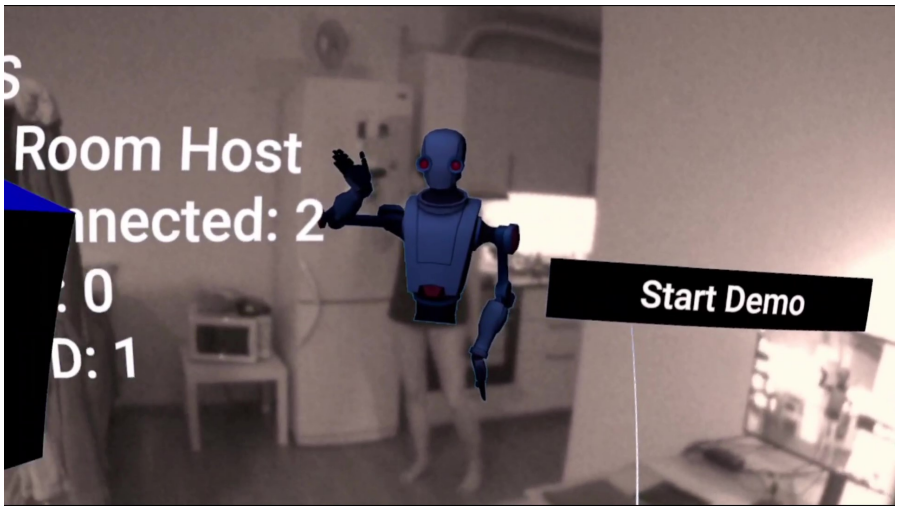
Obrázek 4.1 Kotva ve scéně v levé spodní části - MR

Zároveň je taky dobře vidět avatar v podobě robota jménem Kyle, k dispozici volně z Unity Asset Store. [33]



Obrázek 4.2 Volný postoj - MR

Výstupem zpracovaného obrazu z integrovaných kamer v HMD Oculus Quest 2 je pouze černobílý obraz, tedy i průhled do scény je černobílý.

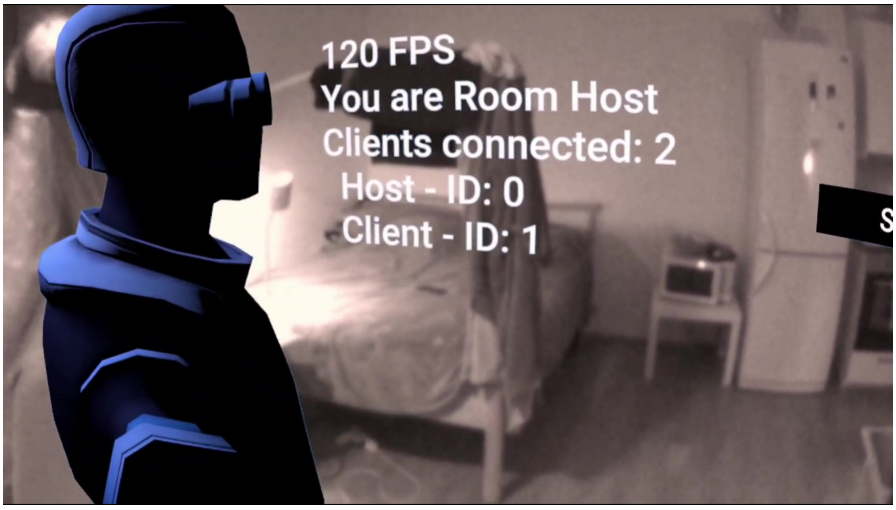


Obrázek 4.3 Mávání - MR

U mávání je vidět lehce opožděné mapování ruky robota, které se snaží dohnat pozici ovladače z průhledu kamer. Zpoždění je velmi nepatrné a nečiní žádnou překážku ve vzájemné interakci.

