VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Rozšířená realita a možnosti brýlí Microsoft Hololens 2

Augmented Reality and Possibilities of Headset Microsoft Hololens 2

Ondřej Fojtík

Bakalářská práce

Vedoucí práce: Ing. Martin Němec, Ph.D.

Ostrava, 2021

Abstrakt

Hlavním cílem této bakalářské práce bylo podívat se na oblast rozšířené reality. Hlavní část je věnována vývoji ukázkových aplikací, které prakticky ukazují mošnosti rozšířené reality na zařízení Hololens 2. V teoretické části práce se lehce zaměřuji na práci v herním enginu Unreal Engine 4, ale vetšína práce se zaměřuje na vývoj s knihovnou OpenXR. V rámci bakalářské práce jsem vytvořil ukázkové aplikace, které prakticky demonstrují možnosti rozšířené reality.

Klíčová slova

Rozšířená realita; DirectX; OpenXR; UWP; CppWinRT; bakalářská práce

Abstract

This thesis covers implementation of an Augmented reality (AR) application, which demonstrates basic capabilities of Hololens 2 headset. This application could be used in places such as museum, where we could use application-rendered text as an exhibit description, or in medicine for its model-rendering capabilities, which we could use for displaying various organs. Practical part of this theses focuses on implementation of an AR application written in C++ using open-source library OpenXR. Furthermore the theses compares Hololens 1 and 2 headsets, and explains the choise of a framework one should choose if developing an AR application.

Keywords

Augmented reality; DirectX; OpenXR; UWP; CppWinRT; bachelor thesis

Poděkování

Rád bych na tomto místě poděkoval Ing. Martinovi Němcovi, Ph.D. za odbornou pomoc a konzultaci při zpracování této bakalářské práce. Dále chci poděkovat kamarádovi Matějovi za pomoc při korekci gramatických chyb.

Obsah

Seznam použitých zkratek a symbolů

AR	–	Augmented Reality (rozšířená realita)
VR	–	Virtual Reality (virtuální realita)
UI	–	User Interface (uživatelské rozhraní)
API	–	Application Programming Interface
MRTK	–	MixedRealityToolKit
SDK	–	Software development kit
UWP	–	Universal Windows Platform
IDE	–	Integrated Development Enviroment
UE4	–	Unreal Engine 4
WinRT	–	Windows Runtime

Seznam obrázků

Seznam tabulek

Seznam výpisu zdrojového kódu

Kapitola 1

Úvod

Rozšířená realita (Augmented Reality) je technologie obohacující reálný svět o interaktivní, počítačem generované prvky napříč více smyslovými věmy včetně vizuálních, sluchových, sematosenzorických nebo čichových. To, co tuto technologii odděluje od virtuální reality, je fakt, že počítač negeneruje veškerý obraz, který je uživateli prezentován, ale pouze svět doplňuje o informace další.

Dnes můžeme rozšířenou realitu potkat v běžném životě nazordíl od dob minulých, kdy k utilizaci rozšířené reality docházelo pouze ve specializované technice v odvětvích jako medicína nebo vojenský průmysl. To, co nám umožnuje mít rozšířenou realitu dnes skoro v každém mobilním zařízení, je hlavně technologický posun v oblasti kamer a výpočetní techniky. Rozšířená realita využívá schopnost kamery rozeznat hloubku obrazu (konkrétně Time-of-Flight kamery) a orientaci zařízení (díky vestavěnému gyroskopu), které zařízení umožňují rozeznat to, kde se zrovna nachází. Tato data jsou v reálném čase posílána do software, který se stará o výpočty spojené s obohacením reality. Tyto výpočty jsou velmi obtížné, ale v dnešní době, kdy se výkon zařízení stále zvyšuje, jsou tyto aplikace stále více aktuální.

Kapitola 2

Aktuální stav AR

Rozšířená realita je technologie insporovaná virtuální realitou. Jedná se o koncept, ve kterém je uživateli prezentován počítačem generovaný obsah, který doplňuje reálný svět [1].

Dnes se s tímto konceptem setkáváme nejčastěji v rámci mobilních zařízení. Speciální kategorii tvoří immersive headsety, které posouvají poholtivosti celého zážitku.

První AR headset byl vytvořen již v roce 1968 [2]. Ve skutečnosti je ale jednalo o VR headset, ve kterém byly informace promítány na průsvitný display, což vyústílo v AR zážitek. AR headsety byly dále vyvíjeny hlavně pro vojenské účely.

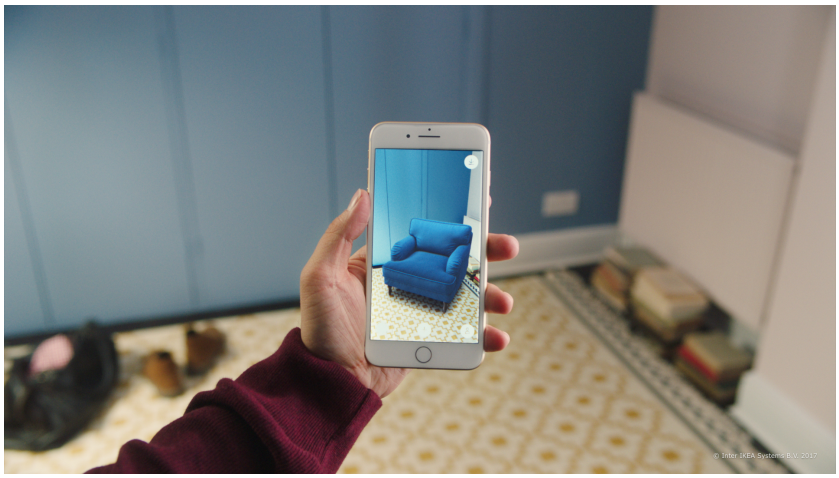
Roku 2014 byly na trh uvedeny brýle Google glass. Jednalo se o první, volně dostupný, AR headset. Tento headset ovšem selhal hlavně na neinformovanosti zákazníků, kteří nevěděli, co s takovým zařízením dělat [3].

Roku 2016 představuje Microsoft immersive headset Microsoft Hololens, který značně překonal brýle Google glass, hlavně pak ve funkcích, které uživateli nabízely.

2.1 Co je to rozšířená realita

Rozšířenou realitu (taky nazývanou Mixed Reality - mixovaná realita) si můžeme představit jako reálný svět doplněný o prvky ze světa virtuálního. Pojem mixovaná realita je ovšem častěji spojovaná s immersive headsety, jako jsou Hololens nebo Moverio. Jedná se o headsety s průhledným hledím, na které jsou promítané počítačem generované prvky.

Dnes se ale častěji setkáme s rozšířenou realitou třeba v telefonu. Jako jeden z příkladů mohu uvést aplikaci Ikea Place, která uživateli umožní pokládat sortiment firmy Ikea do prostoru tak, aby si mohl lépe představit, jak by jeho místnost s tímto prvkem vypadala, viz obrázek 2.1



Obrázek 2.1: Aplikace Ikea Place [4]

2.2 Cíl práce

AR aplikace najdou využití v širokém spektru oborů. AR aplikace většinou pracují s objekty, které v reálném životě usnadňují práci uživatele.

Jako příklad takové aplikace si můžeme představit aplikaci v prostředí muzea, kde je uživateli prezentován textový popis exponátu.

Další příklad najdeme například v medicíně, kde je uživateli umožněno podívat se na holografický objekt orgánu, kterým lze procházet, což mu dává lepší predstavu o jeho celistvosti.

Zajímavé jsou poté aplikace, které mohou elektrikářům ukazovat cesty, kterými vede elektřina skrze zdi nebo automechanikovi model motoru auta, které zrovna opravuje.

Obrázek 2.2: Ukázka AR aplikace [5]

Zajímavá je také aplikace homeAR, která uživateli umožňuje umístění modelu celého domu do prostoru. Uživatel se tak může rovnou podívat na to, jak bude dům v oblasti vypada nebo jaký bude mít výhled z okna, viz obrázek 2.3

Obrázek 2.3: Ukázka aplikace homeAR [6]

Nebo například aplikace Live Home 3D, která uživateli dovoluje komplexněji naplánovat proměnu místností, viz obrázek 2.4

Obrázek 2.4: Ukázka aplikace Live Home 3D [7]

Cílem této práce bylo odzkoušet možnosti AR, srovnat jejich možnosti a odzkoušet vývoj jak s využítím herních enginů, jako Unreal Engine 4 nebo Unity, tak s využitím externích knihoven v kombinaci s jazykem C++.

