FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Možnosti použití Pixel Streamingu pro streamování her ve virtuální realitě
Student:	Pavel Plotnikau
Vedoucí:	Ing. Petr Pauš, Ph.D.
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

Do Unreal Enginu byla přidána podpora pro pixel streaming. Cílem této práce bude zjistit, zda je možné použít Pixel Streaming pro streamování her na zařízení virtuální reality (VR).

1) Analyzujte možnosti pixel streamingu v Unreal Enginu.
2) Analyzujte možnosti pro využití streamingu v aplikacích pro VR.
3) Navrhněte prototyp game clienta a web clienta, který bude streaming využívat.
4) Prototyp implementujte.
5) Prototyp otestujte.

Elektronicky schválil/a Ing. Radek Richtr, Ph.D. dne 12. ledna 2022 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Možnosti použití Pixel Streamingu pro streamování her ve virtuální realitě

Pavel Plotnikau

Webové a softwarové inženýrství
Vedoucí práce: Ing. Petr Pauš, Ph.D.

9. května 2022

Poděkování

Rád bych poděkoval svému šéfovi za svolení k použití daného tématu pro tuto práci, svému vedoucímu za pomoc při psaní této práce a své rodině a přátelům za morální podporu.

Prohlášení

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. Dále prohlašuji, že jsem s Českým vysokým učením technickým v Praze uzavřel dohodu, na jejímž základě se ČVUT vzdalo práva na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona. Tato skutečnost nemá vliv na ustanovení § 47b zákona č. 111/1998 Sb., o vysokých školách, ve znění pozdějších předpisů.

V Praze dne 9. května 2022 . . . . . .

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Plotnikau, Pavel. *Možnosti použití Pixel Streamingu pro streamování her ve virtuální realitě*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

Abstrakt

Tato bakalářská práce se zabývá využitím pluginu Pixel Streaming herního enginu Unreal Engine 4 v oblasti VR. Jeho hlavním cílem je zjistit, zda je možné pomocí tohoto pluginu streamovat VR hry přes internet ze vzdáleného zařízení do brýlí virtuální reality, jako jsou Oculus Quest a Oculus Quest 2. Během práce byly napsány dva plaginy pro Unreal Engine 4 a rozšířena funkionalita webové stránky, kterou plugin poskytuje. Výsledkem je funkční projekt, který umožní streamování hry z počítače do brýlí v rámci stejné lokální sítě.

Klíčová slova UE4, Unreal Engine 4, Pixel Streaming, plugin, VR, Virtuální Realita.

Abstract

This bachelor work focuses on the use of the Pixel Streaming plugin of the Unreal Engine 4 game engine in VR. Its main goal is to see if this plugin can be used to stream VR games over the internet from a remote device to virtual reality glasses such as the Oculus Quest and Oculus Quest 2. While working, I had to write two plugins for Unreal Engine 4 and extend the functionality of the website that the plugin provides. The result is a working project that allows streaming of the game from the computer to the glasses within the same local network.

Keywords UE4, Unreal Engine 4, Pixel Streaming, plugin, VR, Virtual Reality.

Obsah

Seznam obrázků

Úvod

V srpnu 2021 byla oficiálně vydána verze 27 herního enginu Unreal Engine 4. Kromě obrovského množství změn týkajících se tvorby filmů i vylepšení grafiky, konečně vyšel z beta testování plugin „Pixel Streaming“. Na základě popisu, tento plugin dovoluje poskytování přístupu k aplikaci běžící na výkonném serveru odkudkoli na světě. Jediné, co je třeba mít, je dobré internetové připojení a obyčejný web browser.

Po přečtení dokumentace jsem nabyl dojmu, že podle názoru vývojářů tato technologie se musí používat pro prezentace. Ale co se týče cloud gamingu? Zaujalo mě především možnost použití této technologie v zařízeních pro virtuální realitu jako např. Oculus Quest a Oculus Quest 2. Nicméně po dlouhém hledání informací na internetu jsem skoro nenašel žádnou informaci o případech použití pluginu v této oblasti.

Tato práce se bude věnovat výzkumu možnosti použití Pixel Streamingu pro vzdálenou interakci s VR hrami. Výstupy této práce nejsou určeny pro širokou veřejnost, ale spíše pro vývojáře cloud servisu a her v Unrealu.

Kapitola 1

Cíl práce

Hlavním cílem této práce bude zjistit, zda je možné použít Pixel Streaming pro streamování VR her na zařízeni virtuální reality. Zaměřím se na zařízení Oculus Quest 2.

Především bude provedena analýza na základě oficiální dokumentace a vytvořené testovací aplikace s použitím pluginu a předpřipravené šablony Unrealu.

Dalším cílem je návrh řešení za předpokladu, že neexistuje hotové. Unreal Engine je open source, což znamená, že si možně prohlédnout celý zdrojový kód včetně pluginů, které budu potřebovat. Cílem práce je zjistit možnosti originálního pluginu, což znamená, že můžu přidat funkcionalitu, ale nemůžu změnit již hotové řešení.

Na základě návrhu bude implementován prototyp aplikace. Tento prototyp bude testován a na základě výsledku testování budou shrnuté výsledky výzkumu.

Kapitola 2

Analýza Pixel Streamingu v Unreal Enginu

2.1 Použité technologie

Pixel Streaming návřen na základě WebRTC API, poskytujícího podporu streamování dat na základě technologie *peer-to-peer*.

2.1.1 WebRTC

Toto API je určeno především pro realizaci video chatů ve webovém prohlížeči. Open-source projekt WebRTC byl oficiálně zveřejněn firmou Google v květnu roku 2011. Princip jeho fungování je následující [8]:

1. Uživatel otevírá stránku obsahující WebRTC kontent.
2. Browser žádá uživatele o přístup k web-kameře a mikrofonu, pokud to potřebuje. Nezačne fungovat, dokud to uživatel nepovolí.
3. V browseru–iniciátoru se vytváří SDP-balík. V podstatě to je textový soubor obsahující všechny potřebné informace o parametrech spojení (co se předává, jaké kodeky se používají apod).
4. Dále, v závislosti na realizaci, iniciátor přeposílá tento balík ostatním uživatelům.
5. Browser, který tento balík dostane, vygeneruje podobný na základě prvního. Druhý balík se posílá zpět iniciátorovi. Od této doby uživatelé mají potřebné minimum informací o ostatních uživatelích.
6. V závislosti na realizaci, paralelně s předchozími kroky prochází analýza síťového připojení. Klientům je předána adresa serveru STUN. Tento server se používá k získání externích IP adres, pokud jsou klienti odděleni NAT a/nebo firewallem. Ve složitějších případech (např. při použití duálního NAT) se používají servery

TURN. Jsou to v podstatě zprostředkovatelé, kteří mění spojení *klient-klient* (P2P) na spojení *klient-server-klient*. Používají se v případech, kdy nelze navázat přímé spojení, např. při malé šířce pásma některé části sítě.

7. Pokud všechny kroky proběhnou úspěšně, je spojení navázáno. Periodicky se volá událost OnIceCandidate, která přenáší informace o IP adresách, nastavení NAT a pokusech o připojení mezi klienty.

Technologie WebRTC má řadu výhod [8]:

- Není nutná instalace softwaru. Tato technologie je podporována většinou prohlížečů.
- Vysoká kvalita spojení.
- Vysoká úroveň zabezpečení.
- Na základě HTML5 a JavaScriptu lze implementovat libovolné ovládací rozhraní.
- Open source projekt.
- Cross-platform: Stejná aplikace WebRTC poběží stejně dobře na jakémkoli operačním systému.

Kromě toho existuje i několik nevýhod [8]:

- Všechna řešení WebRTC jsou vzájemně nekompatibilní, protože standard popisuje pouze způsob přenosu videa a zvuku. Implementace ostatních funkcí je zcela závislá na programátorech. Jinými slovy, nebude možné volat z jedné aplikace WebRTC do druhé.
- WebRTC pro svůj provoz určuje reálné IP adresy uživatelů.

WebRTC má k disposici dva audio kodeky (G.711 a Opus) a dva video kodeky (VP8 a H.264).

2.1.2 SDP

SDP – síťový protokol, určený k popisu a vyjednávání informací o relaci mezi několika klienty. Ve většině případů je princip použití takový, že si klienti vyměňují SDP-soubory, aby se dohodli na formátu vyměňovaných dat.

Formát SDP-souborů je následující [9]:

```
v=  (verze protokolu, v tuto chvíli existuje jenom jedna verze,
    takže vždy 0)
o=  (identifikátory organizátoru a relace)
s=  (jméno relace)
i=* (informace relace)
u=* (URI-adresa s dodatečným popisem relace)
```

```
e=* (E-mailová adresa zodpovědné osoby)
p=* (telefonní číslo zodpovědné osoby)
c=* (informace o spojení | je vyžadována pouze v případě, že
    není zaznamenána ve všech médiích)
b=* (0 či více řádků informací o šířce pásma)

Tady potřeba uložit 1 či víc řádků definujících čas)
t=  (trvání relace)
r=* (počet pokusů o opakování. Musí se rovnat 0 nebo víc.)

z=* (nastavení časové zóny)
k=* (šifrovací klíč)
a=* (jeden nebo více řádků s popisem atributů)

m=  (typ média a transportní adresa zařízení)
i=* (mediální tituly)
c=* (informace o spojení | není potřebná pokud už zapsána do
    hlavičky)
b=* (informace o šířce pásma, kterou zabírá komunikační kanál)
k=* (šifrovací klíč)
a=* (0 či více řádků informací o atributy média dat)
```

Během vytváření spojení odešle webový klient podobný soubor hernímu klientovi. Tento soubor můžeme rozdělit na tři části:

- hlavička, obsahující základní informaci o relaci,
- první odstavec, který obsahuje informaci o nastavení video kodeku,
- druhý odstavec, popisující nastavení audiokodeku.

Vzhledem k tomu, že v této bakalářské práci se zabývám především vizuální částí pluginu, nebudu se zabývat základním nastavením a nastavením audio kodeku, ale pouze nastavením video kodeku.

V nastavení video kodeku mě nejvíc zajímají dva konkrétní řádky, na které je třeba si dát pozor:

```
a=rtpmap:100 H264/90000
...
a=fmtp:100 level-asymmetry-allowed=1;packetization-mode=1;
profile-level-id=42e01f
```

Z prvního řádku je zřejmé, že k navázání spojení se používá kodek H.264. Další důležitou informaci lze nalézt na konci druhého řádku. *Profile-level-id* – kódované informace o základních nastaveních kodeku. Ale k tomu se dostaneme o něco později.

2.1.3 H.264

H.264 je dnes jedním z nejpoužívanějších open-source kodeků. Tento video kodek je licencovaný standard komprese videa určený k dosažení vysokého stupně komprese video při zachování vysoké kvality. Finální podoba byla vytvořena v roce 2003. Formát byl vyvinut ve spolupráci dvou skupin – ITU-T VCEG (International Telecommunications Union – Video Coding Experts Group) a ISO/IEC JTC1 MPEG (International Organization for Standardization/International Electrotechnical Commission – Moving Picture Experts Group)[10].

Standard definuje sady funkcí, tzv. *profily*, zaměřené na konkrétní třídy aplikací. Kód profilu a uvedená omezení použité v enkodéru umožňují dekodéru rozpoznat požadavky na dekódování konkrétního datového toku. Uvedu jen několik z nich [11]:

- **Baseline Profile** – Nabízí všechny funkce CBP profilu a odolnost proti ztrátě dat. Také umí zpracovávat CBP bitový proud, protože oba mají stejnou hodnotu identifikátoru.
- **Extended Profile** – Je určen pro streamování videa, má poměrně vysokou míru komprese a další funkce pro zvýšení odolnosti proti ztrátě dat.
- **Main Profile** – Používá se pro digitální televizi se standardním rozlišením ve vysílání s kompresí MPEG-4 v souladu se standardem DVB.
- **High Profile** – Je základem pro digitální vysílání a video na optických médiích, zejména pro televizi s vysokým rozlišením. Používá se pro video disky Blu-ray a vysílání DVB HDTV.
- A další...

Kromě profilů H.264 také existuje řada *úrovní*. *Úroveň* definuje stupeň požadovaného výkonu dekodéru pro daný profil. Přesněji řečeno definuje profilu, s jakým rozlišením, bitratem a jakou maximální velikosti makrobloku bude pracovat. Taky definuje maximální rychlost dekódování streamu. Zpět ke kódu nastavení kodeku, který jsem zmínil dříve.

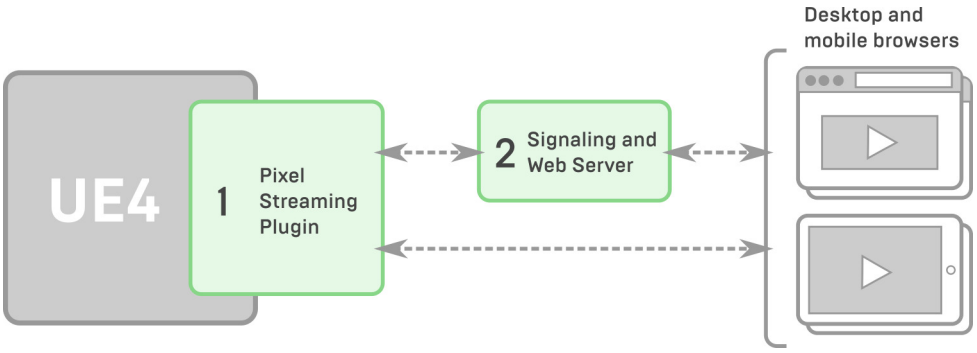
```
profile-level-id=42e01f
```

Informace o nastavení kodeku je zakódována ve třech bitech. První z nich obsahuje id profilu používaného kodekem a poslední – jeho úroveň. **0x42 = 66**, což je id *Baseline* profilu. **0x1f = 31** z čehož vyplývá, že použitá úroveň je *3.1*[12].