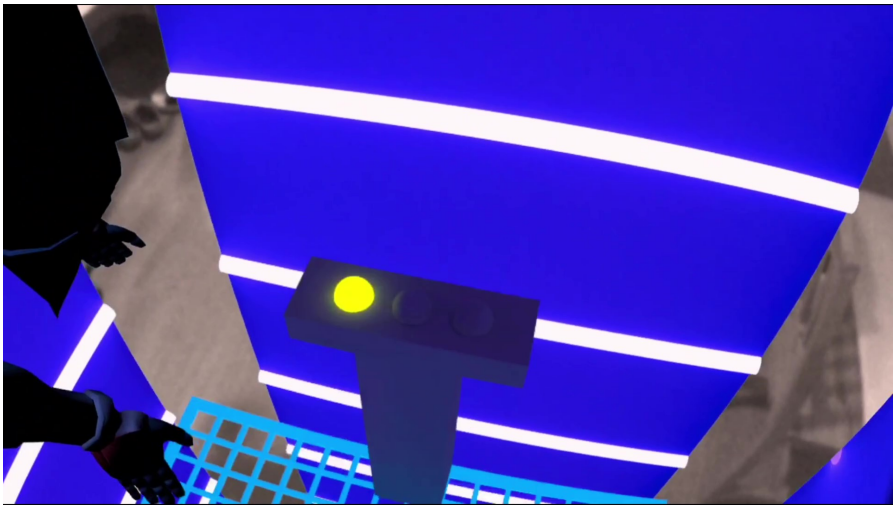
VR systémy byly v průběhu testování připojeny k lokálnímu routeru prostřednictvím WiFi, a to na vzdálenost 5-10 metrů. Komunikace do internetu potom probíhala po optickém kabelu rychlostí 500 Mb/s, tedy lze předpokládat, že napojení na servery Normcore bylo velmi svižné.

Na méně kvalitním síťovém připojení by mohla být synchronizace pomalejší, díky zvolenému protokolu UDP pro přenos pozic a rotací by ale i přesto neměla činit problémy ve vzájemné interakci.



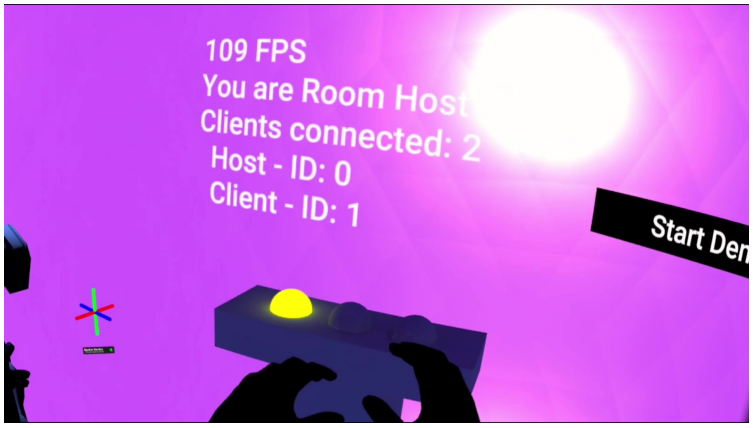
Obrázek 4.4 Viditelná hlava - MR

Zde je vidět, jak je robot mapován na přední část hlavy.



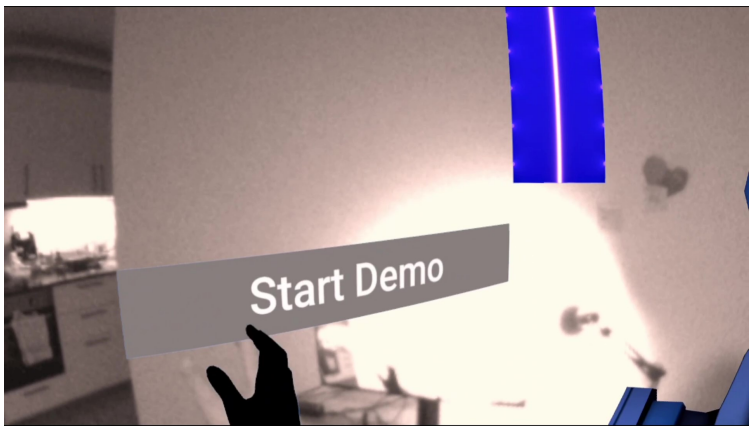
Obrázek 4.5 Uzavírající se výtah - MR to VR

Přechod ze smíšené do virtuální reality.



Obrázek 4.6 Sledování rukou - VR

Obrázek 4.7 Druhý uživatel - VR



Obrázek 4.8 Zapínaní simulace rukou - MR

Kapitola 5
Závěr

Cílem této práce bylo navrhnout řešení multi-uživatelského systému pro mixed reality a jeho implementace pro zvolený systém brýlí virtuální reality s inside-out trackingem.