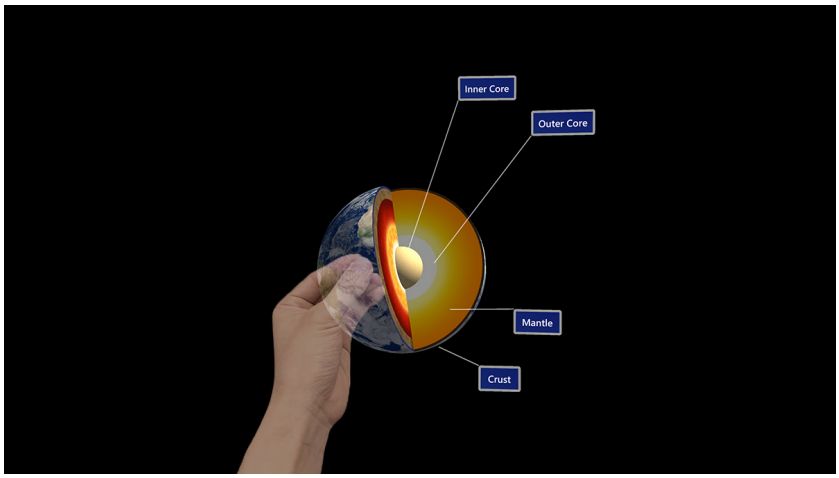
Jedná se o snahu tvorby univerzální aplikace, která je schopna vykreslovat text a objekty. Tuto aplikaci by následně šlo použít pro trorbu jednoduchých AR aplikací.

2.3 Srovnání headsetů Hololens 1 a Hololens 2

Brýle Hololens za dobu jejich vývoje ušly dlouhou cestu. Hololens 2 byly na trh uvedeny tři roky po uvedení Hololens 1 a přinesly nejen hardwarové změny, jako rychlejší procesor nebo větší hledí na kterém je bonusový obsah zobrazován, ale také změny softwarové.

Hlavním rozdílem mezi headsety je způsob, jakým s těmito headsety pracujeme. Hololens 1 nemají schopnost přímé manipulace s rukama[8]. Co to reálně znamená je, že uživatel může použít své ruce, jako by to udělal v reálném světě, pro operace s hologramy, viz obrázek 2.5

Více rozdílných funkcí můžeme vidět znázorněny v tabulce 2.1



Obrázek 2.5: Přímá manipulace s objektem [8]

	Hololens 1	**Hololens 2**
Scene understanding	X	✓
Eye tracking	X	✓
Hand tracking	částečně	✓
Iris scanning	X	✓

Tabulka 2.1: Porovnání funkcí

Dalším velkým rozdílem je počet gest, které jsou na headsetech dostupné. Nejpodstatnějším z přidaných gest je poté Touch, které uživateli umožňuje rukou přímo zasahovat do objektů a tak například zmáčknout holografické tlačítko[9]. Přehled nových gest můžeme vidět v tabulce 2.2

Gesto	**Hololens 1**	**Hololens 2**
Bloom	✓	✓
Air tap	✓	✓
Touch	X	✓
Hand ray	X	✓
Gaze	X	✓
Start menu with two hands	X	✓

Tabulka 2.2: Porovnání gest

V oblasti hardware najdeme největší rozdíl u procesoru, kde došlo k přechodu od x86 na ARM64 platformu. Významnou a velmi vítanou změnou bylo zaké zvětšení hledí, na kterém se zobrazují bonusové informace, které značně přispívá k pohltivosti headsetu[10].

Přehled dalších hardwarových změn můžeme vidět v tabulce 2.3

	Hololens 1	**Hololens 2**
Kamera	2.4 MP (HD)	8 MP (Full HD)
Display (per eye)	1280x720	2048x1080
RAM	2GB	8GB
Bluetooth	4.1	5.0
Field of View	30°	52°
Processor	IA-32 (x86)	Snapdragon 850 (64-bit ARM LTE)

Tabulka 2.3: Hololens specifikace

Kapitola 3

Přehled platforem pro vývoj AR aplikací

Existuje přehršel platforem, které můžeme při vývoji aplikací rozšířené reality použít. Tyto platformy můžeme obecně rozdělit na dvě skupiny podle toho, jestli pro implementaci využívají herních enginů, nebo dovolují uživateli vytvářet s jeho pomocí aplikace vlastní.

3.1 OpenXR

OpenXR je open-souce API od Khronosu, které nám dává přístup k širokému množství platforem pro jak rozšířenou, tak virtuální realitu. Jedná se o kolaboraci mezi téměř každou společností, která je jakkoli spojena s VR nebo AR[11][12].

Obrázek 3.1: Firmy podílející se na vývoji OpenXR [12]

Jednou z hlavních myšlenek při tvorbě tohoto API byla znovupoužitelnost kódu napříč více headsety od různých prodejců. Dnes můžeme OpenXR nalézt například v novém RenderDragon enginu pro Minecraft, Firefox Reality na Hololens 2 a je dokonce podporovaný v Unreal Engine

4.25 nebo Blender 2.83(VR preview) a mnoho dalších. Samozřejmostí je také fakt, že OpenXR API můžeme použít ve vlastním enginu, který si můžeme přizpůsobit vlastním požadavkům.

3.2 Microsoft MixedRealityToolkit

Microsoft MixedRealityToolkit (MRTK) je open-source toolkit, který existuje již od vydání brýlí Hololens 1 [13]. S MRTK jsme schopni vyvíjet aplikace napříč mnoha platformami, jako jsou Microsoft Hololens, Windows Mixed Reality immersive headsety (jedná se o VR headset), OpenVR a Android.

Tento toolkit je pro vývojáře dostupný skrze herní enginy, jako jsou Unity a Unreal Engine 4.

MRTK dříve používal pro vývoj aplikací pro headsety Hololens 1 a 2 Windows Mixed Reality. Dnes ale přechází na platformu OpenXR, která nabízí lepší řešení pro problémy rozšířené reality. Využití OpenXR je zatím dostupné pouze jako Preview verze pro herní engine Unity a to pro verzi 2020.2 a vyšší.

Obrázek 3.2: Logo MixedRealityToolKit [13]

3.3 vuforia

Jedná se o SDK, které vývojáři umožňuje snadno vyvíjet AR aplikace skrze herní enginy Unreal Engine 4 a Unity. Za pomoci vuforia jsme schopni vytvářet AR aplikace pro širokou škálu platforem, jako Android, iOS nebo pro brýle Hololens[14].

Jedná se o komplexní řešení, které dnes stále využívá API Windows Mixed Reality pro řešení problému Spatial anchoru, čímž může upadat oproti MRTK.

Obrázek 3.3: Logo vufiria [14]

3.4 Srovnání

Většina těchto frameworků dokáže řešit všechny problémy, se kterými se při vývoji AR aplikací potkáme. Hlavní rozdíly pak najdeme v tom, jak dobře daný problém řeší. Já jsem během své práce odzkoušel vytvoření aplikace v rámci frameworku Mixed Reality Toolkit v prostředí herního enginu Unreal Engine 4.

Při porovnání této aplikace s aplikací vytvořenou vlastními silami s využitím OpenXR jsem dosáhl mnohem lepších výsledků v aplikaci napsané s OpenXR. Hlavním problémem aplikace vytvořené za pomoci MRTK bylo mapování prostoru. Konkrétně byl tento problém pozorovatelný při pohybu napříč více místnostmi. MRTK nedokázal rozpoznat přechod uživatele z jedné místnosti do druhé, díky čemuž posunul hologramy z jedné místnosti do druhé.

V rámci své práce jsem již nestihnul vytvořit ukázkovou aplikaci ve frameworku vuforia, ale vzhledem k tomu, že pro funkci spatial anchoru používá stejné řešení jako MRTK (totiž skrze Windows Mixed Reality), se domnívám, že by byl výsledek totožný.

Pro vývojáře, kteří plánují využívat herní engine pro vývoj AR aplikace, bych doporučil spíše MRTK, pro jeho Preview verzi OpenXR v herním enginu Unity, které nabízí komplexnější řešení problému mapování a porozumění prostředí.

Kapitola 4

Vývoj aplikací s OpenXR

Následující kapitola popisuje teorii, kterou budeme potřebovat při tvorbě aplikace s využitím knihovny OpenXR.

Vývoj aplikací pro brýle Hololens prostřednictvím OpenXR je velmi rozdílný oproti ostatním možnostem, které využívají už existujících herních enginů, jako je Unreal Engine nebo Unity. To znamená, že největší překážkou při tvorbě těchto aplikací bude vůbec tvorba základního vykreslovacího řetězce předtím, než se dostaneme k práci se samotným API. Jelikož se na brýlích Hololens dají spouštět pouze aplikace typu UWP, jsme nuceni pracovat s takovým vykreslovacím API, které je s tímto typem projektu kompatibilní. V tomto případě jsme tedy nuceni pracovat s API DirectX. Konkrétně jsem při tvorbě aplikace zvolil DirectX 11. To nám sice dává největší kontrolu nad aplikací a dovoluje nám optimalizovat a kontrolovat běh aplikace, ale jedná se o velmi dlouhý a složitý proces.

4.1 Universal Windows Platform

Universal Windows Platform (UWP) je platforma, která se využívá pro univerzální vývoj aplikací pro zařízení s operačním systémem Windows 10, Windows 10 Mobile, Xbox One a Hololens[15]. Samotné UWP nám dává přístup k Windows RunTime API (WinRT), které můžeme například využít pro asynchronní čtění souborů.