2.2 Archtektura

Plugin je založen na použití knihovny WebRTC. Z tohoto důvodu je základní princip podobný. Tato knihovna však neimplementuje způsob přenosu dat mezi klienty. Proto vývojáři pluginu vyvinuli vlastní metodu založenou na dvou hlavních částech: samotný plugin implementovaný na straně herního klienta a signální server.

2.2.1 Vytvoření spojení



Obrázek 2.1: Základní architektura pluginu[1]

Signaling and Web Server – poskytuje prohlížečům prostředí HTML a JavaScript, které přehrává mediální stream a taky zodpovědný za vyjednávání spojení mezi prohlížečem a pluginem ze strany klienta[1].

Pixel Streaming Plugin – kóduje vykreslený obrázek, balí ho spolu s audio do media streamu a posílá do jednoho nebo víc připojených prohlížečů přes přímé *peer-to-peer* připojení[1].

Postup spuštění je následující [1]:

1. Ihned po spuštění se herní klient i webový klient pokusí připojit k signálnímu serveru.
2. Jakmile je navázáno spojení, server odešle webovému klientovi soubory HTML a JavaScript, aby se stránka otevřela v prohlížeči.
3. Jakmile uživatel zahájí streamování, webový klient vygeneruje SDP-paket a odešle jej na server. Server pak předá paket požadovanému hernímu klientovi. Nakonec klient, který už zná adresu odesílatele, vygeneruje odpověď a odešle ji přímo webovému klientovi.
4. Jakmile je navázáno připojení, začne Pixel Streaming plugin komprimovat video a zvuk a přenášet je prostřednictvím media streamu přímo do prohlížeče.
5. Signalizační server mezitím udržuje spojení s oběma klienty a v případě potřeby může spojení ukončit nebo sledovat ukončení relace ze strany webového klienta.

Ještě před začátkem relace obdrží web klient všechny potřebné soubory k vytvoření stránky. Standardně je k dispozici následující sada pro mě důležitých souborů:

- *player.html* a *player.css* – spolu tvoří vzhled standardní stránky pluginu,
- *app.js* – obsahuje všechny funkce zodpovědné za provoz web stránky a interakci s uživatelem,

- *webRtcPlayer.js* – je zodpovědný za spojení mezi webovou stránkou a herním klientem. Implementuje funkce přenosu / příjmu dat prostřednictvím data kanálu a také přijímání mediálního streamu.

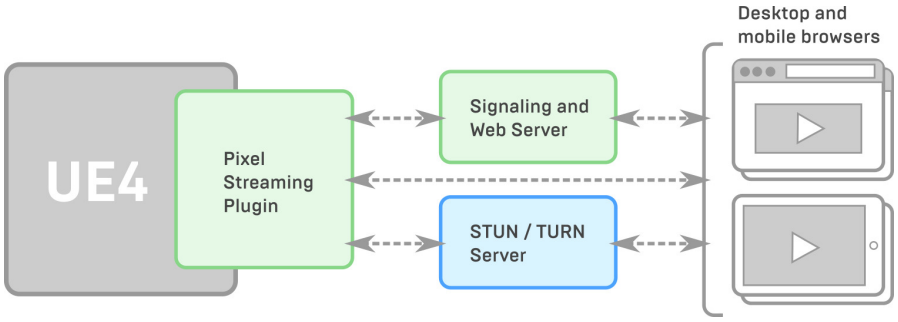
2.2.2 Komunikace mezi klienty

Po vytvoření spojení herní klient začíná přenášet media stream do web klienta. To však nestačí. Web klient musí nějakým způsobem komunikovat s hrou. Z těchto důvodů se při vytváření spojení mezi dvěma klienty kromě media streamu vytváří další stream pro výměnu dat – *data kanál*. Prostřednictvím tohoto kanálu webový klient přenáší vstupní data a příkazy do herního klienta. Plugin rozpozná typ příkazů pomocí id-hodnot, zaznamenaných v souboru *app.js*.

2.2.3 Další způsoby použití pluginu

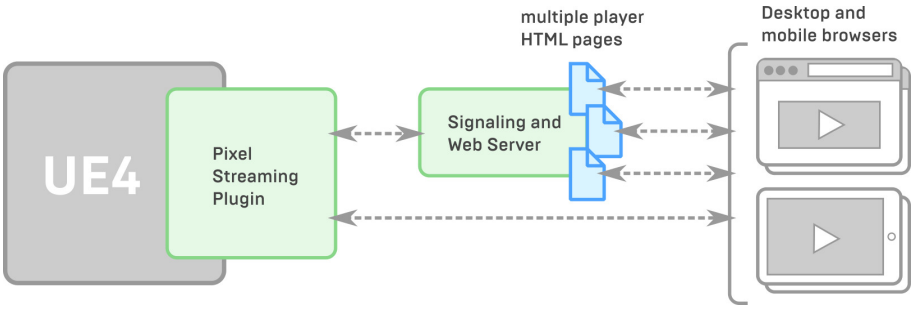
Existuje několik dalších způsobů použití tohoto pluginu, které jsou popsány na oficiálních webových stránkách a které považuji za vhodné zmínit. Protože však takové příklady nebudou v této práci použity, nebudu se jimi podrobně zabývat.

- Výše uvedený způsob vytvoření spojení mezi klienty funguje pouze v případě, že jsou připojeny ke stejné lokální síti. Důvodem je, že klienti potřebují znát navzájem své IP adresy. Pokud jde o globální síť, jsou klienti odděleni firewally a službami NAT, které mění jejich IP adresy. V takových případech se používají servery STUN/TURN.



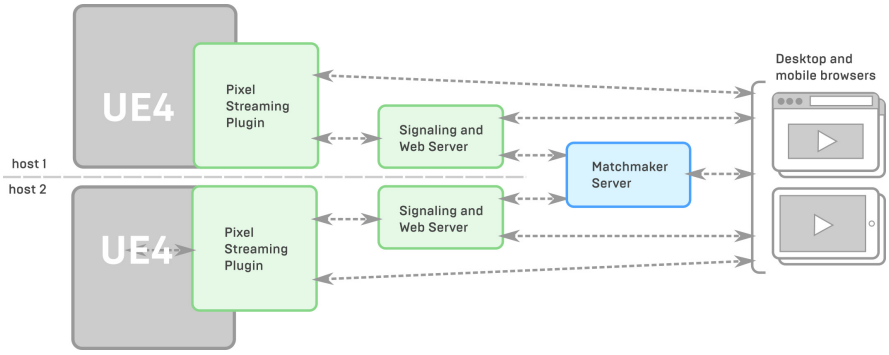
Obrázek 2.2: Architektura využívající servery STUN/TURN[2]

- V závislosti na zadané adrese URL je možné v rámci jedné relace předávat různým uživatelům různé stránky HTML s různou logikou. To může být užitečné, když chce jeden hlavní uživatel provést prezentaci, ale nechce předat ovládání aplikace ostatním uživatelům. Tato metoda také poskytuje možnost rozdělit ovládání aplikace mezi více uživateli (např. párové řízení letadla).



Obrázek 2.3: Architektura zaměřená na více hráčů používajících jednu aplikaci[2]

- Pokud potřebujete každému uživateli poskytnout samostatnou instanci herního klienta, může vám pomoci Matchmaker server. Tento server bude přijímat požadavky klientů na načtení stránky a přesměruje je na příslušný signální server. Matchmaker bude také monitorovat aktivitu již vytvořených relací a zabrání pokusům o připojení k již používaným signálním serverům.



Obrázek 2.4: Architektura použití několika herních klientů a signalizačních serverů[2]

2.3 Jak plugin spustit

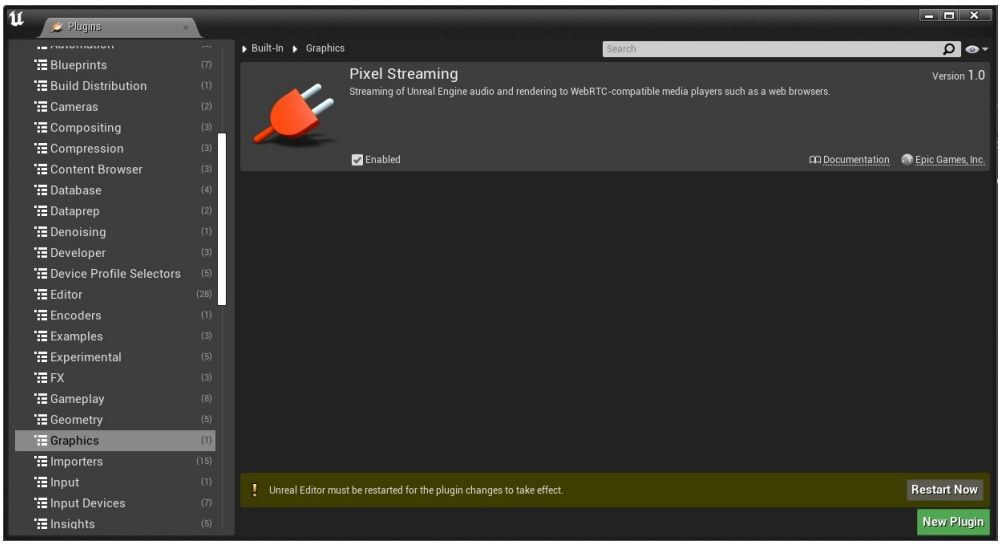
Před spuštěním pluginu je potřeba zařízení připravit [13]:

- Je třeba zkontrolovat operační systém a hardware, protože plugin je schopen přenášet media stream pouze z operačního systému Windows a pouze při určitém nastavení GPU.
- Server vyžaduje nainstalovaný *node.js*. Pokud tedy nemáte v počítači nainstalovaný *node.js*, stáhněte si ho a nainstalujte.
- Je třeba zkontrolovat stav portů, které má zásuvný modul používat. Standardně jsou to porty 80 a 8888.

- Potřebujete znát IP adresy zařízení, na kterých poběží klíčové prvky pluginu, zejména IP adresu zařízení, na kterém běží signální server.
- Vývojáři pluginu také důrazně doporučují zastavit ostatní webové služby, když je plugin používán.

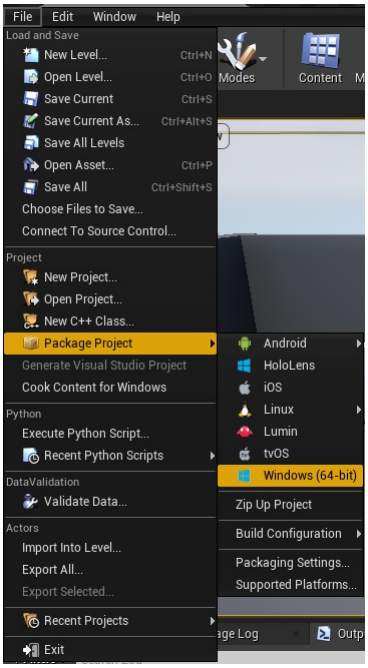
Pokud byly splněny všechny výše uvedené podmínky, lze se začít připravovat na spuštění pluginu. Zde je třeba upřesnit, že tyto pokyny vyžadují, aby klienti i server byli spuštěni na stejném zařízení. Nejprve je třeba si připravit herní klient [13]:

1. Nejprve je nutné vytvořit projekt v Unrealu.
2. Ihned po vytvoření projektu je potřeba přejít do složky pluginů a zapnout Pixel Streaming plugin. Restartujte projekt, aby se změny použily.



Obrázek 2.5: Okno pluginů v Unreal Engine 4

3. Dalším krokem je sestavení projektu pro systém Windows.

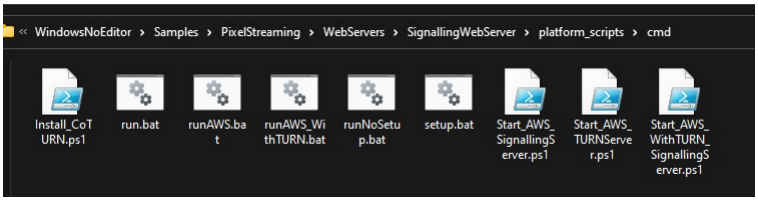


Obrázek 2.6: Umístění tlačítka pro sestavení projektu

4. V sestaveném projektu je potřeba vytvořit odkaz na soubor .exe. Dále je nutné otevřít okno „Properties“ tohoto odkazu a přidat **-PixelStreamingIP=localhost -AudioMixer -PixelStreamingPort=8888** do řádku „Target“.

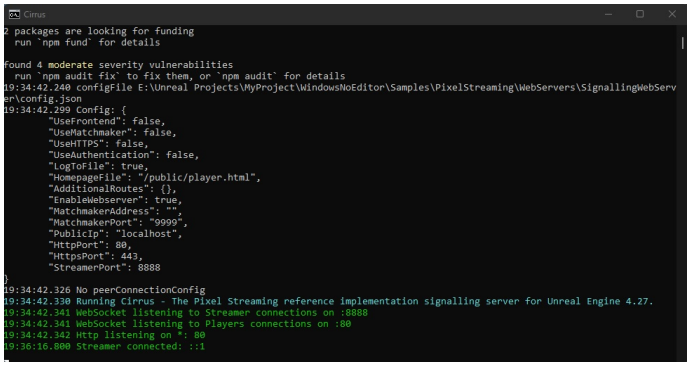
Obrázek 2.7: Jak by měla vypadat složka sestaveného projektu

5. Dále je třeba spustit server. Nejprve je třeba jej nakonfigurovat spuštěním souboru *setup.bat*. Poté jej lze spustit pomocí souboru *run.bat*.