Hlavní náplní práce bylo řešení problému multi-uživatelského systému pro lokální uživatele, tedy zajištění korektní synchronizace pozic a rotací HMDs s ovladači jednotlivých uživatelů, stejně jako ostatních objektů ve scéně. K tomu bylo zapotřebí vyřešit jak síťovou komunikaci mezi zařízeními, tak zejména zajištění a synchronizaci zarovnaného souřadnicového prostoru se stejným sdíleným počátkem.

Pro implementaci tohoto systému (aplikace) byl zvolen herní engine Unity. Alternativou mohla být implementace v enginu Unreal Engine, pro který ale v době psaní práce nebyla hotová implementace Spatial Anchors (prostorových kotev) v Oculus SDK, které byly nezbytnou součástí řešení problému zarovnaného souřadnicového systému.

Druhou alternativou by byla implementace nativní aplikace v OpenGL / Vulkan, která sice implementací Spatial Anchors disponuje, její implementace by ale zabrala několikanásobně více času bez mnoha přínosů a obtížné rozšiřitelnosti pro uplatnění v reálném použití. Pro síťovou komunikaci byl zvolen framework Normcore. Kvůli své přímočaré implementaci v enginu Unity. Tento framework je možné v omezeném počtu uživatelů (30 aktivních v jeden okamžik) využívat zdarma, což bylo pro účel práce postačující řešení.

Zarovnání souřadnicového systému bylo vyřešeno pomocí Spatial Anchors (prostorových kotev), které při prvním spuštění aplikace uživatelé rozmístí na předem určená místa ve fyzickém prostoru. Následně pak jen stačí dopočítat vzdálenosti a rotace vztažené k těmto kotvám a ty posílat do ostatních zařízení po síti. Jelikož ostatní zařízení také znají polohu těchto kotev, stačí k nim přičíst po síti získaná data a na výslednou pozici umístit odpovídající reprezentaci HMD a ovladačů. Tímto je tedy problém vyřešen.

V případě reálného využití v praxi by do budoucna stálo za zvážení, zda by nebylo vhodnější implementovat komunikaci napříč zařízeními napřímo, bez nutnosti serveru poskytnutého v rámci síťového řešení Normcore. Mohlo by tak dojít ke zrychlení síťové komunikace a věrohodnější reprezentaci scény zejména v případě rychlých pohybů uživatelů. Zároveň by tak ale mohlo dojít ke zkrácení vzdálenosti, na kterou jsou HMDs schopné komunikovat, tedy zde by záleželo na konkrétní představě realizovaného řešení, pro kterou možnost se rozhodnout.

Bibliografie

1. LANG, Ben. *The number of VR headsets on steam blasts through 3 million milestone* [online] [cit. 2022-05-07]. Dostupné z: https://www.roadtovr.com/monthly-connected-vr-headsets-steam-survey-january-2022/.
2. *Hardwarový a softwarový průzkum: April 2022* [online] [cit. 2022-05-10]. Dostupné z: https://store.steampowered.com/hwsurvey/Steam-Hardware-Software-Survey-Welcome-to-Steam.
3. *Česká spořitelna startuje školení pro zaměstnance ve VR. Technologii dodala společnost Cleverlance* [online] [cit. 2022-05-10]. Dostupné z: https://www.cleverlance.com/cz/novinky/Stranky/VR_skoleni.aspx.
4. *Windows Mixed Reality Headset* [online] [cit. 2022-05-12]. Dostupné z: https://www.acer.com/ac/en/US/content/model/VD.R05AP.002.
5. *Your Oculus Quest 2 just got a killer upgrade with this free tool* [online] [cit. 2022-05-11]. Dostupné z: https://www.tomsguide.com/news/your-oculus-quest-2-just-got-a-killer-upgrade-with-this-free-tool.
6. *Powered by AI: Oculus Insight* [online] [cit. 2022-05-09]. Dostupné z: https://ai.facebook.com/blog/powered-by-ai-oculus-insight/.
7. *Pico Neo 3 Link* [online] [cit. 2022-05-08]. Dostupné z: https://bestware.com/en/pico-neo-3-link.html.
8. HEANEY, David. *Pico Neo 3 Link vs Quest 2 Specs: What's The Difference?* [Online] [cit. 2022-05-08]. Dostupné z: https://uploadvr.com/pico-neo-3-link-vs-quest-2-specs-comparison/.
9. *HoloLens 2* [online] [cit. 2022-05-11]. Dostupné z: https://www.microsoft.com/en-us/hololens/hardware.
10. *HP Reverb G2 Virtual Reality Headset* [online] [cit. 2022-05-12]. Dostupné z: https://www.hp.com/us-en/shop/pdp/hp-reverb-g2-virtual-reality-headset.
11. *HTC Vive Cosmos* [online] [cit. 2022-05-12]. Dostupné z: https://www.microsoft.com/en-us/d/htc-vive-cosmos/942ts1jmj7zv.
12. *Cosmos tracking issues* [online] [cit. 2022-05-12]. Dostupné z: https://www.reddit.com/r/Vive/comments/i1sfrg/cosmos_tracking_issues/.
13. *VIVE Cosmos External Tracking Faceplate* [online] [cit. 2022-05-12]. Dostupné z: https://www.vive.com/us/accessory/cosmos-external-tracking-faceplate/.
14. *XTAL 3 Mixed Reality* [online] [cit. 2022-05-12]. Dostupné z: https://www.xtal.pro/product/xtal-3-mr.