4.2 DirectX

DirectX byl představen v roce 1995 za účelem zlepšení vývoje her a 3D software na platformě Windows. Jednotlivé API je označeno slovem Direct a pokračuje popisem samotného API, jako je například Direct3D nebo DirectSound. Dnes je DirectX společně s OpenGL nejpoužívanějším vykreslovacím API v oblasti her a 3D software a naprosto dominantní je pak v oblasti herních konzolí, jako je Xbox nebo PlayStation[16]. Z balíčku DirectX jsou pak nejpoužívanější API:

DirectDraw – dnes už zastaralé. V případě 2D využívá hardwarové akcelerace, pokud to hardware umožňuje. Umožňuje vykreslovat i ve 3D, ale v tomto případě již neumožňuje hardwarovou akceleraci a nebylo by to tak efektivnějším řešením jako třeba při využití Direct3D.

Direct2D – jedná se o 2D grafické API, které využívá hardwarové akcelerace skrze GPU. Efektivnější, než DirectDraw.

Direct3D – jedná se o 3D grafické API pro platformy x86 a ARM. Dnes nahradilo DirectDraw i Direct2D. Používá hardwarovou akceleraci skrze GPU.

DirectInput – jedná se o starší API, které obstarává vstupy z klávesnice nebo ovladače. S příchodem nových konzolí, jako je Xbox360, bylo vytvořeno nové API, Xinput. Xinput je kompatibilní s DirectX verze devět a více.

DirectSound – dnes zastaralé API pro obsluhu zvuků v aplikaci. Dnes se používá XAudio2, které se stejně jako Xinput používá od příchodu nových konzolí. V oblasti zvuku se dále můžeme setkat s DirectSound3D a EAX.

DirectPlay – obsahuje API pro síťovou komunikaci. Je založeno na UDP (User Datagram Protocol) protokolu a umožňuje vícevláknový vývoj.

DirectShow – API, které zašťiťuje multimédia a podporuje širokou škálu formátů.

Pro práci s DirectX se nejčastěji využívá jazyků C a C++, ovšem dnes se velmi často setkáme s implmentací aplikací, které využívají DirectX, v C#. DirectX je dnes častěji a častěji upřednostňováno před API jako je OpenGL a důvodů je mnoho. I přes to, že je OpenGL multi-platformní (což není případ DirectX) a rychlejší, než DirectX, tak je OpenGL složitější na naučení. DirectX má perfektní dokumentaci, která umožňuje snadnou a rychlou práci s API. Samotné API je také čistější a obsáhlejší. Vzhledem k tomu, že více lidí dnes používá DirectX je také důvodem, proč má DirectX dnes lepší podporu ovladačů. Práce s DirectX je pak velmi podobná modernímu OpenGL, což dělá přechod pro programátory velmi snadným.

4.3 Kartézská soustava souřadnic

Jedná se o systém, který používá čísla pro vyjádření pozice bodu v prostoru, jako je například prostor Eukleidovský[17]. Nejčastěji vidíme v počítačové grafice koordinační systémy levoruké nebo pravoruké, které se od sebe liší orientací osy Z. DirectX používá levoruký systém, zatím co OpenXR systém pravoruký. S čím se ale běžně při práci s koordinačním systémem nesetkáme je pohyblivý střed soustavy. OpenXR tvoří počátek v místě inicializace aplikace a veškerý prostor, který v průběhu aplikace nalezne, uloží relativně k počátku. Ovšem v rámci optimalizace stability koordinačního systému v blízkosti uživatele pohybuje počátkem referenčního prostoru podle potřeby.

Důležité z pohledu tvorby aplikací pro AR je také velikost objektů. Referenční prostor od OpenXR, se kterým nadále pracujeme, má jako základní jednotku velikosti jeden metr.

4.4 HLSL - Shader

Shader je program, který slouží k programování grafického řetězce grafické karty. K programování tohoto řetězce využíváme speciální programovací jazyky, které jsou často spárovány grafickým API, které programátor používá. Za výjimku můžeme považovat jazyk CG, který je možno použít jak při práci s OpenGL, tak s DirectX. CG ovšem nepodporuje všechny nejnovější funkce, které nalezneme v GLSL nebo HLSL, proto většinou při práci s OpenGL budeme pracovat s GLSL, a ne CG. Je nutné taky zmínit, že vývoj CG byl ukončen, takže je použitelné jen se staršími verzemi HLSL a GLSL.

HLSL (High-Level Shading Language) je shadovací jazyk vyvíjený společností Microsoft původně pro API Direct3D 9[18]. Později se stalo požadovaným pro API Direct3D 10 a výše. Tento shadovací jazyk je možno využít pouze v kombinaci s DirectX, takže neumožňuje vývoj multiplatformních aplikací. HLSL umožňuje programátorovi pracovat s šesti druhy shaderů:

Pixel shader – V GLSL jako fragment shader. Počítají barvu a další atributy pro každý pixel.

Vertex shader – Jeho účelem je transformování 3D pozice každého bodu do 2D virtuálního prostoru, v jakém se zobrazí na obrazovce. Můžeme také využít pro změnu barvy nebo jiných atributů, ale ovlivní tak celou řadu pixelů.

Geometry shader – Dokáže vytvářet nové body. Dovoluje tak zvětšení kvality výsledného modelu bez nutnosti zvětšování jeho polygonální struktury mimo grafickou kartu.

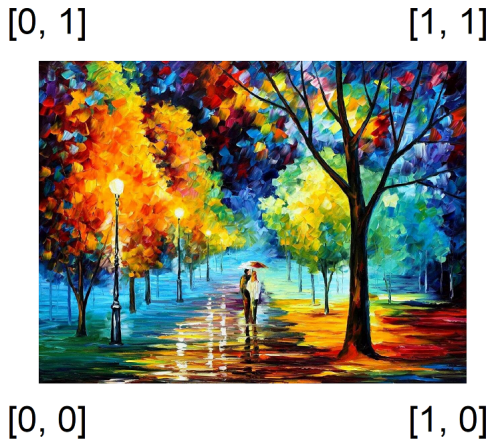
Compute shader – Většinou využívaný pro sdílení dat skrze grafický řetězec.

Tessallation shader – Využívaný pro zjednodušení nebo zdetailnění struktury objektu.

Ray tracing shader – Využívaný pro ray tracing.

4.4.1 Textury

Textury můžeme využít pro nanášení barev jednotlivých pixelů na objekty. Pro jejich mapování na úrovni shaderu se stará sampler, který v DirectX musíme ručně vytvořit. Pro jejich namapování na objekt potřebujeme znát texturovací souřadnice, které odkazují na to, odkud má sampler pixely z obrázku brát, viz. obrázek 4.1



Obrázek 4.1: Ukázka texturovacích souřadnic [19]

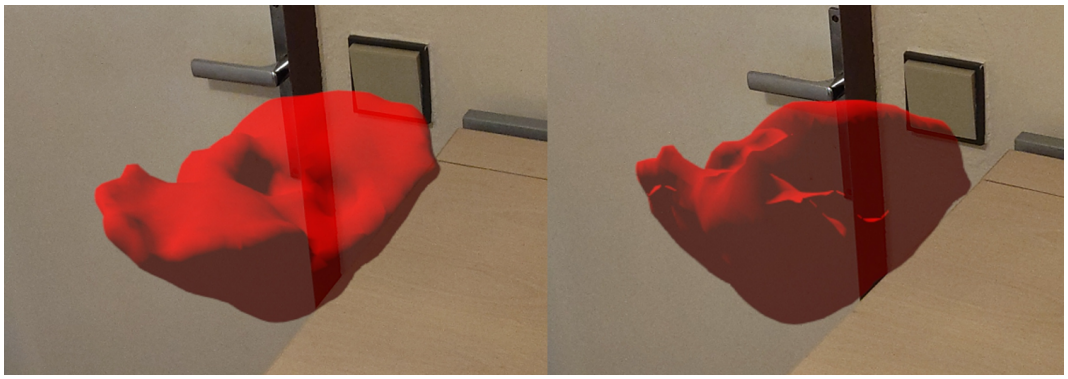
4.4.2 Texture atlas

Texture atlas (často nazývaný jako subtexture) je textura obsahující více textur v jedné. Díky této vlastnosti jsme schopni dostat více textur do shaderu najednou. Odpadá tak také nutnost měnit ID textury, kterou znovna vykreslujeme.

4.4.3 Culling

U vykreslování objektů je třeba dbát na pořadí, ve kterém jsou konktrétní body vykreslovány. Objekty, které vykreslujeme za pomocí trojúhelníků, mohou být uloženy buď v pořadí podle směru hodinových ručiček, nebo proti jejich směru.

Pokud bychom ovšem objekty vykreslovali bez specifického nastavení (tedy s výchozími hodnotami DirectX[18]), tak bychom neviděli takové trojúhelníky, které by byly uloženy ve směru hodinových ručiček. Ukázku, jak by naše objekty mohly ve výsledku vypadat, můžeme vidět na ukázce modelu ledviny, viz. obrázek 4.2



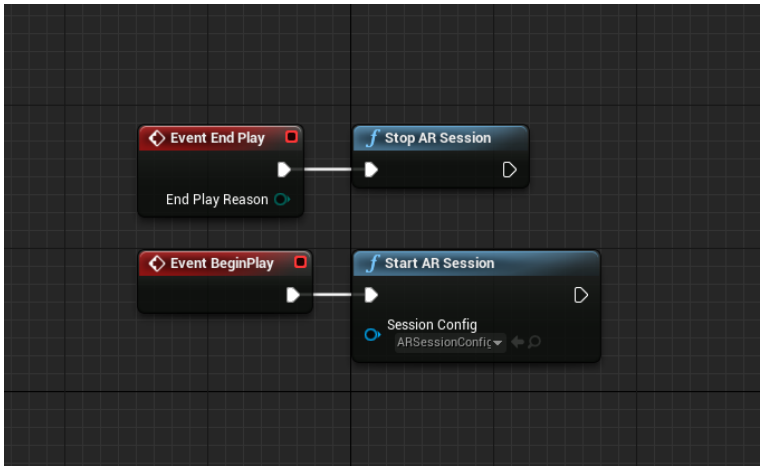
Obrázek 4.2: Ukázka nastavení cullingu

Kapitola 5

Vývoj aplikací v UE4 a MRTK

Vývoj s využitím herního enginu je o poznání rychlejší. V herním enginu Unreal Engine 4 používáme systém blueprintů pro reprezentaci kódu. Unreal Engine 4 nás ale nenutí s blueprint systémem pracovat a dovoluje nám psát C++ kód. MRTK ale obsahuje předpřipravené blueprinty, které značně ulehčují vývoj aplikace.

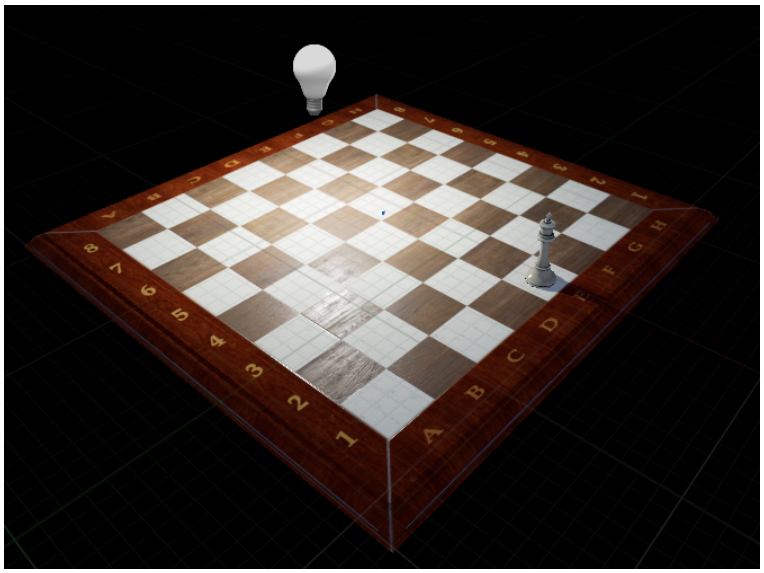
Problém vytvoření AR aplikace lze v UE4 vyřešit jednoduchým blueprintem, viz. obrázek 5.1



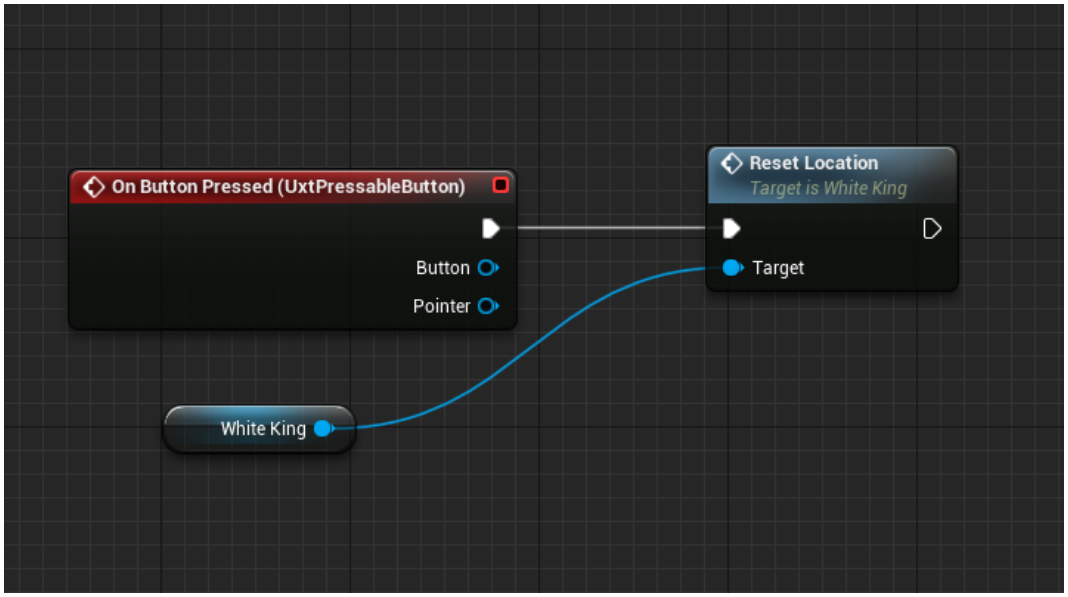
Obrázek 5.1: Zahájení AR aplikace v MRTK

Pro zobrazování objektů ve scéně stačí vytvořit scénu jako v jakékoli jiné 3D aplikaci. Ukázku takové scény můžeme vidět na obrázku 5.2

MRTK blueprinty implementují mnoho pokročilých funkcí. Já jsem pro ukázku snadnosti použití těchto blueprintů zvolil tvorbu tlačítka, které posouvá pozici krále na šachovnici na jeho původní pozici. Tento blueprint můžeme vidět na obrázku 5.3



Obrázek 5.2: Scéna v UE4



Obrázek 5.3: Blueprint pro tlačítko

Kapitola 6

Implementace

V rámci této kapitoly bych se chtěl věnovat zajímavým problémům, které bylo třeba vyřešit v rámci tvorby aplikace s využitím knihovny OpenXR.

6.1 Vývojové prostředí

Pro vývoj UWP aplikace můžeme využít jakyzů C++ nebo C#. Pokud ale chceme pracovat s nízkoprofilovými knihovnami, jako je OpenXR, tak jsme nuceni pracovat s s jazykem C++.

Jako samotné prostředí jsem využil Visual Studio 2019 nejen díky mé největší obeznámenosti s tímto IDE, ale hlavně pak díky jeho schopnosti vzdáleného debugování a nasazování.

6.1.1 Remote debugging

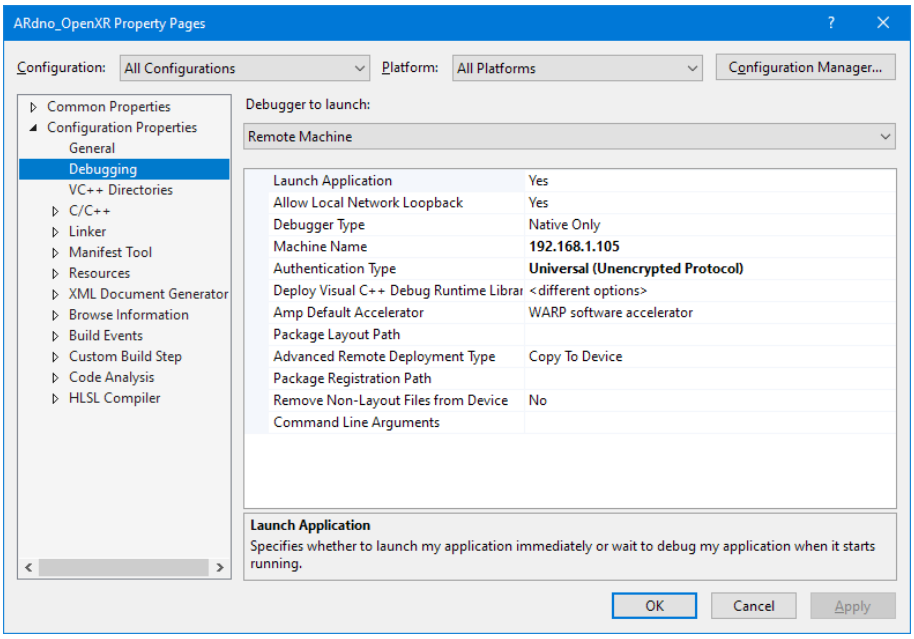
Remote debugging (vzdálené debugování) je funkce Visual Studio 2019, které vývojáři umožňuje debugování takových aplikací, které běží na jiném zařízení než tom, na kterém aplikaci píšeme..

Prerekvizity pro použití vzdáleného debugování jsou

- Propojení zařízení přes Wifi nebo USB kabel. U zařízení, které je připojeno přes Wifi, je třeba znát jeho IP adresu.
- Instalace VS2019 Remote Tools, které tuto funkci implementují.
- Zařízení musí běžet na architektuře x86, x64 nebo ARM64.
- Zaříření musí používat operační systém Windows 7 a vyšší vyjma jejich mobilních verzí (jedná se pouze o Win8 mobile a Win10 mobile).
- Zapnutí Developer mode a Remote machine na obou zařízeních. Jedná se o Windows nastavení. Na mém zařízení nastal problém, kdy možnost Developer mode bylo zešedlé, takže jsem ho nemohl zapnout. Tento problém se dá řešit pomocí Windows Managment Console, kterou

využijeme pro editaci Local Group Policy na našem zařízení. Pro povolení se stačí navigovat skrze konzoli Computer Configuration -> Administrative Templates -> Windows Components -> App package deployment a povolit nastavení Allows development of Windows Store apps and installing them from an integrated development enviroment (IDE) a Allow all trusted apps to install.

Konfiguraci pro Remote debuging v C++ můžeme vidět na obrázku 6.1



Obrázek 6.1: Ukázka nastavení remote debuggingu

6.1.2 Remote deployment

Remote deployment (vzdálené nasazení) vývojáři umožňuje aplikaci nasazovat a instalovat do cílového zařízení. Jedná se o část procesu vzdáleného debugování a má stejné prerekvizity jako právě vzdálené debugování.

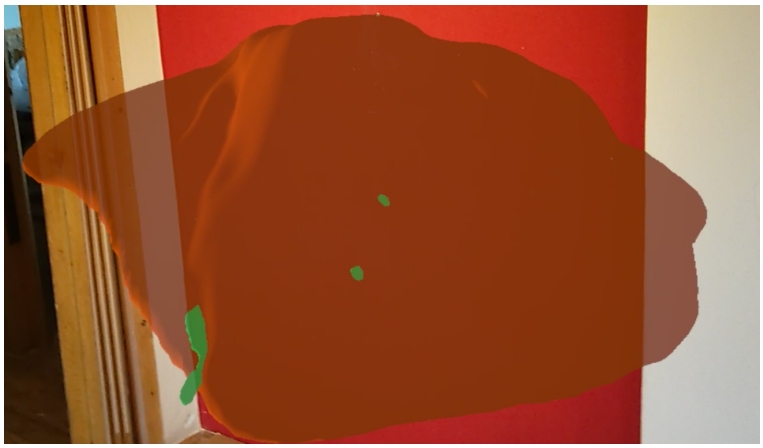
6.2 ARdno_EU4

Aplikace vytvořená v prostředí herního enginu Unreal Engine 4. Jedná se o jednoduchou aplikaci, která vykresluje objekt v prostoru (konkrétně šachovnici, viz. obrázek 6.2). Uživatel může pohybovat s figurkami, demonstruje užití tlačítek a dovoluje uživateli pohybovat a škálovat objekty. Tato aplikace nemá reálné využití, kromě ukázky funkčnosti aplikací vytvořených v rámci herního enginu Unreal Engine 4 a frameworku MRTK.

Obrázek 6.2: Ukázka šachovnice v aplikaci ARdno_UE4

6.3 ARdno_D

Aplikace vytvořená ve vlastním enginu s využitím C++ a knihovny OpenXR. Tato aplikace slouží jako aplikace pro cílového uživatele, protože uživateli dovoluje pouze položení posunutého středu soustavy 6.7.2 gestem pro stisk na pravé ruce. Gestem pro stisk na pravé ruce jsme poté schopni procházet modely, které jsou nahrané ve složce Pictures/ARdno_obj. Aplikace také dokáže vykreslovat jak text, tak i komplexnější hologramy, díky kterých jsme schopni aplikaci použít například v muzeu, pro doplnění textových informací exponátů nebo v medicíne, pro procházení hologramů orgánů. Ukázku objektu vykreslovaného v aplikaci můžeme vidět na obrázkách 6.3 a 6.4.



Obrázek 6.3: Ukázka objektu jater

Obrázek 6.4: Ukázka objektu jater při pohledu dovnitř

6.4 ARdno_E

Tato aplikace slouží pro nastavování hologramů, které jsou následně použity v aplikaci ARdno_D 6.3. Aplikace uživateli umožňuje, stejně jako v aplikaci ARdno_D, položení posunutého středu soustavy 6.7.2 pomocí gesta stisku na pravé ruce. Gesto stisku na levé ruce následně slouží pro položení objektů do prostoru.

6.5 ARdno_parser

Parser pro aplikaci, která konvertuje formát .obj do speciálního formátu, který je dále využívaný v rámci aplikace. Tento formát je dále popsaný v sekci 6.7.5

6.6 Vykreslení textu

U vykreslování textu máme mnoho možností implementací různých částí problému.

Jedním z těchto problémů je zisk texturovacích souřadnic. Pro jejich zisk se nám nabízejí dvě možnosti.

- Jeden obrázek pro každý charakter
- Jeden obrázek pro všechny charaktery

6.6.1 Jeden obrázek pro každý charakter

Nabízí se možnost přepínat texturu pro každý charakter, který cheme vykreslit. Tato možnost je více použitelná pouze v případě, kdy bychom počítali s tím, že každý charakter vykreslujeme zvlášť. Při vykreslování více charakterů dochází k takovému problému, že do Shaderu musíme poslat náležitý počet ID textur tak, abychom věděli, ze které textury máme pixely brát. To je ale při proměnlivém počtu znaků problém.

6.6.2 Jeden obrázek pro všechny charaktery

Jedná se o řešení, kde uvažujeme nad jednou texturou, která obsahuje obrázky všech znaků. Jde o texture atlas, viz. 4.4.2, takže musíme získat správné texturovací souřadnice z textury.

Jednoduchým způsobem, jakým můžeme problém vyřešit, je vytvořit si texturu takovou, ve které máme znaky za sebou seřazené podle jejich ASCII hodnoty, jak můžeme vidět na obrázku 6.5

Obrázek 6.5: Ukázka texture atlasu charakterů

Pokud známe rozměry obrázku a znaku, tak jsme snadno schopni získat texturovací souřadnice, které dále použijeme pro vykreslení. Ukázku toho, jak souřadnice z textury můžeme získat, můžeme vidět na ukázce kódu 6.1 Tyto souřadnice později použijeme při tvorbě vertex a index bufferu.

```
XrVector2f sprite_size;
sprite_size.x = 32.0f;
sprite_size.y = 32.0f;

XrVector2f texture_size;
texture_size.x = 4096.0f;
```

```
texture_size.y = 32.0f;

int32_t x = c - ' ';

XrVector2f* coords = new XrVector2f[4];
coords[0] = { (x * sprite_size.x) / texture_size.x, 0.0f };
coords[1] = { ((x + 1) * sprite_size.x) / (texture_size.x), 0.0f };
coords[2] = { ((x + 1) * sprite_size.x) / (texture_size.x), 1.0f };
coords[3] = { (x * sprite_size.x) / (texture_size.x), 1.0f };
```

Algorithm 6.1: Zisk texturovacích souřadnic

Toto řešení také výrazně usnadňuje práci při změně fontu. Vzhledem k tomu, že pro vykreslování využíváme texture atlas, nám stačí vygenerovat texture atlas s jiným fontem.

6.6.3 Text jako objekt

Při řešení problému se také musíme zaměřit na to, jak budeme text reprezentovat v rámci objektů, se kterými budeme v aplikaci pracovat. Text si můžeme představit jako

- jeden objekt, který obsahuje všechny písmena
- více objektů, kde každý jeden objekt reprezentuje jedno písmeno

6.6.4 Více objektů

Řešení s více objekty, tedy takové, kde každému objektu připadá jeden znak, vypadá implementačně snažší. Pro vykreslení jednoho znaku nám stačí získat texturovací souřadnice náležitého znaku, které použijeme v shaderu, viz sekce 4.4.1

Pro vykreslení více písmen, jako je třeba TEST, musíme zjistit, jaká bude pozice dalších písmen. Pokud bychom zvolili jako hlavní znak první objekt, v tomto případě tedy T, museli bychom vypočítat souřadnice, na kterých se bude vykreslovat další znak.

Dalším problémem je orientace. Pokud bychom text chtěli zarotovat, tak bychom nemohli čekat, že všechny ostatní objekty stačí zarotovat o stejnou hodnotu, jelikož se nám rotuje celý text. Rotace by tedy musely být relativní k hlavnímu bodu. Po takové rotaci by bylo třeba opět přepočítat pozice dalších objektů na korektní místo.

Problematická je taky volba hlavního bodu, tedy bod, podle kterého se budou ostatní rotovat a pozicovat. Pokud bychom například zvolili jako hlavní bod první písmeno, v tomto případě T, tak bychom rotovali kolem něj. To by znamenalo, že pokud bychom chtěli text zarotovat o 30° kolem osy Y, tak bychom měli na stejném místě písmeno T a všechny ostatní písmena by se posunuly na jiné pozice. Ideálnější by pro naše účely bylo zvolit jako hlavní písmeno takové, které je uprostřed.

Problém opět nastává v případě, kdy máme slovo se sudým počtem písmen. Pro takové slovo bychom museli dělat o mnoho složitější počty pro zisk jeho pozice a rotace.

6.6.5 Jeden objekt

Řešení s jedním objektem může na první pohled vypadat jako složitější ze dvou řešení. Ve skutečnosti nám ale odpadá celá řada starostí, které je třeba řešit při implementaci s více objekty.

Pro implementaci budeme muset použít dynamické vertex a index buffery. Dynamický buffer je takový buffer, který za běhu aplikace dokáže změnit svou velikost. Tuto vlastnost využijeme, jelikož text, který chceme vykreslit, nemusí být vždy stejně dlouhý. Dynamický buffer ale není nekonečný a při jeho tvorbě musíme deklarovat jeho maximální velikost. Já jsem ve své implementaci zvolil velikost třiceti dvou znaků. Ukázku, jak vytvořit dynamický vertex a index buffer, můžeme vidět na ukázce kódu 6.2

```
// Dynamic VB
D3D11_BUFFER_DESC vertexBufferDesc;
ZeroMemory(&vertexBufferDesc, sizeof(vertexBufferDesc));
vertexBufferDesc.Usage = D3D11_USAGE_DYNAMIC;
vertexBufferDesc.ByteWidth = sizeof(QuadShader::Vertex) * (4 * 32);
vertexBufferDesc.BindFlags = D3D11_BIND_VERTEX_BUFFER;
vertexBufferDesc.CPUAccessFlags = D3D11_CPU_ACCESS_WRITE;
D3D11_SUBRESOURCE_DATA initial_data{ QuadShader::c_quadVertices };
HRESULT hr = m_device->CreateBuffer(&vertexBufferDesc,
                                    &initial_data,
                                    m_quadVertexBuffer.put());

// Dynamic IB
D3D11_BUFFER_DESC indexBufferDesc;
ZeroMemory(&indexBufferDesc, sizeof(indexBufferDesc));
indexBufferDesc.Usage = D3D11_USAGE_DYNAMIC;
indexBufferDesc.ByteWidth = sizeof(unsigned short) * (6 * 32);
indexBufferDesc.BindFlags = D3D11_BIND_INDEX_BUFFER;
indexBufferDesc.CPUAccessFlags = D3D11_CPU_ACCESS_WRITE;
D3D11_SUBRESOURCE_DATA data{ QuadShader::c_quadIndices.data() };
hr = m_device->CreateBuffer(&indexBufferDesc, &data, m_quadIndexBuffer.put());
```

Algorithm 6.2: Deklarace dynamického bufferu

Dále musíme vertex a index buffer naplnit daty tak, aby tvořil výsledný text. Jeden znak jsme schopni popsat čtyřmi body a šesti indexy, které určují v jakém pořadí budou body vykreslovány.

Generování informací pro index buffer je velmi snadné. Za předpokladu, že nebudeme měnit pořadí bodů, ve kterém body ukládáme do bufferu, jsme schopni ukládat stejné indexy pro každý znak. Tyto indexy stačí náležitě posouvat, viz. ukázka kódu 6.3

```
std::vector<unsigned short> _indices;

for (int i = 0; i < size; i++)
{
    int32_t init_value = i * 4;
    _indices.push_back(init_value + 0);
    _indices.push_back(init_value + 1);
    _indices.push_back(init_value + 2);

    _indices.push_back(init_value + 2);
    _indices.push_back(init_value + 3);
    _indices.push_back(init_value + 0);

}
unsigned short* indices = &_indices[0];
```

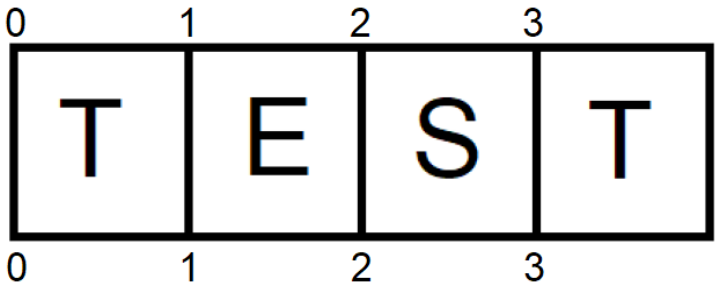
Algorithm 6.3: Zisk index bufferu

Generování informací pro vertex buffer je o něco obtížnější. Musíme si uvědomit, že jediná souřadnice, která se bude napříč objektem posouvat, je ta na ose X.

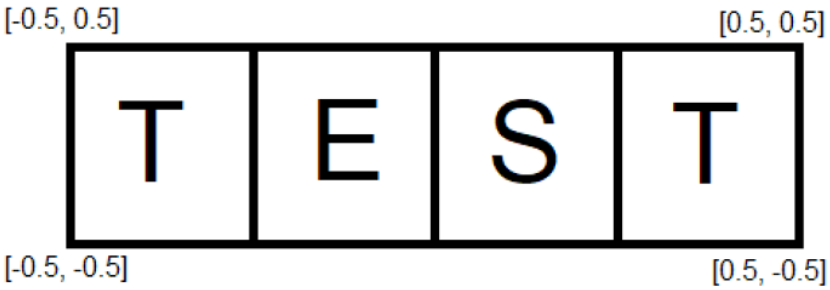
U popisu informací pro jeden objekt bude tedy většina informací stejná s výjimkou těch na ose X. Pozici na ose X můžeme ale například určit pořadím znaku, pro který vertex buffer zrovna tvoříme, v textu za předpokladu, že velikost jednoho znaku bude v lokálním prostoru jedna, viz. obrázek 6.6

Toto řešení by samo o sobě fungovalo, ale práce s ním by byla velmi obtížná, jelikož by střed každého objektu byl jinde. Dalším problémem by bylo třeba jeho škálování. Proto je lepší tyto souřadnice transformovat tak, aby bodu textu, který najdeme v nejlevější části textu, odpovídala pozice -0.5, a nejpravější souřadnici pozice 0.5. Viz. obrázek 6.7.

Jelikož jsme ukládali souřadnice na ose X tak, aby odpovídaly pozicím charakterů textu, tak víme, že nejvyšší číslo, které v souřadnicích dostaneme, bude rovno délce textu. Takto nám dále stačí jen přepočítat jejich současnou pozici na hodnotu mezi nulou a jedničkou, kterou získáme jako jeho současnou relativní pozici k nule a délce textu. Abychom dostali střed objektu doprostřed vykresleného textu, tak výslednou pozici ještě zmenšíme o 0.5, tedy polovinu jeho velikosti.



Obrázek 6.6: Neupravené souřadnice textu



Obrázek 6.7: Upravené souřadnice textu

Tvobru takového vertex bufferu můžeme vidět na ukázce kódu 6.4

```
XrVector2f* coords = get_coords(text[i]);

// Vertex 1
{
    Vertex v;
    v.Position.x = reverse_lerp(i, 0, (float)size);
    v.Position.y = -0.5f * (1 / (float)size);
    v.Position.z = 0.0f;

    v.Color = { 1.0f, 1.0f, 1.0f };
    v.TexCoords = coords[0];
    _vertices.push_back(v);
}
```

Algorithm 6.4: Zisk vertex bufferu

6.6.6 Víceřádkový text

Více řádkový text je realizovatelný stejným způsobem jako bylo popsáno v 6.6.5 s tím rozdílem, že bychom museli transformovat i souřadnice na ose Y.

Jiným řešením, které jsem při vypracování zvolil, byla myšlenka položení více jednořádkových textů pod sebe tak, že tvoří text víceřádkový.

Ukázku víceřádkového textu můžeme vidět na obrázku 6.8

Obrázek 6.8: Ukázka víceřádkového textu

6.6.7 Porovnání

Po stránce funkčnosti jsou obě řešení validní. Nemalé rozdíly ovšem najdeme nejen v jejich ovladatelnosti, ale také výkonu.

Ovladatelnost textu, který je tvořen jedním objektem je o hodně snažší, než u toho, který je tvořen objekty několika. Pokud bychom chtěli text škálovat, posouvat nebo s ním otáčet, nevyhnuli bychom se přepočítávání všech informací všech ostatních objektů. Na druhé straně dokážeme každou takovou operaci vyřešit jednoduchým násobením matic.

Větší rozdíly si dokážeme představit ve výkonu. Je o hodně efektivnější vykreslit jeden veliký objekt, než nekolik malých, protože každý draw-call (@edit - překlad) je velmi drahý. Dalším problémem pro řešení s více objekty, spojeným s výkonem, je nutnost neustále přepočítávat nové souřadnice pro každou operaci. V AR se často potkáme se situací, kdy s objektem pohybujeme nebo rotujeme. Ve svém řešení jsem například pro usnadnění umisťování objektů do prostoru nechal vykreslovat textovou ukázku na levé ruce uživatele tak, aby si dokázal lépe představit, jak

bude výsledný text vypadat, viz. obrázek 6.9. Ruka uživatele se samozřejmě neustále pohybuje. V takovém případě by docházelo k velikému výkonnostnímu rozdílu. Bohužel jsem již v rámci své práce nestihnul druhé řešení naimplementovat a ukázat tak na konkrétních datech, o kolik je jedna metoda od druhé efektivnější.

Obrázek 6.9: Ukázka textu vykreslovaného na ruce

6.7 Ukládání prostoru

Ukládání objektů v prostoru je velmi zajímavý problém, se kterým se ve 3D nebo VR aplikacích nepotkáme. Ve většině takových aplikacích máme objekty, tedy jejich pozici a rotaci, uloženy do prostoru. Tyto data o jejich pozici zůstávají vždy stejná, bez ohledu na to, kdy aplikaci otevřeme, jelikož to, co je uživateli zobrazováno, je skrze kameru, kterou dokážeme vždy inicializovat na stejné pozici.

V rozšířené realitě ale realisticky nemáme možnost inicializovat kameru vždy na stejném místě, jak je tomu v ostatních 3D aplikacích, jelikož pohled kamery je ten pohled, kam se dívá uživatel. Pro vyřešení tohoto problému existuje mnoho řesení.

- Inicializace kamery na stejném místě
- Pomocí posunutého středu souřadné soustavy
- Gramova–Schmidtova ortogonalizace
- Ray tracing a image recognition

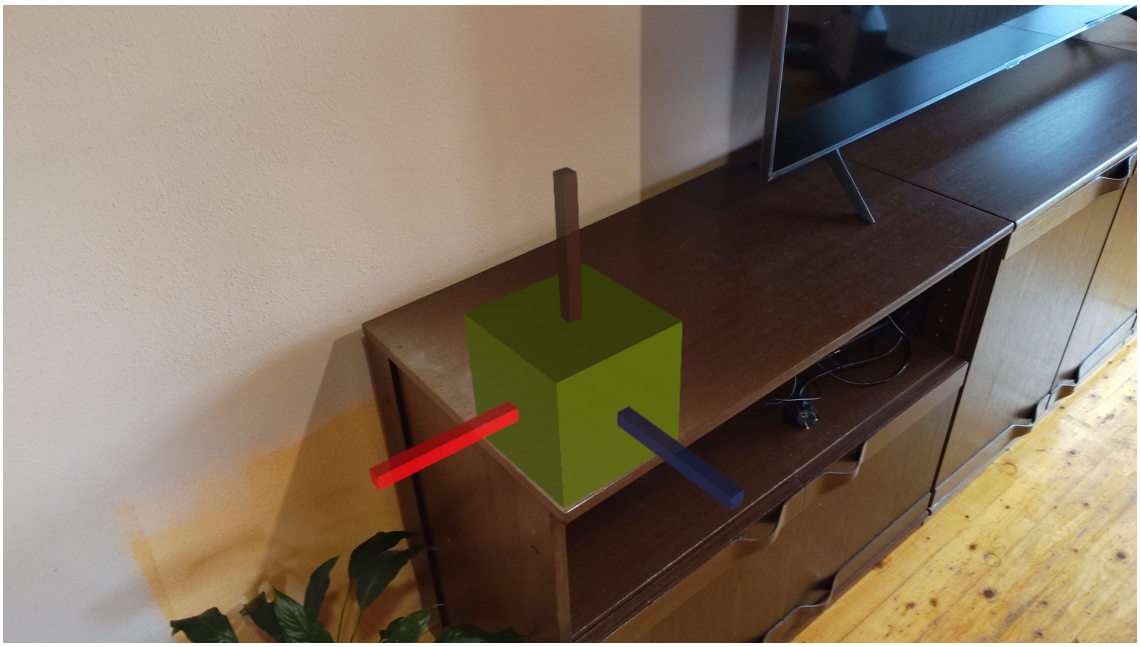
6.7.1 Inicializace kamery na stejném místě

Jedná se o nejsnažší řešení, které není realisticky proveditelné. Vyžaduje, aby uživatel při otevírání aplikace, byl přesně na stejném místě a jeho hlava byla stejně orientovaná jako byla při jejím nastavování. Takto bychom mohli simulovat nastavení pohledu kamery. Toto řešení je ale velmi težko realizovatelné a i při sebemenším pohybu uživatele by došlo ke špatné kalibraci a hologramy by tak byly mimo pozici, kterou jsme jim uložili. Proto bych jí při implementaci nedoporučoval.

6.7.2 Pomocí posunutého středu souřadné soustavy

Při tomto řešení probíhá kalibrace za pomoci uživatele. Uživatel musí po zapnutí aplikace položit orientační bod do prostoru. Tento bod funguje jako posunutý střed souřadné soustavy. Body v prostoru pak tedy nejsou uloženy do prostoru relativně ke středu soustavy, ale k posunutému bodu.

Jedná se o řešení, které jsem využil ve své implementaci. Jde o lepší řešení, než řešení 6.7.1, ale k perfektnímu řešení má daleko. Abychom dosáhli co nejpřesnějších pozic, se kterými jsme objekty ukládali, musíme vynaložit poměrně dost úsilí na to, aby jejich pozice byla co nejpřesnější, což může být nepraktické. Při implementaci jsem jako model referenčního bodu zvolil krychli s orientací os, viz. obrázek 6.10, který usnadňuje orientaci. Můžeme tak například tuto krychli položit do rohu místnosti, popřípadě jiné, námi vyznačené, orientační místo, a orientovat se podle jejich hran, zda jsme objekt položili na stejné místo.



Obrázek 6.10: Ukázka posunutého středu

Jednou z možností, jak tuto kalibraci ulehčit by bylo pomocí detekce a porozumění prostoru. Pokud bychom totiž například detekovali podlahu nebo jinou stěnu, tak bychom mohli objekt k této

stěně “přilepit”. To by znamenalo, že minimálně orientace by byla perfektně zarovnaná. Toto řešení jsem již v rámci své práce nestihl implementovat.

6.7.3 Gramova–Schmidtova ortogonalizace

Jedná se pořád o řešení, kde je nutná akce uživatele. Tato metoda je ovšem mnohem konzistentnější, než popsaná v 6.7.2. Gram–Schmidtova metoda nám umožňuje vytvořit ortogonální bázi, kterou bychom mohli použít jako posunutý střed soustavy. Úkol uživatele by bylo položit čtyři body do prostoru tak, aby tři z nich reprezentovaly osy soustavy. Čtvrtý bod by potom sloužil jako pozice středu sousavy.

Tato metoda by se opět dala vylepšit pomocí detekce a porozumění prostoru, jak bylo popsáno v 6.7.2

6.7.4 Ray tracing a image recognition

Jedná se o nejkomplexnější a v tuto chvíli asi nejideálnější možnost, jak problém ukládání prostoru řešit. Tato metoda nevyžaduje žádnou akci uživatele. Vzhledem k pokročilejší kameře na zařízení Hololens 2 jsme schopni dobře rozeznávat prostor. Pro funkčnost této metody je třeba, aby v místnosti byl nějaký marker nebo objekt, který by brýle za pomoci ray tracingu a image recognitionu dokázaly odhalit. Po nalezení takového předmětu by se posunutý střed soustavy posunul do detekovaného místa a automaticky se tak nakalibroval.

6.7.5 Formát uloženýh dat

Formát, ve kterém ukládám data uložených objektů, můžeme vidět na ukázce 6.5. Každý řádek reprezentuje jeden objekt a popisuje pozici, orientaci, škálu (@edit - preklad (scale)), typ objektu a text. Typ objektu říká aplikaci, jaký typ objektu má vytvořit.

Aplikace má v současném stavu dva typy objektů

- model - v uložených datech reprezentován jako 0
- text - v uložených datech reprezentován jako 1

Objekt typu model vykresluje aktivní model, který si uživatel vybere. Text označuje objekt vykreslující text. Uživatel může tyto data kdykoli přepsat a ručně tak upravovat scénu, která bude v rámci aplikace vykreslována.

```
-1.063035;0.323608;4.237508;-0.116299;0.727993;0.666530;0.110624;0.100000;
0.100000;0.100000;1;SAMPLE;
-2.940711;0.312193;1.625429;-0.072517;0.974939;0.209292;0.020809;0.500000;
0.500000;0.500000;1;MONITOR;
```

```
1.069815;0.206465;0.628414;-0.117146;-0.073440;0.964544;0.224807;0.500000;
0.500000;0.500000;0;-;
```

Algorithm 6.5: Formát uložených dat

6.8 Práce se soubory

Práce se souborovým systémem je na platformě UWP velmi omezená, jelikož Windows značně omezuje systém, do kterého můžeme přistoupit. Konkrétně jsme schopni přistoupit pouze k datům, které se nachází v root složce naší aplikace nebo ve "známých složkách". Jedná se o složky typu Obrázky, Dokumenty, 3D Objekty a podobně.

Pro přistup k datům navíc nesmíme využít C++ fstreamy. Jediným způsobem, jakým můžeme přistoupit k datům, je skrze knihovnu WinRT (v našem případě CppWinRT). Tuto knihovnu používáme k asynchronnímu přistupu k datům.