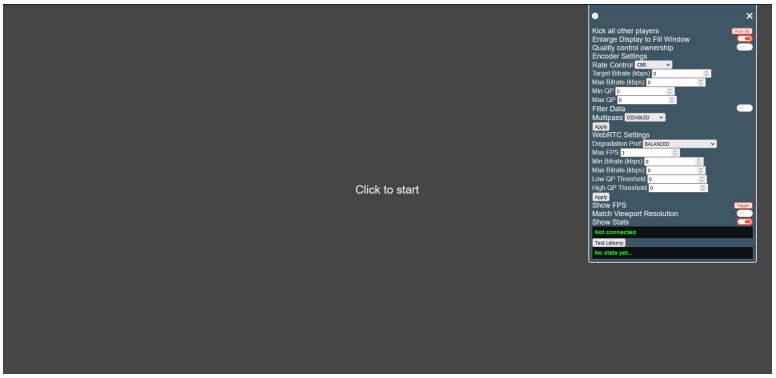
Obrázek 2.8: Složka se soubory pro nastavení a spuštění signálního serveru

Pokud je vše provedeno správně a herní klient i signální server jsou spuštěny, měl by se klient okamžitě připojit k serveru.



Obrázek 2.9: Jak by měl vypadat úspěšně spuštěný server

6. Posledním krokem je spuštění web klienta. To lze provést zadáním IP adresy serveru do vyhledávacího řádku prohlížeče. Protože jsou všechny aplikace spouštěny lokálně, je IP adresa 127.0.0.1.



Obrázek 2.10: Jak vypadá standardní webový klient

2.4 Srovnání s jinými službami

Dnes je k dispozici poměrně dost cloudových herních služeb. Jelikož je však účel mé práce zcela specifický, nepovažuji za správné plugin porovnávat s každým z nich. Ze všech dostupných služeb bych proto rád vyzdvihl dvě, které jsou tématu této práce nejbližší: **Virtual Desktop** a **Oculus Air Link**.

2.4.1 Virtual Desktop

Jedná se o aplikaci, která umožňuje přenést plochu počítače do trojrozměrného prostoru virtuální reality a interagovat s ní. Kromě toho umožňuje také spustit VR hry nainstalované v počítači. Virtual Desktop je primárně určen pro samostatné brýle virtuální reality, jako jsou Oculus Quest a Oculus Quest 2. K fungování aplikace nepotřebujete kabel. Aby to fungovalo, je třeba nainstalovat počítačovou verzi aplikace i verzi pro brýle a obě zařízení připojit ke stejné lokální síti.

Na základě dostupných nastavení a testů používá Virtual Desktop jako hlavní kodek H.264. S datovým tokem kolem 130 Mbps je aplikace schopna poskytovat kvalitní obraz se snímkovou frekvencí 90 fps a latencí kolem 40 ms, což je pro hraní her skutečně velmi příjemné[14]. Aplikace také podporuje použití kodeku H.265. Díky tomuto kodeku může Virtual Desktop podporovat vyšší kvalitu obrazu při mírně nižším bitratu. To však zvyšuje latenci.

2.4.2 Oculus Air Link

V současné době se jedná o rozšíření pro Oculus Link, které v podstatě plní stejné funkce jako Virtual Desktop. Oculus Link slouží k interakci s pracovní plochou v prostoru virtuální reality. Tato aplikace, stejně jako její rozšíření Air Link, se původně instaluje na všechny brýle řady Quest a Quest 2.

Toto rozšíření je v současné době ve fázi beta testování, takže nemá skoro žádné nastavení jako Virtual Desktop. Nicméně, i když je ve fázi beta testování, je schopen dosahovat skoro stejných výsledků jako Virtual Desktop, je zdarma a je od počátku integrován do operačních systémů Quest a Quest 2[14].

Kapitola 3

Nárvh VR rozšíření pro Pixel Streaming

Nejprve je třeba identifikovat problémy, které je třeba vyřešit. A největším problémem je, že plugin je zcela nevhodný pro zařízení VR. V oficiální dokumentaci se nikde neuvádí, že je lze nakonfigurovat pro VR zařízení. Na oficiálním fóru Unreal Engine o tom také nejsou žádné informace. A přestože lze najít několik příspěvků s podobnými otázkami, nejsou o tom téměř žádné informace.

Je tedy možné spustit web klienta na brýlích pro virtuální realitu? Ano, ale vypada to, jako by ve virtuálním prostoru byla počítačová obrazovka, na které se zobrazuje hra. Normální hry pro virtuální realitu naopak vykreslují scénu přímo na displeje brýlí. Tak se objevuje první problém. Pixel Streaming odesílá vykreslený obraz scény do prohlížeče uvnitř brýlí. Musím je nějakým způsobem odeslat přímo na displeje.

Po vyřešení prvního problému by se měl objevit druhý: obraz nepůsobí trojrozměrně. Důvodem je to, že výsledný obraz je monoskopický. Musel by však být stereoskopický, aby vznikl pocit objemu. Rozdíl mezi monoskopickým a stereoskopickým obrazem a to, jak fungují vysvětlím později.

Posledním problémem je, že herní klient nerozpoznává pohyby hlavy a ovladačů, a taky vstupní data z ovladačů. Tento problém je také způsoben tím, že plugin nebyl původně návržen pro práci se zařízeními VR.

Byly tak zjištěny tři hlavní problémy:

- nutnost posílat obraz přímo na displeje,
- absence stereoskopie,
- nutnost posílat vstupní data.

3.1 Posílání obrazu na brýlové displeje

Jak je tedy možné poslat obraz, který přišel do prohlížeče, přímo na displeje brýlí? Nejdřív mě napadlo podívat se do Oculus API, ale nenašel jsem žádné potřebné informace

o prohlížeči a displejích. Při hledání dalších řešení na internetu jsem narazil na hru Moon Rider.

Moon Rider – bezplatná VR hra pro prohlížeče, podobná jiné známé hře Beat Saber. To však není důležité. Důležité je, že tuto hru lze spustit v prohlížeči Oculus Quest 2 a přepnout ji do celoobrazovkového režimu. To je možné, protože aplikace byla napsána na základě rozhraní WebVR.

3.1.1 WebXR

WebVR – rozhraní pro vytváření VR aplikací pro prohlížeč. První verze WebVR byla zveřejněna týmem Mozilla VR a týmem Google Chrome v březnu roku 2016[15].

Protože toto rozhraní bylo zaměřeno na VR aplikace a neobsahovalo funkcionalitu pro vytváření AR aplikací, bylo následně v roce 2018 nahrazeno standardem WebXR. Tento standard zase umožňuje vytvářet aplikace pro VR i AR[16].

Životní cyklus aplikací založených na WebXR má obvykle tyto kroky [17]:

1. Nejprve zkontroluje, zda zařízení a prohlížeč podporují provozní režimy XR i samotný WebXR.

2. Dále, když se uživatel pokusí povolit WebXR, aplikace odešle požadavek na rozhraní XRSession, aby jej zapnula v režimu zadaném uživatelem/aplikací.

   XRSession je rozhraní, které reprezentuje aktuální relaci v režimu zvoleném uživatelem (VR, AR, ...) a umožňuje s ní pracovat.

3. Po přijetí požadavku aplikace vrátí objekt XRSession, prostřednictvím kterého pak může aplikace řídit procesy aktuální relace, jako je zpracování vstupů, animace, vykreslování atd.

   Proces vykreslování probíhá v cyklu, v němž aplikace každý snímek odesílá požadavek WebXR na vykreslení. WebXR zase přijme požadavek, vykreslí další snímek pomocí WebGL nebo jiné grafické knihovny a odešle výsledek na obrazovku.

4. Na konci, když chce uživatel ukončit aplikaci, odešle požadavek na ukončení relace.

Protože WebXR je pouze rozhraním a není zaměřeno na renderování nebo zpracování 3D dat, obvykle úzce spolupracuje s grafickými knihovnami, jako je WebGL. Existují však samozřejmě mnohem lehčí knihovny, které implementují WebXR. Jednou z nich je poměrně populární knihovna Three.js.

3.1.2 WebGL a Three.js

WebGL – multiplatformní knihovna pro práci s 3D a 2D grafikou v prohlížeči. Oficiálně vydána v roce 2011 neziskovou organizací Khronos Group. První verze knihovny byla založena na jiné multiplatformní grafické knihovně OpenGL ES 2.0[18].

Three.js je naopak lehčí multiplatformní grafická knihovna, která využívá funkce knihovny WebGL. První alfa verze se objevila na github.com 3. března v roce 2013[19].

Jak název napovídá, jako programovací jazyk se používá JavaScript. Kód vytvořený na základě Three.js lze použít jako součást webové stránky ve formě skriptu a díky WebGL aplikace nevyžaduje použití externích rozšíření pro prohlížeč. Ve své práci budu tuto knihovnu používat.

3.1.3 Způsob použití Three.js

Tato knihovna bohužel poskytuje poměrně základní sadu funkcí pro práci s 3D grafikou. Tím chci říct, že nemá funkci pro přímou práci s displeji, ale automaticky na ně odesílá vyrenderovaný obraz. Navíc se video na webu přehrává prostřednictvím kontejneru videa HTML. Nicméně lze zkusit přenést video proud pomocí textury.

Myšlenka je jednoduchá: před kameru nasadit rovinu a použít na ní materiál s texturou. Dobrou zprávou je, že textury v Three.js umožňují streamovat video, a to prostřednictvím kontejneru videa.

3.2 Absence stereoskopie

3.2.1 Existující typy zobrazení 3D scény

V této části je třeba lépe pochopit, co je to stereoskopie. V současné době existují čtyři typy zobrazení scény, které se běžně používají v počítačové grafice:

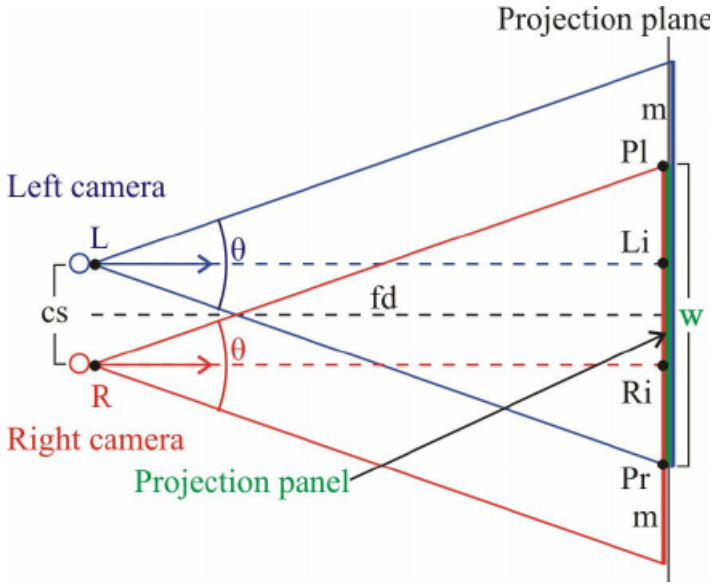
- monoskopické zobrazení,
- stereoskopické zobrazení,
- hybridní monoskopie,
- obrázek 360°.

3.2.1.1 Monoskopické zobrazení

Monoskopie je nejčastěji používanou metodou zobrazování 3D scén v počítačových hrách i mimo ně. Jeho podstatou je, že pozorovatel vidí scénu pouze z jednoho bodu v prostoru. Tady není moc co vysvětlovat, takže přejděme k další části.

3.2.1.2 Stereoskopické zobrazení

Stereoskopie je ta samá metoda zobrazení prostoru, která vytváří iluzi objemu ve hrách pro virtuální realitu. Je v podstatě napodobeninou lidského vidění. Pomocí dvou kamer umístěných v určité vzdálenosti od sebe lze získat dva obrazy, z nichž lze vytvořit jeden stereoskopický obraz, i když samotné obrazy byly původně monoskopické.



Obrázek 3.1: Princip fungování stereoskopie[3]

Obrázek 3.2: Příklad stereoskopického snímku[4]

Při pohledu na obrázek výše je vidět, že jeden z nich je vůči druhému trochu posunutý. Stejný efekt se projeví, pokud začít rychle mrkat oběma očima za sebou.

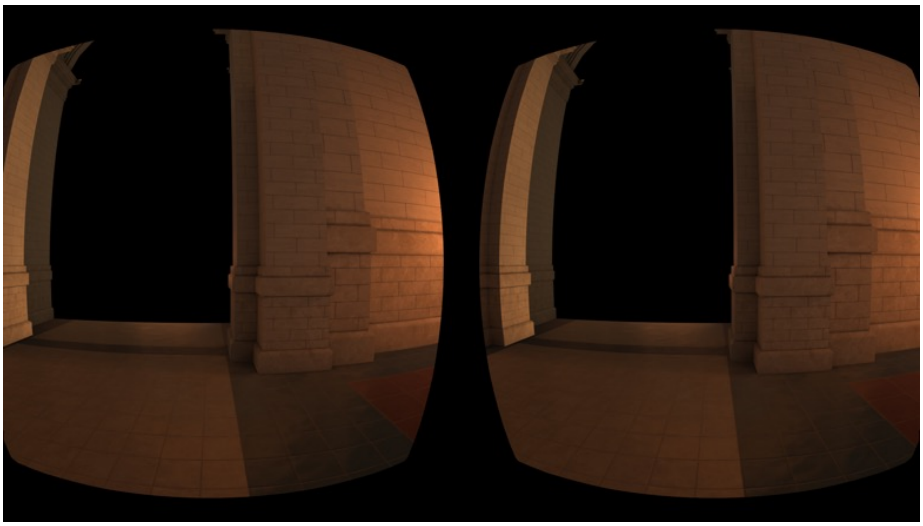
3.2.1.3 Hybridní monoskopie

Ačkoli je stereoskopie skvělým způsobem, jak ve VR hrách vytvořit iluzi objemu, zabírá také hodně výpočetního výkonu GPU. K vytvoření stereoskopického obrazu je třeba

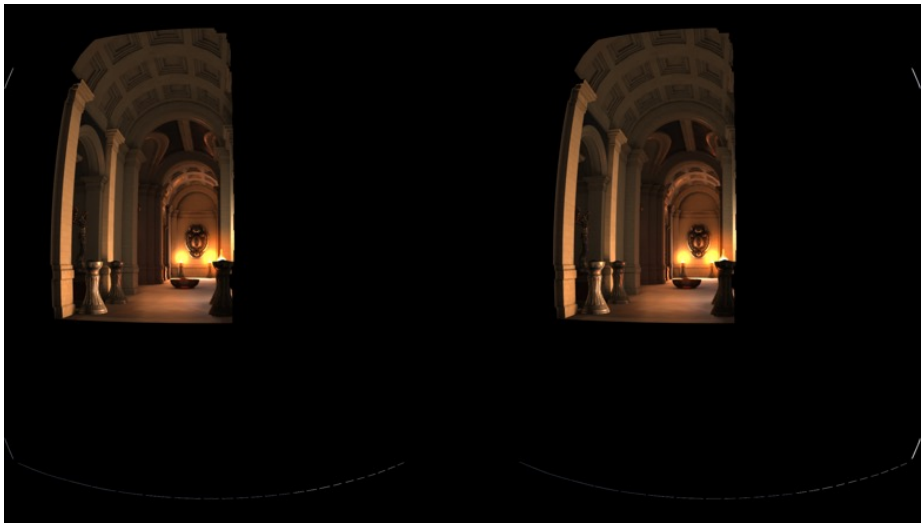
scénu vykreslit dvakrát. Při poměrně vysokém rozlišení každého snímku (minimálně 2K), lze získát velmi náročnou operaci, kterou musí počítač opakovat každý snímek po celou dobu trvání aplikace. Zde přichází na řadu hybridní monoskopie.

Jak název napovídá, je hybridem monoskopického a stereoskopického zobrazení. Jaký to má však smysl? Zpět na obrázek 3.2. Stereoskopie je v něm dobře viditelná na objektech, které jsou relativně blízko kamery, jako je například sloup veřejného osvětlení nebo budova za ním. Jak je to však s velmi vzdálenými objekty, například s městem v pozadí fotografie? Téměř žádný rozdíl. Dalo by se dokonce říci, že rozdíl mezi oběma obrázky je v rámci chyby. Nikdo si tak ani nevšimne, když pozadí z levého obrázku nahradíte pozadím z pravého obrázku.

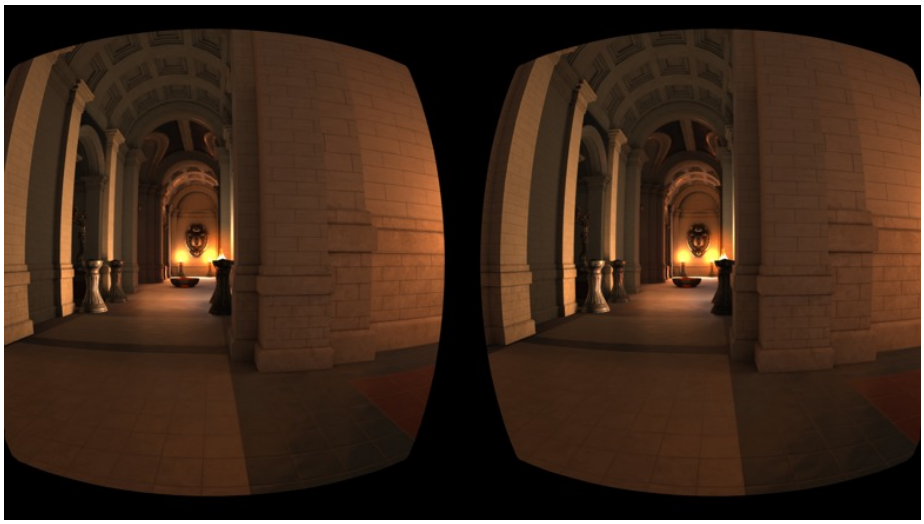
To je podstata hybridního zobrazení scény. Všechny objekty, které jsou blíže než určitá hranice, se vykreslí dvakrát, čímž vznikne stereoskopický obraz. Objekty, které jsou dále od hranice, se vykreslí pouze jednou.



Obrázek 3.3: Blízké objekty, které se vykreslují dvakrát[5]



Obrázek 3.4: Vzdálené objekty, které se vykreslí jednou[5]



Obrázek 3.5: Konečný obrázek[5]

V roce 2016 společnost Oculus aktivně vyvíjela tento režim vykreslování v herních enginech Unreal Engine 4 a Unity[5]. Tento režim byl zaveden i v dřívějších verzích UE4, ale následně byl ve verzi 4.22 odstraněn[20].

3.2.1.4 Obrázek 360°

V podstatě se nejedná o metodu vykreslování jako v předchozích třech případech. Tato metoda se však také někdy používá k vytváření zážitků ve virtuální realitě.

Princip je podobný jako u skyboxů ve hrách. Kolem statické kamery se vytvoří krychle/koule s normálami směrem dovnitř, na kterou se aplikuje materiál s texturou, která je 360° obrazem. Místo obrázků lze také použít video.

Obrázek 3.6: Příklad 360° snímku

Tento způsob vytváření VR zážitků se nejčastěji používá pro prezentace, kde není potřeba pohybovat se po herní scéně. Ve hrách se naopak používají méně často, s výjimkou skyboxů.

3.2.2 Získání stereoskopického obrázku

Pokud se sestaví projekt pro brýle pro virtuální realitu, je pro ně automaticky zapnutý stereoskopický režim. V případě projektu zaměřeného na systém Windows je to však trochu složitější. Zatím jsem našel pouze tři způsoby, jak spustit stereoskopii u aplikací pro Windows:

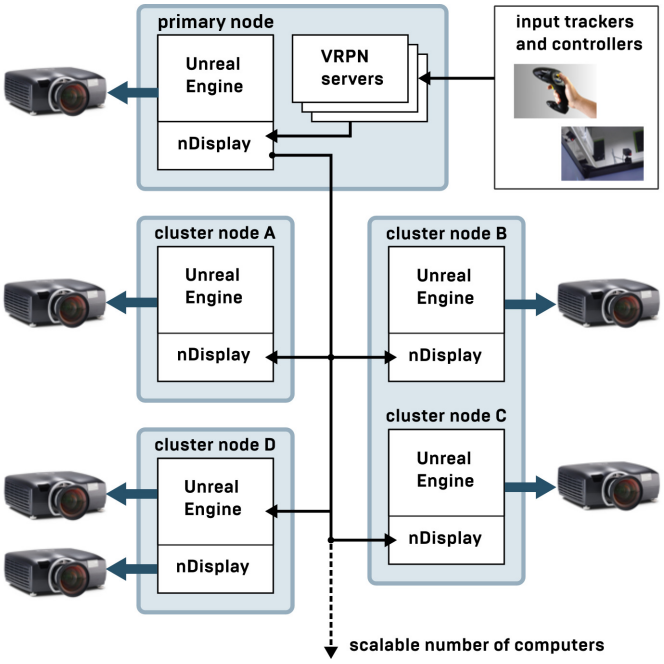
- nDisplay plugin,
- pomocí dvou kamer na herní scéně,
- pomocí příkazu emulatestereo.

3.2.2.1 nDisplay plugin

Tento plugin byl návržen pro případy, kdy „interaktivní obsah není omezen na zobrazení na jediné obrazovce nebo dokonce na jediném zařízení se dvěma obrazovkami, jako jsou VR brýle“[21].

Princip jeho fungování je následující [6]:

- K síti je připojeno několik zařízení. Jedno z nich je hlavním počítačem.
- Na každém zařízení v síti se spustí instance programu sestaveného z vašeho projektu s zapnutým a nakonfigurovaným pluginem nDisplay.
- Každé ze zařízení vykresluje scénu z pozice a směru, ve kterém se zařízení nachází ve fyzickém světě, a odešle obraz na připojený displej nebo projektor.
- Hlavní počítač je mezitím zodpovědný za příjem vstupů ze sledovacích zařízení a ovladačů prostřednictvím VRPN (Virtual-Reality Peripheral Networks) serveru a jejich předávání ostatním zařízením v síti.



Obrázek 3.7: Základní architektura nDisplay[6]

Jedním z příkladů, který vývojáři uvedli v dokumentaci, byla kopule monitorů, z nichž každý byl připojen k jednomu ze zařízení a zobrazoval scénu z perspektivy místa, kde se nacházel samotný monitor, což pozorovateli poskytovalo iluzi, že se nachází uvnitř herní scény.

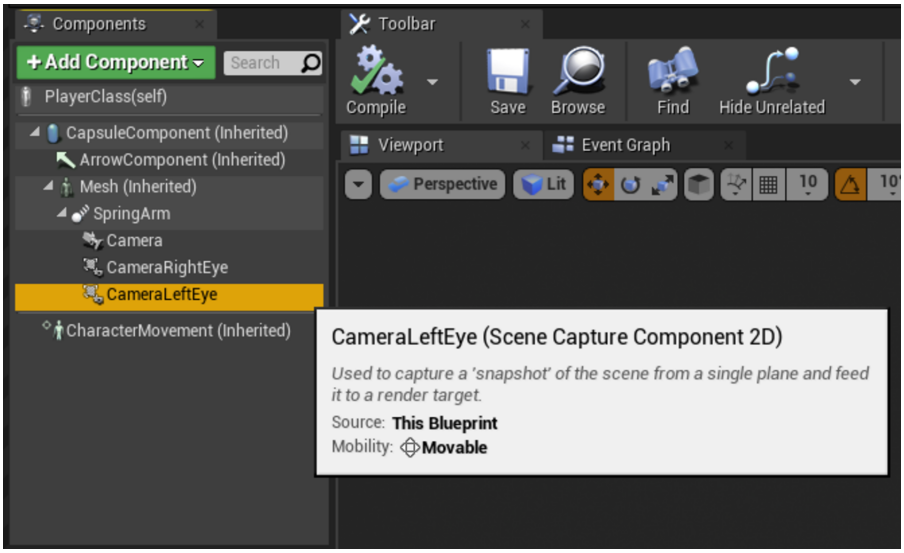
Myšlenkově je to podobné jako 360° textura, kterou jsem zmínil dříve. Tento plugin lze také přirovnat k webovým systémům SAGE a SAGE2, které umožňují synchronizovat více obrazovek a vytvořit tak jednu velkou obrazovku s vysokým rozlišením, se kterou lze následně pracovat prostřednictvím prohlížeče.

Ačkoli tento plugin poskytuje také možnost vytvořit stereoskopický obraz, pro mé účely není vhodný.

3.2.2.2 Dvě kamery ve scéně

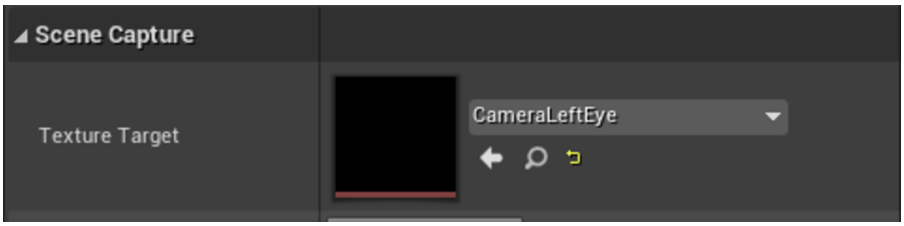
Tuto metodu jsem našel v jednom z článků na fóru UE4[7].

1. Na místě očí postavy musí být umístěny dvě kamery. Nebudete však používat běžné kamery, ale *SceneCaptureComponent2D*.



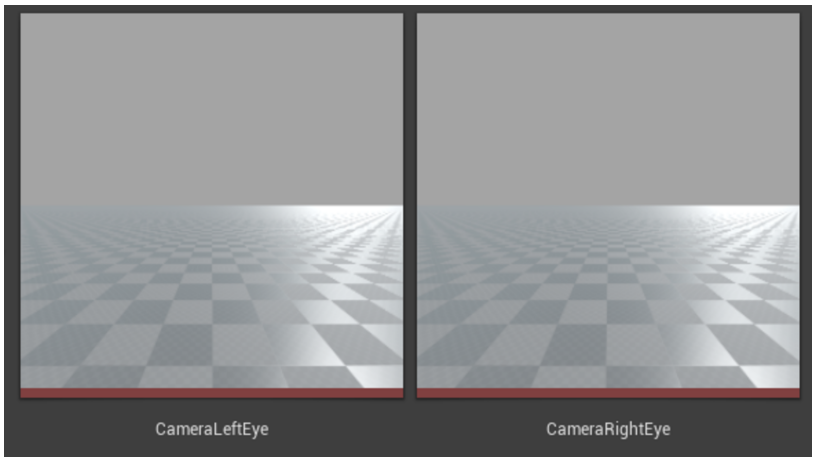
Obrázek 3.8: Jak by měla vypadat hierarchie komponent v rámci třídy postavy[7]

Hlavním rozdílem oproti běžnému *CameraComponent* je existence slotu pro texturu, do které bude kamera posílat vykreslený obraz.