15. *Varjo XR-3* [online] [cit. 2022-05-12]. Dostupné z: https://store.varjo.com/xr-3.
16. *Integrate Guardian* [online] [cit. 2022-05-10]. Dostupné z: https://developer.oculus.com/documentation/unity/unity-ovrboundary/.
17. *Oculus Quest Maximum Guardian Space Soon Growing To 15×15 Meters* [online] [cit. 2022-05-10]. Dostupné z: https://uploadvr.com/oculus-quest-guardian-size-increasing/.
18. VINCEY, Hugues. *Hugues Vincey on Twitter* [online] [cit. 2022-05-10]. Dostupné z: https://twitter.com/HuguesVincey/status/1453743928252018690.
19. CLANCEY, James. *James Clancey on Twitter* [online] [cit. 2022-05-10]. Dostupné z: https://twitter.com/jtclancey/status/1453775339969126402.
20. BERKEBILE, Bob. *Bob Berkebile on Twitter* [online] [cit. 2022-05-10]. Dostupné z: https://twitter.com/pixelplacement/status/1453744918279737349.
21. *Passthrough API Overview* [online] [cit. 2022-05-10]. Dostupné z: https://developer.oculus.com/documentation/unity/unity-passthrough/.
22. *Oculus Integration SDK* [online] [cit. 2022-05-09]. Dostupné z: https://developer.oculus.com/downloads/package/unity-integration/.
23. *Spatial Anchors Overview* [online] [cit. 2022-05-10]. Dostupné z: https://developer.oculus.com/experimental/spatial-anchors-overview/.
24. *Spatial Anchors for Unity API Reference* [online] [cit. 2022-05-10]. Dostupné z: https://developer.oculus.com/experimental/spatial-anchors-api-unity/.
25. *Multiplayer and Networking* [online] [cit. 2022-05-10]. Dostupné z: https://docs.unity3d.com/Manual/UNet.html.
26. *Netcode for GameObjects 1.0.0-pre.8* [online] [cit. 2022-05-10]. Dostupné z: https://docs.unity3d.com/Packages/com.unity.netcode.gameobjects@1.0/changelog/CHANGELOG.html.
27. *Normcore. Seamless multiplayer for Unity.* [Online] [cit. 2022-05-10]. Dostupné z: https://normcore.io/.
28. *Unity Client* [online] [cit. 2022-05-10]. Dostupné z: https://normcore.io/documentation/architecture/client.html.
29. *Cloud Infrastructure* [online] [cit. 2022-05-10]. Dostupné z: https://normcore.io/documentation/architecture/cloud.html.
30. *Transport* [online] [cit. 2022-05-10]. Dostupné z: https://normcore.io/documentation/architecture/transport.html.
31. *[PLM] LIGHTMAPPING IGNORES LIGHTS CULLING MASK* [online] [cit. 2022-05-09]. Dostupné z: https://issuetracker.unity3d.com/issues/cullingmask-lightmapping-ignores-lights-culling-mask.
32. *Add a Spatial Anchor Session Script* [online] [cit. 2022-05-10]. Dostupné z: https://developer.oculus.com/experimental/spatial-anchors-persist-content-unity/#add-a-spatial-anchor-session-script.
33. *Space Robot Kyle* [online] [cit. 2022-05-12]. Dostupné z: https://assetstore.unity.com/packages/3d/characters/robots/space-robot-kyle-4696.

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF