VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

4K VIDEO PŘEHRÁVAČ PRO VIRTUÁLNÍ REALITU

4K VIDEO PLAYER FOR VIRTUAL REALITY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **VÁCLAV CHVÍLA**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. JOZEF KOBRTEK**
SUPERVISOR

BRNO 2018

Zadání bakalářské práce/21155/2017/xchvil03

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Chvíla Václav**
Obor: Informační technologie
Téma: **4K video přehrávač pro virtuální realitu**
4K Video Player for Virtual Reality
Kategorie: Počítačová grafika

Pokyny:

1. Prozkoumejte rozhraní OpenVR pro práci s virtuální realitou a framework Qt.
2. Seznamte se s API od firmy Comprimato pro dekompresi 4K prostorového videa na GPU.
3. Navrhněte Qt aplikaci pro přehrávání 360° prostorového 4K videa v prostředí virtuální reality, navrhněte také uživatelské rozhraní použité uvnitř virtuální reality pro ovládání přehrávání.
4. Přehrávač naimplementujte.
5. Získejte zpětnou vazbu od uživatelů.
6. Vyhodnoťte dosažené výsledky a zpětnou vazbu od uživatelů.

Literatura:

- dle doporučení vedoucího

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Kobrtek Jozef, Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
L.S. 612 66 Brno, Božetěchova 2

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Práce se zabývá implementací přehrávače videa pro virtuální realitu o kvalitě 4K. Teoretická část práce pojednává o vzniku 360° videí, jejich kompresi a následného zobrazení. V návrhu aplikace bylo zvoleno řešení s knihovnami OpenGL, OpenVR, Qt a komerčního dekodéru formátu jpeg2000 od firmy Comprimato a jako zařízení pro virtuální realitu HTC Vive. V rámci části implementace práce je popis problematiky sdílené paměti s obrazem videa mezi knihovnami a samotné optimalizace přehrávače. Aplikace byla otestována z pohledu výkonu a zpětné vazby uživatelů.

Abstract

The subject of this work is the implementation of 4K video player for virtual reality. Theoretical part describes the origin of 360° videos, their compressions and rendering. In design of application was choosed solution with OpenGL, OpenVR, Qt libraries and commercial codec jpeg2000 developed by Comprimato company and for virtual reality device the HTC Vive headset. In part of implementation is analysed problem of shared video memory between libraries and discusses optimalization of the video player. Application was tested in perfomance and by users.

Klíčová slova

Virtuální realita, 4K, přehrávač videa, 360° video, jpeg2000, OpenGL, CUDA, OpenVR, sférické mapování, cube mapping, C++

Keywords

Virtual reality, 4K, video player, 360° video, jpeg2000, OpenGL, CUDA, OpenVR, sphere mapping, cube mapping, C++

Citace

CHVÍLA, Václav. *4K video přehrávač pro virtuální realitu.* Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jozef Kobrtek

4K video přehrávač pro virtuální realitu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jozefa Kobrtka. Další informace mi poskytl pracovník firmy Comprimato pan Mgr. Martin Jirman. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Václav Chvíla
16. května 2018

Poděkování

Chtěl bych poděkovat vedoucímu Ing. Jozefovi Kobrtkovi za konzultace a neocenitelné připomínky, které přispěly ke zkvalitnění této práce. Velký dík patří panu Mgr. Martinovi Jirmanovi z firmy Comprimato za jeho rady a množství cenných informací.

Obsah

1 Úvod **2**

2 Teorie **4**
2.1 Imersivní obraz 4
2.2 Standard prostorových medií 9
2.3 Užité kodeky ve VR videích 10

3 Analýza a návrh **14**
3.1 Integrace s HTC Vive 14
3.2 Vykreslení videa ve VR 14
3.3 Knihovna jpeg2000 16
3.4 Návrh aplikace 17
3.5 Optimalizace přehrávání 4K videa 17
3.6 Testování a zpětná vazba 18

4 Implementace **19**
4.1 Práce s frameworkem Qt 19
4.2 Integrace OpenVR 19
4.3 Struktura přehrávače 20

5 Testování **22**
5.1 Hodnocení FPS 22
5.2 Výhodnocení zpětné vazby 24

6 Závěr **25**

Literatura **26**

A Snímky obrazovky z běhu aplikace **29**

B Obsah přiloženého DVD **31**

Kapitola 1

Úvod

Pojem *virtuální realita* (dále jen VR) se začal skloňovat už v 60. letech 20. století, avšak o videu ve VR se pravděpodobně začalo nejvíce hovořit ve spojitosti s tvorbou virtuálních prohlídek, kdy v roce 2007 začal projekt Google Street View[1]. Dalším mezníkem v historii vývoje byl počátek roku 2015, kdy Youtube oznámil možnost libovolně nahrávat a přehrávat 360° videa, a k tomuto asi nejznámějšímu webu se streamovaným videem se o dva roky později přidává i Vimeo. Dále se již užité technologie pouze zlepšují např. širší nabídkou HMD (*angl. head-mounted display*[2]), podporou více typů projekcí a hardwarovou akcelerací dekóderů vstupních videí. Velkým trendem je přehrávání těchto videí v chytrých telefonech, které již dnes disponují dostatečnou technologii.

Základním problémem plynulého přehrávání ve VR je velká šířka přenosového pásma vstupních dat, protože uživateli není zprostředkován snímek jen s běžným zorným úhlem, který nabízí fotografie či film, ale celé jeho okolí tedy, 360° pohled v horizontálním a taktéž i ve vertikálním směru. Z tohoto důvodu je kladen nárok na to jakýmkoliv způsobem zmenšit šířku pásma, tak aby video neztratilo svoji obrazovou kvalitu a zároveň bylo spustitelné i na slabší hardwarové sestavě.

Cílem práce je implementace přehrávače ve VR s užitím kodeku jpeg2000 a jeho následné otestování. Z tohoto důvodu je práce rozčleněna do několika kapitol, které se postupně věnují teoretickým informacím, návrhu aplikace, implementace a testování. V kapitole 2 je definice videa ve VR popsaná od jeho vzniku až po samotnou reprodukci. V závěrečné podkapitole 2.3 je shrnutí stávajícího řešení aplikované v přehrávačích a na samotném konci představení kodeku jpeg2000.

V další kapitole 3 je obsažen popis použitého HMD zařízení, návrhu uživatelského rozhraní a představení navržených knihoven a jejich vzájemné integrace. Klíčovou částí je podkapitola 3.3 popisující komerční dekodér jpeg2000 od firmy Comprimato a jeho způsob integrace do struktury aplikace. V podkapitole 3.6 je stanoven návrh testování a dotazník zpětné vazby.

kapitola 4 je věnována implementaci aplikace s důrazem na důležitá místa např. podrobnější popis užití Qt frameworku a obsluhy knihovny OpenVR. V závěru je rozebráno řešení týkající se nahrávání rastrových dat z disku počítače do OpenGL, které je třeba zpřístupnit i pro aplikační rozhraní CUDA, ve kterém pracuje komerční dekodér jpeg2000, jenž je v této práci využit.

[1] https://www.google.com/streetview/

[2] Taktéž *headset* pro virtuální realitu.

V poslední kapitole 5 se zabýváme testováním a jejich následnému vyhodnocení. Detailně je rozebrán výsledek testů a okomentována je i zpětná vazba uživatelů.

Kapitola 2

Teorie

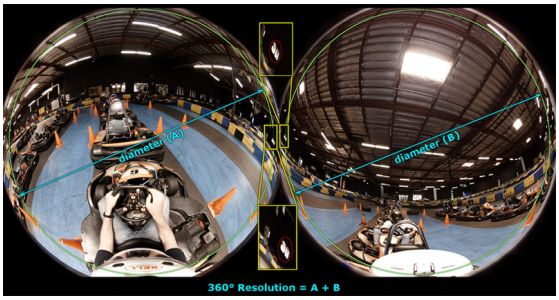
Cílem úvodních podkapitol je popsat vznik panoramatického nebo 360° obrazu a jejich jednotlivé způsoby projekce do roviny. Důraz je kladen na samotná 360° videa, v jaké podobě jsou dnes zakódována a jakým způsobem se s nimi pracuje s ohledem na metadata, která jsou součástí video kontejnerů [5]. V závěru je rozebrána teorie kodeku jpeg2000, protože je součástí návrhu a implementace aplikace.

2.1 Imersivní obraz

Tak se nazývají obrazová data, která se užívají jako vstupní materiál pro virtuální hloubkové zobrazení. Vzniká zachycením širšího zorného pole například spojením několika fotografií a následné nezbytné projekce do roviny, protože ve výsledku se taktéž jedná o rastrová data. Do této kategorie spadá např. panoramatická fotografie, snímek vyfocený technologii rybího oka (*angl. fisheye lens*) a taktéž jakýkoliv více stupňový snímek, ať už do horizontálního či vertikálního směru [11].

Problematika spojování fotografií spočívá v hledání společných znaků a jejich vzájemného překrývání s nevyhnutelnou deformací obrazu, ale taktéž i v úpravě odstínu barev a jejich jasu. Tato úprava je nezbytně nutná, neboť zachycené fotografie vznikají buď v jiném čase nebo pod jiným úhlem osvětlení, což ovlivňuje celou barevnou složku [2].

Jako praktická ukázka, ze které vzniká přímo 360° video poslouží například GoPro Fusion[1] (Obrázek 2.1b), které spojuje dva 180° snímky z kamerových snímačů, které jsou umístěny „zády“ k sobě a každý tak snímá svou část polokoule [17].



(a) Snímky dvou kamer typu *fisheye lens*. Žluté rámečky ukazují společné body pro sjednocení obrazu. (Převzato z [17])

(b) Zařízení GoPro Fusion z obou stran[2]

Obrázek 2.1: Ukázka vzniku 360° videa pomocí technologie rybí oko.

Dalším příkladem může být firma Airpano[3], která produkuje videa ve VR propojením 4 a více fotoaparátů s nastavením pevně dané šířky záběru (Obrázek 2.2), které společně vytvoří sférické zobrazení [25].

Obrázek 2.2: Zařízení firmy Airpano pro snímání VR videa. (Převzato z [25])

2.1.1 Typy projekcí

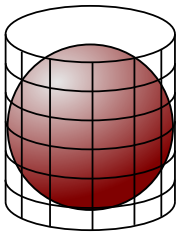
Tyto spojené snímky, zachycující okolí v daném horizontálním i vertikálním směru, lze vybranou projekcí promítnout do roviny. Uvedeme zde několik základních projekcí, ale také i ty, které jsou ve spojení s 360° videi nejvíce diskutovány [3].

- *Přímočará projekce (angl. rectilinear projection)*

  Tato projekce zplošťuje perspektivu do roviny a díky tomu dokáže zachovat všechny přímočaré linky bez zakřivení. Bohužel její hlavní nevýhodou je právě ztráta perspektivy, která vede k zakřivení objektů na hranách snímku. Proto není doporučena k užívání s více než 120° zorným úhlem [14].

- *Ekvidistantní válcová projekce (angl. equirectangular projection)*

  Nebo taktéž sférická projekce či pouze ekvidistantní. V geografii se používá jako jedno ze základních mapování Země na plochu, které není plochojevná projekce ani konformní zobrazení[4]. Mapuje zeměpisnou šířku (rovnoběžky) a zeměpisnou délku (poledníky) jako souřadnice koule přímo na rovinu obdélníkové mřížky viz obrázek 2.3 [4].



Obrázek 2.3: *Equirectangular projection* – obraz vzniká projekcí koule na vnitřní stěnu pláště válce (bez užití podstavců).

[1] https://shop.gopro.com/EMEA/cameras/fusion/CHDHZ-103.html
[2] Převzato z: https://www.provideocoalition.com/first-look-gopro-fusion-360-cam/
[3] http://www.airpano.com/
[4] Spojité zobrazení, které zachovává úhly.

Definice ekvidistantní projekce sférických souřadnic do souřadnic plochy [19]:

$$x = (\lambda - \lambda_0) * cos(\varphi_1) \tag{2.1}$$

$$y = \varphi \tag{2.2}$$

Kde:

- λ je zeměpisná délka
- φ je zeměpisná šířka
- φ_1 je parametr rovnoběžek (severně a jižně od rovníku), který určuje skutečný rozsah správné projekce
- λ_0 je centrální poledník mapování
- x je horizontální souřadnice projektovaného bodu v ploše
- y je vertikální souřadnice projektovaného bodu v ploše

Výsledkem projekce je plně 360° horizontální panorama a 180° vertikální zorné pole, které umožňuje pozorovateli podívat se libovolným směrem. Při zobrazení v rovině můžeme pozorovat zakřivení horizontálních přímek, kdežto vertikální zůstávají zachovány. Důležitým poznatkem je, že pomyslné póly koule jsou roztaženy po celé délce a dochází tak ke značné deformaci jejich obrazu (Obrázek 2.4) [14].

Obrázek 2.4: Ukázka ekvidistantní projekce do roviny[5]

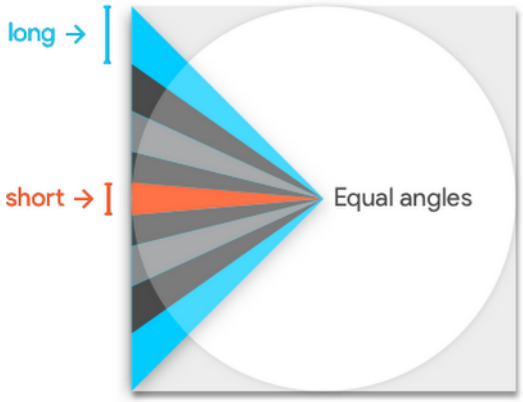
- *Válcová projekce (angl. cylindrical projection)*

  Je podobná ekvidistantní projekci, ale s tím rozdílem, že nemapuje plný 180° vertikální úhel pohledu. Důsledkem je, že vertikálně natahuje objekty čím blíže jsou severního nebo jižního pólu až do nekonečných hodnot. Z tohoto důvodu není vhodná pro snímky s širokým úhlem vertikálního zorného pole [14].

- *Projekce na krychli (angl. cubemap projection)*

  Jedná se o radiální projekci koule do krychle, kde jejím výsledkem je 6 snímků neboli stěn tvořící krychli (Obrázek 2.5).

[5]Převzato z: https://commons.wikimedia.org/wiki/File:Rheingauer_Dom,_Geisenheim,_360_Panorama_(Equirectangular_projection).jpg



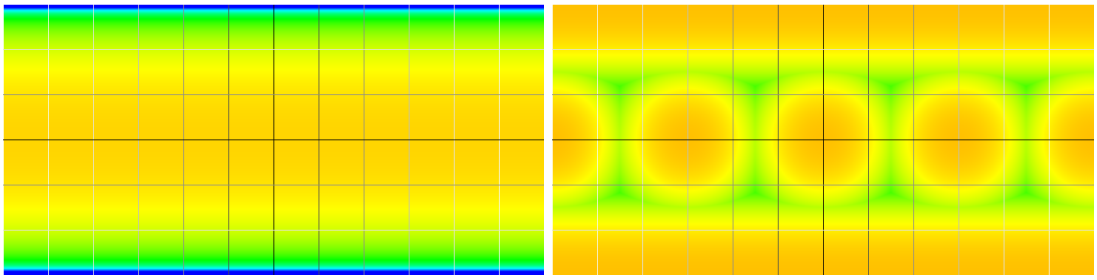
Obrázek 2.5: Radiální projekce cube mapy – rovnoměrně promítnuté úhly vrhají na stěnu krychle nerovnoměrné rozložení rastrových bodů, které znázorňuje modrá a oranžová úsečka (Převzato z: [1])

- *Pyramidová projekce (angl. pyramid projection)*

  Pyramidová projekce byla navržena Facebokem s účelem snížit šířku přenosového pásma pro video ve VR. Projekce vykresluje video do 30 různých výstupních obrazů, kde základ pyramidy je v plném rozlišení ze směru pozorovatele, zatímco boční stěny postupně ztrácí kvalitu rozlišení. Společnost facebook deklaruje snížení velikosti šířky pásma přenosu internetem až o 80%. Nevýhodou tohoto řešení je větší počet výstupních obrazů, které je potřeba vykreslit a uložit [10].

- *Equi-angular projection – EAC*

  Vznik projekce byl uskutečněn díky studii, na které se podílely firmy Youtube a Daydream. Vývoj projekce byl oblivněn výsledky ze *saturačních map*, které ukazují poměr hustot pixelů vstupního obrazu ku hustotám zobrazených pixelů po projekci (Obrázek 2.6) [24].



(a) Ekvidistantní válcová projekce (b) Projekce na krychli

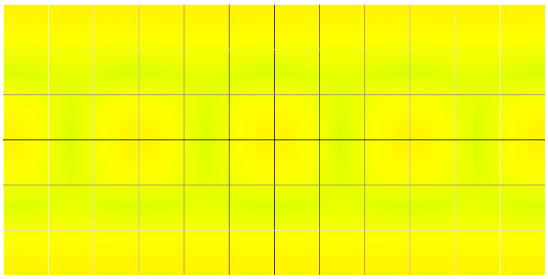
Obrázek 2.6: Porovnání saturačních map (Převzato z [24])

Barevné spektrum saturační mapy začíná na červené (nízký poměr hodnot) a dále pokračuje k oranžové, žluté, zelené až nakonec modré (vysoký poměr hodnot). Zelená indikuje optimální poměr hustoty pixelů v hodnotě blízké 1:1. Žlutá a oranžová barva znázorňuje nedostatečnou hustotu (příliš málo pixelů vstupního obrazu ku možných

zobrazitelných pixelů) a naopak modrá poukazuje na plýtvání zdrojem obrazu (příliš mnoho vstupních pixelů na dostupný počet zobrazitelných pixelů). Ideální projekce vyžaduje takovou saturační mapu, kde její barva bude téměř jednolitá, takže v jiném rozlišení videa by se jevila jako plně zelená [1].

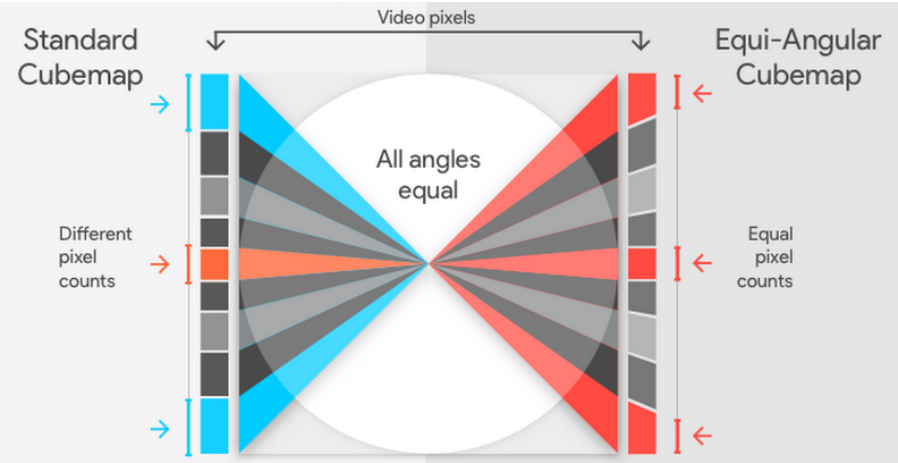
V saturační mapě užité při ekvidistantní projekci jsou oba póly modré, což indikuje zbytečné pixely a horizont je oranžový, což značí nedostatečný počet pixelů. Oproti tomu cubemap má zelená místa v oblastech horizontu a jejím dalším pozitivem je, že zcela postrádá modrá místa s nadbytečnými vstupními pixely. Přesto cubemap ve svém výsledku obsahuje i oranžové plochy, které se nacházejí v totožné oblasti spolu se zelenými. Tento jev je způsoben tím, že rohy stěn krychle se lépe vzorkují a mají tedy větší množství pixelů než jejich středy. Na základě tohoto porovnání dvou typů projekcí se Youtube rozhodl použít cubemap jako výchozí formát k dalším optimalizacím (Obrázek 2.6) [24].

Díky equi-angular projekci Youtube dosáhl značného zlepšení výsledku v saturační mapě, kde její barvy jsou více uniformní než v předchozích případech (Obrázek 2.7).



Obrázek 2.7: Saturační mapa – Equi-angular Cubemap (EAC) (Převzato z [24])

Oproti tradiční cubemap, která pod rovnoměrným úhlem distribuuje pixely ve stejnoměrné hustotě na plochu krychle, tak Equi-Angular Cubemap rovnoměrně distribuuje pixely na změnu úhlu. Zcela názorné porovnání můžeme vidět na obrázku 2.8, kde je u obou metod užito radiálních „paprsků“ s totožným rozestupem, které promítají obraz koule na stěnu krychle, avšak u EAC dochází k deformaci, která zajišťuje co nejrovnoměrnější hustotu promítnutých pixelů na stěnu [1].



Obrázek 2.8: Porovnání cubemap a EAC projekce z pohledu hustoty pixelů (Převzato z [1])

Youtube od zveřejnění této optimalizace prohlásil, že každé nahrané 360° video převádí do tohoto typu projekce [24].

2.2 Standard prostorových medií

Na základě spuštění služby s možností nahrávat a přehrávat videa ve VR na webovém streamu Youtube tak Google vydává pracovní verzi standardu s názvem *Spatial Media*[6], který má popisovat obsažená metadata ve vybraných video kontejnerech. Díky službě Google Groups je tento standard plně otevřen veřejné diskusi[7], která má za cíl vytvořit co nejšířeji přijatelný standard [23].

Standard popisuje metadata, která mají upřesnit informace o typu prostorového zvuku a obrazu. Obraz videa může být zprostředkován různými typy projekcí, jenž byly zmíněny v kapitole 2.1.1, a krom typu projekce je zde popsán i způsob vykreslení [5].

Aktuální část standardu zabývající se obrazem, se nazývá *Spherical Video V2 RFC* a podporuje multimediální kontejnery MP4 (ISOBMFF) a WebM (Matroska) [5]. První zmíněný formát MP4 podporuje kodeky zejména řady MPEG, protože je součástí specifikace MPEG-4[8] a druhý kontejner jménem WebM, který je založen na profilu Matroska, podporuje jen malé spektrum kodeků (pouze VP8, VP9 a AV1)[9]. Mimo některé z uvedených typů projekcí v předchozích kapitolách, obsahuje standard i „nezávislou projekci“ (*z angl. Projection Independent Mesh*), která umožňuje definovat téměř libovolnou projekci [5].

Zde je ukázka několika podstatných metadat obsažených v kontejneru MP4:

- *Stereoscopic 3D Video Box (st3d)*

  Parametrem `stereo_mode` typu `int` lze rozlišit nejen monoskopický a stereoskopický pohled kamery, ale například i specifické rozložení stereoskopických obrazů ve snímku videa.

- *Projection Header Box (prhd)*

  Popisuje prostorovou rotaci výstupní projekce.

- *Cubemap Projection Box (cbmp)*

  Specifikuje video stopu pouze jako cubemap projekci. Parametr identifikovaný jako `layout` určuje rozložení stěn krychle v ploše, kde je standardizována pouze jedna hodnota (Tabulka 2.1).

vpravo	vlevo	nahoře
dole	vpředu	vzadu

Tabulka 2.1: Rozložení stěn krychle – popisky odpovídají směr stěny z pohledu pozorovatele uvnitř krychle

Další parametr `padding` nastavuje počet pixelů, který odděluje jednotlivé hrany stěn krychle. Toto odsazení se používá v případě, kdy užitý kodek by svojí kompresí deformoval hrany mezi jednotlivými stěnami krychle, které spolu nesousedí a nenavazují tak přirozeně na sebe. Ve výsledném zobrazení by tak vznikl viditelný defekt.

[6] https://github.com/google/spatial-media
[7] Diskuse dostupná zde: https://groups.google.com/forum/#!forum/spatial-media-discuss
[8] Specifikace MP4: https://www.iso.org/standard/38538.html
[9] Viz: https://www.webmproject.org/about/

- *Equirectangular Projection Box (equi)*

  Definuje video stopu jako ekvidistantní typ projekce. Přítomné 4 parametry pouze určují ořez každého snímku z každé strany zvlášť.

- *Mesh Projection Box (mshp)*

  Definuje zmíněnou *nezávislou projekci*. Udává, že video stopa se bude rozkládat do konkrétní 3D topologie sítě objektu. První parametry popisují jaké bude bitové kódování dalších dodatečných informací, a posledním parametrem je objekt `Mesh Box (mesh)`, který popisuje konkrétní body (*vertices*) a souřadnice textury.

Klíčovým poznatkem standardu je, že díky *Projection Independent Mesh* je možno nadefinovat zcela vlastní projekci a zobrazit ji bez předchozí znalosti například jejího matematického pozadí a dále dovoluje zobrazovat i takové snímky, které nejsou projekcí plného 180° vertikální zorného úhlu. Při reprodukci se tedy jedná už o běžný rendering např. v OpenGL s mapováním textury na těleso [24].

2.3 Užité kodeky ve VR videích

V následujícím textu vedeme úzký profil kodeků, které jsou přímo podporovány těmi kontejnery, které mají standardizovaná metadata, jenž popisují video ve VR. Úmyslem této podkapitoly bude reprezentace stávajícího řešení komprese video obrazu ve VR, jehož záměrem bude vyzdvihnout pozitiva i negativa, kvalitu videí ovlivňují. Jedná se o kodeky H.264, H.265 a VP9.

Již v podkapitole 2.1 bylo zmíněno, že cílem implementace této práce je použití kodeku jpeg2000. V poslední části bude tento kodek představen s ohledem na jeho zcela unikátní vlastnosti, které by mohly ovlivnit video ve VR.

2.3.1 Kodek H.264

H.264 neboli MPEG-4 AVC je kodek, se kterým se lze setkat v široce dostupných službách jako je např. Youtube[10] a Vimeo[11], které obě nabízejí přehrávání 360° videí, ale i v jiných odvětvích, jako jsou přenosná média (DVD, Blue-Ray) nebo kabelové či satelitní televizní vysílání (DVB, HDTV) [12]. Toto široké využití bylo způsobeno několika vlastnostmi např. lepším kompresním poměrem oproti předchozím kodekům v řadě MPEG, rozdělením kodeku do různých *profilů*, které jsou zacíleny na specifické užití, a dále oproti předchozím standardům dobře vyvážená efektivita kódování a komplexnost implementace [21].

Kodek pracuje se třemi typy snímků, kde každý má mezi sebou specifický vztah, díky jemuž lze zvyšovat kompresní poměr. Jedná se o tyto typy:

- I-frame *(Intra-coded picture)* – klíčový snímek

  Obsahuje obraz, který je uložen v obdobné podobě jako je obraz komprimovaný kodekem jpeg nebo BMP. V porovnání se snímky typu P-frame a B-frame, které obsahují pouze část informací o obraze (pouze změny mezi jednotlivými snímky), je velikost dat v I-frame mnohonásobně větší.

[10] Specifikace Youtube: https://support.google.com/youtube/answer/4603579?hl=en
[11] https://vimeo.com/help/compression

- P-frame *(Predicted picture)* – predikát

  Drží pouze ten typ informace, který je rozdílem od předchozího snímku. Například ve scéně, kde se pohybuje pouze auto a za ním je statické pozadí, postačí, když bude zakódován pouze pohyb auta. Kodér tak nepotřebuje ukládat neměnné pixely pozadí do P-snímku z důvodu ušetření místa. P-frame je taktéž znám pod *delta frame*.

- B-frame *(Bidirectional predicted picture)* – obousměrný predikát

  Šetří mnohem více nadbytečných dat díky rozdílům mezi aktuálním snímkem, předchozím a následujícím.

Sady těchto typů snímků mají mezi s sebou pravidla, podle kterých se mohou za sebou nejen vyskytovat, ale mají i dané svoje vlastní pořadí při dekódovaní. Identickým skupinám těchto snímků se říká GOP (*Group Of Pictures*). [9]

Komprese je tak založená na porovnávání obrazu v následujících i předchozích snímcích. Toto porovnání vzniká díky rozdělení obrazu na čtvercové skupiny sousedících pixelů, které jsou nazvány *macroblocks*, o typické velikost 16x16 nebo 8x8 pixelů. Tyto skupiny pixelů nebo bloků jsou porovnávány mezi dvěma rozdílnými snímky a komprese videa je založená pouze na jejich vzájemných rozdílech. V oblastech, kde dochází k větší proměně obrazu, musí být zakódováno větší množství dat, které se změnilo. Díky tomuto dochází k jevu, že video má variabilní bitrate, jelikož okamžité změny scén (např. výbuchy nebo rychlé střihy) navyšují počet změn v mezisnímcích. Tento kodek taktéž disponuje mnoha různými kompresními profily a levely. Profilem rozumíme kompresní poměr, kvalitu vzorkování, typ přenosové rychlosti (variabilní nebo konstantní) a jiné parametry ovlivňující rychlost a kvalitu kódování a dekódování. Level je omezení, které způsobují parametry jako maximální rozlišení a datové toky [21].

2.3.2 Kodek H.265

H.265 neboli HEVC (*High Efficiency Video Coding*) je aktuálně nejnovějším standardem video kodeku ITU-T *Video Coding Experts Group* a ISO/IEC *Moving Picture Experts Group* a je tedy dalším kodekem v řadě MPEG. V porovnání s předchozím standardem (H.264) je výsledkem zlepšení kompresního poměru, přesněji snížení až o 50% z podílu šířky pásma při rovnocenné kvalitě obrazu. Ačkoliv kodek H.265 vychází z totožného konceptu jako jeho předchůdce, není zpětně kompatibilní [20]. Příčinou, která vedl k vývoji tohoto kodeku, byla zvyšující se dostupnost videa s větším rozlišením nabízené ve všech stávajících službách, kde předchozí standard svými kvalitami nedostačoval. Jednalo se zejména o větší zátěž z pohledu přenosu dat na internetu, kde služby stále více cílily na připojení tabletů a mobilních zařízení. HEVC byl tedy navrhnut tak, aby došel zastoupení ve všech službách, kde je užit jeho předchůdce, avšak se záměrem umožnit zakódovat větší rozlišení a zefektivnit tento proces na paralelních architekturách [20].

Rozdíl oproti H.264 je např. v rozšíření velikosti *macroblock* z maximální velikosti 16x16 až do 64x64, je taktéž vylepšena i segmentace obrazu po blocích a dále rozšíření z 8 na 33 směrových módů *intra* predikce [20].

2.3.3 Kodek VP9

Ze stejných důvodů, ze kterých vznikl kodek HEVC, firma Google zahájila projekt WebM, jehož cílem je vytvořit otevřený – *royalty free*[12] – webový media formát. Tento formát definuje nejenom strukturu stejnojmenného kontejneru, ale také video a audio formáty. Speciálně se zaměřuje na streamované audio (*open-source* kodek Vorbis) a video komprimované *open-source* kodekem VP8 a zejména novějším VP9 [16].

Kodek VP9 je jako jeho konkurenti taktéž blokově kódován. Užívá *super-block* s velikostí až 64x64, které mohou být rekurzivně dekomponovány až do 4x4 bloků pixelů. Taktéž lze tyto bloky kódovat např. v predikčním módu či s pohybovým vektorem, který má 8 směrových módů podobně jako H.264 [15].

Velká část výhod, které přináší VP9 proti VP8 je přirozeným vývojem jeho nové generace, ale v důsledku je cílem sjednotit jejich práci a dosáhnout co nejefektivnější komprese [16].

2.3.4 Kodek jpeg2000[13]

Kodek je znám i pod *j2k* a jeho standard byl vydán za účelem zlepšit kompresní poměr a nabídnout nové vlastnosti, které nabízí dekompoziční metoda obrazu postavená na diskrétní vlnkové transformaci (dále jen DWT) oproti jeho předchůdci kodeku JPEG, který užívá diskrétní kosinové transformace [13].

Jpeg2000 je užíván ve spojitosti se zpracováním statického obrazu a zejména fotografiemi s nadstandardními rozlišeními dosahujícími více než 8K. Do jeho užití spadá práce s velkoformátovými rastrovými daty, například vesmírné snímky či letecké mapy. Avšak nachází využití i ve filmovém průmyslu a to zejména jako formát pro práci ve střižnách a nejvíce jako standardní kodek formátu D-Cinema pro komerční projekci filmů [8].

Mezi největší přednosti patří jeho jedinečné vlastnosti jako je DWT, možnost ztrátové i bezeztrátové komprese, vícerozměrná reprezentace, progresivní přenos pixelů a přesnost rozlišení (známé též pod *progressive decoding* a *signal-to-noise ratio (SNR)*) a široká podpora bitové hloubky včetně barevných prostorů [18]. Tyto vlastnosti jsou vhodné pro dynamickou práci s obrazem například uvedené letecké mapy, kdy uživatel prohlížením map často mění měřítko i kvalitu zobrazení a taktéž i u videí ve VR by tato vlastnost mohla být velkou výhodou, jelikož uživatel nikdy nepojme svým pohledem celou plochu svého okolí a taktéž nepotřebuje maximální kvalitu bitové hloubky v okrajových oblastech zorného pole. Můžeme tedy mluvit o dekódování pouhého výřezu vstupního obrazu a predikce jeho dalšího posunu způsobené interakcí s HMD [3].

Pokud kodek užíváme ve spojitosti s videem, jeho hlavním negativem je kompresní poměr, což může být u videí ve VR s nadměrně vysokým rozlišením a kvalitou velký problém. Ačkoliv z tabulky 2.2 vyplývá (při srovnání s intra-frame kódováním HEVC), že se jedná o malý rozdíl, tak hlavním důvodem špatného poměru komprese je to, že každý snímek je zakódován zvlášť bez znalosti několika následujících či předešlých. Na druhou stranu je tento způsob kódování naprosto bezesporně výhodný v případě, že se jedná o streamovaný zdroj videa, ve kterém snadno dochází ke ztrátě dat. Pokud dojde k porušení informací, tak zpravidla dojde k *frame dropping* (zahazování jednotlivých snímků), které je z pohledu lidského vnímání zanedbatelnější než ztráty v běžných videových kodecích, kde jsou delší časové úseky ovlivněny znatelnou obrazovou deformací.

[12] Právo užívat produkt nebo duševní vlastnictví, které podléhá autorským právům, bez nutnosti platit licenční poplatky.

[13] Oficiální web kodeku jpeg2000: https://jpeg.org/jpeg2000/

Standardně kódovaný statický obraz (testovací metoda)	Redukce průměrné přenosové rychlosti HEVC v porovnání s	
	JPEG 2000 4:2:0	JPEG
HEVC (PSNR)	-20.26%	-61.63%
HEVC (MOS)	-30.96%	-43.10%

Tabulka 2.2: Srovnání kvality komprese na základě totožných hodnot PSNR a MOS. Při HEVC bylo užito intra-frame kódování [7].

Kapitola 3

Analýza a návrh

V kapitole budou popsány vybrané HMD, knihovny pro dekódovaní kodeku jpeg2000 a další knihovny které byly potřeba při implementaci přehrávače. Součástí návrhu je uživatelské rozhraní, zpracování vstupních dat, prvky optimalizace a v jeho závěru jsou zmíněny podmínky testovaní.

3.1 Integrace s HTC Vive

V zadání práce je splnit úkol implementovat přehrávač ve VR. Uživatelský návrh aplikace je koncipován tak, aby byla možnost přehrávač přenést na jakékoliv zařízení podporující VR. Jedno z dostupných API pro spolupráci s HMD je OpenVR[1], které tuto přenositelnost mezi jednotlivými zařízeními umožňuje. Pro splnění úkolu této práce se naskytla příležitost pracovat s brýlemi HTC Vive, které disponují bezdrátovými ovladačemi a *trackovacími* stanicemi, které dokáží získávat přesné souřadnice všech ovladačů i HMD. Na tomto zařízení lze vyvíjet i pod pokročilejšími nástroji jako jsou např. *Unity* nebo *Unreal Engine*, avšak při implementaci této práce by rozhraní těchto softwarů mohlo ztížit programování přehrávače na nižší úrovni kódu.

HTC Vive je první verze řady tohoto typu zařízení, které vyvinula firma HTC[2]. HMD disponuje displejem s maximálním rozlišením 2160x1200 pixelů (1080x1200 pro jedno oko), jeho obnovovací frekvence je 90Hz a zorné pole nabývá hodnoty 110°.

Ideálním rozlišením 360° videa, dle uvedených parametrů displeje (šířky zorného pole a rozlišení), by v případě ekvidistantní projekce ve své horizontální ose bylo přibližně 7 069 pixelů. Lze tedy konstatovat, že tento typ brýlí bude schopen zobrazit vstupní 4K[3] video ve více než nadstandardní kvalitě, tedy že počet zobrazitelných bodů ve své horizontále téměř 2x převyšuje vstupní video a ve vertikále dosahuje přibližně totožných hodnot.

3.2 Vykreslení videa ve VR

Vykreslení lze zjednodušit do dvou kroků, kde v prvním se zpracovává samotná 3D scéna a v dalším dojde k úpravě, která vede k realistickému zobrazení přímo v brýlích uživatele. V podkapitolách jsou podrobněji uvedeny knihovny, které tyto dílčí kroky umožňují.

[1] https://github.com/ValveSoftware/openvr
[2] https://www.vive.com/eu/
[3] 3840x1920 pixelů

3.2.1 OpenGL

Pro práci k vykreslování scény ve VR byla zvolena knihovna OpenGL, která nabízí plnou přenositelnost mezi jinými operačními systémy. Tato knihovna má taktéž velkou dostupnost podpory na různých hardwarových grafických adaptérech a nabízí širokou škálu rozmanitých speciálních efektů a vizuálních funkcí [6].

3.2.2 OpenVR[4]

API této knihovny zcela plně obstarává práci s brýlemi a umožňuje programátorovi pracovat i s ovladači a zjednodušit celý proces přípravy a zpracování obrazu do brýlí, a to bez znalosti či specifikace hardware HMD nebo jeho výrobce. API může být aktualizováno zcela nezávisle na produktu, v kterém je užito, a zajistit tak podporu například dalšího hardwaru. OpenVR je implementováno jako třídní rozhraní čistě virtuálních funkcí v jazyce C++. Při inicializaci systému OpenVR je aplikaci vráceno rozhraní, které odpovídá HMD SDK, které je v ní užito. OpenVR dále deklaruje, že další nové verze SDK budou zpětně plně kompatibilní i s dalším vývojem hardware či software.

OpenVR nabízí programátorovi využít své API v několika režimech, které jsou k dispozici skrze různá aplikační rozhraní:

- *IVRSystem*

  Hlavní rozhraní pro zobrazení na displeji (zkreslení vstupního obrazu), trackování, ovládání a přístup k událostem celého zařízení.

- *IVRChaperone*

  Umožňuje přístup k *chaperone bounds* – virtuálně stanovené hranice ohraničující pohyb uživatele v reálném světě.

- *IVRCompositor*

  Poskytuje aplikaci vykreslení 3D obsahu skrze VR.

- *IVROverlay*

  Poskytuje aplikaci vykreslení 2D obsahu skrze VR.

- *IVRRenderModels*

  Poskytuje aplikaci přístup k vykreslení modelů.

- *IVRScreenshots*

  Umožňuje aplikaci zažádat nebo dodat snímek obrazovky.

Z těchto typů rozhraní bylo vybráno *IVRCompositor*, které zprostředkovává vykreslení celé projekce VR videa a *IVRSystem*, které umožňuje získání přesné polohy zařízení v prostoru. Polohu zařízení lze získat ve dvou režimech. Prvním je *TrackingUniverseSeated*, které zajišťuje polohu ve vztahu k nastavené „nulové“ pozici a druhé *TrackingUniverseStanding*, které jsou ve vztahu k hranicím VR vytyčených uživatelem (dané maximálním prostorem trackovacích zařízení). Byla vybrána druhá varianta, která umožňuje uživateli využít celý prostor určený k pohybu mezi snímači pro větší komfort přehrávání videa.

[4] Čerpáno z dokumentace: https://github.com/ValveSoftware/openvr/wiki/API-Documentation

3.3 Knihovna jpeg2000

Cílem návrhu přehrávání videa je integrovat implementaci kodeku jpeg2000 od firmy Comprimato. Kodek je znám nejen v této implementaci, ale existují i jiné alternativy, které taktéž podporují svůj výpočet na grafické kartě. Obecný přehled jiných implementací se nachází v tabulce 3.1.

Název (firmy nebo aplikace)	*basic*		*advanced*		Jazyk	Licence
	čtení	zápis	čtení	zápis		
Comprimato	ano	ano	ano	ano	C, C++	proprietární
ERDAS ECW JPEG2000 SDK	ano	ano	?	?	C, C++	proprietární
Fastvideo SDK	ano	ano	ano	ano	C, C++	proprietární
FFmpeg	ano	ano	?	?	C	LGPL
Grok	ano	ano	ano	ano	C, C++	AGPL
J2K-Codec	ano	ne	ano	ne	C++	proprietární
JasPer	ano	ano	ne	ne	C	MIT Licence
Kakadu	ano	ano	ano	ano	C++	proprietární
LEADTOOLS	ano	ano	ano	ano	C++, .NET	proprietární
OpenJPEG	ano	ano	ano	ano	C	BSD
BOI codec	ano	ano	ne	ne	Java	BOI Licence

Tabulka 3.1: Několik příkladů knihoven implementace jpeg2000[5]

K řešení této práce byla již ze zadání zvolena implementace firmou Comprimato, a to z důvodu jejich unikátních výsledků díky využití rozhraní CUDA. Už od samého počátku byla navázána bližší vzájemná spolupráce při řešení jednotlivých problémů.

Knihovna od firmy Comprimato je produktem, který nese název UltraJ2K[6] a byl firmou vyvinut především pro zpracování fotografii, obrazových dat ve zdravotnictví a následně se svojí aplikací zaměřil na video obraz. Aktuálně je UltraJ2K implementován do produktu UltraPix, který se užívá při zpracování videa v profesionálních editorech a do transkodéru, který zpracovává živé videostreamy profesionálního plošného vysílání.

Implementace knihovny UltraJ2K je založená na paralelním řešení s využitím zejména CUDA procesorů. Krom výpočtu na CUDA jádrech nabízí i alternativu v podobě CPU a OpenCL implementace, které ale nedosahují tak dobrých výsledků jako právě CUDA. Z důvodu úspěšnosti CUDA optimalizace bylo právě toto řešení zakomponováno do návrhu implementace přehrávače. Z důvodu užití CUDA bylo nutností integrovat jedno z jeho dostupných API (*driver* nebo *runtime*). Na základě rozboru rozhraní produktu UltraJ2K, bylo zvoleno *runtime*, jelikož v dané problematice není zapotřebí užití např. komplexnější práce s CUDA, vlastních kernelů, správa kontextu a nezávislost na jazyku kódu aplikace.

Struktura práce s rozhraním UltraJ2K se dělí na inicializaci kodéru nebo dekodéru a stanovením formátu vstupních a výstupních dat. Proces dekódování nebo kódování je ovlivněn způsobem „podání“ nebo „odebrání“ dat. Přesněji řečeno, jedná se o to, zdali je správa paměti vstupně/výstupních obrazových dat na programátorovi nebo na interním mechanismu knihovny. V našem případě bylo navrhnuto zajištění vlastní správy paměti z důvodu potřeby vyrovnávací paměti pro plynulé přehrávání videa. Jednotlivé části paměti

[5] Převzato z: https://en.wikipedia.org/wiki/JPEG_2000

[6] https://comprimato.com/products/comprimato-jpeg2000/

určené pro dekódované snímky jsou v návrhu přiřazeny dekodéru a paměť pro vstupní zakódované snímky je z pohledu grafické paměti v režii knihovny.

3.4 Návrh aplikace

Celý program byl rozčleněn do tří částí: práce s uživatelským rozhraní, zpracování vstupních dat a jejich dekódování a obsluha API k HMD pro zobrazení ve VR. V této podkapitole budou jednotlivé návrhy těchto částí podrobně rozebrány.

3.4.1 Grafické uživatelské rozhraní

Ovládání aplikace bylo ve svém návrhu rozděleno do dvou variant, kde první nabízí přistupovat k aplikaci běžným způsobem, který je znám z desktopových počítačů, tedy ovládaní klávesnicí a myší. Další variantou je ovládat přehrávač pomocí ovladačů, které jsou dostupné ke zmíněnému zařízení HTC Vive, díky kterým bude možnost ovládat základní funkce přehrávače videa.

Samotné video bude přehráváno v brýlích a taktéž z pozice levého oka v okně aplikace, které je primárně určeno pro grafické ovládání včetně svého kontextového menu.

Pro implementaci byl vybrán framework Qt[7], který zejména nabízí plnou přenositelnost mezi jinými operačními systémy a disponuje širokou škálou možností uživatelského rozhraní, k čemuž je primárně určen. Framework je schopen nabídnout uživateli běžnou interakci s operačním systémem (prohlížeč souborů, zachycení myši nebo klávesových zkratek a vznik samotného okna aplikace). Zásadní výhodou, která měla dopad pro výběr tohoto frameworku je plná dispozice OpenGL API. Jako nemalou výhodou bylo bráno i to, že je tento framework užit jako jeden z oficiálních referenčních ukázek kódu při užití OpenVR.

3.4.2 Zpracování vstupních dat

Formát vstupních dat byl z pohledu funkce knihovny zjednodušen na sekvenci snímků (jednotlivých souborů) ve formátu jpeg2000. Pro vykreslení prostorového videa byla zvolena ekvidistantní projekce z důvodu, že jiné formáty jsou buď proprietárními a nebo neexistuje jejich exaktní standardizovaný způsob vykreslení.

3.5 Optimalizace přehrávání 4K videa

Základním kamenem optimalizace bude dvouvláknové řešení, kde hlavní vlákno čeká na dekódované snímky z grafického adaptéru zprostředkovávané knihovnou UltraJ2K, a vedlejší obstará nahrávání snímků z disku počítače do totožné instance dané knihovny.

Dalším prvkem, který umožní plynulé přehrávání, je saturace dekodéru dostatečným počtem snímků. Z tohoto důvodu je potřeba vytvořit vyrovnávací paměť určenou pro snímky, které byly čerstvě dekódovány. Tyto snímky budou uloženy v CUDA paměti a připraveny pro vykreslení v OpenGL. Z tohoto důvodu bude potřeba převést rastrová data do textury, jenž se namapuje na síť objektu koule. Pro spolupráci CUDA a OpenGL se nabízí *CUDA interoperability*[8], které umožňuje mapování paměťových zdrojů z OpenGL do CUDA.

[7] Dokumentace API: http://doc.qt.io/

[8] https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__INTEROP.html

3.6 Testování a zpětná vazba

Cílem testování bude zjistit kvalitu přehrávaného videa. Hlavním ukazatelem této kvality bude FPS (*angl. frames per second*) neboli počet snímků za vteřinu, subjektivní hodnocení uživatelů obrazové kvality videa a dále hodnocení samotné funkcionality přehrávače.

3.6.1 Návrh dotazníku

Testování proběhne ve dvou fázích. V první fázi bude uživateli spuštěn přehrávač a jeho úkolem bude nahlas komentovat jeho postup práce s aplikací a na závěr subjektivně ohodnotí celkové ovládání aplikace.

V druhé fázi bude hodnocena kvalita obrazu, která je rozdělena na spuštění videa a zobrazení jednoho statického snímku vyňaté ze sekvence totožného videa. Uživatel v obou případech zhodnotí kvalitu obrazu odpovězení na tyto otázky:

- Bylo video plynulé? (Odpovídá pouze v případě videa: ano – ne)
- Odpovídal pohyb vaší hlavy okamžitému pohybu ve scéně? (Odpověď: ano – ne)
- Jak hodnotíte kvalitu obrazu? (Uveďte známkou od 1 do 5, kde 1 je nejlepší hodnocení a odpovídá vysoké kvalitě detailů oproti známce 5, která značí že obraz byl zcela znatelně nekvalitní.)

Poté bude uživateli postupně představena jiná statická scéna v různých variantách rozlišení (2K, 4K a 8K)[9], ze kterých bude muset vybrat, které dosahovalo největší obrazové kvality.

[9] Odpovídá rozlišením 1920x960, 3840x1920 a 7680x3840 pixelů.

Kapitola 4

Implementace

V kapitole je obsažen rozbor klíčových částí implementace, ve kterých je kladen důraz na to, jakým způsobem je dosaženo přehrávání 4K videa ve formátu jpeg2000. Budou zde zmíněny všechny zásadní problémy, které vyvstaly a jakým způsobem byly vyřešeny.

Aplikace byla implementována v jazyce C++ a je přeložitelná pro 64 bitový operační systém windows. Volba 32 bitového systému omezovala práci s většími texturami, které vyžadovali větší adresní prostor aplikace. Všechny potřebné knihovny i prostředí plně podporuje překlad i na systému typu Unix, ale v takové sestavě nebyl program otestován.

4.1 Práce s frameworkem Qt

Implementace pomocí Qt je složená ze dvou hlavních komponent jmenovitě `QMainWindow` a `QWindow`. První z nich zajišťuje celé okno aplikace s jednotlivými prvky jako jsou kontextové menu, `status bar` (spodní lišta aplikace) s textovou nápovědou a činnosti jako je tvorba dialogových oken a zachycení systémových zkratek. Druhá slouží pro přímé vykreslování OpenGL, proto třída `OpenGLWindow`, která ji dědí, obsahuje základní inicializaci a zachycení vlastních kláves aplikace. Třídu `OpenGLWindow` následně dědí `VRPlayer`, která čistě pracuje s OpenGL.

4.2 Integrace OpenVR

Všechny prvky knihovny OpenVR (třídy, struktury, výčtové typy a jiné) jsou v jmenném prostoru `vr`. Přístup k API je tedy jednotně oddělený a v naší aplikaci se s knihovnou pracuje pouze ve třídě `VRPlayer`, která se stará o OpenGL a samotnou logiku aplikace. Knihovna vyžaduje nahrávání vykreslených snímků scény, kde v případě OpenGL je zprostředkované pomocí dvěma framebuffery odpovídající očím. K tomuto vykreslení nabízí dvě *projection* a *view* matice, taktéž odpovídající každému oku zvlášť. Celé obnovení snímku v brýlích se skládá ze získání souřadnic v podobě jednotlivých matic odpovídající všem trackovatelným zařízením funkcí `vr::VRCompositor()->WaitGetPoses()` a dvojitým voláním funkce `vr::VRCompositor()->Submit()` s parametrem oka a odpovídajícím framebufferem převedeným do vlastního typu `vr::Texture_t`. Po druhém zavolání funkce `Submit()` je spuštěn mechanizmus zpracování, vykreslení a obnovení scény v brýlích, přičemž zde nezáleží na pořadí očí. Nahrávání snímků do brýlí musí být vždy odděleno voláním funkce pro získání pozic zařízení.

4.3 Struktura přehrávače

Jádro aplikace, které se stará o chování přehrávače, je ve třídě `VRPlayer`. Třída zajišťuje přehrávání videa na základě seznamu (`QStringList`), který obsahuje získanou cestu k jednotlivým snímků, které jsou abecedně seřazeny bez citlivosti na velká písmena, ale taktéž umožňuje zobrazení i jednoho statického snímku. Veškerá práce přehrávače s videem je tedy závislá na tomto seznamu, např. při posunu videa v čase je zprostředkován pomocí skoku v tomto seznamu. Rychlost přehrávání určuje dané FPS, které lze libovolně měnit z uživatelského rozhraní. Z důvodu různých demoverzí v průběhu implementace je aplikace krom formátu jpeg2000 schopna zobrazit snímky i v těchto variantách:

- png – *.png
- jpeg – *.jpg, *.jpeg
- bmp – *.bmp
- jpeg2000 – *.jp2, *.j2k, *.j2c

Integrace dekodéru UltraJ2K je vnořena do třídy `DecoderProvider`, která obstarává jeho základní činnost, ale zásadním prvkem je třída `MoviePlayer`, která se stará o vznik saturačního vlákna a zaštiťuje práci s vyrovnávací pamětí pro snímky uložené na kartě. V průběhu implementace vyrovnávací paměti se naskytla dvě řešení. Obě uvedené metody jsou založené na CUDA interoperabilitě.

Metoda cudaMemcpyToArray()

Využívá přímého mapování OpenGL textury do CUDA, pomocí funkce `cudaGraphicsGLRegisterImage()`, která v jednom ze svých parametrů vyžaduje identifikátor již dříve vygenerované OpenGL textury. Nevýhodou řešení je, že následným voláním funkce `cudaGraphicsSubResourceGetMappedArray()` je vrácen ukazatel typu `cudaArray`, který není plně kompatibilní s ukazatelem do paměti typu `void`, který je nazýván v kontextu CUDA runtime jako `devPtr`. Rozdíl je zřejmý již v samotném typu, avšak pro podrobnější porovnání je `devPtr` ukazatelem do souvislého bloku paměti, kdežto `cudaArray` je složitější interní struktura. Pro vzájemnou spolupráci musí být užito funkce `cudaMemcpyToArray()`[1], která mezi těmito dvěma typy překopírovává data.

Metoda PBO

CUDA interoperabilita mimo jiné nabízí mapování OpenGL bufferů. Mezi tyto buffery patří PBO (*Pixel Buffer Object*), který je využitelný pro obrazová data. Při získávání ukazatele na jakýkoliv namapovaný buffer se užívá funkce `cudaGraphicsResourceGetMappedPointer()`, která vrací již výše zmíněný ukazatel `devPtr`, jenž je plně kompatibilní respektive totožný s potřebným ukazatelem v knihovně UltraJ2K.

Pro implementaci vyrovnávací paměti byla vybrána metoda PBO, která nahrává snímky do tohoto typu bufferu. V OpenGL se běžně využívá k asynchronním operacím nad pixely[2] a dále nahrávání textur bez závislosti na činnosti procesoru, jelikož je řízeno přes řadič

[1] https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__MEMORY.html
[2] http://www.khronos.org/opengl/wiki_opengl/index.php?title=Pixel_Buffer_Object&oldid=13691

DMA[3] [22]. V našem případě se sice nejedná o nahrávání textur z disku, ale z celé metody je využito pouze „překlopení“ dat z PBO do textury, které je prakticky cenou své zátěže zanedbatelné. Vyrovnávací paměť tvoří sada PBO, které musí být po inicializaci namapovány do CUDA. Ihned po dekódování jsou snímky připraveny k překlopení dat do textury, následně vykresleny a poté uvolněny k dalšímu použití. Jelikož tento proces probíhá ve dvouvláknovém řešení, tak bylo třeba vyrovnávací paměť opatřit mechanismem výlučného přístupu v podobě dvou uzamykatelných semaforů objektů třídy `Semaphore`. Semafor s názvem `semPBOlist` svým počtem uvolněných zámků indikuje počet volných bufferů a druhý semafor (`semThread`) výlučně uzamyká ovládací prvky vlákna. Zde je ilustrační ukázka týkající se CUDA interoperability a PBO, která je obsažena v saturačním vlákně aplikace 4.1:

```
GLuint pbo;
glGenBuffers(1, &pbo);
glBindBuffer(GL_PIXEL_UNPACK_BUFFER_ARB, pbo);
int frameSize = size_x * size_y * component_size;
glBufferData(GL_PIXEL_UNPACK_BUFFER_ARB,
             frameSize,
             NULL,
             GL_STREAM_DRAW_ARB);
glBindBuffer(GL_PIXEL_UNPACK_BUFFER_ARB, 0);

cudaGraphicsResource *resource;
cudaGraphicsGLRegisterBuffer(&resource, pbo, cudaGraphicsMapFlagsNone);
cudaGraphicsMapResources(1, &resource, 0);
void *mappedBuffer;
size_t bufferSize;
cudaGraphicsResourceGetMappedPointer(&mappedBuffer, &bufferSize, resource);

// Working with data ...

cudaGraphicsUnmapResources(1, &resource, 0);
cudaGraphicsUnregisterResource(resource);
```

Výpis 4.1: Zjednodušený kód práce s CUDA interoperability s využitím PBO.

V následující ukázce kódu 4.2 je demonstrace překlopení dat z PBO, které je dosaženo díky předchozího volání `glBindBuffer()` pro texturu a PBO a následného zavolání funkce `glTexImage2D()` s posledním parametrem `NULL` pro zdroj nahrávaných dat.

```
GLuint texture;
glGenTextures(1, &texture);
glBindTexture(GL_TEXTURE_2D, texture);
glBindBuffer(GL_PIXEL_UNPACK_BUFFER_ARB, pbo);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA8, size_x, size_y,
             0, GL_RGB, GL_UNSIGNED_BYTE, NULL);
```

Výpis 4.2: Ukázka překlopení dat z PBO do textury.

[3] Přímým přístupem do paměti

Kapitola 5

Testování

Cílem kapitoly je představit výsledky testování a zpětné vazby, která byla provedena podle návrhu z podkapitoly 3.6. Na základě některých průběžných výsledků bylo spuštěno i několik dodatečných testů, které zde budou taktéž uvedeny. V závěru kapitoly je jejich celkové zhodnocení. Součástí přílohy této práce je několik snímků obrazovky z výsledné aplikace, které jsou umístěny v příloze A.

Všechny testy byly provedeny na uvedené počítačové sestavě:

- Procesor: Intel Core i5-6500 @ 3.20GHz[1]
- Nvidia GeForce GTX 1060, OC, 6 GB[2]
- 16 GB RAM, DDR4
- SSD disk
- Windows 10 64-bit

Každý test, pokud není uvedeno jinak, proběhl s videem od společnosti Airpano, jehož jednotlivé snímky byly zakódovány do jpeg2000 s kvalitou komprese 40% a s ponecháním jeho původní velikosti 4K (3840x1920) s podvzorkováním 4:4:4. Pro účel porovnání kvality statického snímku bylo užito panoramatické fotografie[3], která byla z 8K zmenšena na 4K a 2K a vytvořila tak sadu tří fotografií.

5.1 Hodnocení FPS

Na základě měření bylo prokázáno, že dekódování pomocí UltraJ2K za výše uvedených podmínek a bez jiné zátěže hardware je schopno dosáhnout na 1000 snímcích průměrných 118.47 FPS. Testování probíhalo speciální aplikací od firmy Comprimato, která umožňuje simulovat různé podmínky, např. vlákna, která plní dekodér snímky, nebo naopak ta, která je odebírají, a taktéž lze testovat i jiné typy implementací (CPU, OpenCL a CUDA). Na základě těchto a jiných dalších možností byl test plně přizpůsoben návrhu našeho přehrávače.

[1] https://ark.intel.com/products/88184/Intel-Core-i5-6500-Processor-6M-Cache-up-to-3_60-GHz

[2] https://www.techpowerup.com/gpudb/b3755/asus-dual-gtx-1060-oc

[3] Převzaté z: http://www.justpano.com/images/USA-road-trip-VR-360-Photo-17

Účelem dalšího měření bylo zjistit, jakého maximálního FPS jsme schopni dosáhnout pouhým vykreslováním statického obrazu do brýlí. Abychom zamezili odběru výkonu inicializací knihovny UltraJ2K, dekódovali jsme statický snímek pomocí integrovaného dekodéru v knihovně Qt. Výsledkem měření bylo, že aplikace byla schopna aktualizovat snímek v brýlích maximálně 89.55 FPS, což odpovídá garantované obnovovací frekvenci zařízení HTC Vive, která je 90Hz. Dle pozdějšího profilování bylo ověřeno, že maximální rychlost běhu aplikace je omezená odezvou HMD v momentech zajištění souřadnic polohy a při nahrávání snímků pro obě oči.

Z těchto dvou úvodních měření, lze tedy předpokládat, že aplikace nebude schopna prolomit hodnotu vyšší než ~90 FPS, které odpovídá druhé variantě měření.

V závěrečné fázi implementace přehrávač vykazoval vůči maximálnímu předpokladu FPS značnou ztrátu, protože dosahoval maximálně ~20 FPS. Na základě tohoto zjištění bylo nalezeno několik chyb v samotné optimalizaci dvouvláknové spolupráce a sdílené vyrovnávací paměti k vykreslení. Po jejich opravě bylo dosaženo hodnoty 22.14 FPS. Celkové vytížení procesoru grafické karty dle aplikace GPU-Z[4] bylo 85% a vytíženost CPU dle *Správce úloh* systému Windows přibližně 55%. Na základě těchto informací a nedostačujícího výkonu v měřítku FPS bylo provedeno více testů s jinými parametry, které jsou včetně jejich výsledků uvedeny v následující tabulce 5.1:

Kvalita videa	FPS	Zátěž CPU	Zátěž GPU
1K[5] 100%	22.50	53%	78%
4K 40%	22.14	55%	86%
4K 75%	21.30	50%	90%
4K 100%	17.34	52%	91%
8K 50%	13.35	55%	100%

Tabulka 5.1: Testování FPS na jednotlivých kvalitách videí.

Dle naměřených výsledků, které poukazují na změnu FPS v závislosti na kvalitě videa, vyplývá, že při markantním snížení kvality vstupního videa není hodnota FPS příliš ovlivněna.

Úpravami maximálního počtu snímků ve vyrovnávací paměti a omezením maximální saturace dekodéru[6] jsme docílili přehrávání videa po shlucích. Tato skutečnost vedla k profilování aplikace s uvedenými úpravami parametrů, které vyvolali „shlukové“ přehrávání. V tabulce 5.2 je souhrn volání, které zabraly > 1 ms.

Název funkce	Průměrný čas vykonání funkce [ms]	
	při dekódování	bez dekódování
Submit(vr::Eye_Left, ...) – (1)	0.87	0.75
Submit(vr::Eye_Right, ...) – (2)	5.67	0.31
WaitGetPoses()	7.46	5.57

Tabulka 5.2: Výsledky časů měřených po dobu vykonání dané funkce.

[4] Oficiální WWW stránka aplikace: https://www.techpowerup.com/gpuz/

[5] 960x480 pixelů

[6] Jedná se o umělé navýšení či snížení počtu snímků, které lze dekódovat paralelně, této úpravy je možné dosáhnout s úměrnou závislostí na dostupné paměti na GPU.

Při podrobném výpisu hodnot zpoždění druhého volání funkce `Submit()`[7], bylo zjištěno, že v celém běhu aplikace dosahuje totožného průměru jako při běhu bez dekódování, až na moment, kdy je spuštěno dekódování. V tento moment je trvání funkce prodlouženo až na 100 ms a další volání (pro následující snímek) dosahuje přibližně 50 ms. Je tedy průkazné, že se jedná o dobu, která je potřebná pro trvání dekódovacího kernelu v knihovně UltraJ2K, která v daný moment dekóduje. K tomuto zjištění odpovídá i aktuální saturace, potvrzená debugovacím výpisem.

Z důvodu vyvrácení teorie, že je chyba v implementaci přehrávače, bylo užito úvodních testů, které byly spuštěny souběžně. Spuštěný testovací program UltraJ2K dosahoval dekódování snímků v rychlosti 90 FPS, ale přehrávač, který pouze vykresloval statickou scénu, nedosáhl ani 16 FPS. Výsledkem těchto všech měření je, že knihovny ve společném užití zdrojů CUDA nejsou schopny spolupracovat tak, aby se výlučně jejich chod neomezil.

Nejjednodušším a přímočarým řešením tohoto problému by bylo užití dvou grafických karet, kde by nedocházelo k vzájemnému výlučnému zabírání dostupných zdrojů. Muselo by být ovšem zajištěno překopírování dekódovaných dat z jedné karty do druhé, kde by docházelo k vykreslení do brýlí. Toto řešení by bylo značně neekonomické a neodpovídalo by tak jednomu ze záměru snížit náročnost aplikace na dostupný hardware.

5.2 Výhodnocení zpětné vazby

Testování prvních tří uživatelů probíhalo ještě v době, kdy byl přehrávač ve vývoji a to mělo za následek úprav některých prvků uživatelského rozhraní. Jednalo se zejména o zanedbatelné detaily, jakými jsou např. jméno aplikace uvedené v záhlaví, přirozenější uzpůsobení zkratek a obsah úvodní obrazovky. Nejvíce kritizována byla rychlost přehrávání videa, tu se bohužel nepodařilo znatelně zlepšit, proto i další hodnocení tuto nedokonalost reflektovalo. Celkem se hodnocení zpětné vazby účastnilo 8 uživatelů.

V druhé fázi testu 6 z 8 uživatelů hodnotilo přehrávání videa, jako trhané nebo pomalé. Totožná skupina uživatelů si stěžovala na zpomalený pohyb hlavy, který byl způsoben nízkou hodnotou FPS. Kvalita obrazu byla veskrze pozitivní. Bezprostředně po této části si 2 uživatelé stěžovali na problémy spojené s nevolností v simulátoru (*z angl. simulator sickness*).

V druhé části této fáze, kde se testoval statický snímek z předchozího videa, se všichni uživatelé jednomyslně shodli na plynulosti pohybu hlavy. Hodnocení kvality obrazu bylo o něco vyšší oproti hodnocení videa, ačkoliv se jednalo o jeden ze snímků z jeho sekvence.

Poslední otázkou byl výběr ideálního rozlišení. Uživatelé se shodli na 4K a to zřejmě proto, že 8K převyšovalo možné maximální zobrazení brýlí HTC Vive v šířce ekvidistantní textury (viz 4.2). V obraze tak docházelo k aliasingu v místech s velkým kontrastem na hranách objektů.

[7] Měření odpovídá jejího druhého volání, které spouští mechanizmus obnovení scény v brýlích viz 4.2.

Kapitola 6

Závěr

Cílem práce bylo implementovat přehrávač videa pro virtuální realitu o kvalitě 4K, provést jeho testování a zajistit zpětnou vazbu uživatelů.

V průběhu práce jsme se seznámili s více variantami projekce 360° videa a dalšími obohacujícími informacemi jako například různá aktuální řešení kompresí a znalosti týkající se kodeku jpeg2000. Jako hlavní zdroje informací, které se týkali reprodukce 360° videa, byly developerské weby softwarových společností jako je Google a Facebook a z tohoto důvodu byla práce se zdroji ztížena, jelikož tyto informace nejsou nijak oficiálně standardizovány nebo vědecky podloženy. Nemalou problematikou bylo zajištění vstupních videí v potřebném formátu projekce, velikosti rozlišení a takových které by mohli odpovídat pracovní verzi standardu *Spatial Media*. Z pohledu integrace kodeku byla klíčová vzájemná spolupráce s firmou Comprimato, která poskytla pro účel této práce demo svého produktu UltraJ2K, který implementuje kodek jpeg2000 s optimalizací na rozhraní CUDA.

Výsledkem implementace, s přihlédnutím na hodnocení uživatelů, se stala aplikace s jednoduše přístupným uživatelským rozhraním pro přehrávání videa ve VR. Bohužel i přes veškerou snahu optimalizace nebylo dosaženo vyššího přehrávání videa než průměrného ~22 FPS. Na základě testů a měření bylo prokázáno, že kumulované zpoždění je způsobeno vzájemnou spoluprací knihoven UltraJ2K a OpenVR, které se výlučně blokují se zdrojem grafického výkonu jader CUDA. Jedním z řešení může být využití dvou karet s podporou CUDA, nebo novější technologie v oblasti CUDA paralelismu či optimalizovanější verze produktu UltraJ2K.

Práce by dál mohla být rozšířena jednoduchou záměnou dekódovací knihovny. Nabízí se implementace kodeku HTJ2K (High-Throughput JPEG 2000), který vychází ze specifikace jpeg2000, nebo kodeku AV1, který je vyvíjen za účelem nahradit VP9 a H.265. Jednou z dalších výhod u obou těchto kodeků je *royality free* specifikace, avšak stále jsou ještě ve fázi vývoje.

Dalším posunem by samozřejmě bylo uzavření a oficiální vydání specifikace standardu *Spatial Media* a jeho následnou implementací včetně podpory jeho vlastních projekcí.

Literatura

[1] Brown, C.: *Bringing pixels front and center in VR video*. Google AR and VR, 2017, [Online; navštíveno 10.04.2018].
URL https://blog.google/products/google-vr/bringing-pixels-front-and-center-vr-video/

[2] Brown, M.; Lowe, D. G.: *Automatic Panoramic Image Stitching using Invariant Features*. *International Journal of Computer Vision*, ročník 74, č. 1, 2007: s. 59–73, doi:10.1007/s11263-006-0002-3.

[3] Corbillon, X.; Simon, G.; Devlic, A.; aj.: *Viewport-Adaptive Navigable 360-Degree Video Delivery*. *EEE International Conference on Communications*, 2007: s. 1–7, doi:10.1109/ICC.2017.7996611.

[4] Furuti, C. A.: *Map Projections*. 2015, [Online; navštíveno 04.05.2018].
URL http://www.progonos.com/furuti/MapProj/Dither/ProjTbl/projTbl.html

[5] Google: *Spherical Video V2 RFC*. 2015, [Online; navštíveno 10.04.2018].
URL https://github.com/google/spatial-media/blob/master/docs/spherical-video-v2-rfc.md

[6] Group, K.: *OpenGL Overview*. 2018, [Online; navštíveno 06.05.2018].
URL https://www.opengl.org/about/

[7] Hanhart, P.; Rerabek, M.; Korshunov, P.; aj.: *Subjective evaluation of HEVC intra coding for still image compression*. Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, 2013, [Online; navštíveno 30.04.2018].
URL http://phenix.it-sudparis.eu/jct/doc_end_user/current_document.php?id=7167

[8] Initiatives, D. C.: *DCI specification*. DCI Digital Cinema Initiatives, LLC, 2012, [Online; navštíveno 30.04.2018].
URL http://www.dcimovies.com/specification/index.html

[9] ITU: *Recommendation ITU-T H.264*. 2017, [Online; navštíveno 28.04.2018].
URL https://www.itu.int/rec/T-REC-H.264-201704-I/en

[10] Kuzyakov, E.; Pio, D.: *Next-generation video encoding techniques for 360 video and VR*. Engineering Blog – Facebook Code, 2016, [Online; navštíveno 10.04.2018].
URL https://code.facebook.com/posts/1126354007399553/next-generation-video-encoding-techniques-for-360-video-and-vr/

[11] LaValle, S. M.: *Virtual Reality*. Cambridge University Press, 2017, [Online; navštíveno 08.05.2018].
URL http://vr.cs.uiuc.edu/vrbook.pdf

[12] Marpe, D.; Wiegand, T.; Sullivan, G. J.: *The H.264/MPEG4 Advanced Video Coding Standard and its Applications*. *IEEE*, ročník 44, 2006: s. 134–143, doi:10.1109/MCOM.2006.1678121.

[13] Matela, J.: *Implementace JPEG2000 komprese na GPU*. Diplomová práce, Masarykova univerzita, Fakulta informatiky, Brno, 2009.
URL https://is.muni.cz/th/m6lmk/

[14] McHugh, S.: *Panoramic Image Projections*. Cambridge in Colour, 2018, [Online; navštíveno 06.05.2018].
URL https://www.cambridgeincolour.com/tutorials/image-projections.htm

[15] Mukherjee, D.: *A Technical Overview of VP9: The latest royalty-free video codec from Google*. 2014, [Online; navštíveno 09.05.2018].
URL https://files.meetup.com/9842252/Overview-VP9.pdf

[16] Mukherjee, D.; Bankoski, J.; Grange, A.; aj.: *The latest open-source video codec VP9 – An overview and preliminary results*. *Picture Coding Symposium*, 2013: s. 390–393, doi:10.1109/PCS.2013.6737765.

[17] Newman, D.: *Beyond Counting Pixels: Defining Resolution in Spherical*. GoPro Official Website, 2017, [Online; navštíveno 24.04.2018].
URL https://gopro.com/news/beyond-counting-pixels--defining-resolution-in-spherical

[18] Skodras, A.; Christopoulos, C.; Ebrahimi, T.: *The JPEG 2000 Still Image Compression Standard*. IEEE SIGNAL PROCESSING MAGAZINE, 2001, [Online; navštíveno 30.04.2018].
URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.591.4975&rep=rep1&type=pdf

[19] Snyder, J. P.: *Map Projections–A Working Manual*. Washington, DC: U. S. Government Printing Office, první vydání, 1987, ISBN 9998605067.

[20] Sullivan, G. J.; Ohm, J.-R.; Han, W.-J.; aj.: *Overview of the High Efficiency Video Coding (HEVC) Standard*. *IEEE Transactions on Circuits and Systems for Video Technology*, ročník 22, č. 12, 2012: s. 1649–1668, doi:10.1109/TCSVT.2012.2221191.

[21] Sullivan, G. J.; Topiwala, P. N.; Luthra, A.: *The H.264/AVC Advanced Video Coding standard: overview and introduction to the fidelity range extensions*. *Proceedings of the SPIE*, ročník 5558, 2004: s. 454–474, doi:10.1117/12.564457.

[22] Venkataraman, S.: *Optimizing Texture Transfers*. GPU Technology Conference, 2012, [Online; navštíveno 13.02.2018].
URL https://developer.download.nvidia.com/GTC/PDF/GTC2012/PresentationPDF/S0356-GTC2012-Texture-Transfers.pdf

[23] Verma, S.: *A new way to see and share your world with 360-degree video*. Youtube Creator Blog, 2015, [Online; navštíveno 22.04.2018].
URL https://youtube-creators.googleblog.com/2015/03/a-new-way-to-see-and-share-your-world.html

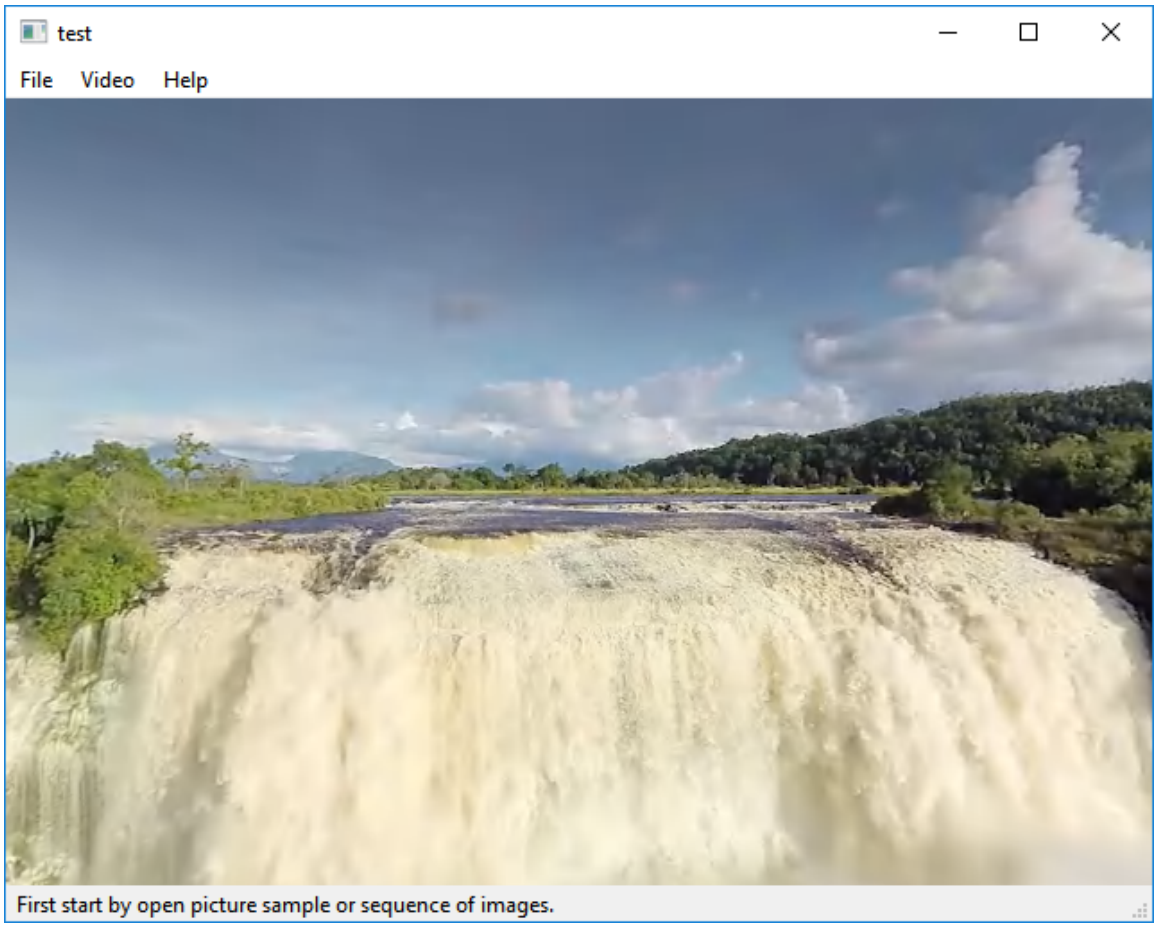
[24] Wheeler, A.: *Improving VR videos*. Youtube Engineering and Developers Blog, 2017, [Online; navštíveno 10.04.2018].
URL https://youtube-eng.googleblog.com/2017/03/improving-vr-videos.html

[25] Zubetz, A.; Gaponyuk, O.: *Spherical 360 Video, Test Shooting*. AirPano, 2011, [Online; navštíveno 21.04.2018].
URL http://www.airpano.com/Articles-AirPano.php?article=101606

Příloha A

Snímky obrazovky z běhu aplikace

Obrázek A.1: Okno aplikace s pohledem levého oka na virtuální ovladač HTC Vive.



Obrázek A.2: Okno aplikace s pohledem levého oka při přehrávání.

Příloha B

Obsah přiloženého DVD

Obsah přiloženého DVD je rozdělen do těchto adresářů:

- *bin* – přeložené binární soubory aplikace přehrávače
- *src* – obsahuje zdrojové kódy pro překlad aplikace
- *text* – obsahuje tento dokument ve formátu PDF
- *text_src* – zdrojové kódy tohoto dokumentu a obrázky použité v textu
- *manual* – obsahuje manuál k aplikaci a jeho celou dokumentaci
- *others* – obsahuje doplňující materiál např. testovací obrázky