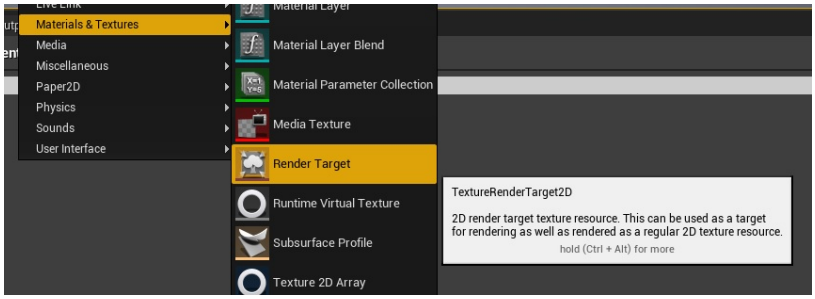
Obrázek 3.9: Umístění nastavení cílové textury[7]

2. Dále je potřeba vytvořit ve složce projektu dvě textury a sloučit je s každým „okem“ zvlášť.



Obrázek 3.10: Textury pro levé a pravé „oko“[7]

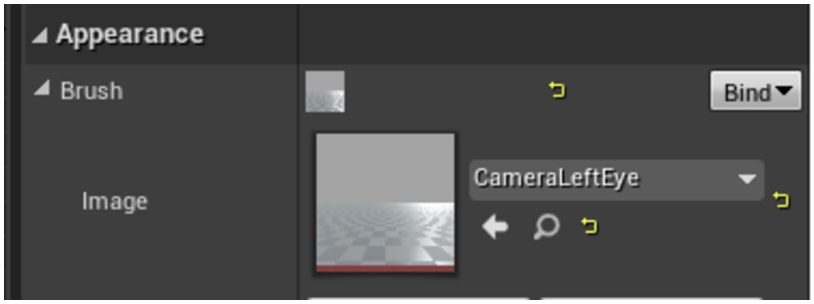
To vyžaduje speciální třídu textur nazvanou Render Target.



Obrázek 3.11: Požadovaný typ textury

3. Posledním krokem je vytvoření HUD (uživatelského rozhraní hry) a jeho umístění na obrazovku. Pro to ve složce projektu potřeba vytvořit dědice třídy *UserWidget*, přidat do něj dva obrázky vedle sebe tak, aby vyplňovaly celý prostor obrazovky, a nakonec na každý obrázek aplikovat jednu z textur, které byly vytvořeny předtím.

Obrázek 3.12: Jak by měla vypadat hierarchie třídy uživatelského rozhraní[7]



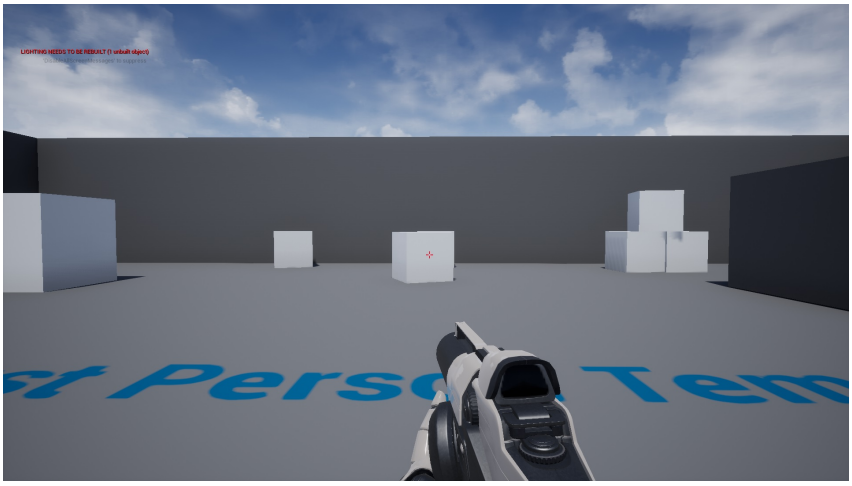
Obrázek 3.13: Kam mají být vloženy dříve vytvořené textury[7]

Tuto metodu zobrazení stereoskopie může zopakovat kdokoli. Má však jednu zásadní nevýhodu: stojí obrovské množství prostředků procesoru. Zejména při vysokém rozlišení textur. Samozřejmě je možné snížit rozlišení a tím snížit zátěž procesoru, ale bylo by téměř nemožné hrát VR hru s takovou kvalitou obrazu. Kromě toho jsem si také všiml, že na výsledném snímku chybí *Anti Aliasing*.

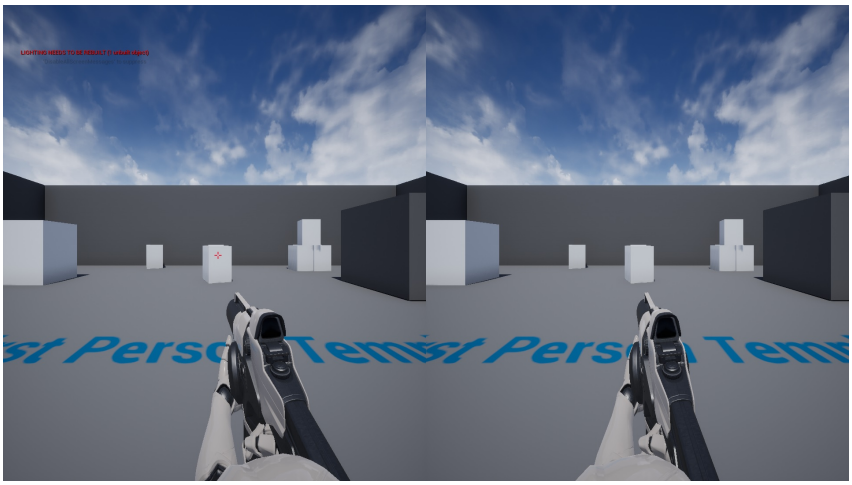
Na základě výše uvedených problémů pro mě tato možnost není vhodná.

3.2.2.3 Příkaz emulatestereo

Jak název napovídá, tento příkaz spouští emulaci stereoskopie. Je tu ale jeden závažný problém. Obraz je deformovaný.



Obrázek 3.14: Obrazovka s vypnutým *emulatestereo*



Obrázek 3.15: Obrazovka se zapnutým *emulatestereo*

Je to zřejmé, pokud se podíváte na snímky shora. Nepodařilo se mi najít řešení tohoto problému, takže tato varianta pro mě taky nevhodná.

3.2.3 Řešení

Přestože mi žádné z výše uvedených řešení nepomohlo, stále se s tím dá něco dělat. Při pohledu na zdrojový kód UE4 lze v souboru *UnrealEngine.cpp* najít ten samý kus kódu, který spouští simulaci stereoskopie při volání příkazu *emulatestereo*.

```
1 ...
2 static TAutoConsoleVariable<int32> CVarEmulateStereo(TEXT("r.EnableStereoEmulation"), 0,
  ↪  TEXT("Emulate stereo rendering"));
3 if (FParse::Param(FCommandLine::Get(), TEXT("emulatestereo")) ||
  ↪  CVarEmulateStereo.GetValueOnAnyThread() != 0)
4 {
5         TSharedPtr<FFakeStereoRenderingDevice, ESPMode::ThreadSafe> FakeStereoDevice(new
          ↪  FFakeStereoRenderingDevice());
6         StereoRenderingDevice = FakeStereoDevice;
7 }
8 ...
```

Důležitým detailem je tady *StereoRenderiongDevice*. Tato proměnná je ukazatel na objekt, jehož třída implementuje rozhraní *IStereoRendering*. Dalším důležitým prvkem je třída *FFakeStereoRenderingDevice*. V podstatě se jedná o třídu, která implementuje toto rozhraní.

Z toho lze vyvodit, že rozhraní *IStereoRendering* je rozhraním odpovědným za stereoskopii. To se projeví i při pohledu na další pluginy, které mají něco společného s virtuální realitou, jako jsou GoogleVR, Oculus, OpenXR atd. Všechny mají třídu, která implementuje metody rozhraní *IStereoRendering*.

Třída *FFakeStereoRenderingDevice* se bohužel nachází v souboru *UnrealEngine.cpp*, což znemožňuje její načtení a další opravu mimo tento soubor. Kromě toho se při spuštění projektu pomocí příkazu bude použita nefunkční verze třídy. Bez příkazu nehraje tato třída žádnou roli a jediným způsobem spustit stereoskopii, je připojení zařízení implementujícího *IStereoRendering*.

V této situaci je podle mého názoru jediným řešením napsat vlastní plugin, který pomocí tohoto rozhraní bude moci spustit stereoskopii bez nutnosti připojení dalších zařízení. Tento plugin nebude přímou součástí enginu, takže v něm mohu snadno provádět vlastní změny.

3.3 Přenos vstupu z webového klienta do herního klienta

Standardně má hráč jako vstupní zařízení k dispozici klávesnici, myš a touchpad mobilního zařízení. V oficiální dokumentaci se uvádí, že zpracování vstupů se provádí pomocí souboru *app.js*[22], který server odesílá spolu s HTML a dalšími soubory ještě před spuštěním relace. Tento soubor obsahuje seznam dostupných příkazů pro přenos do herního klienta.

```
1  // Must be kept in sync with PixelStreamingProtocol::EToUE4Msg C++ enum.
2  const MessageType = {
3
4          /**********************************************************************/
5          /*
6           * Control Messages. Range = 0..49.
7           */
8          IFrameRequest: 0,
9          RequestQualityControl: 1,
10         MaxFpsRequest: 2,
11         AverageBitrateRequest: 3,
12         StartStreaming: 4,
13         StopStreaming: 5,
14         LatencyTest: 6,
15         RequestInitialSettings: 7,
16         /**********************************************************************/
17         /*
18          * Input Messages. Range = 50..89.
19          */
20
21         // Generic Input Messages. Range = 50..59.
22         UIInteraction: 50,
23         Command: 51,
24
25         // Keyboard Input Message. Range = 60..69.
26         KeyDown: 60,
27         KeyUp: 61,
28         KeyPress: 62,
29
30         // Mouse Input Messages. Range = 70..79.
31         MouseEnter: 70,
32         MouseLeave: 71,
33         MouseDown: 72,
34         MouseUp: 73,
35         MouseMove: 74,
36         MouseWheel: 75,
```

```
37
38          // Touch Input Messages. Range = 80..89.
39          TouchStart: 80,
40          TouchEnd: 81,
41          TouchMove: 82
42          /*****************************************************************/
43  };
```

V souboru *ProtocolDefs.h* zdrojového kódu pluginu najdete enumerátor *EToStreamerMsg* se stejnými hodnotami. Tento enumerátor se aktivně používá v souboru *InputDevice.cpp*, který přijímá a zpracovává příkazy z prohlížeče.

3.3.1 Přenos vstupních dat

Protože nebudu plugin přepisovat, nemůžu přidat nové typy příkazů pro sledování pozice a otáčení ovladačů a VR brýlí, ani pro zpracování vstupů. Nicméně, při pohledu na seznam dostupných příkazů lze uvidět kromě standardních klávesnice, myši a touchpadu ještě další 3 typy příkazů:

- pro odeslání přímo do příkazového řádku Unreal Engine,
- pro přenos vstupů z uživatelského rozhraní webové stránky,
- a také standardní sadu příkazů pluginu.

Z nich mě nejvíce zajímá ten, který je zodpovědný za vstup z uživatelského rozhraní webové stránky – *UIInteraction*. Tento typ příkazu se předává z pluginu do komponenty *PixelStreamerInputComponent*. Výstupem této komponenty je řetězec, což znamená, že v tomto typu příkazu lze zakódovat jakoukoli informaci, kterou potřebuji. Takže to je to, co budu v budoucnu používat.

3.3.2 Zpracování vstupních dat

K zadání pozice a rotace můžu použít funkce typu *SetLocation* a *SetRotation*, které poskytuje herní engine. Nicméně, co dělat se vstupy od ovladačů. Nejjednodušším řešením by bylo vytvořit několik eventů v rámci komponenty, která bude zpracovávat vstupní data. Takové eventy by však měly omezená nastavení ve srovnání s těmi, které poskytuje Unreal. To by také velmi zkomplikovalo další vývoj, protože vývojáři by museli měnit úplně všechny vstupní eventy. Samotný engine bohužel neposkytuje žádnou funkcionalitu pro přímé zadávání vstupů od uživatele kromě ovladačů.

Pixel Streaming řeší tento problém vytvořením třídy, která implementuje rozhraní *IInputDevice*. Toto rozhraní je zodpovědné za předávání vstupních dat enginu k následnému zpracování. Na základě tohoto rozhraní chci vytvořit vlastní plugin, který mi umožní odesílat data přímo do enginu. Tímto způsobem mohu používat eventy poskytované Unrealem, aniž bych musel měnit kód projektu, aby se přizpůsobil mým změnám.

Kapitola 4

Implimentace VR rozšíření pro Pixel Streaming

4.1 Web klient

4.1.1 Inicializace virtuální scény

Nejprve je třeba stáhnout soubor s knihovnou Three.js a připojit jej ke stránce. To lze udělat prostřednictvím *player.html* stejným způsobem jako soubory *app.js* m *webRtcPlayer.js*.

```
1 <script type="text/javascript" src="scripts/webRtcPlayer.js"></script>
2 <script type="text/javascript" src="scripts/app.js"></script>
3 <script type="text/javascript" src="scripts/three.js"></script>
```

Dalším krokem je vytvoření souboru, který bude obsahovat inicializační a vykreslovací funkce pro herní scénu. Jako základ budu používat příklad z oficiální stránky Three.js[23]. Tento příklad ukazuje použití 360° stereoskopického videa jako video textury aplikované na kouli kolem stereoskopické kamery. Zde použité video je vloženo do video objektu stránky HTML, který je následně použit k vytvoření video textury.

V tomto příkladu musím přepsat inicializační funkci. Část se získáním video objektu odstraním. Místo toho předám tento objekt prostřednictvím parametrů funkce. Také odstraním části s nalezením kontejneru a sledováním kliknutí myší na objekt ze začátku. Tato funkcionalita je již implementována v souboru *app.js*. Na závěr je třeba nahradit koule, na které se aplikuje video textura plochami, které mají být připojeny ke kameře.

Logika změny UV souřadnic u ploch je potřebná, ale je třeba ji trochu přepsat. Jde o to, že obrázek, který přijde do webového klienta, bude *side-by-side*, tj. bude vypadat jako na obrázku 3.15. Takže pomocí UV souřadnic rozdělím texturu na dvě části pro každé oko zvlášť.

Takto to vypadá u levého oka:

```
// left eye
let geometry_L = new THREE.PlaneGeometry(2, 2);
var uvAttribute_L = geometry_L.attributes.uv;
for (var i = 0; i < uvAttribute_L.count; i++) {
var u = uvAttribute_L.getX(i);
uvAttribute_L.setX(i, u == 1 ? 0.5 : u);
}
stream_plane_L = new THREE.Mesh(geometry_L, stream_material);
stream_plane_L.layers.set(1); //1
stream_plane_L.position.z = -1; camera.add(stream_plane_L);
```

Zbývá už jen tlačítko pro přepnutí do celoobrazovkového VR režimu. Třídu *VRButton*, která se pro tento účel používá, lze najít v jiném souboru ve stejném Git repozitáři. Takže si také stáhnu tento soubor a připojím ho ke stránce.

Pro spuštění virtuální scény je třeba zavolat inicializační a vykreslovací funkce. To provedu po stisknutí tlačítka Start. Logika pro stisknutí tlačítka je zapsána ve funkci *showPlayOverlay()* v souboru *app.js*.

```
function showPlayOverlay() {
    var img = document.createElement('img');
    img.id = 'playButton';
    img.src = '/images/Play.png';
    img.alt = 'Start Streaming';
    setOverlay('clickableState', img, event => {
        if (webRtcPlayerObj)
            webRtcPlayerObj.video.play();

        requestInitialSettings();
        requestQualityControl();

        showFreezeFrameOverlay();
        hideOverlay();

        // Virtual scene initialization and animation functions
        initThreejsScene(webRtcPlayerObj.video);
        animateThreejsScene();
    });
    shouldShowPlayOverlay = false;
}
```

Pokud se nyní pokusím spustit webového klienta na jiném zařízení, nebudu moci přepnout do celoobrazovkového VR režimu. To vyžaduje použití zabezpečeného připojení.

Lze jej zapnout ve složce signalizačního serveru v souboru *config.json*. Je také nutné mít certifikát a klíč ve složce *SignallingWebServer/certificates*. Pro testování vygeneruji svůj certifikát a klíč pomocí knihovny OpenSSL.

4.1.2 Sledování vstupních dat

4.1.2.1 Sledování polohy a otáčení brýlí a ovladačů

Pozice a rotace kamery jsou automaticky synchronizovány, jakmile je aktivován celoobrazovkový režim. Co se týče kontrolerů, je třeba je nejprve získat pomocí funkce *renderer.xr.getController(int controllerIndex)*, kde *renderer* je objekt třídy *WebGLRenderer*, kterou poskytuje Three.js, a *controllerIndex* je číslo požadovaného kontroleru. Po přidání kamery i ovladačů do scény bude možné sledovat jejich pozici a rotaci.

Při přenosu těchto dat do herního klienta je třeba mít na paměti několik důležitých bodů:

- Za prvé, WebGL, na kterém je Three.js založen, a Unreal Engine mají odlišné souřadnicové systémy. Začneme tím, že WebGL používá pravoruký souřadnicový systém, zatímco Unreal levoruký. Souřadnicový systém Unrealu je navíc otočen tak, aby osa Z směřovala nahoru, osa X dopředu a osa Y doprava. Takže ještě před odesláním je třeba převést všechna data z pravorukého souřadnicového systému do levorukého s ohledem na jeho rotace v UE4.
- Za druhé, je třeba zvážit, jak odeslat rotaci objektů. Když jsem se nejprve snažil poslat rotaci jako Eulerovy úhly, a to i s ohledem na rozdíl v souřadnicových systémech, narazil jsem na problém. Když jsem se snažil otočit hlavu o více než o 90 stupňů, kamera se prudce otočila na opačnou stranu, po čemž se ovládání obrátilo. Tento problém se vyřešil, když jsem místo Eulerových úhlů začal předávat kvaterniony.

4.1.2.2 Sledování vstupů z ovladačů

Pro získání vstupu od hráče je třeba nejprve získat objekt třídy *XRSession*, který poskytuje WebXR. Tento objekt obsahuje pole objekty, z nichž každý charakterizuje ovladač připojený k současné relaci. Tyto objekty obsahují informace o aktuálním stavu ovladače a jednotlivých tlačítek a os. Tlačítka jsou sledována nejen stisknutím, ale i dotykem, pokud to platforma podporuje (např. Oculus Quest 2 podporuje sledování dotyků tlačítek). Tyto objekty mohou také ukázat, zda je ovladač pravý nebo levý.

Na základě výše uvedeného napíšu funkci, která bude každý snímek kontrolovat stav každého tlačítka a osy na obou ovladačích a zapisovat je do bufferu, který pak budu předávat hernímu klientovi.

4.1.3 Odesílání vstupních dat hernímu klientovi

Přestože vývojáři v oficiální dokumentaci doporučují použít řetězec ve formátu Json, nebudu tuto metodu používat. Za prvé by taková zpráva byla poměrně velká a za druhé by její dekódování trvalo poměrně dlouho, protože bych musel pracovat s řetězcem. Takže všechna data týkající se stisknutí nebo dotyku tlačítek zakóduji do proměnné *short int* pomocí bitové masky.

```
1  const ButtonsInputCodes = {
2
3      LTriggerTouch: 0x0001,          // 0000 0000 0000 0001
4      LStickTouch: 0x0002,            // 0000 0000 0000 0010
5      LXTouch: 0x0004,                // 0000 0000 0000 0100
6      LYTouch: 0x0008,                // 0000 0000 0000 1000
7
8      RTriggerTouch: 0x0010,          // 0000 0000 0001 0000
9      RStickTouch: 0x0020,            // 0000 0000 0010 0000
10     RXTouch: 0x0040,                // 0000 0000 0100 0000
11     RYTouch: 0x0080,                // 0000 0000 1000 0000
12
13     LStick: 0x0100,                 // 0000 0001 0000 0000
14     LX: 0x0200,                     // 0000 0010 0000 0000
15     LY: 0x0400,                     // 0000 0100 0000 0000
16
17     RStick: 0x0800,                 // 0000 1000 0000 0000
18     RX: 0x1000,                     // 0001 0000 0000 0000
19     RY: 0x2000                      // 0010 0000 0000 0000
20
21 };
```

Pro kódování tlačítek, která mají stupeň stisku, použiji proměnné typu *unsigned byte* a pro osy – *short int*. Rotaci ovladačů a VR brýlí budu předávat pomocí čtyř hodnot *short int*, které dohromady tvoří kvaternion. Pro přesný přenos pozice potřebuji přesnější hodnoty, takže je budu posílat jako *float*.

Někteří lidé mohou mít otázky ohledně přesnosti přenosu hodnot s plovoucí desetinnou čárkou pomocí celých čísel, ale mohu ujistit, že chyba je v tomto případě minimální a není fyzicky citelná.

Po zakódování všech dat je třeba je předat hernímu klientovi. Jak jsem se již dříve zmínil, pro to budu používat příkaz typu *UIInteraction*. V souboru *app.js* již byla napsána funkce, která to umožňuje, ale je určena pro text. Takže napíšu vlastní, která podobně jako již existující funkce vloží zakódovaná data do bufferu a předá je přímo hernímu klientovi.

Funkce pro odeslání příkazu se nazývá *sendInputData(data)*, kde *data* je buffer, do kterého je tento příkaz zapsán. Jako buffer pro příkaz budu používat objekt třídy *ArrayBuffer* a také *DataView*, který umožní zápis dat do něj.

Struktura příkazu je následující:

- hlavička obsahující id použitého příkazu,
- délka datového bufferu v bajtech děleno dvěma,
- samotný datový buffer.

Konečná funkce vypadá následovně:

```
1  function sendVRInputData(data_array) {
2          let data = new DataView(new ArrayBuffer(77));
3          data.setUint8(0, MessageType.UIInteraction);
4          data.setUint16(1, 37, true);
5
6          // set click input data
7          data.setUint16(3, data_array[0], true);
8
9          // set analog input of left controller
10         data.setUint8(5, data_array[1], true);      // left trigger
11         data.setUint8(6, data_array[2], true);      // left grip
12         data.setInt16(7, data_array[3], true);      // left left-right stick
13         data.setInt16(9, data_array[4], true);      // left up-down stick
14
15         // set analog input of right controller
16         data.setUint8(11, data_array[5], true);     // right trigger
17         data.setUint8(12, data_array[6], true);     // right grip
18         data.setInt16(13, data_array[7], true);     // right left-right stick
19         data.setInt16(15, data_array[8], true);     // right up-down stick
20         //-----------------------------------------------------------------
21         //set camera position
22         data.setFloat32(17, data_array[9], true);   // x position
23         data.setFloat32(21, data_array[10], true);  // y position
24         data.setFloat32(25, data_array[11], true);  // z position
25         //set camera rotation by quaternions
26         data.setInt16(29, data_array[12], true);    // x quaternion
27         data.setInt16(31, data_array[13], true);    // y quaternion
28         data.setInt16(33, data_array[14], true);    // z quaternion
29         data.setInt16(35, data_array[15], true);    // w quaternion
30
31         //set left controller position
32         data.setFloat32(37, data_array[16], true);  // x position
```

```
33          data.setFloat32(41, data_array[17], true);  // y position
34          data.setFloat32(45, data_array[18], true);  // z position
35          //set left controller rotation by quaternions
36          data.setInt16(49, data_array[19], true);    // x quaternion
37          data.setInt16(51, data_array[20], true);    // y quaternion
38          data.setInt16(53, data_array[21], true);    // z quaternion
39          data.setInt16(55, data_array[22], true);    // w quaternion
40
41          //set right controller position
42          data.setFloat32(57, data_array[23], true);  // x position
43          data.setFloat32(61, data_array[24], true);  // y position
44          data.setFloat32(65, data_array[25], true);  // z position
45          //set right controller rotation by quaternions
46          data.setInt16(69, data_array[26], true);    // x quaternion
47          data.setInt16(71, data_array[27], true);    // y quaternion
48          data.setInt16(73, data_array[28], true);    // z quaternion
49          data.setInt16(75, data_array[29], true);    // w quaternion
50          //-----------------------------------------------------------------
51
52          sendInputData(data.buffer);
53  }
```

4.2 Herní klient

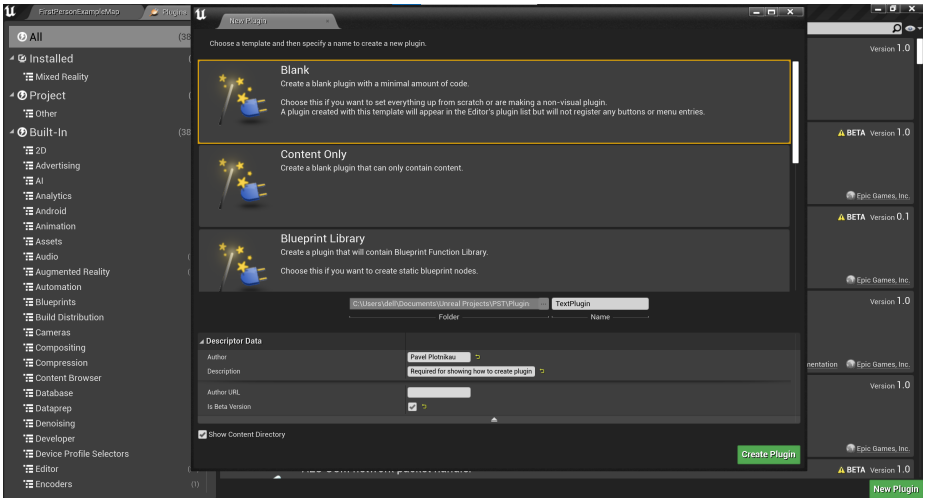
4.2.1 Moduly v Unreal Engine 4

Než začnu popisovat proces psaní pluginů, je třeba upozornit na jednu důležitou věc. Unreal Engine 4 je postaven na velkém množství modulů. Každý z těchto modulů plní zvláštní úkol. Některé jsou připojeny zpočátku a některé lze připojit jako plugin. Moduly mohou také implementovat veřejné rozhraní pro použití jinými moduly nebo vývojáři.

To bylo zmíněno, protože rozhraní, o kterých jsem se zmínil dříve a kolem kterých jsem chtěl vytvořit pluginy, nebudou fungovat sama o sobě. Každé z nich bude fungovat pouze v rámci určitého modulu. Proto je nutné nejprve implementovat požadovaný modul a pak už lze implementovat potřebné rozhraní pro tento modul.

4.2.2 Input Device Emulator Plugin

Nejprve je třeba vytvořit nový plugin. To je možné udělat na stránce pluginů v otevřeném projektu. Je nutné vybrat prázdný základ.



Obrázek 4.1: Jak vytvořit plugin

Vytvořený plugin pak lye najdít v automaticky vytvořené složce *„VášProjekt“/Plugins*. Současně budou vygenerovány soubory *.h a *.cpp. Soubor *.h by měl vypadat takto:

```
1  // Copyright Epic Games, Inc. All Rights Reserved.
2
3  #pragma once
4
5  #include "CoreMinimal.h"
6  #include "Modules/ModuleManager.h"
7
8  class FTextPluginModule : public IModuleInterface
9  {
10 public:
11
12     /** IModuleInterface implementation */
13     virtual void StartupModule() override;
14     virtual void ShutdownModule() override;
15 };
```

Výsledný základní modul musí být přepsán do modulu vstupního zařízení IInputDeviceModule.

Nejdříve opravím soubor s příponou *.Build.cs*, který se nachází ve složce pluginu. Tento soubor je zodpovědný za to, jak bude modul sestaven pro daný projekt. Obsahuje také cestu do původního modulu. Pokud tento soubor není opraven, modul neuvidí některé důležité knihovny.

Protože se plugin nenachází ve složce herního enginu, je třeba změnit proměnné *PublicIncludePaths* a *PrivateIncludePaths* takto:

```
PublicIncludePaths.Add(Path.Combine(ModuleDirectory, "Public"));

PrivateIncludePaths.Add(Path.Combine(ModuleDirectory, "Private"));
```

Překladač tak bude znát umístění modulu a bude moci poskytnout přístup k požadovaným knihovnám a třídám.

Dále je třeba přidat potřebné moduly. Pro testování bude stačit zveřejnit všechny již připojené moduly a přidat moduly *InputCore* a *InputDevice*. Mělo by to vypadat takto:

```
PublicDependencyModuleNames.AddRange(
            new string[]
            {
                "Core",
                "CoreUObject",
                "Engine",
                "Slate",
                "SlateCore",
                "InputCore",
                "InputDevice",
                // ... add other public dependencies that you statically link with here
                ↪  ...
            }
);
```

Nyní lze připojit knihovnu *InputDevice/Public/IInputDeviceModule.h* a změnit rozhraní standardního modulu na rozhraní modulu vstupního zařízení. Dále je třeba *IInputDeviceModule* implementovat. Jedinou abstraktní metodou tohoto rozhraní je *CreateInputDevice*. Mám v plánu volat tento modul během hry, abych získal vstupní zařízení, takže dodatečně vytvořím pár statických metod *Get* a *IsAvailable*. Základem jsou stejné metody implementované v rozhraní *IInputDeviceModule*.

V souhrnu by třída měla vypadat takto:

```
class FInputDeviceEmulatorPluginModule : public IInputDeviceModule
{
public:

    static inline FInputDeviceEmulatorPluginModule& Get()
    {
```

```
7           return FModuleManager::LoadModuleChecked<FInputDeviceEmulatorPluginModule>
8           (
9               "InputDeviceEmulatorPlugin"
10          );
11      }
12
13      static inline bool IsAvailable()
14      {
15          return FModuleManager::Get().IsModuleLoaded("InputDeviceEmulatorPlugin");
16      }
17
18      virtual TSharedPtr< class IInputDevice > CreateInputDevice(const TSharedRef<
        ↪  FGenericApplicationMessageHandler >& InMessageHandler);
19
20      /** IModuleInterface implementation */
21      virtual void StartupModule() override;
22      virtual void ShutdownModule() override;
23  };
```

Také nezapomínám na funkce *StartupModule* a *ShutdownModule*. Při zapnutí modul musí být zaregistrován a stejně tak při vypnutí musí být odregistrován.

```
1   void FInputDeviceEmulatorPluginModule::StartupModule()
2   {
3       // This code will execute after your module is loaded into memory; the exact timing
        ↪  is specified in the .uplugin file per-module
4
5       IModularFeatures::Get().RegisterModularFeature(
6           IInputDeviceModule::GetModularFeatureName(), this
7       );
8   }
9
10  void FInputDeviceEmulatorPluginModule::ShutdownModule()
11  {
12      // This function may be called during shutdown to clean up your module.  For modules
        ↪  that support dynamic reloading,
13      // we call this function before unloading the module.
14
15      IModularFeatures::Get().UnregisterModularFeature(
16          IInputDeviceModule::GetModularFeatureName(), this
17      );
18  }
```

Zbývá už jen metoda *CreateInputDevice*, ale předtím je třeba vytvořit třídu vstupního zařízení. V projektu vytvořím novou třídu a umístím ji do složky pluginu. Třída musí být prázdná. Připojuji knihovnu *IInputDevice.h* a rozhraní *IInputDevice*. Dále je třeba implementovat všechny abstraktní metody rozhraní a také konstruktor a destruktor třídy. Potřebuji však pouze konstruktor a metodu *SetMessageHandler*, protože oba umožňují načtení objektu třídy *FGenericApplicationMessageHandler*. Podle oficiální dokumentace je *FGenericApplicationMessageHandler* „Rozhraním, které definuje způsob interakce s uživatelem prostřednictvím hardwarového vstupu a výstupu“[24]. Tomuto objektu budu pak předávat všechna vstupní data. Abych mohl s objektem externě interagovat, vytvořím další dvě funkce: *SetButtonState* a *SetAxisState*. Vypadají následovně:

```
1  bool FInputDeviceEmulator::SetButtonState(const FName& Key, int32 ControllerID, bool
   ↪  IsPressed, bool IsRepeat)
2  {
3      bool Result = false;
4      if (IsPressed)
5      {
6          Result = MessageHandler->OnControllerButtonPressed(Key, ControllerID, IsRepeat);
7      }
8      else
9      {
10         Result = MessageHandler->OnControllerButtonReleased(Key, ControllerID,
           ↪  IsRepeat);
11     }
12     return Result;
13 }
14
15 bool FInputDeviceEmulator::SetAxisState(const FName& Key, int32 ControllerID, float
   ↪  Value)
16 {
17     bool Result = false;
18     Result = MessageHandler->OnControllerAnalog(Key, ControllerID, Value);
19     return Result;
20 }
```

Zde dokončím psaní třídy vstupního zařízení a vrátím se k metodě *CreateInputDevice*. V této metodě budu vytvářet objekt vstupního zařízení a vrácet na něj ukazatel. Nezapomenu také uložit tento ukazatel a napsat funkci, která mi ho vrátí.

```
1  TSharedPtr< IInputDevice > FInputDeviceEmulatorPluginModule::CreateInputDevice(const
   ↪  TSharedRef< FGenericApplicationMessageHandler >& InMessageHandler)
2  {
3      InputDeviceEmulator = MakeShareable(new FInputDeviceEmulator(InMessageHandler));
```

```
4
5       return InputDeviceEmulator;
6   }
7
8   TSharedPtr<FInputDeviceEmulator> FInputDeviceEmulatorPluginModule::GetInputDevicePtr()
9   {
10      return InputDeviceEmulator;
11  }
```

4.2.3 Stereo Renderer Emulator Plugin

Nový plugin vytvořím stejně jako v části 3.2.2. Stejným způsobem změním soubor *.Build.cs*, ale místo modulů *InputCore* a *InputDevice* přidám *HeadMountedDisplay*, *RenderCore*, *Renderer* a *RHI*. Rovněž přejdu přímo do souboru modulu, připojím soubor *IHeadMountedDisplayModul.h* a změním standardní rozhraní na *IHeadMountedDisplayModul*.

Zde je třeba implementovat dvě abstraktní metody tohoto rozhraní: *CreateTrackingSystem* a *GetModuleKeyName*. Stejně přidám statické metody *IsAvailable* a *Get* a přepíšu metodu *IsHMDConnected* tak, aby vždy vracela *true*. Nezapomínám na funkce *StartupModule* a *ShutdownModule*. Protože při zapínání a vypínání modulu nepotřebuji nic dělat, budu používat jejich implementaci z rozhraní *IHeadMountedDisplayModule*.

Nyní je potřeba probrat, jak přesně se stereoskopický renderer spouští. Zpět na místo v souboru *UnrealEngine.cpp*, kde se vybírá stereoskopický renderer. Pokud není použit příkaz *emulatestereo*, je princip výběru následující:

1. Na začátku se ze všech HMD modulů vybírá pomocí metody *IsHMDConnected* jeden aktivní.

2. Vybraný modul pak vytvoří objekt třídy, která implementuje rozhraní *IXRTrackingSystem*, a uloží ukazatel na ně do proměny XRSystem.

3. Pokud je ukazatel platný, vytváří se pomocí tohoto objektu stereo-renderer.

Na základě toho musím pro vytvoření stereoskopického rendereru nejprve implementovat rozhraní *IXRTrackingSystem*. Nicméně podle toho, co se stane, když používám příkaz *emulatestereo*, není *XRSystem* vyžadován. To znamená, že si mohu zjednodušit práci a neimplementovat celé rozhraní. Pouze funkce, která je zodpovědná za vytvoření rendereru. Pak, aby *XRSystem* nezasahoval do aplikace, vymažu ze systému ukazatel na něj. V tomto případě bude celá aplikace fungovat, jako kdybych použil *emulatestereo*.

Vytvořím pomocí editoru dvě prázdné třídy. Aby to bylo ještě jednodušší, zkopíruji do třídy, která bude implementovat rozhraní *IStereoRendering*, kód ze třídy *FFakeStereoRenderingDevice*. Jediné, co musím udělat, je opravit funkci *GetStereoProjectionMatrix*. To je příčinou problému deformaci obrazu. Při výpočtu matice se použije výška a šířka obrazovky. Abych se tohoto problému zbavil, rozdělil jsem šířku obrazovky dvěma.

Ve druhé třídě implementuji rozhraní *IXRTrackingSystem*, ale ponechám všechny metody prázdné kromě té, která vytváří renderer. Protože *IXRTrackingSystem* a *IStereoRendering* jsou obvykle implementovány ve stejné třídě, rozhodl jsem se renderer vytvořit pomocí operátoru ***new***.

```
1  TSharedPtr<class IStereoRendering, ESPMode::ThreadSafe>
   ↪  FEmulatorXRSystem::GetStereoRenderingDevice()
2  {
3      FStereoEmulatorPtr Emulator = MakeShareable(new FStereoRendererEmulator);
4      OnEmulatorCreatedEvent.ExecuteIfBound(Emulator);
5      return Emulator;
6  }
```

Zbývá už jen dopsat funkci *CreateTrackingSystem*. Způsob vytvoření systému XR jsem převzal z jiných pluginů zaměřených na virtuální realitu, které také používají rozhraní *IXRTrackingSystem*. K tomu však musí třída XR systémy implementovat další rozhraní – *FSceneViewExtentionBase*. Metody tohoto rozhraní lze také ponechat prázdné. Jediné, co musím dopsat, je konstruktor třídy.

```
1  FEmulatorXRSystem::FEmulatorXRSystem(const FAutoRegister& AutoRegister)
2      : FXRTrackingSystemBase(nullptr)
3      , FSceneViewExtensionBase(AutoRegister)
4  {
5  }
```

Plugin je připraven. Zbývá jej připojit k projektu a nakonfigurovat všechny potřebné proměnné, z nichž nejdůležitější jsou vzdálenost mezi očí a úhel pohledu kamery.

4.2.4 Rozšíření komponenty PixelStreamerInputComponent

Při spuštění aplikace předá každá komponenta *PixelStreamerInputComponent* ukazatel na sebe sama modulu vstupního zařízení implementovanému na straně Pixel Streamingu. Samotná komponenta zase implementuje *delegát*, prostřednictvím kterého modul následně předá data přijatá po síti pomocí příkazu *UIInteaction*.

Delegáti jsou objekty schopné uložit ukazatel na funkci pro následné vyvolání. Tento typ callbacku je implementován pouze v UE4 a vytváří se pomocí makra.

Prostřednictvím delegáta *PixelStreamerInputComponent* si chci svoje zakódovaná data získávat. Proto nejprve vytvořím dědice této komponenty. V něm vytvořím metodu, která bude dekódovat přijatá data. Zde je několik důležitých detailu, které je třeba při dekódování pamatovat.

- Řetězec, který tato metoda získá, používá dva bity na každé písmeno. Ze stejného důvodu musela být délka datového bufferu v bajtech při odesílání příkazu vydělena dvěma.
- Je třeba věnovat pozornost pořadí zápisu bajtů (Little-endian, Big-endian), protože to může mít vliv na dekódování jednobajtových hodnot i masky tlačítek.

V konstruktoru třídy předám metodu-dekodér delegátovi a také uložím ukazatel na modul vstupního zařízení, který jsem napsal dříve. Ukazatel na samotný objekt vstupního zařízení však musí být uložen až po spuštění hry, protože se často inicializuje až po inicializaci komponenty.

Vstupní data budu zpracovávat stejným způsobem, jako je zpracovává plugin Pixel Streaming. Veškeré vstupní údaje, kromě údajů o pozici a rotaci ovladačů a VR brýlí, uložím do fronty, ze které pak budou předány objektu vstupního zařízení ve funkci *TickComponent*, která se volá každý snímek.

Je třeba také zmínit, že komponenta a objekt vstupního zařízení běží v různých vláknech. A přestože objekt *TQueue* lze bezpečně používat mezi více vlákny, znamená to také, že do komponenty může přicházet více vstupních dat. Z nich je nejlepší vybrat ty nejaktuálnější a ostatní ignorovat.

A to je v podstatě všechno. Zbývá jen sestavit projekt, spustit hru, server a web klienta a otestovat.

Kapitola 5

Testování VR rozšířeí pro Pixel Streaming

K testování budu používat brýle pro virtuální realitu Oculus Quest 2 a herní notebook Alienware AREA-51m R2.

Konfigurace Oculus Quest 2:

- Typ obrazovky: Single Fast-Switch LCD, 1832×1920px na každé oko
- CPU: Qualcomm® Snapdragon XR2
- GPU: Qualcomm® Adreno™ 850 GPU
- Operační paměť: 6 GB

Konfigurace notebooku:

- CPU: Intel® Core™ i9-10900K CPU @ 3.70GHz 3.70 GHz
- GPU: NVIDIA® GeForce® RTX 2080 SUPER™
- Operační paměť: 64.0 GB

Aplikaci a server spustím na notebooku a webového klienta na brýlích. Testovat budu při rozlišení obrazovky 3000×1500px s bitratem 20 Mb/s. Provedu testy pomocí standardního okna statistiky dostupného na stránce web klienta.

Při těchto parametrech se konečné zpoždění pohybovalo mezi 45 a 80 ms. Zpoždění se projevilo i fyzicky. Kromě efektu duchů (ghost effect) bylo také výsledné rozlišení na oko mnohem nižší (1500×1500) než u standardního obrazu na displeji (1832×1920). Z tohoto důvodu byla výsledná kvalita snímku značně zhoršena.

Příkazem **-PixelStreamingWebRTCMaxBitrate=30000000** zkusím zvýšit bitrate do 30 Mb/s. Se zvýšením rychlosti přenosu dat by se mělo zpoždění snížit.

Podle čísel se zpoždění nezměnilo. Během testování se mi však zdálo, že se zpoždění zvětšilo.

Pokusím se také zvětšit rozšíření zaslaného obrázku. Nicméně při pokusu o spuštění media-streamu klesne snímková frekvence do 1-2 fps. Mezitím se ve Správci zařízení zvýšilo zatížení GPU do 100 procent a aplikace se ukončila s fatální chybovou zprávou.

Možným důvodem všech výše uvedených problémů může být to, že neexistuje žádný způsob, jak nastavit úroveň profilu video kodeku. Jak jsem zjistil dříve, aktuální nastavení video kodeku je Baseline profil s úrovní 3.1. Maximální rozšíření pro tuto úroveň je 1280x720 s maximální snímkovou frekvencí 30 fps[25]. Já naopak přenáším obraz s rozlišením téměř pětkrát větším, než je maximální dostupné rozlišení. Nemá také smysl měnit samotný profil, protože ačkoli se tím zvýší komprese videa, sníží se také rychlost kódování, což zvýší latenci.

Dalším možným řešením problému by mohlo být oddělení vykreslování obrázku a jeho kódování. To by však znamenalo, že místo pluginu bych musel napsat samostatnou aplikaci, která by obraz kódovála a odesílala, a ještě jednu, která by obraz přijímala, dekódovála a přenášela přímo na displeje brýlí.

Závěr

Cílem této práce bylo zjistit, zda je možné použít plugin Pixel Streaming z herního enginu Unreal Engine 4 pro streamování her ve virtuální realitě. Hlavní podmínkou bylo, že jsem nemohl nijak přepsat samotný plugin. Výjimkou jsou soubory, které signální server posílá webovému klientovi, protože ty nejsou fyzickou součástí zásuvného modulu a sami vývojáři doporučují, aby byly upraveny podle potřeby.

Nejdříve bylo třeba pochopit, jak plugin funguje. Bylo zjištěno, že plugin používá knihovnu WebRTC, která slouží k vytvoření spojení mezi několika klienty a následnému přenosu media dat. Video data jsou komprimována pomocí video kodeku H.264 ze strany herního klienta. Nakonec se na straně webového klienta přehrává video stream vytvořený pomocí souborů HTML, CSS a JavaScript, které do prohlížeče posílá signální server.

Protože Pixel Streaming nepodporoval režim VR, bylo nutné najít způsob, jak doplnit plugin beze změny základního kódu. Bylo třeba vyřešit tři hlavní problémy: neschopnost posílat obraz přímo na displeje brýlí, chybějící stereoskopie a nemožnost sledovat pohyby hlavy a ovladačů. K vyřešení prvního problému byla použita knihovna Three.js, což je v podstatě vysokoúrovňová grafická knihovna určená pro použití ve webovém prohlížeči. Pro vyřešení dalších dvou problémů bylo nutné rozšířit funkcionalitu herního enginu napsáním dvou pluginů a rozšířením možností již existující komponenty používané Pixel Streamingem.

Výsledkem je funkční verze aplikace, kterou lze spustit na počítači a ovládat pomoci brýlí. Obraz vyrenderovaný počítačem se posílá přímo na displeje brýlí prostřednictvím lokální sítě.

A přestože aplikace není příliš uživatelsky přívětivá, je zde prostor pro zlepšení, pro které by pravděpodobně musel být upraven samotný plugin.

Literatura

[1] Epic Games: *Pixel Streaming Overview [online]*. [cit. 2022-04-21]. Dostupné z: https://docs.unrealengine.com/4.27/en-US/SharingAndReleasing/PixelStreaming/PixelStreamingOverview/

[2] Epic Games: *Hosting and Networking Guide [online]*. [cit. 2022-04-21]. Dostupné z: https://docs.unrealengine.com/4.27/en-US/SharingAndReleasing/PixelStreaming/Hosting/

[3] Petkov, E.: Generation of Stereo Images in 3D Graphics Applications for Stereoscopic and Nonstereoscopic Displays [online]. *ResearchGate*, leden 2012, [cit. 2022-04-21]. Dostupné z: https://www.researchgate.net/publication/278033341_Generation_of_Stereo_Images_in_3D_Graphics_Applications_for_Stereoscopic_and_Nonstereoscopic_Displays

[4] Leisure; Dundee, C.: Introduction to Stereoscopy [online]. *Leisure and Culture Dundee*, květen 2016, [cit. 2022-04-21]. Dostupné z: http://www.leisureandculturedundee.com/introduction-stereoscopy

[5] Palandri, R.; Green, S.: Hybrid Mono Rendering in UE4 and Unity [online]. *Oculus for developers*, říjen 2016, [cit. 2022-04-21]. Dostupné z: https://developer.oculus.com/blog/hybrid-mono-rendering-in-ue4-and-unity/

[6] Epic Games: *nDisplay Overview [online]*. [cit. 2022-04-21]. Dostupné z: https://docs.unrealengine.com/4.26/en-US/WorkingWithMedia/nDisplay/Overview/

[7] Fortnite50v50: How to add a stereoscopic view in your game [online]. *Unreal Engine Forums*, únor 2020, [cit. 2022-04-21]. Dostupné z: https://forums.unrealengine.com/t/how-to-add-a-stereoscopic-view-in-your-game/138329

[8] TrueConf: *What is WebRTC? [online]*. [cit. 2022-04-21]. Dostupné z: https://trueconf.ru/webrtc.html#examples

[9] Perkins, C.; Handley, M.; Jacobson, V.: *SDP: Session Description Protocol [online]*. Network Working Group, [cit. 2022-04-21]. Dostupné z: https://www.rfc-editor.org/rfc/rfc4566.html

[10] Richardson, I. E.: *THE H.264 ADVANCED VIDEO COMPRESSION STANDARD*, kapitola 4.2.2. John Wiley and Sons, druhé vydání, 2010, str. 82.

[11] MediaCoder: H.264 profiles and levels [online]. *MediaCoder*, duben 2008, [cit. 2022-04-21]. Dostupné z: http://blog.mediacoderhq.com/h264-profiles-and-levels/

[12] Schierl, T.; Eleftheriadis, A.; Wenger, S.; aj.: *RTP Payload Format for Scalable Video Coding [online]*. Internet Engineering Task Force (IETF), [cit. 2022-04-21]. Dostupné z: https://www.rfc-editor.org/rfc/rfc6190.html

[13] Epic Games: *Getting Started with Pixel Streaming [online]*. [cit. 2022-04-21]. Dostupné z: https://docs.unrealengine.com/4.27/en-US/SharingAndReleasing/PixelStreaming/PixelStreamingIntro/

[14] VR, C. N.: Test: Oculus Link vs. Air Link vs. Virtual Desktop [online]. *AR/VR Journey: Augmented and Virtual Reality Magazine*, duben 2021, [cit. 2022-04-21]. Dostupné z: https://arvrjourney.com/test-oculus-link-vs-air-link-vs-virtual-desktop-7e0828fab0d3

[15] Yee, C.: Introducing the WebVR 1.0 API Proposal [online]. *Mozilla Hacks*, březen 2016, [cit. 2022-04-21]. Dostupné z: https://hacks.mozilla.org/2016/03/introducing-the-webvr-1-0-api-proposal/

[16] Medley, J.: Welcome to the immersive web [online]. *web.dev*, květen 2018, [cit. 2022-04-21]. Dostupné z: https://web.dev/welcome-to-immersive/

[17] Mozilla.org: *WebXR application life cycle [online]*. [cit. 2022-04-21]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/WebXR_Device_API/Lifecycle

[18] Group, K.: Khronos Releases Final WebGL 1.0 Specification [online]. *Khronos Group*, březen 2011, [cit. 2022-04-21]. Dostupné z: https://www.khronos.org/news/press/khronos-releases-final-webgl-1.0-specification

[19] mrdoob: First alpha release. Dostupné z: https://github.com/mrdoob/three.js/releases?page=14

[20] Oculus: *Monoscopic Far Field Rendering [online]*. Epic Games, [cit. 2022-04-21]. Dostupné z: https://docs.unrealengine.com/4.26/en-US/SharingAndReleasing/XRDevelopment/VR/DevelopVR/MonoFarFieldRendering/

[21] Epic Games: *Rendering to Multiple Displays with nDisplay [online]*. [cit. 2022-04-21]. Dostupné z: https://docs.unrealengine.com/4.26/en-US/WorkingWithMedia/nDisplay/

[22] Epic Games: *Customizing the Player Web Page [online]*. [cit. 2022-04-21]. Dostupné z: https://docs.unrealengine.com/4.27/en-US/SharingAndReleasing/PixelStreaming/CustomPlayer/

[23] Three.js: 360 stereo video. Dostupné z: https://threejs.org/examples/?q=webxr#webxr_vr_video

[24] Epic Games: *FGenericApplicationMessageHandler [online]*. [cit. 2022-04-21]. Dostupné z: https://docs.unrealengine.com/4.26/en-US/API/Runtime/ApplicationCore/GenericPlatform/FGenericApplicationMessageHandle-/

[25] ITU Telecommunication Standardization Sector (ITU-T): *ITU-T Recommendation H.264 (V14) (08/2021) [online]*. [cit. 2022-04-21]. Dostupné z: https://www.itu.int/rec/T-REC-H.264-202108-I

Příloha **A**

Seznam použitých zkratek

UE4 Unreal Engine 4

VR , Virtuální Realita

STUN Session Traversal Utilities for NAT

TURN Traversal Using Relay NAT

NAT Network Address Translation

IP Internet Protocol

SDP Session Description Protocol

URI Uniform Resource Identifier

ITU-T VCEG International Telecommunications Union – Video Coding Experts Group

ISO/IEC JTC1 MPEG International Organization for Standardization/International Electrotechnical Commission – Moving Picture Experts Group

CBP Constrained Baseline Profile

DVB Digital Video Broadcasting

HDTV High-definition television

URL Uniform Resource Locator

GPU graphics processing unit

CPU Central processing unit

HUD Head-Up Display

Příloha B

Obsah přiloženého CD

- `exe` ..... adresář se spustitelnou formou implementace
  - `PixelStreamingVR` ..... adresář obsahující hotovou verzi aplikace a serveru
    - `Engine`
    - `Plugins` ..... adresář obsahující dva napsané pluginy použité v projektu
    - `PST`
    - `Samples` ..... adresář obsahující uvnitř signální server
    - `Manifest_NonUFSFiles_Win64.txt`
    - `PST.exe`
    - `PST.exe-shortcut`
- `src`
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
    - `images` ..... obrázky použité v práci
    - `csn690.bst`
    - `cvut-logo-bw.pdf`
    - `cvut-logo-bw-en.pdf`
    - `FITthesis.cls`
    - `mybibliographyfile.bib`
    - `prohlaseni6.tex.txt`
    - `Sablona_BP_UTF-8.tex`
    - `zadani.pdf`
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
- `priklad-pouziti.mp4` .... video s příkladem použití pluginu s brýlemi pro virtuální realitu
- `readme.txt` ..... stručný popis obsahu CD a stručný návod ke spuštění