Trochu nešťastná je potom dokumentace samotné knihovny. WinRT se používá hlavně v kombinaci s jazykem C#, což je taky důvodem, proč je veškerá dokumentace v kombinaci právě s tímto jazykem. Naštěstí je syntaxe pro C++ velmi podobná té v C#, takže si většinu věcí dokážeme domyslet.

Ukázku toho, jaké rozdíly implementačně čekat, můžeme vidět na ukázkách kódu 6.6 a 6.7, které ukazují, jak můžeme číst data ze souboru.

```
StorageFolder picturesFolder = KnownFolders.PicturesLibrary;
StorageFile file = await picturesFolder.GetFileAsync("application_data.txt");
string fileContent = await FileIO.ReadTextAsync(file);
```

Algorithm 6.6: Čtení dat v C#

```
using namespace winrt;
using namespace winrt::Windows::Foundation;
using namespace winrt::Windows::Storage;

StorageFolder picturesFolder = KnownFolders::GetFolderForUserAsync(
                                nullptr,
                                KnownFolderId::PicturesLibrary).get();
StorageFile file_ = picturesFolder.GetFileAsync(L"application_data.txt").get();
hstring file_content_ = FileIO::ReadTextAsync(
                        file_,
```

```
                    Streams::UnicodeEncoding::Utf8).get();
std::string data = winrt::to_string(file_content_);
```

Algorithm 6.7: Čtení dat v C++

6.9 Vykreslování modelů

Vykreslování modelů je problém, jelikož pracujeme s platformou ARM64 a UWP. Většina zažitých knihoven pro načítání objektů jsou tvořeny pro platformu x86, ale například Assimp má knihovnu i pro ARM64 platformu.

Problémová je UWP, která značně omezuje přístup uživatele do souborového systému. Tyto knihovny také neřeší čtení dat s využitím knihovny WinRT, ale s pomocí fstreamů. Tato problematika je popsána v rámci sekce 6.8

Tento problém jsem vyřešil pomocí vlastního parseru. Jedná se o velmi neefektivní řešení, kdy na jiném zařízení rozeberu a zapíšu data do souboru tak, abych je mohl v brýlích načíst.

V rámci svého řešení jsem chtěl přepsat knihovnu pro načítání objektů s pomocí WinRT tak, abych ji mohl na zařízení používat. Bohužel jsem to již v rámci své implementace nestihl.

Namísto toho jsem zvolil možnost vygenerování vertex a index bufferu na jiném zařízení. Tato data jsem následně uložil do souborů, které aplikace v Hololens načítá a tvoří pomocí ní objekty.

V rámci svého řešení jsem vytvořil složku na zařízení, která je dedikovaná pro objekty, které chceme v zařízení zobrazovat. Uživatel si tak může libovolně přidávat parsované objekty do složky bez nutnosti rekompilace kódu.

6.10 Limitace aplikace v reálném použití

V této sekci bych chtěl zmínit problémy, které vznikly v souvislosti s implementací aplikací.

Na brýlích Hololens se kvůli omezení operačního systému nedají spouštět aplikace, které se otevírají delší, než maximální stanovenou dobu. Vzhledem k neefektivnosti mého řešení v rámci načítání objektů, může jejich načítání přesahovat maximální povolenou dobu, což vyustí v nespouštějící se aplikaci. Při testování tohoto problému jsem došel k závěru, že aplikace dokáže načíst maximálně objekty, které mají dohromady méně než 35 000 vertexů.

V rámci současného řešení jsem se ještě rozhodnul otestovat maximální počet vrcholů, které je schopna aplikace v současném stavu vykreslit, bez většího počtu ztracených snímků za vteřinu. Výslednou hodnotou byl počet zhruba 55 000 vrcholů. Spíše bych ale doporučil držet orientačně 40 000 vrcholů, u kterých je aplikace přece jenom plynulejší.

Kapitola 7

Závěr

Cílem práce bylo navrhnout a odzkoušet vývoj aplikací pro brýle Hololens 2 a to jak s využitím externích knihoven a jazyka C++, tak v prostředí herního enginu, jako je Unreal Engine 4.

V rámci bakalářské práce jsem vytvořil ukázkové aplikace, které prakticky demonstrují základní možnosti rozšířené reality, které brýle Hololens 2 nabízí. Při vývoji těchto aplikací jsem se zaměřil hlavně na úlohy demonstrující zobrazování textu a modelů. U těchto úloh jsem v rámci praktické části navrhnul možná řešení, popsal jejich výhody a nevýhody a popřípadě implementoval.

Tyto aplikace bychom mohli využít například v rámci textového popisu exponátů v muzeu nebo v oblasti medicíny pro vykreslování holografických objektů orgánů.

Hlavním problémem současné aplikace je ukládání prostoru, u kterého jsem nezvolil nejideálnější řešení, což se na výsledné aplikaci odráží.

Práce na této bakalářské práci pro mne byla velmi přínosná. Nemusel jsem totiž pochopit pouze princip fungování rozšířené reality, ale také se naučit pracovat s velikým množštvím knihoven a frameworků, se kterými jsem nikdy dříve nepracoval.

V práci bych rád pokračoval s zaměřil se pak hlavně na řešení problému inicializace prostoru, který je v tuto chvíli nejslabším článkem aplikace.

Implementační část bakalářské práci jsem si maximálně užil. Teorie rozšířené reality spojené s tvorbou enginu byl pro mne jedinečným zážitkem a jsem velice vděčný za možnost pracovat se zařízením Microsoft Hololens, a pracovat tak na něčem, co mne zajímá.

Literatura

1. ROSENBERG, Louis B. *The Use of Virtual Fixtures as Perceptual Overlays to Enhance Operator Performance in Remote Environments.* [B.r.]. Dostupné také z: https://apps.dtic.mil/docs/citations/ADA292450.
2. POETKER, Bridget. *A Brief History of Augmented Reality Future Trends Impact.* [B.r.]. Dostupné také z: https://learn.g2.com/history-of-augmented-reality.
3. YOON, Clara. *Assumptions that led to the failure of Google Glass.* NYC Design, 2018-08. Dostupné také z: https://medium.com/nyc-design/the-assumptions-that-led-to-failures-of-google-glass-8b40a07cfa1e.
4. HOUSKA, Filip. *S novou aplikaci IKEA Place si muzete diky rozsirene realite zkusit umisteni nabytku v domacnosti.* 2017-09. Dostupné také z: https://www.czechcrunch.cz/2017/09/s-novou-aplikaci-ikea-place-si-muzete-diky-rozsirene-realite-zkusit-umisteni-nabytku-v-domacnosti/.
5. TWENTYMAN, Jessica. *AR enabled industrial wearables the next growth market.* 2018-03. Dostupné také z: https://internetofbusiness.com/ar-enabled-industrial-wearables-the-next-growth-market/.
6. *See new homes in augmented reality.* [B.r.]. Dostupné také z: https://www.homear.io/.
7. SOFTWARE, BeLight. *Live Home 3D.* [B.r.]. Dostupné také z: https://www.livehome3d.com/.
8. CASEYMEEKHOF. *Direct manipulation with hands - Mixed Reality.* [B.r.]. Dostupné také z: https://docs.microsoft.com/en-us/windows/mixed-reality/design/direct-manipulation.
9. *HoloLens 2 vs HoloLens 1 whats new 4Experiences AR VR Blog.* 2020-01. Dostupné také z: https://4experience.co/hololens-2-vs-hololens-1-whats-new/.
10. *Microsoft HoloLens 2 review mixed reality smartglasses holographic.* [B.r.]. Dostupné také z: https://www.aniwaa.com/product/vr-ar/microsoft-hololens-2/.
11. KHRONOSGROUP. *KhronosGroup OpenXR SDK.* [B.r.]. Dostupné také z: https://github.com/KhronosGroup/OpenXR-SDK.

12. THETUVIX. *OpenXR Mixed Reality*. [B.r.]. Dostupné také z: https://docs.microsoft.com/en-us/windows/mixed-reality/develop/native/openxr.

13. MICROSOFT. *microsoft/MixedRealityToolkit-Unity*. [B.r.]. Dostupné také z: https://github.com/microsoft/MixedRealityToolkit-Unity.

14. [B.r.]. Dostupné také z: https://developer.vuforia.com/.

15. QUINNRADICH. *What is a Universal Windows Platform*. [B.r.]. Dostupné také z: https://docs.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide.

16. GRANTMESTRENGTH. *DirectX graphics and gaming Win32 apps*. [B.r.]. Dostupné také z: https://docs.microsoft.com/en-us/windows/win32/directx.

17. *Coordinate System*. [B.r.]. Dostupné také z: https://mathworld.wolfram.com/CoordinateSystem.html.

18. STEVEWHIMS. *Reference for HLSL Win32 apps*. [B.r.]. Dostupné také z: https://docs.microsoft.com/en-us/windows/win32/direct3dhlsl/dx-graphics-hlsl-reference.

19. *Nature paintings exhibitions at PNG Sons*. 2018-06. Dostupné také z: https://pngadgilandsons.com/nature-paintings-exhibitions-at-png-sons/.