SLEZSKÁ UNIVERZITA V OPAVĚ

Filozoficko-přírodovědecká fakulta v Opavě

DIPLOMOVÁ PRÁCE

Opava 2016 Michael Kocián

SLEZSKÁ UNIVERZITA V OPAVĚ

Filozoficko-přírodovědecká fakulta v Opavě

Michael Kocián

Informatika a výpočetní technika

Senzory mobilních zařízení a jejich využití při programování her pro platformu Android

Sensors in Mobile Devices and their Use for Game Development on Android Platform

Diplomová práce

Opava 2016

Vedoucí diplomové práce:

RNDr. Šárka Vavrečková, Ph.D.

Abstrakt:

Práce se zabývá technologií mobilních senzorů, hlavně je popsán způsob fyzického fungování a možnosti využití v programování. Dále se práce zabývá operačním systémem Android a pochopením, jak se pro něj programují aplikace. Výsledkem je vytvoření programu (hry), který využívá senzory a podle získaných dat uzpůsobuje grafický výstup. Tento program je popsán z funkční i programátorské stránky.

Klíčová slova:

Android, Java, Programování, OpenGL ES, Virtuální realita, Unity3D, senzory

Abstract:

This thesis deals with the phone sensors technology. We mainly describe the method of physical functioning and options of usage in programming. Furthermore, the thesis focuses on the Android operating system and understanding how Android applications are made for. The aim of this thesis is to create a program (game) which uses sensors, and its graphics output is adapted according to obtained data graphics. The created program is described from the functional and programming point of view.

Keywords:

Android, Java, Programming, OpenGL ES, Virtual reality, Unity3D, sensors

Sem vložte kopii podkladu zadání práce ze STAGu, a to podepsanou.
Buď naskenovat a vložit jako obrázek, anebo vložit vytisknuté podepsané.

Prohlášení

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškerou literaturu a další zdroje, z nichž jsem při zpracování čerpal, v práci řádně cituji a jsou uvedeny v seznamu použité literatury.

V Opavě dne …….

……..

Michael Kocián

Poděkování

Rád bych poděkoval za odborné vedení, rady a cenné poznatky k danému tématu vedoucímu práce RNDr. Šárce Vavrečkové Ph.D.

Také bych rád poděkoval mé rodině a přátelům za podporu a pomoc během mého studia.

Obsah

Úvod

Tato diplomová práce vznikla z potřeby ukázat možnost využití telefonu se systémem Android jako výpočetního prvku pro brýle zobrazující virtuální realitu. V dnešní době většina lidi vlastní telefon s operačním systémem Android. Výpočetní výkon takového mobilního zařízení v poslední době dosahuje hranice, kdy je schopen zobrazovat virtuální prostředí, například hru, při udržení 60 snímků za sekundu. Tato schopnost je zásadní, jelikož ve virtuálních brýlích není možné tolerovat záseky jako na obyčejném monitoru, jelikož dochází k velmi negativnímu vnímání ze strany uživatele, které se může projevit dokonce fyzickou nevolností.

Telefon se vsune dovnitř plastové násady s dvěma konvexními čočkami, které míří na displej telefonu a z druhé strany může uživatel pozorovat virtuální realitu. Simulovat realitu je možné díky senzorům orientace. Telefon zjišťuje svou polohu, ze které odvodí natočení uživatele v reálném světě a podle toho namíří i svou vykreslovací kameru v simulovaném prostředí. Výsledkem jsou brýle, které si uživatel přidržuje nebo připevní k hlavě a je mu umožněno nahlížet do virtuální reality pomocí jeho fyzické rotace. Přirozený pohyb v této virtuální realitě je možný jen v případě přidání dalších zařízení jako je například kamera, a proto se pohyb řeší nejčastěji jako procházka po předdefinované cestě, případně ve směru pohledu.

Cílem práce je popsat možnosti vykreslování grafických aplikací na platformě Android s využitím senzorů při jejich ovládání. Praktickou částí práce je vytvoření herního prostředí, v němž má být kamera ovládána vestavěným gyroskopem.

1 Platforma

1.1 Operační systém Android

OS Android je nejrychleji rostoucí systém pro přenosná zařízení s dotykovou obrazovkou. Vývoj systému započal ve firmě Android již v roce 2003. Jejich cílem bylo vytvořit otevřenou mobilní platformu, která by byla schopna konkurovat tehdejším největším systémům Symbian a Windows Mobile. V roce 2005 firmu koupila společnost Google. První Android telefon byl HTC Dream, vydaný dne 5. listopadu 2008. Během následujících 8 let se podíl operačního systému Android na trhu rozrostl na více než 80 % [1]. V roce 2016 je pro operační systém Android konkurencí Windows Phone a iOS.

1.1.1 Verze

Každá verze systému má kromě číselného označení také jméno po nějaké pochoutce, a to v abecedním pořadí. První volně šiřitelná verze byla 1.5, do té doby se jednalo pouze o *beta* verze, které neběžely na žádných prodávaných telefonech.

Číselné označení a jméno	Hlavní přidané novinky
1.5 Cupcake	Nahrávání videa, softwarová klávesnice
1.6 Donut	Multitouch, GPS, větší rozlišení
2.0 Eclair	Softwarový zoom u kamery, podpora HTML5
2.2 Froyo	Podpora Adobe Flash, podpora více jazyků
2.3 Gingerbread	Podpora více kamer, NFC a jiných senzorů
3.0 Honeycomb	Přizpůsobená pro tablety, pro telefony nedostupná, Podpora nových grafických čipů a vícejádrových procesorů
4.0 Ice Cream Sandwich	Odemykání obličejem, zkratky na zamčené obrazovce
4.1 Jellybean	Podpora více uživatelů na jedno zařízení, vylepšení odezvy
4.4 KitKat	Integrace Google Now, nižší náročnost na RAM
5.0 Lollipop	Kompletně předělaný vzhled
6.0 Marshmallow	Možnost aplikacím zakázat různé komponenty telefonu
7.0 N (beta)	Možnost více oken na jedné obrazovce

Tabulka 1: Historie Verzí OS Android

1.1.2 Grafické rozhraní

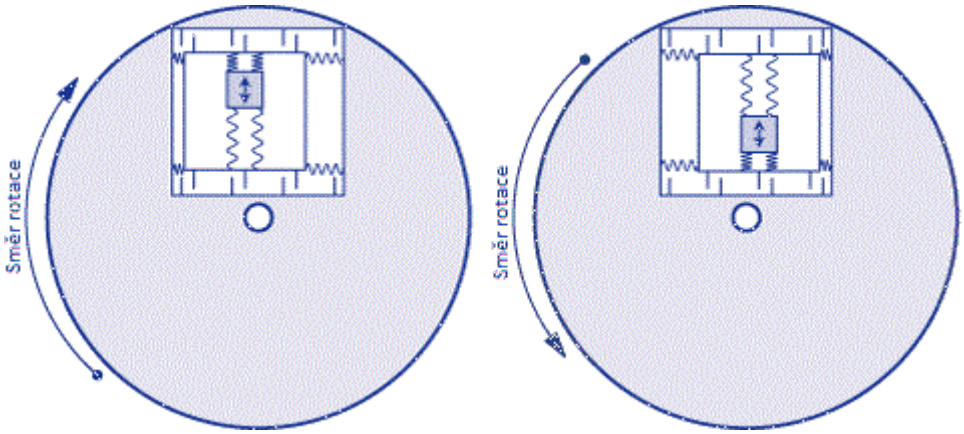
Uživatel komunikuje se systémem pomocí doteků obrazovky a gest. Odezva na tyto doteky a gesta by měla být co nejrychlejší a nejplynulejší, protože to je nejvíc viditelná složka kvality systému. Klasické doteky obrazovky nahrazují použití myši. Gesta přidávají funkčnost, která nahrazuje potřebu použití většího množství tlačítek nebo klávesových zkratek, což je na malém displeji vhodné.

1.2 Senzory

Senzory v telefonu jsou vyrobeny technologií MEMS (Micro Electric Mechanical Systems). Tato sofistikovaná technologie je využívaná nejčastěji pro pohybové senzory, mikropohony, mikrocívky a podobně. Takovéto čipy mají mechanickou komponentu, která je nezbytná pro detekci fyzikálních veličin, a elektroniku, která snímá změny ve vlastnostech mechanického subsystému, například změny elektrického odporu.[12]

1.2.1 Senzor Gyroskop

Gyroskop je zařízení, které měří, jak rychle se objekt, ke kterému je připevněn, otáčí. Zjišťujeme úhlovou rychlost vzhledem k osám X, Y a Z, která je vyjádřená ve stupních nebo radiánech za sekundu. Senzor je složen z rámu a hmoty na mikro pružinách, která vibruje, čímž nahrazuje rotaci. Na takto vibrující hmotu stejně jako v případě rotace působí Coriolisova síla, což je veličina kolmá na odstředivou sílu a je přímo úměrná k rychlosti rotace.[10][8] Když telefon otáčíme, posouvá se rámeček s vibrující hmotou a kapacitní snímače zjišťují míru Coriolisovy síly; když je telefon v klidu, vibrující hmota rámeček ustálí.

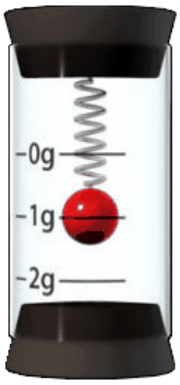


Obrázek 1: Zjednodušené schéma MEMS gyroskopu [10]

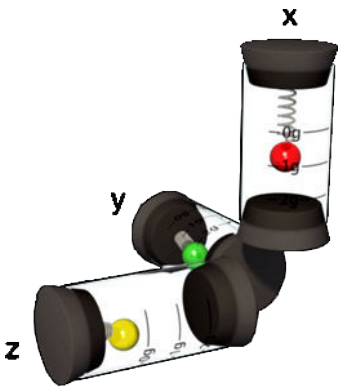
1.2.2 Senzor Akcelerometr

Jedná se o senzor zrychlení. Tento senzor měří změnu v pohybu zařízení v jednotkách m/s^2. Je ovlivněn gravitačním zrychlením, které míří přímo ke středu země. Akcelerometr podobně využívá pohyblivou hmotu, na rozdíl od gyroskopu si hmota uchovává svou potenciální energii a při změně rychlosti pohybu se snaží pokračovat ve své trajektorii, to jí ovšem není dovoleno opět kvůli mikro pružince, ale dojde k dočasnému vychýlení z ustálené polohy. Zde využijeme míru vychýlení, která

je přímo úměrná zrychlení, což si můžeme všimnout na Obrázek 1. K získání všech potřebných údajů o zrychlení musíme jednotlivé akcelerometry uspořádat do všech tří os, jak ukazuje obrázek 3.



Obrázek 2: Accelerometer[1]



Obrázek 3: Osové uspořádání accelerometeru[1]

1.2.3 Elektromagnetický kompas

Tento kompas se skládá ze dvou (pro 2D), nebo tří (pro 3D) senzorů magnetického pole. Základní část je magnetoresistivní plíšek vyrobený z feromagnetického materiálu (trvalý magnet) vyrobeného ze směsi železa a niklu. Takový plíšek má při nulovém působení vnější magnetické síly při stanovém proudu určitý odpor. Tento odpor se mění v závislosti na magnetickém působení.

Tato konstrukce má dva nedostatky. První nedostatek je nelinearita zejména pro krajní hodnoty, která se řeší překrytím plíšku hliníkovými pásky s úhlem 45°. Hliník má lepší vodivost a tím zlepší charakter měřených hodnot odporu plíšku. Druhým nedostatkem je nemožnost určení směru magnetického působení, při otočení o 180° budou naměřené hodnoty stejné. Proto se přidává druhý senzor otočený o 90° a výsledný směr magnetického působení je softwarově dopočítán.

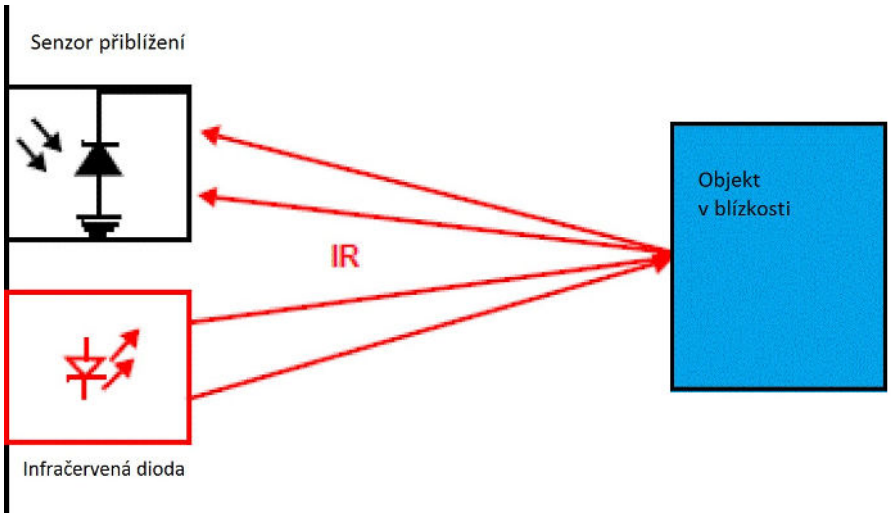
1.2.4 Senzor Světla

Senzor světla je pouhá fotocitlivá dioda, která adaptuje svůj odpor v závislosti na světelných podmínkách. Tento senzor bývá kalibrován na velmi nízké světelné podmínky a není určen pro měření intenzity světla ve vnějším prostředí. Jeho využití spočívá v naměření ambientního osvětlení, a následně se adaptuje intenzita svícení displeje tak, aby ve tmě telefon příliš nezářil a tím neoslepoval uživatele a zároveň neplýtval baterií. S rostoucím světlem se intenzita svícení zvyšuje tak, aby obsah zůstal dobře viditelný.

[1] https://mobilenet.cz/clanky/techbox-vas-telefon-je-prospikovany-senzory-12496

1.2.5 Senzor Přiblížení

Tento senzor se skládá ze dvou částí. První část je pouhá infračervená dioda, a druhá část je světlo-citlivá dioda kalibrovaná na světlo z první diody. Účelem využití tohoto senzoru není poskytovat údaj o vzdálenosti, ale pouze o tom, zdali se v blízkosti senzoru něco nachází nebo ne. Obrázek 4 ukazuje situaci, když se nachází nějaký objekt v blízkosti senzoru. Kdyby žádný objekt v blízkosti nebyl, tak se světlo neodrazí, ale rozptýlí se v prostoru. Ve skutečnosti je maximální vzdálenost přibližně 1cm.



Obrázek 4: Senzor přiblížení

1.3 OpenGL

OpenGL je API sloužící pro rychlou komunikaci s grafickou kartou. Toto API nám pomáhá vykreslovat scénu na obrazovku. Od verze 2.0 se s OpenGL pracuje tak, že máme 3D objekt v paměti ve formě vertexů, což jsou vrcholy trojúhelníků, ze kterých se celý objekt skládá, a také máme v paměti uloženou texturu, což je bitmapa. Na každý takto uložený objekt ve scéně zavoláme funkci pro vykreslení objektu z API společně s pozicí a natočením. Na konci vykreslování dostaneme obrázek scény s vyřešenou viditelností.

1.3.1 Varianta OpenGL ES

Pro mobilní zařízení se používá varianta ES (Embedded System), která vychází se značným zpožděním oproti hlavní variantě z důvodu, že celý mobilní ekosystém je slabší, co se týká výkonu, a tedy jsou nutné rozsáhlejší optimalizace.. Toto omezení ale velmi klame, protože mobilní čipy pracují obecně efektivněji a také varianta ES je lépe optimalizovaná právě na úkor včasnosti uvedení.

1.3.2 Jazyk grafických karet

Pro grafickou kartu je nutné napsat program. Programovací jazyk používaný pro tento účel se nazývá GLSL a vytvořené jednotky nazýváme shadery. Tento jazyk je syntaxí velice podobný jazyku C. Shadery se distribuují v textové formě, a jejich kompilace probíhá až po spuštění na grafické kartě. Zpracování na grafické kartě probíhá paralelně pro více vstupů (SIMD). Další informace lze nalézt ve zdrojích [6][7][15][17]. Rozlišujeme dva druhy takovýchto shaderů.

- *Vertex shader* – tento shader je použit jako první a je aplikován pro každý vrchol každého objektu, na který jsme zavolali API pro vykreslení. Vstupní data jsou aplikována na uložené modely v paměti grafické karty a vznikají modely nové, tentokrát zasazené v reálné scéně. Je tedy určena přesná pozice, natočení, zvětšení či zmenšení modelu a libovolné další geometrické operace. Výsledkem je scéna v souřadnicovém systému kamery.
- *Pixel shader* – Pracuje s výstupem z vertex shaderu. Přeneseně řečeno pro každý pixel v obraze je spuštěn algoritmus, který určí pixelu výslednou barvu podle jeho textury, osvětlení a dalších podmínek. Textury se většinou skládají ze dvou složek, první složka je barevná mapa a druhá je tzv. normálová mapa, která určuje míru a směr odraženého světla. Je to také velice oblíbený nástroj pro přidávání dodatečných efektů (anglicky After Effects).

1.4 Unity

Unity je rozsáhlý framework pro tvorbu her, který zapouzdřuje všechny základní potřeby jako je vykreslování, osvětlení, správa animací, částicové efekty, fyzika nebo umělá inteligence. Unity je často preferováno, protože zbavuje povinnosti tvořit zvlášť ty části zobrazení, které jsou ve většině her společné, a umožňuje programátorům soustředit se čistě na hru. Toto je na úkor výkonu, jelikož Unity zahrnuje mnoho komponent a je tedy i poměrně náročné na výpočetní výkon.

Chování hry se vytváří pomocí skriptů, které mohou být napsány v jazycích C# nebo JavaScript. Přímo v prostředí lze hru průběžně spouštět a ladit. Na konci lze hru zkompilovat pro různé platformy (WebGL, Windows; Mac a Linux, Android, Windows Phone, iOS, tvOS, Tizen, Xbox 360 a One, PS 3;4 a Vita, SamsungTV). Některé z těchto platforem mají jiný výkon a jiné vstupní možnosti, a proto je při vývoji nutné vědět, na kterou nebo které platformy míříme.

2 Virtuální realita

Virtuální realita je technologie, která umožňuje uživateli pohybovat se v simulovaném prostředí. Znamená to, že se snažíme oklamat smysly člověka tak, aby nevnímal, že není přítomen tam, kde se jeho fyzické tělo skutečně nachází, ale aby nabyl dojmu, že je v prostředí, které je mu ukazováno. Současná technologie, ani ta komerční, ještě není ani zdaleka tak dokonalá, aby člověk opravdu uvěřil, že se nachází na jiném místě, ale dokáže uživatele natolik pohltit, aby se v simulovaném prostředí dokázal pohybovat, pracovat v něm, nebo hrát hry alespoň s dostatečnou mírou důvěryhodnosti, která u 2D obrazovek neexistuje.

2.1 Historie

Virtuální realita se začala objevovat počátkem devadesátých let. Tehdy i samotná počítačová grafika byla velice primitivní, natož zobrazovací prostředky. Zobrazování bylo uskutečňováno v „obřích krabicích“, které musel uživatel nasadit na hlavu. Tyto krabice byly velké, těžké a nepohodlné, a proto došlo k útlumu virtuální reality na minimálně dalších 20 let. Během této doby byla virtuální realita provozována jen v laboratořích. Jediný užitek z toho měla americká armáda a NASA, protože bylo levnější cvičit piloty ve virtuálním prostředí, než uskutečňovat lety.

V poslední době ale vývoj mobilních zařízení předběhl tato drahá dedikovaná zařízení svými parametry natolik, že je z jejich komponent umožněno sestavení výkonné náhlavní soupravy bez příliš drahého vývoje zaměřeného na virtuální realitu a její hardware.

2.2 Využití

Virtuální realita má příhodné využití v lékařství, kde se už v současnosti pořizují 3D snímky orgánů, nicméně jejich prohlížení je značně nemotorné. V oblasti konstrukčního inženýrství vzniká také mnoho 3D modelů. Strojní výrobky, stavby, automobily a podobné věci, které jsou nákladné na výrobu, si lze ve virtuální realitě snadno prohlédnout. Tímto způsobem lze nejen odhalit případné nedostatky, ale také ukázat dopředu zákazníkovi to, co si bude kupovat, ne ve smyslu vidět, ale ve smyslu zažít.

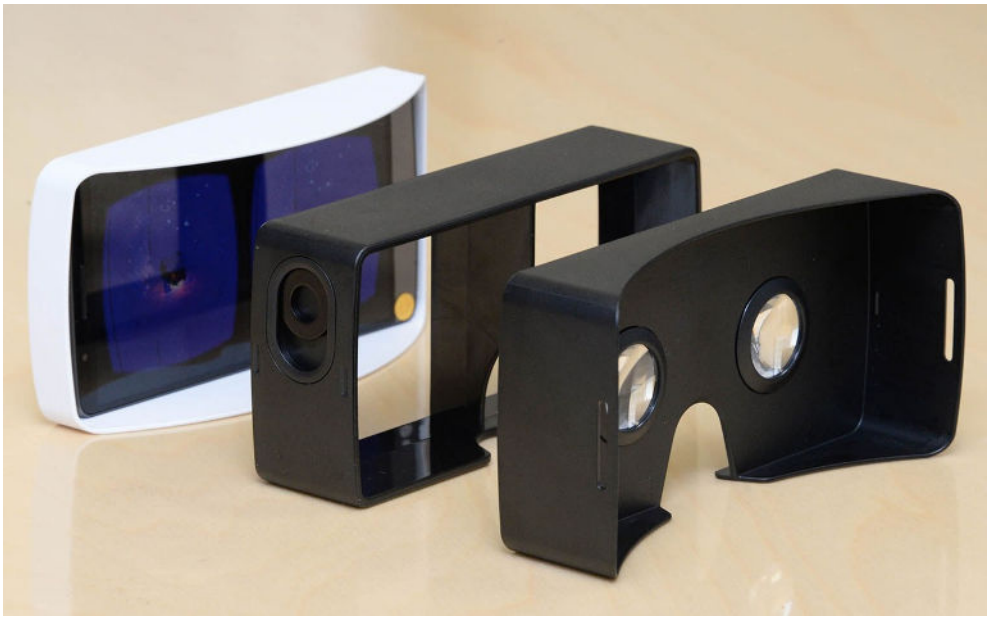
Samostatnou kapitolou je zábava. Grafické karty jsou nejvíce rozvinutou součástí počítače, a to hlavně díky vysoké poptávce. V současnosti se vymýšlejí

způsoby, jak zužitkovat vysoký výkon grafických karet v algoritmech, kde je snaha o jejich paralelizaci. Lze očekávat, že právě komerční směr virtuální reality zaměřený na zábavu umožní vývoj kvalitních souprav, ze kterého poté budou těžit i ostatní obory.

2.3 Android

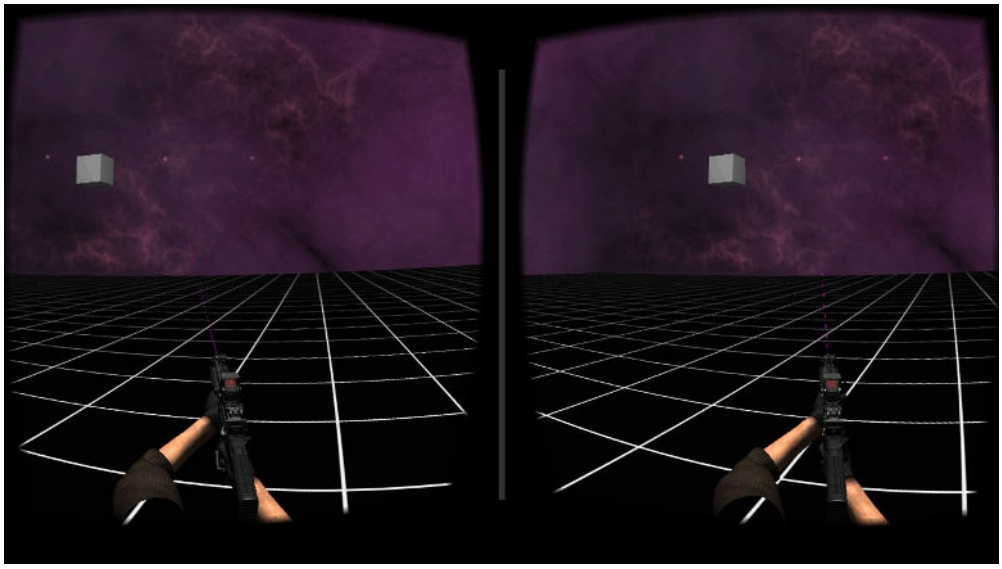
Nejde přímo o operační systém, ze kterého virtuální realita těží, ale jde o hardware, na kterém tento systém běží. Silný konkurenční boj velkých společností vyrábějících mobilní telefony a jejich mohutné investice mají za následek vysoký výkon a kvalitní displeje, ve spojitosti s jejich velikostí tvoří vhodné jádro pro brýle určené pro virtuální realitu.

Obrázek 5 ukazuje složení brýlí. V bílém rámečku je vidět vložený telefon. Dále vidíme střední část s tlačítkem a styčnou část, která se přikládá k obličeji, obsahující dvě konvexní čočky, které umožňují oku zaostřit na blízkou vzdálenost. Obrázek 6 ukazuje, že displej musí být rozdělen na dvě poloviny tak, aby zobrazení odpovídalo vždy jednomu oku. Renderovací kamera musí být pro každou polovinu posunutá o pár centimetrů ve virtuální scéně a tím se dosáhne stereoskopického vidění, které je pro člověka přirozené.



Obrázek 5: Skladba virtuálních brýlí s Android telefonem[2]

[2] http://www.lgnewsroom.com/2015/02/lg-g3-and-google-cardboard-bring-mobile-virtual-reality-to-everyday-life/



Obrázek 6: Zobrazení telefonu je rozděleno na dvě podobné části určené každému oku zvlášť

2.4 Požadavky

Hardwarové požadavky jsou přímo úměrné tomu, co se zrovna zobrazuje, ale virtuální realita si nárokuje mnohem více než běžné zobrazení na monitoru.

2.4.1 Obnovovací frekvence obrazu

Nejdůležitějším faktorem je udržení stabilních 60 snímků za sekundu. Tento požadavek se jeví jako velmi přísný, ale ve skutečnosti se jedná o absolutně nejnižší hranici pro používání. Bez dodržení stabilní maximální obnovovací frekvence mobilních displejů je vnímání virtuálního světa značně nepohodlné a pozorovatel může trpět nevolností. Nevolnost je způsobena podobně jako při cestování v autě: lidské rovnovážné ústrojí vnímá pohyb, ale člověk se vůči autu nehýbe, tím dochází k odlišnému vnímání, které mozek není schopný na podvědomé úrovni zpracovat.

Pokud displej telefonu zobrazuje i na tak krátké časové okamžiky stejný obraz, ačkoliv člověk například otáčí hlavou, podvědomí začne vnímat nesrovnalosti a může přijít nevolnost.

2.4.2 Věrohodné snímání pohybu

Ve virtuální realitě potřebujeme znát údaje o pohybu hlavy uživatele. Tento pohyb v programovacím prostředí chápeme jako dva vektory. Jeden vektor určuje pozici hlavy uživatele vztaženou k nehybné soustavě a druhý vektor reprezentuje směr pohledu. Je důležité, aby snímání pohybu bylo co nejpřesnější. Není možné, aby poloha v prostoru nebo v rotaci neodpovídala skutečnosti, protože uživatel pozná nesrovnalosti.

Snímání v prostoru je výpočetně i hardwarově náročná záležitost a ještě nebylo nalezeno optimální řešení. Provozování virtuální reality na méně sofistikovaných zařízeních často nesnímá pohyb v prostoru, ale jen rotaci hlavy. Při tomto přístupu by se uživatel neměl v prostoru pohybovat, je také nutné uplatit model krku (více v kapitole 3.3.1 Sledování hlavy).

2.4.3 Měřítko

Pro zachování uvěřitelnosti virtuální scény musí všechny objekty působit takovým dojmem, jaký uživatel očekává i v reálném světě.

Velikost

Faktor, který by neměl být opomíjen ve vývoji aplikací s virtuálním prostředím, je zachování rozumného poměru objektů. I přes neurčitost tohoto termínu je jasné, že například budova musí na člověka působit určitou velikostí. Kdyby budova byla příliš malá, působení scény na uživatele nebude realistické. Obdobně si lze představit situaci, kdy uživatel prochází budovou a dveřmi, které by nepůsobily na uživatele přirozeně. Například kdyby uživatel procházel v polovině výšky, pokazí se dojem realističnosti scény.

Neplatí striktně, že všechny velikosti objektů musí být pro člověka přirozené. Při virtuální simulaci mravencova života se z lidského hlediska bude vše jevit obrovské, ovšem poměr mezi zobrazovanými objekty musí vycházet z reálného prostředí. V projektu z praktické části jsem se dlouhou dobu snažil najít optimální velikost budovy, aby působila na uživatele přirozeně, ale nebylo možné najít ideální velikost. Problém jsem objevil ve velikosti stromů okolo budovy, které byly moc malé a tím zkreslovaly celou scénu.

Rychlost

Při tvorbě různých scén bylo vypozorováno, že rychlost pohybu objektů může působit ve virtuálním světě stejně nepřirozeně jako na běžné obrazovce. Velmi odlišné je vnímání rychlosti osoby, která ve virtuální realitě reprezentuje tělo uživatele. Zatímco je vhodné, aby uživatel měl ve virtuální realitě model svého vlastního těla, i když nebude odpovídat pohybům uživatele, není vhodné upravovat rychlost jeho pohybu. Při nasazených brýlích je uživatel daleko citlivější na rychlost pohybu jeho postavy, oproti zobrazení na klasickém monitoru. Uživatel vnímá zvýšenou rychlost jako průlet, a proto

by měl mít při větších rychlostech okolo sebe nějaký dopravní prostředek tak, jak by tomu bylo ve skutečnosti.

Důsledek

I když se přechod do virtuálního prostředí jeví pouze jako úprava stylu renderování a namapování vstupů, většina her je po tomto přechodu velmi nevěrohodná, jelikož porušují stanovená pravidla.

2.5 Google Cardboard

Google Cardboard je kartonové řešení držáku, který obsahuje optické čočky a do kterého se vsune telefon. Výsledkem je krabička, se kterou lze v omezené míře pozorovat virtuální realitu. Toto řešení je dostatečné pro ukázku, že už standardní mobilní telefon je schopen zobrazovat virtuální prostor. Tato ukázka slouží jen pro motivování lidí a v žádném případě není možné toto řešení používat nebo srovnávat s dedikovaným řešením. Google Cardboard je možné si zakoupit za cenu od 3 dolarů přes internet, nebo si ho lze vyrobit podle mnoha dostupných návodů.

Praktická část bude vypracovávána pro použití touto metodou kvůli jednoduchosti a snadné dostupnosti.

Obrázek 7: Google Cardboard[3]

2.6 Komerční řešení

V době odevzdávání této práce je situace na trhu taková, že existuje mnoho tzv. Google Cardboard klonů, několik kvalitních dedikovaných řešení, a několik dalších očekávaných.

[3] http://www.irishtimes.com/business/technology/1.2168860

Google Cardboard klony jsou funkčně identické s kartonovým řešením popsaným výše, ale jsou vyrobeny z lepších materiálů. Pro konstrukci je použit plast, mnohdy postačuje výroba 3D tiskárnou, a také čočky jsou kvalitnější a s větším průměrem. Výhodou bývá také kvalitní a měkký materiál, který se stýká s obličejem uživatele. Mezi nejkvalitnější, ale také nejdražší klon patří HOMIDO VR[4] headset, který je kompatibilní s velkou řadou telefonů.

Dedikované zařízení nazvané GearVR nabízí firma Samsung, prozatím jen pro několik málo určených telefonů[5]. Toto zařízení má výhodu, že všechny podporované telefony mají vysoký výpočetní a vykreslovací výkon, a tedy vytvořené aplikace mohou obsahovat komplexnější prostředí. GearVR má kvalitní čočky a také vlastní orientační senzory jako je gyroskop, akcelerometr a elektromagnetický kompas, které dosahují větší přesnosti, než ty zakomponované v telefonu. Software podporovaných telefonů také implementuje některé optimalizační metody popisované v kapitole 4 Optimalizace.

Další zařízení pro virtuální realitu už nevyužívají telefon jako výpočetní ani zobrazovací jednotku. Hlavními produkty jsou Oculus Rift, který pomocí serveru KickStarter[6] v roce 2012 započal moderní vývoj VR headsetů. V současnosti je to také nejkvalitnější dedikované zařízení, hodnotíme-li podle ergonomie a přesnosti sledovacích senzorů. Druhý produkt se nazývá HTC Vive, a v současnosti je to jediná konkurence. Tento headset je v mnoha ohledech podobný s Riftem, ale výhodou je přibalené ovládání, jehož poloha je také sledována a přenesena do virtuálního prostředí. Obě zařízení musí být propojena kabelem se stolním počítačem, který zpracovává data ze senzoru, modeluje virtuální realitu a posílá obraz zpátky do headsetu k vykreslení.

V budoucnu očekáváme uvedení dalších dvou VR headsetů, které se zaměřují na odlišnou část trhu. První z těchto očekávaných zařízení je Sony Morpheus zaměřený na vlastníky konzolí PlayStation 4, které budou sloužit obdobně jako počítač v předcházejících případech. Posledním zmíněným zařízením je Sulon Q od firmy AMD, který je průsečíkem variant s telefonem i bez něj, protože i když se dovnitř nevkládá telefon, tak veškeré zobrazovací i výpočetní komponenty se nacházejí v zařízení a není nutné připojovat počítač či konzoli. Je to první kompletně dedikované zařízení.

---

[4] http://www.homido.com/
[5] Samsung Galaxy S7, Galaxy S7 Edge, Galaxy Note 5, Galaxy S6, Galaxy S6 Edge, Galaxy S6 Edge+
[6] https://www.kickstarter.com/ – Crowdfunding server

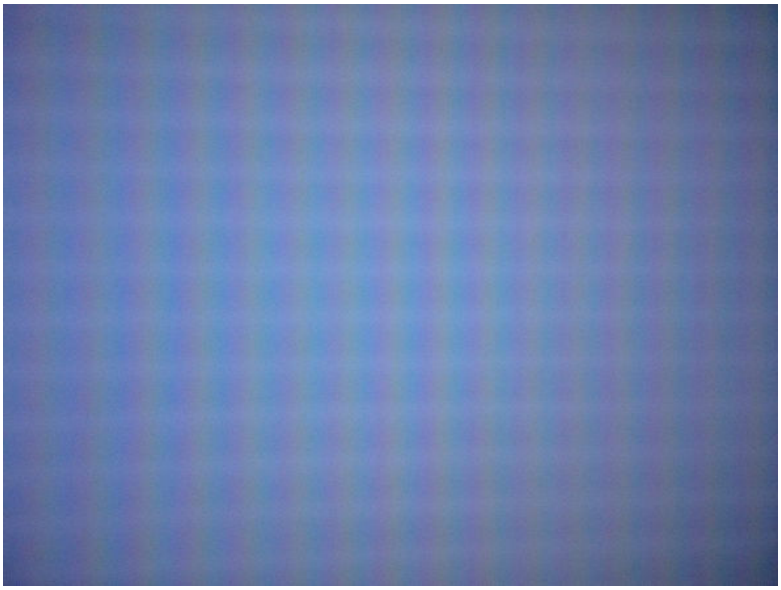
3 Technologie

3.1 Displeje

Displej je stejně důležitá část jako výpočetní výkon. Nebylo by možné se dlouhodobě dívat na obraz, který není příjemný na pohled. V současnosti mají nejlepší telefony Quad HD (2560×1440) displej, a to znamená 1280×1440 zobrazovaných bodů pro jedno oko. Z odhadů plyne, že pro jedno oko je potřeba alespoň 16K, to znamená 15360×8640 bodů, což je celkem přes 265 milionů pixelů. Tato hranice je pro dnešní obrazovky a výpočetní výkon v blízké budoucnosti nereálná, a proto dochází k několika nepříjemnostem.

Malé rozlišení

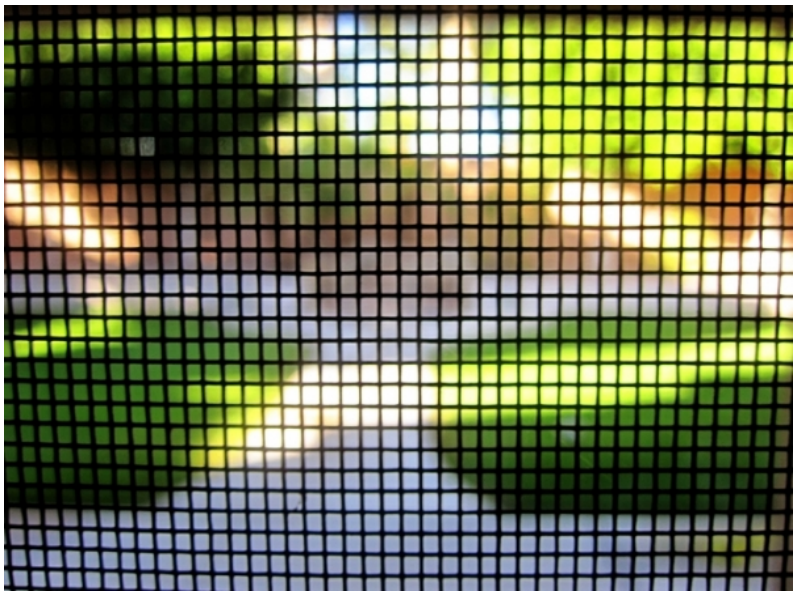
Je třeba si uvědomit, že čočka v brýlích slouží jako lupa, a proto se opět stávají viditelné jednotlivé subpixely. Vezměme rozlišení FullHD (1920×1080); na televizích je toto rozlišení dostatečné, na telefonech více než dostatečné, ale virtuální brýle zabírají větší úhel zorného pole, a proto je potřeba větší hustota pixelů. Nízká hustota pixelů znamená, že jednotlivé pixely musejí zabírat více plochy. Ze znalosti konstrukce pixelů vyplývá, že při větším přiblížení budou jednotlivé složky rozeznatelnější jedna od druhé, a kvůli tomu jim bude zabráněno ve vytvoření jedné plné barvy. Obrázek 8 znázorňuje velké přiblížení obrazu pod čočkou, a je z něj patrný rozklad obrazu na jeho konstrukční prvky.



Obrázek 8: Viditelné subpixely

Mezera mezi pixely

Druhým rušivým faktorem je výrobní technologie displejů. Jednotlivé pixely nejsou těsně vedle sebe, a proto vzniká šum s pevným vzorkem (anglicky nazýváno „screen door effect“). Tento šum se projevuje jako černá pravidelná mřížka. Při delším používání tento jev mozek přestane vnímat jako rušivý.



Obrázek 9: Screen door efekt[7]

Krátké přetrvání

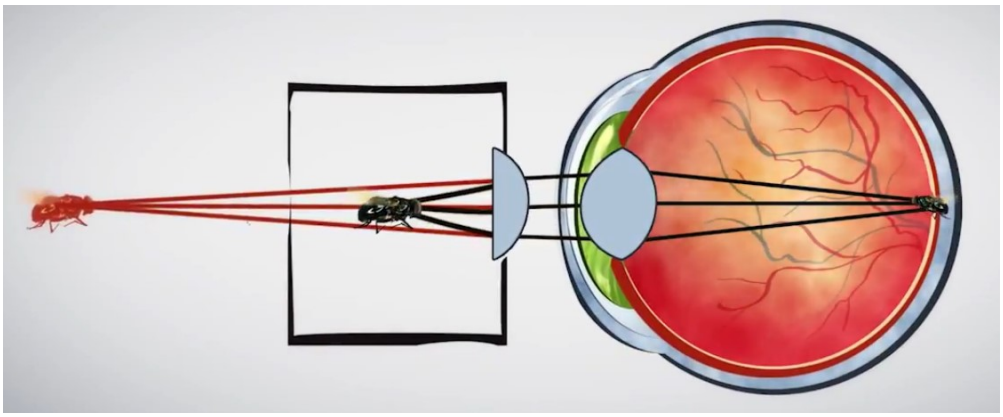
Paralelně se musíme zabývat i časovou složkou zobrazení. Minimální hodnota zobrazovací frekvence 60 snímků za sekundu znamená jeden obrázek každých zhruba 16 milisekund. Výrobci komerčních řešení odhadují optimální hodnotu obnovovací frekvence na 1000 obrázků za sekundu. Opět současná technologie displejů, výpočetní výkon ani rychlost přenosových kabelů takový datový tok neumožňují, a proto v některých displejích byla implementována funkce nazvaná „Low persistence“.

Displej s technologií „Low persistence“ zobrazí vyrenderovaný obraz jen na 1-2 milisekundy a na zbytek doby zhasne. Na zpomaleném záběru by tento displej vypadal jako blikající, pro lidské vnímání je ale lepší, když obraz zhasne, než když je delší dobu špatný obraz, například při posouvání nebo otáčení hlavou, i když se jedná „jen“ o desítky milisekund.

[7] https://atomicsupermen.wordpress.com/2016/02/29/samsung-gear-vr-my-critical-view-of-it-and-vr/

3.2 Čočky

Jelikož je obrazovka jen pár centimetrů před očima, lidské oko není schopné na tuto vzdálenost zaostřit, a proto se mezi obrazovku a oči umisťují konvexní čočky. Obrázek 10 ukazuje, jak čočka oddaluje objekt na displeji. Černé čáry znázorňují skutečnou dráhu světelných paprsků a červené čáry naznačují, kde by se objekt nacházel při klasickém vnímání světa bez čočky. Černé paprsky mezi okem a čočkou jsou téměř rovnoběžné, a to znamená, že oko je v uvolněném stavu, podobně jako kdyby se dívalo do dálky, tudíž nedochází k takovému namáhání očních svalů jako při používání klasického monitoru.[9]



Obrázek 10: Ostření očí přes čočku [9]

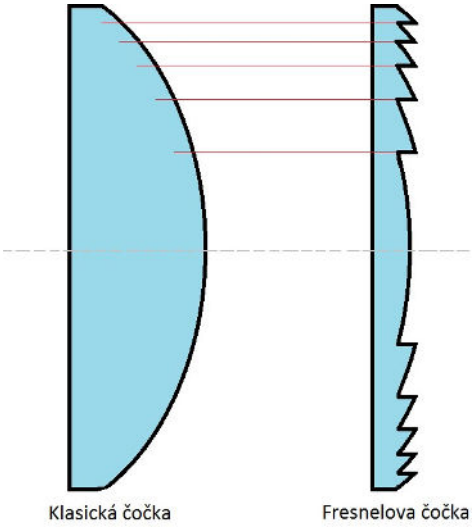
3.2.1 Typy čoček

Normální čočka

Klasická konvexní čočka láme rovnoběžně procházející světlo ke svému středu, kdežto světlo procházející středem svoji dráhu nemění. Místo, kde se všechny paprsky protínají, se nazývá ohnisko.

Fresnelova čočka

Fresnelova čočka má díky svým vlastnostem, zejména nízké hmotnosti, vhodné uplatnění v brýlích pro VR, neboť je snaha co nejvíce snížit váhu zařízení upevňovaného na hlavu. Konstrukce této čočky nezachovává povrch v celku, ale je rozložen na jednotlivé části. Sklon oblouku čočky je zachován jako u nedělené čočky v každém segmentu, jak ukazuje Obrázek 11. Tato čočka není úplně vhodná pro zobrazovací techniku, jelikož vznikají nepatrné kruhy v místě odskoku, a proto specializované firmy vytvářejí různé proprietární modifikace jako například úprava sklonu odskoku, k co možná největší redukci negativních projevů této čočky.



Obrázek 11: Rozdíl mezi klasickou a Fresnelovou čočkou v řezu

3.2.2 Negativa

Vše je ostré

Čočky vždy oddalují oblast vnímání displeje tak, že se oči zaměřují do nekonečna, uživatel tímto způsobem pozoruje i předměty, které se nacházejí blízko. V přirozeném prostředí mozek určuje vzdálenosti objektů podle relativní polohy oproti pozadí v každém oku a podle míry zaostření oka. Jelikož zaostření je vždy do nekonečna, mozek určuje polohu předmětu vždy jen prvním způsobem, a to může způsobovat menší nepohodlí v očích, zejména u nezkušených uživatelů.

Chromatická aberace

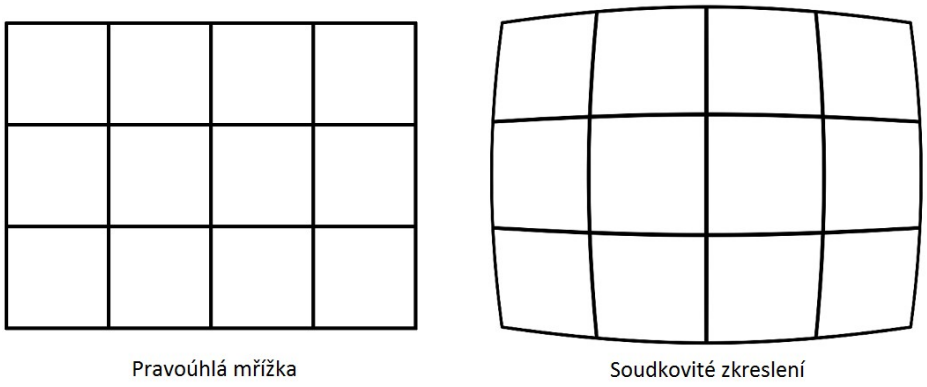
Světlo procházející čočkou se rozkládá podobně jako při průchodu hranolem na barevné složky. Tento jev má za následek barevné „roztřepení“ světlých hran, které postupně narůstá ve směru od středu čočky. Displej generuje obraz pomocí tří základních barev a kvůli různému lomu světla každé barevné složky výsledný obraz vypadá jako složený ze tří překrytých barevných obrazů. Tento defekt lze snadno vyrušit softwarovým filtrem, který předpokládá zkreslení čočkou a namapuje subpixely s negativním odstupem.

Obrázek 12: Ukázka obrázku[8], včetně chromatické aberace a odstranění pomocí filtru

Obrázek 12 ukazuje tři různá zobrazení stejného obrazu. Levá část je klasické zobrazení, přičemž takové zobrazení očekáváme i po průchodu čočkou. Pokud ale na displeji zobrazíme tento obraz, a podíváme se na něj skrze čočku, uvidíme obraz nacházející se uprostřed. Čím dál se díváme od středu, tak je rozklad barev patrnější. Softwarový filtr tedy předpokládá, jak se barvy budou postupně vzdalovat od sebe, a namapuje je na zobrazovaný obraz v opačném pořadí, což můžeme vidět vpravo. Po průchodu takového obrazu čočkou výsledný obraz odpovídá původnímu obrazu vlevo. V *příloze A* na konci práce je program, který tuto chromatickou aberaci simuluje nebo případně dokáže odstranit, podle toho, v jakém pořadí barevné složky oddalujeme od středu.

Zkreslení

Další vadou pohledu přes čočku je „zkulatění“ obrazu, které se odborně nazývá „soudkovité zkreslení“. Toto zkreslení je nejvýraznější v rozích obrazu. Tato vada je opět snadno řešitelná aplikováním jednoduchého filtru, který odečte zkreslení před zobrazením.



Obrázek 13: Soudkovité zkreslení po průchodu čočkou

[8] Ze zdroje https://processing.org/tutorials/pixels/

3.3 Snímání pohybu

Ve virtuální realitě máme tendenci se zbavovat veškerých rušivých elementů. Ačkoliv máme potřebu pohybu ve VR, řešení pomocí šipek na ovladači nebo klávesnici, když máme na hlavě headset, není zdaleka optimální, a proto snímáme pozici uživatele, podle které zobrazujeme prostředí [4].

Rozlišujeme dva základní celky pro sledování. Důležitějším je sledování polohy hlavy, a tedy i displeje, a méně důležitým je sledování polohy zbytku těla.

3.3.1 Sledování hlavy

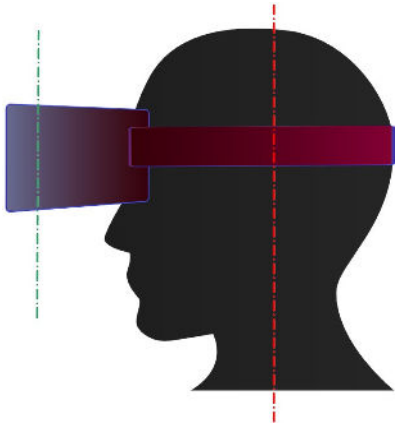
Při sledování hlavy rozlišujeme sledování natočení a sledování pozice v prostoru. Sledování natočení probíhá pomocí senzoru nazývaného gyroskop.

Elektronický gyroskop má problém vycházející z jeho konstrukce, a to je slabé konstantní klouzání, které mírně mění svou intenzitu a směr z dlouhodobého hlediska, a proto není snadné ho softwarově natrvalo odrušit. Pro nejsnadnější odrušení se gyroskop kombinuje s elektromagnetickým kompasem. Gyroskop zachovává přesnost a rychlou odezvu měření a kompas přispívá pevným bodem v prostoru, díky kterému lze rozeznat klouzání. Nejspolehlivějším způsobem, jak odrušit negativní jevy gyroskopu, je kombinace s polohovým snímáním popsaným dále.

Model krku

Model krku (anglicky též Neckmodel) je způsob kompenzace absence polohového snímače. Předpokládá se, že uživatel stojí rovně nebo sedí na točící se židli a točí se pouze okolo své osy. Toto točení neodpovídá otáčení renderovací kamery, protože osa otáčení v realitě prochází krkem člověka, a osa kamery je umístěna ve středu obrazu viz Obrázek 14. Červená osa na obrázku je správné místo, okolo kterého má renderovací kamera obíhat v závislosti na údajích z gyroskopu. Zelená osa znázorňuje špatný přístup při aplikování výstupu gyroskopu jako hodnoty úhlu kamery.

Tento model je použit u mobilních zařízení ze dvou důvodů. Mobilní zařízení nejsou vázána na konkrétní místo a pevně umístěná snímací stanice odstraňuje mobilnost. Metoda snímání pohybu okolí je výpočetně náročná a při využití současných algoritmů není dostatečně přesná.



Obrázek 14: Silueta s osou krku a osou obrazovky

Snímací stanice

Snímací stanice hraje roli nehybného pozorovatele a určuje, kde se zobrazovací zařízení v prostoru nachází. Stanice bývá nejčastěji kamera nebo systém kamer a na zařízení je umístěna snadno rozeznatelná značka. Značku může tvořit například soustava infračervených diod, a pro co nejmenší nutnost předzpracovaní obrazu kamera bude citlivá jen na infračervené světlo. V tomto ideálním případě máme soustavu bodů a časově nenáročným algoritmem zjistíme jejich vzájemnou polohu, ze které lze vypočítat pozici zařízení v prostoru.

V této situaci nás můžou obtěžovat dvě věci: odraz v zrcadle, který můžeme vyřešit nepravidelnou strukturou značky. A také zakrytí značky třeba odvrácením od kamery nebo rukou, které lze částečně kompenzovat použitím více kamer a více rozlišitelných značek.

Snímač je na zařízení

V tomto případě má zařízení v sobě zabudovanou kameru nebo čidlo. Jde-li o kameru, pak jde o náročný a ne zcela vyřešený problém rozpoznání obrazu. Rozpoznání obrazu si lze zjednodušit umístěním značek na zdi, případně i podlahu a strop. Značky mohou být například zvětšené QR kódy, ovšem umisťování takovýchto značek je vhodné maximálně v nějaké dedikované místnosti nebo laboratoři, ale pro mnoho lidí není možné takovýto přístup standardně používat.

Při použití čidla je určujícím faktorem zvolená měrná hodnota, nejčastěji se používají tzv. majáky, které vysílají některou frekvenci z elektromagnetického vlnění, ze kterého lze spočítat pozici triangulací. Jako náhradu vlnění lze použít kalibrované laserové záblesky s konstantním krokováním nebo magnetické pole.

GPS

Obecně je systém GPS považován za velmi nepřesný. Za tuto nepřesnost ale částečně můžou výrobci čipů určujících polohu a nepříznivé podmínky. V každém GPS čipu probíhá triangulace a predikce. Zatímco díky triangulaci se určuje poloha správně, použitím predikce se přesnost snižuje. Predikce je výhodná, když je signál GPS slabý, a je možné, že byl dokonce odražen například od okolních budov nebo objektů.

Společnost Samsung společně s University of Texas oznámila, že pracují na technologii zvané „GPS přesný na centimetr“. Využití této technologie je možné pouze mimo budovy. V budoucnu si díky této technologii lze představit současně se pohybující uživatele na velké ploše pozorující totéž virtuální prostředí s modely postav reprezentujícími ostatní uživatele. Již v současnosti existují možnosti tak přesného zaměření, ale jedná se o drahé zařízení s velkými uspořádanými anténami. Novou technologií jde dosáhnout stejné přesnosti za použití levných antén, které jsou již v současnosti v mobilních zařízeních přítomny. [13]

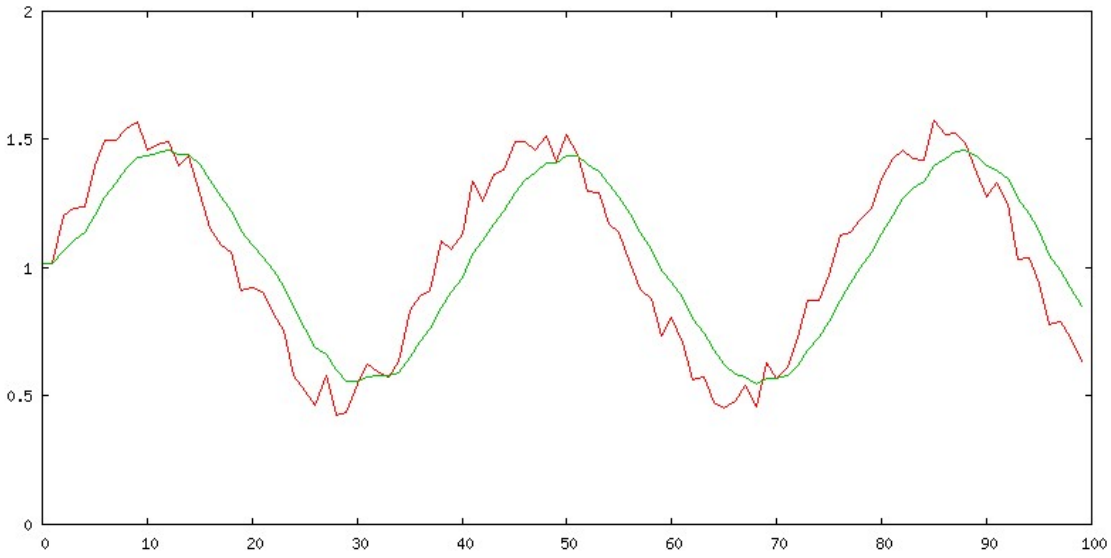
Kalmanův filtr

Kalmanova filtrace je matematický algoritmus pro filtraci zarušeného signálu vyznačující se schopností získat čistý signál i bez znalosti způsobu a míry rušení. Je tedy vhodná pro použití v mobilních zařízeních, protože jednotlivé senzory nejsou nejvyšší kvality a obsahují systémové chyby měření. Některé chyby mají konstantní velikost a je snadné jejich působení vyrušit. Některé chyby konstantně pouze vypadají, a například u senzoru typu gyroskop je chyba konstantní jen po určitou dobu (například jedna hodina až jeden den), a to znamená potřebu kalibrování před každým použitím nebo častěji.

Jakmile odrušíme všechny konstantní a snadno predikovatelné chyby, musíme se vypořádat s nepravidelným a náhodným šumem, který je přítomný u všech méně kvalitních zařízení nebo vychází z principu měření.

Kalmanův filtr sleduje naměřené hodnoty systému a vypočítává jejich trajektorie. Veličiny naměřených hodnot jsou směr, rychlost, zrychlení, střední hodnota chyby a rozptyl chyby, které se určují z několika předešlých měření, a to vytváří tzv. model systému. Tento model poté predikuje hodnoty systému v novém čase (v dalším měření) a porovná je se skutečně naměřenými hodnotami. Tento model poté určí několik chybových pásem anebo takovou funkci, která zohledňuje pravděpodobnost chybného měření způsobeného náhodným šumem a odhadne skutečný stav systému.

Obrázek 15 ukazuje naměřené hodnoty znázorněné červenou křivou původně sinusové funkce s náhodnou chybou při měření. Zelená křivka vznikla aplikací Kalmanova filtru na červenou křivku, přičemž daleko přesněji reprezentuje původní funkci. Můžeme si povšimnout, že zelená křivka je mírně zpožděná v x-ové ose, která znázorňuje čas, a proto nastavení Kalmanova filtru je kompromisem mezi jeho aproximační přesností a časovou prodlevou.



Obrázek 15: Aplikace Kalmanova filtru na signál s neznámým rušením

3.3.2 Snímání rukou

Obecně považujeme za první vstup do virtuální reality samotný pohyb hlavy. I když model hlavy může interagovat s herním prostředím a může například rozrážet zdi nebo posouvat krabice, není to postačující. Nabízí se přidat základní ovládání herním ovladačem používaným u konzolí. Ve virtuální realitě se snažíme dosáhnout co možná největší přirozenosti, ale ovladač značně kazí pocit přítomnosti. Nejdůležitějším vstupem do virtuální reality tak jako v normálním životě jsou naše ruce, a proto se je do virtuálního prostoru snažíme promítnout. Rozlišíme zde dva základní principy, kde jeden z nich zachovává prvek ovladače a druhý ne.

Zachování prvku ovladače

Můžeme si představit rozpůlení klasického ovladače tak, aby se obě ruce mohly pohybovat nezávisle na sobě. Pro každý ovladač zvlášť je zapotřebí implementovat sledování jeho pohybu dle výše uvedených metod. Sledování ovladačů nemusí mít takovou frekvenci jako sledování hlavy, ale je třeba zaručit, že se ovladače nebudou

virtuálně vzdalovat, kvůli konstantní chybě ve sledování, od sebe, ani od žádného jiného sledovaného zařízení, hlavně toho zobrazovacího.

Na ovladači se nacházejí tlačítka, dotykové plochy a páčky, pomocí kterých interagujeme ve virtuálním prostředí v souladu se snímači polohy. Fyzickými pohyby ukazujeme na objekty a pomocí ploch a páček volíme operace. Snímání ovladače znázorňuje směr dlaně, ale neříká nic o poloze prstů, a proto se aplikuje více sofistikovaný přístup. Tlačítka kromě klasické funkce zmáčknuto/nezmáčknuto mívají senzor míry přitlačení, navíc bývají konstruovány z elektrovodného materiálu, který pozná, že se prst na tlačítku nachází, i když je v nezmáčknuté poloze, a dotykové plochy také vypovídají o poloze prstů, a proto lze vymodelovat ruce ve virtuální realitě alespoň orientačně.

Mezi nejvýznamnější komerční ovladače patří Oculus Touch a HTC Vive Sticks. Jejich tvar značně přesahuje rozsah uchopení lidské ruky, aby nedocházelo k zakrývání senzorů.

Obrázek 16: Ovladač Oculus Touch[9]

Obrázek 17: Ovladač od HTC Vive[10]

Promítnutí rukou

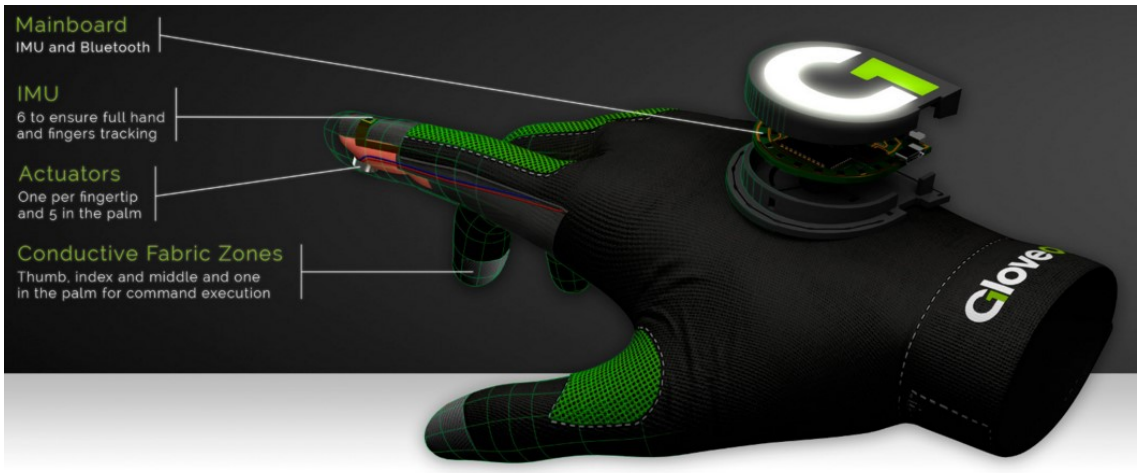
Ačkoliv ovladač umí rozeznat několik poloh ruky, nejde o zcela kompletní snímání, a proto existuje tendence přenést kompletní informace o rukách do virtuálního světa. Jelikož svět virtuální reality je stále ve vývoji, ani zde není rozhodnuto o nejlepším ani uspokojivém řešení, ale vyvíjí se dva přístupy.

První přístup je založen na počítačovém rozpoznání obrazu. Rozpoznat přesnou polohu rukou je složitý úkol, a proto obecně není znám software, který by mohl být

Obrázky jsou z oficiálních stránek:
[9] https://www.oculus.com/en-us/touch/
[10] https://www.htcvive.com/eu/

masově používán, ale když výpočetní výkon postoupí, budou moci být aplikovány náročnější algoritmy a otázkou zůstane, kam umístit kameru. Nabízí se kombinace jedné kamery upevněné v prostoru a druhé kamery umístěné na VR brýlích, která má přímý výhled na ruce uživatele; v případě, že kamera na ruce neuvidí, nebude ani zobrazen jejich model, a tedy nebude vyžadováno tak přesné trasování a funkci převezme vnější kamera. Komerční příklady jsou Leap Motion nebo Intel RealSense.

Nepatrně odlišný přístup je nasazení si speciálních rukavic, které mají tahové a tlakové senzory, podle nichž určíme polohu prstů, a v kombinaci s trasováním zápěstí získáme úplnou informaci a pohybu a poloze rukou.



Obrázek 18: Obrázek komerčního produktu GloveOneVR z oficiální stránky

Nevýhodou přístupu bez ovladače jsou doslova prázdné ruce. Pro člověka neexistuje zpětná vazba, když drží virtuální předmět a tedy může cokoliv promáčknout, a i když se snaží něco držet, tak do mozku přicházejí odporující si informace z očí a z hmatu.

3.3.3 Snímání nohou

Metody snímaní pohybu těla dělíme do dvou kategorií. Pro snímání pohybu nohou dnes již existují uspokojivá řešení, zatímco snímání celého těla je v dnešní době ve fázi raného vývoje.

Pro snímání nohou používáme „omnidirectional threadmill“ (všesměrový běžecký pás), nejčastěji nepohyblivý. Existuje několik komerčních řešení, kde každé využívá jiný princip kompenzace pohybu na místě. Všechna uvedená zařízení jsou v době psaní práce teprve ve fázi předprodeje a hodnocení je psáno podle dostupných informací na oficiálních stránkách a podle reportů z technických veletrhů.

Virtuix Omni

První představené zařízení se skládá ze 140 cm širokého konkávního podstavce a obruče ve výšce pasu, která zabraňuje uživateli spadnout. Nutností je použití speciálních bot, které mají dvě funkce. První funkcí bot je zaručení nízkého tření mezi nohami a podstavcem, a proto jsou jejich podrážky vyrobeny ze speciálního materiálu. Během používání se boty i nášlapná plocha vyhlazují a tření se ještě více snižuje. Druhou funkcí, kterou boty zastávají, je snímání pohybu. Na patách je laser a laserový senzor, který funguje podobně jako u počítačové myši, a je tedy zaznamenáván analogový pohyb pro každou nohu zvlášť.

Negativem tohoto zařízení je hlučnost, jelikož jde o plastový výrobek. Při používání vzniká hluk o síle 70–75 decibelů, což odpovídá hlučnosti běžného běžeckého pásu. Dalším problémem je nepřirozenost pohybu, jelikož noha, která jde dopředu, se dotkne konkávního povrchu o něco dříve než by tomu bylo na rovině, a také zbytek kroku přední noha klouže dolů. Na tento rozpor si lidské tělo zvykne asi po půl hodině používání, ale chůze nikdy nebude přirozená.

Obrázek 19: Obrázek zařízení Virtuix Omni z oficiální stránky[11]

[11] http://www.virtuix.com/

Cyberith Virtualizer

Druhé zařízení už nemá konkávní podstavu, ale rovnou s nízkým třením. Obruč je tentokrát připevněná k pasu uživatele a pohybuje se nahoru a dolů zároveň s ním, což umožňuje rozpoznat další pohyby a postoje, jako je skákání, klečení, dřepění a sedění. V tomto případě je pohyb realizován umělým klouzáním nohou po podstavě s tím, že tělo je zapřené o obruč v pase. Klouzání nevyžaduje speciální obuv, ale používají se hrubší ponožky, případně výrobce nabízí vhodné ponožky, které lze přetáhnout přes obuv.

Toto klouzání také není přesná simulace lidského pohybu, ale výhodou je nízká hlučnost a žádná doba učení. Snímání pohybu je opět analogové, a tentokrát je snímač konstruován stejně jako rezistivní dotykové obrazovky. Toto snímání logicky probíhá jen při klouzání a ne když se noha pohybuje dopředu ve vzduchu, a proto jsou na stojkách další optické senzory, které vyhodnocují i tento pohyb.

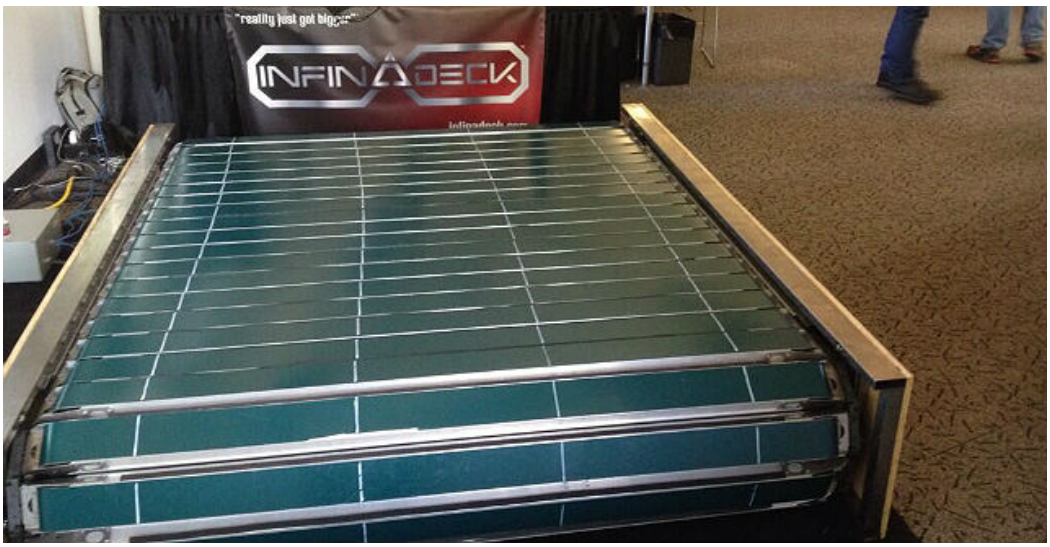
Obrázek 20: Obrázek zařízení Cyberith Virtualizer z oficiální stránky[12]

[12] http://cyberith.com/

InfinAdeck

Toto zařízení používá aktivní posun podstavy, což se projevuje na jeho plánované ceně, která je násobná v porovnání s předchozími modely. Podstava se skládá z gumových pásů, které rotují pro pohyb v jedné ose a celé pásy poté rotují okolo jádra zařízení, což umožňuje pohyb ve směru kolmém k předchozím. Složením těchto pohybů ve dvou osách můžeme dosáhnout jakéhokoliv vektoru pohybu v horizontální rovině. Pohyb na tomto zařízení je prakticky přirozený. Umožňuje chození v kruzích i běh. Zařízení využívá pouze jeden senzor, který je připevněn k pasu a snímá náklon těla.

Problémem je vysoká hlučnost motorů a velikost, což jsou důvody, kvůli kterým je pravděpodobné, že si lidé toto zařízení nikdy nebudou kupovat domů a cena se nebude snižovat. Nicméně to neznamená, že si toto zařízení nenajde své místo na trhu.



Obrázek 21: Fotografie InfinAdeck z veletrhu CES2015

3.3.4 Snímání celého těla

Snímání celého těla je složitá záležitost, neboť se setkáváme s mnoha problémy. Postavení těla reprezentujeme virtuální kostrou, mnoho kloubů na lidském těle má za následek širokou možnost konfigurací kostry. Z toho vyplývá, že je potřeba umístit senzor na každou samostatně pohyblivou část těla. Jelikož snímač senzorů bývá nejčastěji kamera nebo obdobné zařízení, tak se stává, že jedna část člověka zakryje jinou nebo je otočená zády, a proto je třeba ještě navýšit počet senzorů a/nebo snímačů, aby bylo za veškerých podmínek dostatečný počet senzorů vidět. Tyto sestavy jsou většinou konstruovány ve vývojových nebo filmařských studiích, a slouží pro získání věrohodných dat o pohybu k následné tvorbě animací.

Amatérsky lze dosáhnout velmi uspokojivých výsledků za použití různobarevných diod rozmístěných po těle a konstelaci čtyř kamer snímajících uživatele ze všech stran. Je potřeba napsat šikovný software, který po kalibraci umožní správně vyhodnocovat pozici ze všech čtyř obrazů a tím trasovat pohyb uživatele.

Komerční řešení

V tomto případě se již komerční řešení používají u herních konzolí Xbox od Microsoftu a PlayStation od Sony a jedná se o výrobky Kinect a PlayStation Eye. Obě zařízení využívají mírně odlišnou technologii, ale jejich funkce je stejná. Ať už se využívá ke snímání infračervené světlo s hloubkovým senzorem či jen čisté rozpoznání obrazu, výsledkem je kostra umístěná do 3D prostoru. Chybějící rozpoznávací značky (například LED diody a jiné) mají za následek horší rozlišitelnost snímaného subjektu a tím je výpočetní algoritmus náročnější, a proto rozlišení snímací kamery zatím bývá nízké, z čehož vyplývá i nízká přesnost, která nám v pokročilejší virtuální realitě nedostačuje.

Určitě je to ale technologie budoucnosti, protože nevyžaduje absolutně žádné periferie umístěné na uživateli, což je spojeno s vysokou mírou pohodlí, a to znamená o něco uvěřitelnější virtuální realitu.

4 Optimalizace

Z předchozích kapitol jednoznačně vyplývá, že virtuální realita má daleko větší požadavky na grafickou kartu i procesor. Renderovací náročnost je lineárně úměrná zvyšujícímu se požadavku na hodnotu obnovovací frekvence a exponenciálně úměrná zvyšujícímu se rozlišení. Nad koncem Moorova zákona o zdvojnásobování výkonu při zachování ceny se polemizuje již dlouho, ale je třeba si uvědomit, že zvyšování výkonu probíhá určitou rychlostí, a není možnost masově během krátké doby výrazně zvýšit výkon u koncových uživatelů, a proto musíme přemýšlet, jak snížit požadavky softwarovou cestou.

4.1 Méně příkazů k vykreslování

Abychom pochopili následující optimalizační metodu, je třeba si nejprve popsat renderovací techniky současnosti.

Hry se po načtení a inicializaci skládají ze základní smyčky

```
1.  while (gameRunning == true)
2.  {
3.      processInput();
4.      update();
5.      render();
6.  }
```

Nejprve v metodě processInput() zpracujeme všechny vstupy od uživatele, které sledujeme. V druhé metodě update() změníme stav světa v reakci na čase, interakci objektů a vstupů získaných v první metodě a nakonec přejdeme do poslední fáze, kde probíhá vykreslování, u nás v metodě nazvané render()

```
1.  void render()
2.  {
3.      setCamera();
4.      foreach model in models
5.      {
6.          pushToGraphicsCard(model.data);
7.      }
8.      invalidate();
9.  }
```

Metoda render() pracuje následovně. Nastavíme renderovací kameru, která reprezentuje pohled uživatele do prostředí, a poté iterujeme pole všech objektů nacházejících se v našem prostředí. V této iteraci se předávají údaje o objektech, jejich pozice a stav, dle kterého probíhá vykreslování. Toto místo je často nazýváno úzkým hrdlem, protože neprobíhají výpočty pro simulaci světa, procesor i grafická karta jsou

využity na relativně dlouhou dobu a navíc jsme omezeni tím, že s grafickou kartou může pracovat jenom jedno jádro procesoru. Metoda invalidate() nakonec vykreslí obraz složený ze všech viditelných objektů na obrazovku.

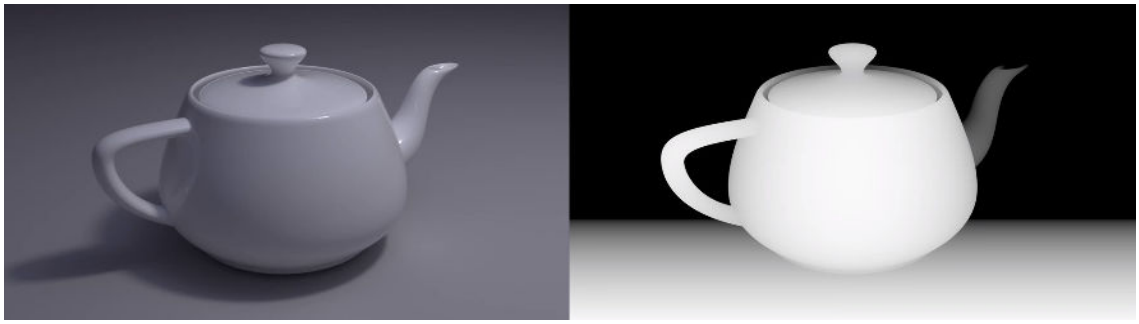
Uvažujeme-li hru pro virtuální realitu, pak původní smyčka platí s tím, že poslední metoda je zavolaná zvlášť pro každé oko, tedy pro každou polovinu obrazovky, což znamená téměř shodné vykreslení, jen s posunutou kamerou. Přirozeně se nabízí myšlenka, že metoda render() dělá zbytečnou práci, když musí znovu iterovat všechny objekty. Proto DirectX od verze 11 a OpenGL od verze 4 pro PC umožňuje ponechat buffer objektů a spustit vykreslování s jinou pozicí kamery. Díky tomuto přístupu šetříme procesorový čas. Plánované API Vulkán[13] a DirextX verze 12 navíc podporuje paralelní komunikaci více jader procesoru s grafickou kartou, což významně rozšiřuje zmíněné úzké hrdlo. Vulkán přenáší tyto výhody i na mobilní platformu Android.

4.2 Asynchronous Timewarp

Jeden z požadavků na virtuální realitu stanovený v kapitole „2.4 Požadavky“ se týkal velikosti obnovovací frekvence obrazu. Bylo řečeno, že minimální limit odpovídá maximální schopnosti dnešních mobilních displejů, a to je 60 snímků za sekundu. U dedikovaných displejů pro virtuální realitu jsou frekvence 90 (Oculus Rift, HTC Vive), 120 (Sony Morpheus) snímků za sekundu a lze předpokládat razantní růst.

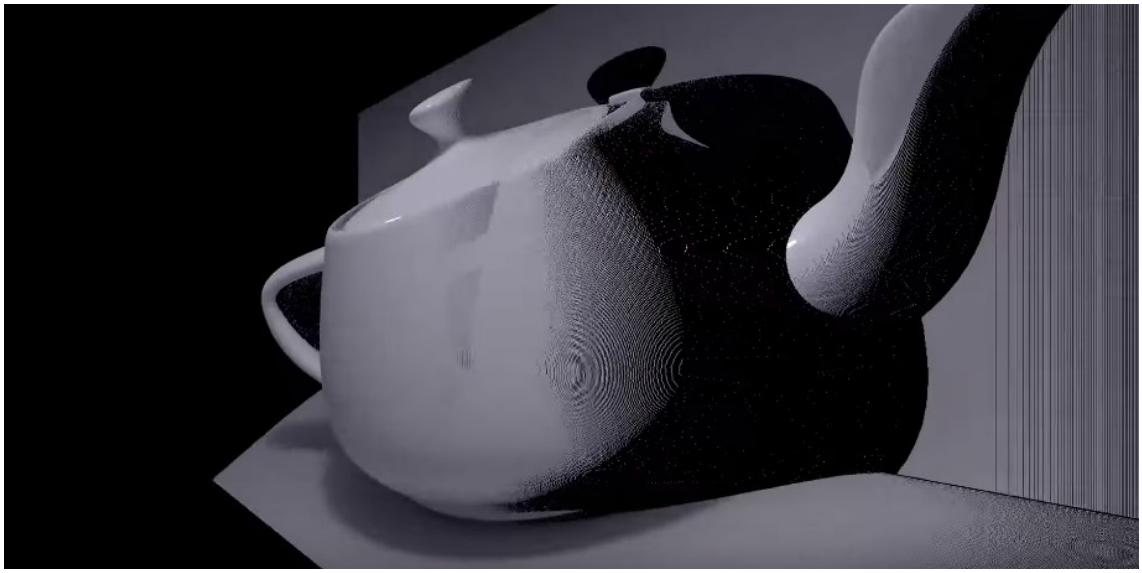
Společnost Nvidia představila experimentální displej s téměř nulovou dobou odezvy. Obnovovací frekvence obrazu tohoto displeje dosahuje 1700 snímků za sekundu a byl představen na veletrhu GTC 2016 [14]. Displej byl snímán vysokorychlostní kamerou pod lupou a byl na něm zobrazen text, který se nepohyboval relativně k prostředí, i když bylo s celým displejem pohybováno. Text zůstával na místě a bylo patrné pouze mírné rozostření. Takováto frekvence obnovení znamená potřebu nového snímku každých 588 nanosekund (asi půl jedné milisekundy), což je v současnosti nepředstavitelné pro cokoliv jiného než vypsání textu, a to je jeden z důvodů, proč byla vymyšlena metoda nazvaná Asynchronous Timewarp. Ke každému obrazu, což je matice pixelů, existuje při počítačovém renderování hloubková matice nazvaná Z-Buffer. Tato matice obsahuje pro každý bod na obrazovce vzdálenost mezi kamerou a skutečnou polohou zobrazovaného bodu.

[13] https://www.khronos.org/vulkan/



Obrázek 22: Renderovaný obrázek (vlevo) a jeho zobrazený Z-Buffer (vpravo)[14]

Při otáčení hlavou se jednotlivé předměty v okolí pohybují po obraze různou rychlostí, platí úměra, že čím je objekt blíž, tím se pohybuje pomaleji, a čím je dál, tím rychleji. Díky této znalosti můžeme jednotlivé pixely posouvat jejich přirozenou rychlostí založenou na vzdálenosti od pozorovatele. Touto metodou můžeme doplnit pouze snímky, během kterých nedošlo k výraznému pohybu kamery, jelikož informací o obraze není mnoho a při větším posunu se předměty jeví, jakoby vrhaly stín prázdnoty, jak ukazuje 23.



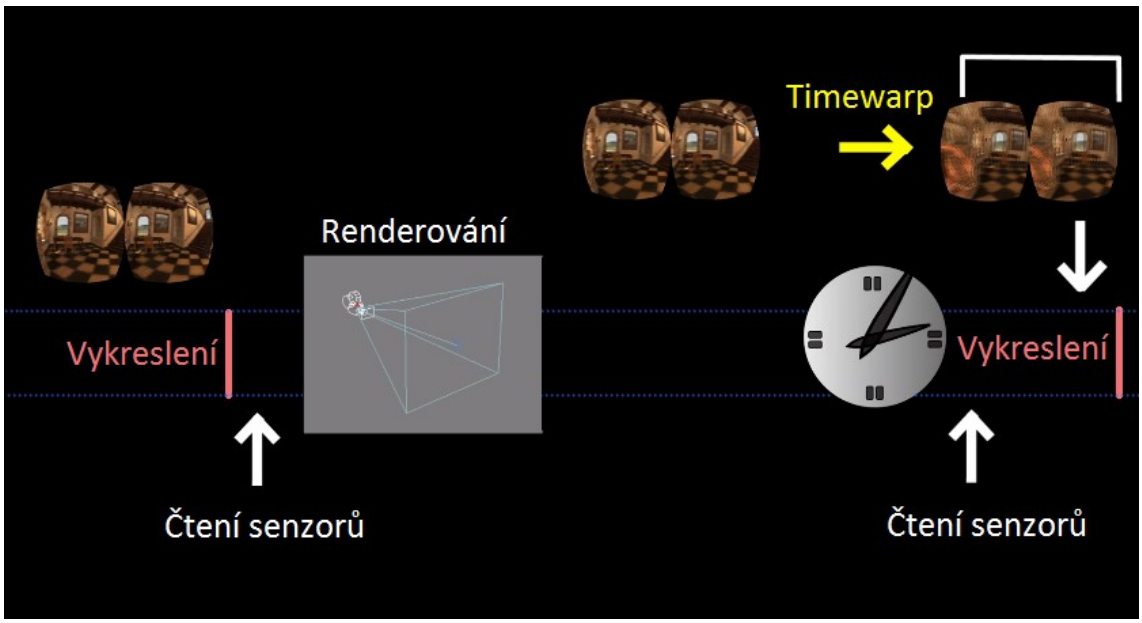
23: Timewarp se stínem prázdnoty[14]

Využití timewarpu se předpokládá jen v krátkých časových rozmezích, a proto dochází ke generování prázdných pixelů jen minimálně a je možné je vyplnit některou průměrovací metodou nebo vyplňovacím algoritmem.

[14] https://www.youtube.com/watch?v=WvtEXMlQQtI

Snížení odezvy

Prvním způsobem, jak využít tuto metodu, je snížení odezvy. Předpokládejme, že máme displej s obnovovací frekvencí 60 snímků za sekundu, to znamená každý snímek co 16,6 milisekundy. Pokud máme dostatečný výkon, stihneme snímek vyrenderovat během 10 milisekund, poté se ale dalších 6.6 milisekundy čeká, než bude možné snímek zobrazit, a to znamená, že uživatel dostává zbytečně zpožděný snímek, který již neodpovídá jeho aktuální pozici. Mohli bychom sice začít renderovat později, ale není záruka, že bychom to dokončili včas, a proto je lepší začít renderovat ihned, poté počkat, a jak už je téměř čas na vykreslení, tak provést timewarp podle nejaktuálnějších dat ze senzorů a výsledný obraz vykreslit. Je zaručeno, že vykreslení stihneme, jelikož timewarp má časovou náročnost přímo úměrnou k rozlišení obrazu.



Obrázek 24: Zobrazení časové posloupnosti vykreslování obrazu[14]

Zvýšení počtu snímků za vteřinu

V opačném případě, tedy pokud nemáme dostatečný výkon, můžeme použít asynchronní timewarp s tím, že ho aplikujeme na poslední vyrenderovaný snímek. Je tedy možné vykreslovat stejný snímek několikrát za sebou, což znamená, že prostředí bude stát, ale díky timewarpu bude pohled kamery aktuální podle senzorů, a to je pro uživatele přijatelnější.

Díky této metodě si v blízké budoucnosti lze představit násobně vyšší počet snímků za sekundu i se současným výpočetním výkonem, lze ji totiž provádět na hardwaru.

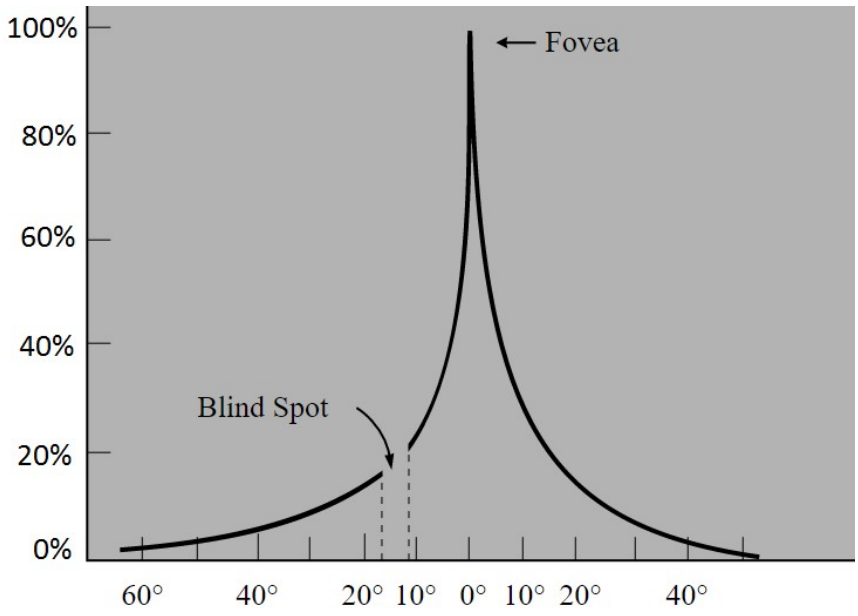
4.3 Low persistence

Low persistence neboli v překladu krátké přetrvání je technika OLED displejů, kdy se obraz rozsvítí pouze na 1–2 milisekundy a poté zhasne. Obraz při obnovovací frekvenci 60 snímků za sekundu tedy svítí asi jen 10 % času.

Pro uživatele vnímajícího virtuální realitu je lepší, když dostává informaci o obraze jen krátkou dobu, než aby byl obraz celou dobu špatně. Blikání uživatel nevnímá, jelikož oko pracuje obdobně jako fotoaparát s dlouhou závěrkou. Obraz, respektive jeho pohyb, si uživatelův mozek doplní sám a nevznikají žádné nevolnosti z rozporu konstantního obrazu a pocitu pohybu.

4.4 Foveated rendering

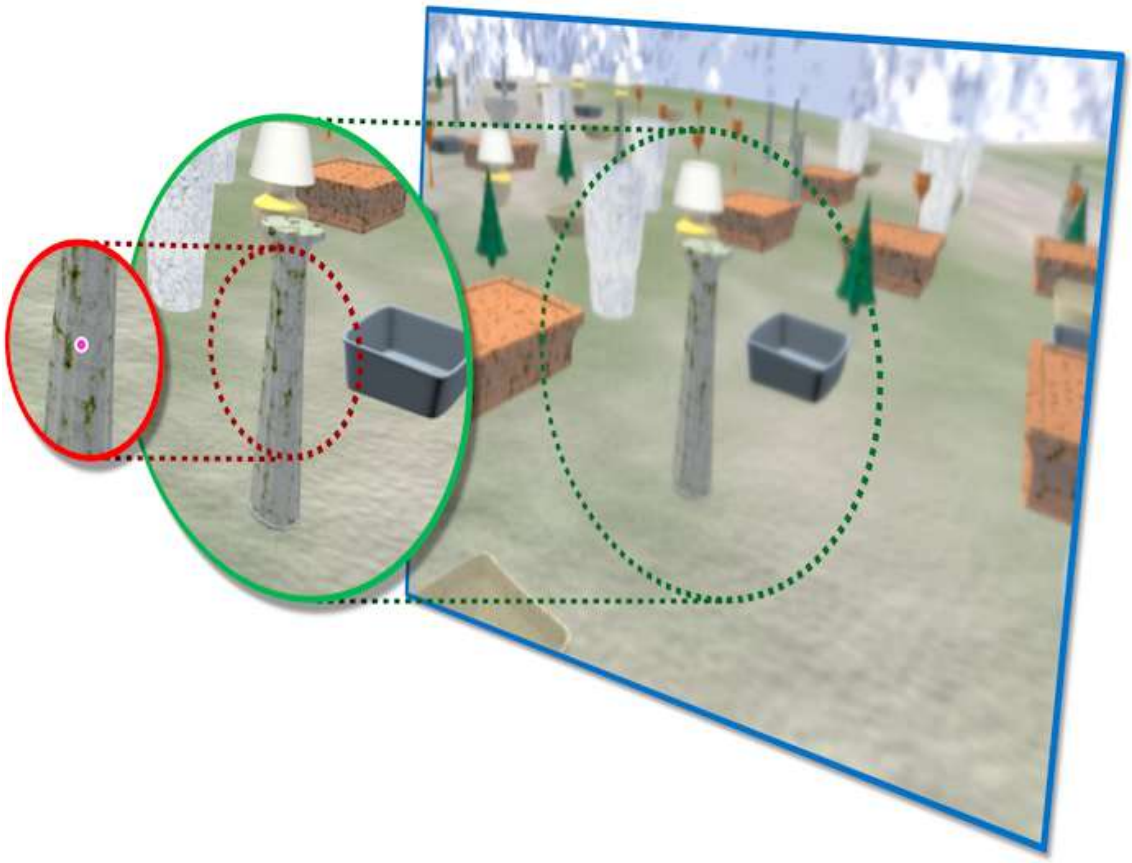
Technika, která je prozatím využívaná pouze experimentálně, se snaží rozdělit obraz na několik kruhových částí. Tyto kruhy vycházejí ze stavby oka a jeho neschopnosti vidět celý obraz ostře. Obrázek 25 zobrazuje graf, jak ostře lidské oko vidí v procentech v porovnání se zorným úhlem. Lze vidět, že nejostřejší vize je pouze v centru zorného pole, a toto místo se nazývá fovea. Při pohledu vpřed jedním okem prvních 10° vidí oko ostře, do 40° vidí oko pouze tvary a barvy a nad 40° už není schopno rozeznat tvary, pouze barvy. Lidský mozek toto omezení nevnímá a na základě barvy usuzuje, že předmět, který na tom místě viděl naposledy, se nepohnul.



Obrázek 25: Rozložení ostrosti vize lidského oka[15]

[15] https://en.wikipedia.org/wiki/Fovea_centralis

Této znalosti lze využít pro usnadnění renderování obrazu, což je činnost velmi náročná na výpočetní výkon, takže budeme renderovat celý obraz v horší kvalitě (v nižším rozlišení) a poté vyrenderujeme jenom malou část obrazu v nejvyšší kvalitě tam, kde se oko zrovna dívá. Případně můžeme obraz rozdělit na několik zón, a s narůstající vzdáleností kvalitu postupně zhoršovat. Tímto lze dosáhnout až 90% úspory renderovaných pixelů, což v budoucnu u 4K displejí a více bude třeba.



Obrázek 26: Foveated rendering[16]

[16] http://research.microsoft.com/en-us/projects/foveateddisplay/

5 Další vstupy a výstupy

Virtuální realita si klade za cíl co nejvěrohodněji přenést uživatele do jiného prostředí, a proto je našim cílem přesvědčit všechny lidské smysly, že zažívají námi zvolené podmínky.

5.1 Zvuk

Nedílnou součástí normální reality je reálný zvuk. Za celou dobu existence herního průmyslu se na zvuku vylepšoval maximálně datový tok a šíře frekvencí. Celý koncept stereo audia je funkční, snadný, ale neodpovídá realitě. Chceme-li zdůraznit, že je něco na levé nebo na pravé straně, potom zvuk z příslušné strany zesílíme a z opačné zeslabíme. Ve skutečnosti je zvuk sice o něco slabší, ale hlavně je zpožděný o dobu, než se dostane do odvráceného ucha, a také jeho tóny jsou mírně pozměněny kvůli odrazu od okolí.

Pro virtuální realitu by nebylo možné nahrát všechny možné situace, kde se může zdroj zvuku nacházet pro každou nahrávku zvlášť, proto se snažíme implementovat alespoň zvukové zpoždění pro jednotlivé uši. Zpoždění je v řádu jednotek milisekund v závislosti na natočení uší od epicentra zdroje zvuku. Technika 3D prostorového zvuku se zpožděním se nazývá Binaural Audio.

Binaural Audio volně překládáme jako audio pro obě uši, a to je technika, kdy se zvuk nahrává dvěma mikrofony umístěnými v uších vymodelované hlavy člověka. Tento přístup je nutný, protože neexistuje žádný jiný způsob jak pozměnit zvuk natolik, aby odpovídal skutečnému zvuku, který doputuje až do ušního bubínku. Jelikož mikrofony jsou již v umělých uších, zpětná reprodukce musí probíhat při použití sluchátek. Výsledkem je vysoce přesvědčivě reálný zvuk. Bez použití sluchátek se časové zpoždění znovu změní a poslech přestane uživateli předávat informaci o lokaci zdroje zvuku.

5.2 Sledování očí

Sledování očí (Eye Tracking) je snímání polohy zorné části oka. Toto snímání je důležité pro optimalizační metodu z kapitoly 4.4 Foveated rendering. Sledování očí je prozatím možné jen pomocí optických senzorů, například kamery, která je umístěna v zařízení připevňovaném na hlavu a míří přímo do očí. Podmínkou je vysoká rychlost snímání, protože oči se pohybují velmi rychle.

Dalšími výhodami využití sledování očí jsou:

- Hloubkové vidění způsobené rozostřováním okolí objektu, na který máme zaostřit.
- Lepší korekce soudkovitého zkreslení.
- Nový vstup – směr pohledu – kterým lze mířit na prvky uživatelského rozhraní.
- Oční kontakt v multiplayer hrách.
- Sledování uživatelovy pozornosti.
- Automatické vypnutí obrazovky, když oči nejsou detekovány.

5.3 Hmat

Rukavice – v kapitole 3.3.2 Snímání rukou jsme si představili rukavice pro snímání pohybu rukou. Tyto rukavice mohou být dále využity za pomocí několika malých servo motorů a tahel k zaručení jistého odporu proti sevření ruky, a tím mohou vzbuzovat dojem zpětné vazby od uchopování předmětů.

Ultrazvuk lze využít pro formování virtuálních tvarů ve vzduchu. Zvuk je totiž tlaková vlna, a při využití frekvenčního pásma, které člověk neslyší, lze pomocí více ultrasonických reproduktorů vytvořit bod, kde se vlny střetávají, a místo je vnímáno, jako kdyby se v něm nacházel pevný objekt. Systém potřebuje senzory, které sledují ruce, aby věděl, na kterém místě má rezonovat.[18]

6 Praktická část

Cílem praktické části je ukázat možnost implementace hry, ve které by uživatel byl schopen pozorovat virtuální prostředí prostřednictvím VR headsetu, a zároveň by mohl provozovat nějakou činnost, která je pro hry typická.

6.1 Analýza požadavků a výběr implementace

6.1.1 Prostor

Hry pro virtuální realitu se musejí vždy odehrávat v 3D prostoru, neboť ve 2D prostoru se nemůžeme pohybovat jako v reálném světě a další dimenze jsou jen hypotetické, přičemž čas v tomto případě za dimenzi nepovažujeme.

Pro tvorbu 3D her bychom museli naprogramovat engine, který by prostřednictvím jazyka pro komunikaci s grafickou kartou (například OpenGL) byl schopen vykreslovat pohled virtuální kamerou do 3D prostředí na 2D obrazovku. Možností je použít některý již naprogramovaný engine.

Jelikož vlastní engine je na naprogramování časově velmi náročná záležitost, pro práci byl vybrán komerční engine, kde podmínkou bylo získání bezplatné licence. Nejlépe vyhověl framework Unity, konkrétně ve verzi 5.3.4f1, protože nabízí osobní licenci zdarma, pokud roční příjem individuální osoby nepřesahuje 100 000 dolarů. Navíc se jedná o velmi populární engine na platformě Android a je snadný k používání.

6.1.2 Senzory

Projekt musí využívat senzory, aby byl schopen přizpůsobovat zobrazení bez nutnosti externího ovládání uživatele.

Obecně platí, že čím více typů senzorů používáme, tím lépe. Velmi také záleží na jejich přesnosti. V ideálním případě by chybovost senzorů měla být menší než hodnota 1 mm od reality a obnovovací frekvence alespoň 1000 Hz.

Na platformě Android jsme velmi omezeni zařízením, které používáme. Je sice možno dodatečné senzory připevnit, ale zatím, kromě GearVR – které není obecné, neexistuje obecné řešení, a proto používáme zabudované senzory v telefonu, jejichž výstup vylepšujeme aplikací kalmanova filtru. Android také nemá dostupnou možnost sledovat telefon v prostoru, proto se omezíme na Neck Model známý z kapitoly 3.3.1.

6.1.3 Stereoskopické zobrazení

Při používání VR headsetu je nutné mít pro každé oko odlišný obraz, aby uživatel vnímal hloubku prostředí jako ve skutečnosti.

Zde je na výběr několik variant řešení. První je dosud používané 3D řešení, které se nachází v televizích. To znamená jedna obrazovka, ale před každým okem se nachází filtr, který rozděluje množiny pixelů pro každé oko. Další možnosti mají pro každé oko svůj vlastní zdroj obrazu. Používáme buď jeden displej rozdělený na dvě poloviny, dva displeje anebo dva sítnicové projektory[17].

Platforma Android nás opět omezuje k používání jediného řešení, a to jeden displej rozdělený na dvě poloviny, kde je každá polovina určena jednomu oku.

6.1.4 VR platforma

Aby uživatel mohl pozorovat virtuální prostředí, musí být zvolen konkrétní VR headset.

Na platformě Android je v současnosti nejlepší volbou headset GearVR, který ale podporuje jen některé telefony. Google Cardboard je ze všech možností nejdostupnější, ale vhodné jsou také jeho klony, které mívají alespoň stejné, nebo lepší parametry.

Pro vypracovávání byl zvolen klasický kartonový Google Cardboard, a to především kvůli jeho dostupnosti, podpoře všech telefonů a ceně. Pro projekt tohoto rozsahu je všemi parametry dostatečný.

6.1.5 Herní náplň

Účel hry je libovolný, ale uživatel musí mít možnost interakce s okolím. Nesmí se tedy jednat o pouhé všesměrové zobrazení pořízené fotografickými metodami (panorama).

Herních žánrů je široké množství, Nejzajímavější Cardboard hry jsou ale se střílením, protože míříme pohledem, což přináší nový a nezvyklý prvek ovládání. Hry pro cardboard, který nemá žádné senzory pohybu, je možné implementovat několika způsoby.

- Postava se nepohybuje, ale okolí ano.
- Postava se pohybuje konstantní rychlostí vpřed nebo po předdefinované dráze.
- Využije se externí ovladač spojený s Androidem, například přes Bluetooth.

[17] https://en.wikipedia.org/wiki/Virtual_retinal_display

V projektu bude postava nehybná, ale bude interagovat s prostředím pomocí pohledu. Účel hry bude triviální, neboť není vhodné Google Cardboard používat výrazně delší dobu (nepřetržitě přes hodinu), kvůli jeho kvalitě.

6.2 Implementace

Nejprve bylo zapotřebí si stáhnout framework Unity[18]. V instalačním programu jsme provedli registraci a zvolili si stažení podpory pro platformu Android.

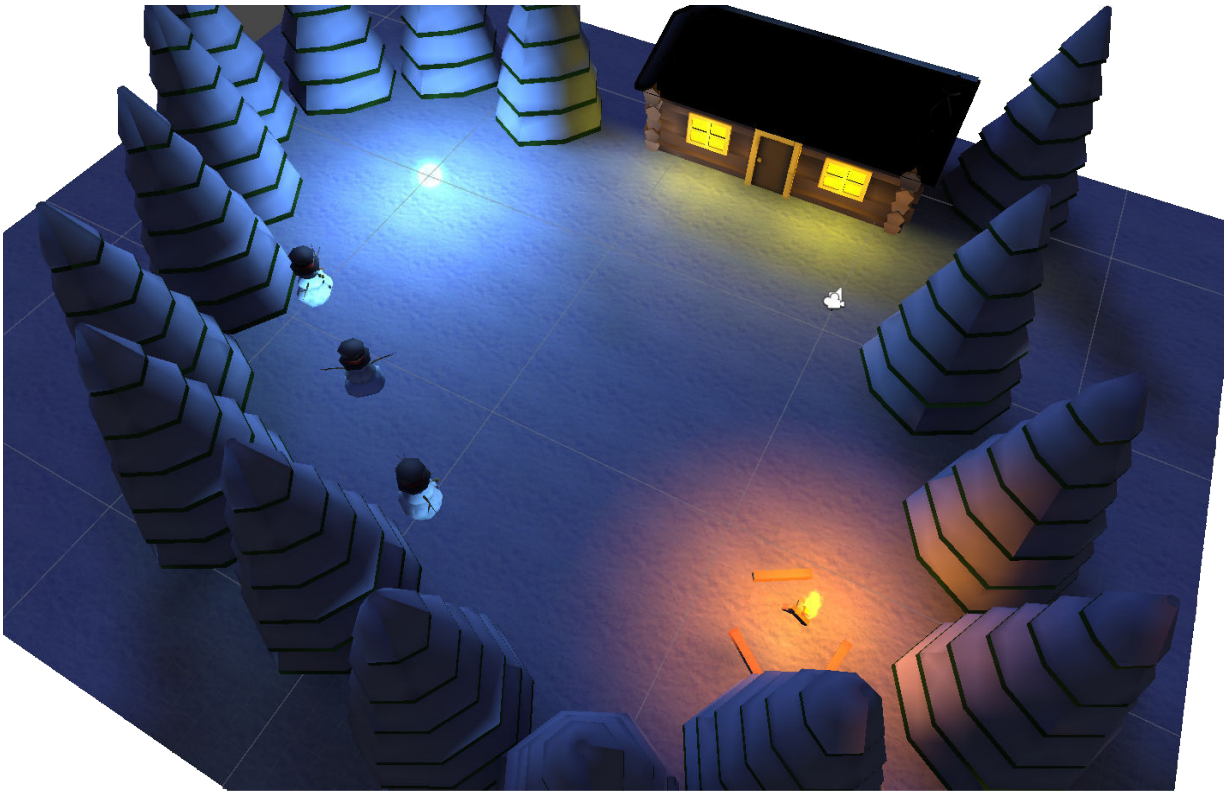
Tento framework se nám stará o vykreslování všech objektů, které se nacházejí v naší hře. Veškeré chování hry a jejich objektů bude napsané v jazyce C#. Pro naši hru potřebujeme nejprve zprovoznit snímání polohy pomocí senzorů, podle kterých se nastavuje poloha kamery, a dále změnit vykreslování tak, aby bylo rozdělené na dvě poloviny, ve kterých se zobrazuje mírně posunutý obraz. Společnost Google má k dispozici rozšiřující SDK pro Unity současně ve verzi v0.6.0, které nám zpřístupní senzory a stereoskopické vykreslování. Dále také ulehčuje několik záležitostí, které mají všechny hry pro virtuální realitu společné. Jsou to:

- Sledování hlavy pomocí gyroskopu včetně modelu krku.
- Stereoskopické renderování obrazu vedle sebe.
- Prostorové audio.
- Korekce soudkovitého rozostření.
- Automatická korekce konstantní chyby gyroskopu.

V této fázi máme připravený framework a je možné tvořit hru. Do hry potřebujeme modely, které byly vytvořeny v modelovacím programu Blender[19] za pomocí online návodů[20][2][3]. Modely importujeme do Unity a poté je zasadíme do scény, tím nám vznikne virtuální prostředí.

Obrázek 27 ukazuje pohled z ptačí perspektivy na herní prostředí. Zvolil jsem minimalistický styl modelování nazývaný často jako „low poly minimalism“. Je to technika, kdy modely jsou tvořeny pouze malým počtem vertexů. Reprezentativní je tvar a barva, například korunu listnatých stromů netvoří větve a listy, ale jedná velká zelená koule. Výsledek takového modelování připomíná pohádkovou grafiku. Jelikož je svět tvořen menším počtem polygonů, je šetřen výpočetní výkon, kterého je na mobilních platformách nedostatek.

[18] https://unity3d.com/ aktuálně ve verzi 5.4.3f1
[19] https://www.blender.org/ aktuálně ve verzi 2.76b
[20] https://www.youtube.com/user/pigartt



Obrázek 27: Ukázka herního prostředí

Dále jsem zvolil vysoký kontrast scény, což znamená kombinace světlých a tmavých barev, zejména světlé žluté a tmavě modré. Důmyslným osvětlením a stínováním je nahrazena potřeba detailně vymodelovaných objektů. Scéna se odehrává v noci, což má za následek menší intenzitu barev, a tím se potlačuje negativní projev chromatické aberace a screen door efektu.

Scéna je ohraničena stromy a jednou chatou. Místo, kde se uživatel nachází, je vyznačeno symbolem kamery (vpravo dole od chaty). Aby scéna nebyla příliš prázdná, přidal jsem kosmetické objekty ohniště a mrazivý orb. Pro celou hru jsem dále implementoval efekt sněžení.

Pohyb byl implementován pomocí prvního způsobu „Postava se nepohybuje, ale okolí ano“, z požadavku 6.1.5. Rozhodl jsem se tak, neboť pohyb by vyžadoval mnohem větší prostředí, které by bylo časově náročné na modelování, což není předmětem této práce.

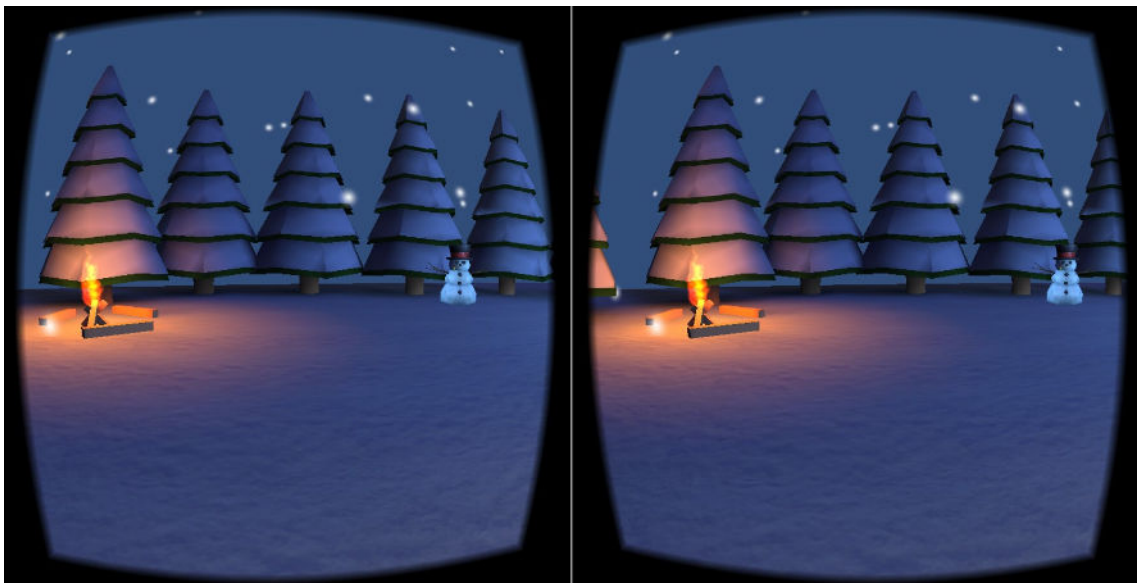
Optimalizace

Hra měla problém s nízkým počtem snímků za sekundu, ten byl vyřešen zvolením méně náročných shaderů a „*vypálením*“ světel přímo do textur objektů. To znamená, že během hraní se nevypočítává osvětlení – je statické.

6.3 Účel hry

Hra se koná v zasněžené oblasti poblíž horského srubu, ohniště a mnoha stromů. Účelem je házet sněhové koule na postupně se objevující sněhuláky. Nejprve se na ploše nacházejí tři sněhuláci.

Na sněhuláka je potřeba nejprve namířit a poté zůstat chvilku nehybně, než vyletí sněhová koule. Během míření na sněhuláka se okolo kurzoru obtočí červené kolečko, což znázorňuje dobu napřahování. Jakmile jsou všichni sněhuláci ve scéně mrtví, objeví se nová vlna, přičemž jejich počet se inkrementuje o jednoho. Sněhuláci jsou náhodně rozmístěni na ploše před uživatelem.



Obrázek 28: Screenshot z vytvořené hry

6.4 Popis hlavních algoritmů

Hra ve frameworku Unity se tvoří pomocí skriptů, ty najdeme v projektu v adresáři Assets/myScripts. Přiblížíme si některé základní skripty, které jsou ve hře využívány.

Skripty v jazyce C# musí používat balíček UnityEngine a být potomky třídy MonoBehaviour. Skripty překrývají metody Start(), která je zavolána při inicializaci objektu a Update(), která je volaná každý snímek.

Prvním vytvořeným skriptem testujeme, zdali se ve směru našeho pohledu nachází nepřátelský objekt, ten poznáme podle předdefinované vrstvy snowman s ID 9;

Zároveň nastavujeme míru vyplnění kurzoru (crosshair) při míření.

```
public class shooting_script : MonoBehaviour
{
    public GameObject snowball;                         //Odkaz na objekt sněhové koule
    public int speed = 40;                                    //Rychlost sněhové koule
    public Image loadingImage;                                    //Odkaz na crosshair

    float lastshottime = -1;                       //Čas, kdy bylo naposled vystřeleno

    // Update je volán jednou za snímek
    void Update()
    {
        if (Time.realtimeSinceStartup - lastshottime < 1)     //Pokud jsme vystřelili
        {                                                     //před méně než sekundou
            loadingImage.fillAmount = 0;                      //neprovádíme další akci
            return;
        }
                                                              //Určíme si směr střelby
        Vector3 fwd = transform.TransformDirection(Vector3.forward);
        if (Physics.Raycast(transform.position, fwd, 100, 1 << 9))
        {         //Vytvoříme virtuální paprsek skrz nepřátelské objekty ve vrstvě 9,
                  //a pokud něco trefíme a čas je 0, tak si uložíme čas začátku míření
            if (aimingStartTime == 0) aimingStartTime = Time.realtimeSinceStartup;
                                                    //Odečteme si čas od začátku míření
            float deltaTime = Time.realtimeSinceStartup - aimingStartTime;
            if (deltaTime > 2)                            //Míříme déle než dvě sekundy
            {
                                                          //Instancujeme sněhovou kouli
                GameObject thrownsnowball = Instantiate(snowball) as GameObject;
                thrownsnowball.transform.position =
                                    transform.position + Camera.main.transform.forward;
                                                                  //Přidáme ji fyziku
                Rigidbody rb = thrownsnowball.GetComponent<Rigidbody>();
                                                               //Přidáme rychlost střelby
                rb.velocity = Camera.main.transform.forward * speed;
                aimingStartTime = 0;                          //Vyresetujeme čas míření,
                                                 //a nastavíme čas poslední střelby
                lastshottime = Time.realtimeSinceStartup;
                loadingImage.fillAmount = 0;          //zrušíme crosshair napřahování
            }
            else             //pokud máme méně, než 2 sekundy pak nastavíme crosshair
                loadingImage.fillAmount = deltaTime/2;
        }
        else
        {                                          //Nemíříme-li na nic, pak se vše zruší
            aimingStartTime = 0;
            loadingImage.fillAmount = 0;
        }
    }
}
```

Dalším potřebným skriptem je myGazePointer. Ve hře pro virtuální realitu nelze umístit kurzor na střed obrazu, Kurzor se musí fyzicky promítat na místo pohledu, aby jeho stereoskopické zobrazení odpovídalo místu ostření očí.

Kdybychom umístili statický kříž na střed obou obrazů, dosáhneme výsledku obdobného, jako když si přiložíme prst před nos, a podíváme se do dálky – uvidíme dva nesloučené obrazy.

```
1.  public class myGazePointer : MonoBehaviour {
2.
3.      public GameObject headPtr;                                          //Odkaz na kameru
4.      public GameObject pointerObject;                                    //Odkaz na objekt,
5.                                                                  //který se promítá do prostředí
6.      float lastDistance = 0;                                             //Vzdálenost od kamery
7.      RaycastHit hit;                                                  //Nalezený objekt paprsek
8.
9.      void Update ()
10.     {                                                          //Namíříme paprsek vpřed od kamery
11.         Vector3 direction = headPtr.transform.TransformDirection(Vector3.forward);
12.                                                 //Pokud najde cíl tak uloží do proměnné hit
13.         if (Physics.Raycast(headPtr.transform.position, direction, out hit))
14.         {
15.                                                       //Změříme vzdálenost bodu od kamery
16.             float distance = Vector3.Distance(headPtr.transform.position, hit.point);
17.                                                          //Vrátíme předchozí vzdálenost
18.             pointerObject.transform.Translate(0, 0, -lastDistance);
19.             this.lastDistance = distance-0.3f;
20.                                     //Posuneme na místo střetu s objekty nebo krajinou
21.             pointerObject.transform.Translate(0, 0, distance-0.3f);
22.                                      //Úměrně zvětšíme objekt, aby byl vidět i z dálky
23.             pointerObject.transform.localScale = Vector3.one * (distance/50);
24.         }
25.     }
26. }
```

Další skript ColisionDetectionForSnowman detekuje zásah sněhuláka sněhovou koulí. Tento skript obsluhuje i sněhulákovo objevení (spawn).

```
1.  public class CollisionDetectionForSnowman : MonoBehaviour {
2.
3.      spawningScript spawnScript;                                 //Odkaz na skript pro spawn
4.
5.      void Start() {
6.          startSpawning = Time.realtimeSinceStartup;                  //Uložíme čas spawnu
7.          spawnScript = (spawningScript)spawnregion
8.                    .GetComponent(typeof(spawningScript));                //Naplníme odkaz
9.      }
10.
11.
12.     bool dead = false;                                                    //bool umírání
13.     bool spawning = true;                                                //bool spawnutí
14.     int angle = 0;                                                     //Úhel pádu pro smrt
15.
16.     float startSpawning = 0;                                           //Čas začátku spawnu
```

```
17.     float startDying = 0;                                   //Čas začátku animace smrti
18.     void Update() {
19.
20.         if (spawning)                                           //Jestliže se spawnuje,
21.         {                                                         //tak sněhulák roste
22.             float size = (Time.realtimeSinceStartup - startSpawning) / 2;
23.             if (size > 2)
24.             {
25.                 size = 2;                                     //Vyrostl dostatečně,
26.                 spawning = false;                               //a už nebude růst
27.             }
28.             transform.localScale = new Vector3(size, size, size);
29.         }
30.
31.         if (dead)                                                           //Pokud umře,
32.         {                                                         //tak začne padat k zemi
33.             gameObject.transform.Rotate(-5, 0, 0, Space.Self);
34.             angle++;
35.             if (angle > 18)                                     //Jakmile spadl k zemi,
36.             {                                                 //vymažeme objekt ze scény,
37.                 Destroy(gameObject);
38.                 OnDeadEvent();                                  //a vytvoříme událost
39.             }
40.             gameObject.transform.Translate(0, -0.05f, 0, Space.Self);
41.
42.         }
43.     }
44.
45.     void OnCollisionEnter(Collision col)             //Pokud dojde ke kolizi s koulí,
46.     {                                               //nastavíme proměnnou pro umírání,
47.         dead = true;
48.         Destroy(col.gameObject);                                        //a zničíme kouli
49.     }
50.
51.
52.     void OnDeadEvent()
53.     {                                                   //Po dopadu k zemi se volá
54.         spawnScript.spawn();                            //metoda ze skriptu spawn,
55.     }                                                    //která má vlastní logiku
56. }
```

6.5 Požadavky pro spuštění

Minimální požadavky pro hru jsou:

- Operační Systém Android 4+,
- dvoujádrový procesor s frekvencí 1.7Ghz,
- grafický čip Adreno 320, nebo ekvivalentní,
- zabudovaný senzor gyroskop.

Jelikož hra není distribuována přes oficiální obchod s aplikacemi Google Play, je třeba v nastavení Android telefonu povolit instalaci aplikací z neznámých zdrojů. Poté musíme instalační soubor přenést do telefonu a nainstalovat. Následně spustíme aplikaci

a telefon zasuneme do Google Cardboardu tak, aby středová čára na displeji odpovídala té na cardboardu, poté přiložíme cardboard k obličeji.

6.6 Zhodnocení

Hra byla testována na zařízení Sony Xperia SP s verzí operačního systému Android 6.0.1, a náhlavní soupravě Google Cardboard. Stereoskopické vidění fungovalo správně. Trasování hlavy pomocí gyroskopu bylo bezproblémové.

Bylo dosaženo stabilních 60 snímků za sekundu. Tento telefon nepodporuje technologii low persistence, a proto obraz působil mírně trhaně, což ale nekazilo dobrý dojem a reálnost scény.

Zkompilovaná hra i její zdrojový kód, včetně kompletního projektu z frameworku Unity, se nachází na přiloženém CD.

6.7 Možnosti rozšíření

Unity i rozšiřující Cardboard SDK nativně podporují platformu iOS, a proto by bylo možné zkompilovat hru také na tuto platformu a spouštět ji na telefonech z řady iPhone.

Pro podporu dalších VR platforem jako je GearVR, Oculus Rift a HTC Vive přechod není složitý, ale je potřeba nahradit Cardboard SDK od Googlu za příslušné SDK dodávané výrobcem konkrétního VR headsetu.

Závěr

Ze začátku práce jsme se seznámili s elektromechanickými senzory MEMS v mobilních telefonech s operačním systémem Android. Nejvíce nás zajímaly senzory, které snímají pohyb telefonu, a to akcelerometr, gyroskop a kompas.

Dále jsme probírali systém Android, virtuální realitu a jejich propojení. U virtuální reality jsme si prošli nejlepší praktiky a podmínky, které musí zařízení pro VR splňovat pro dobrý virtuální zážitek.

Postupně jsme se dostali ke vstupům, k virtuální realitě, možnostem pohybu a manipulaci s předměty, které jsme později rozšířili i o výstupy jako jsou fyzické dotyky.

Důležitým přínosem práce bylo vyhodnocení optimalizačních technik a zdůraznění jejich důležitosti.

Praktickou částí práce bylo naprogramování ukázkové hry, která je spustitelná na mobilním telefonu s operačním systémem Android společně s headsetem Google Cardboard. Aplikace představuje základní prvky virtuální reality, jako je renderování obrazu pro každé oko zvlášť a možnost rozhledu po okolí pouhým otáčením hlavy.

Cíl práce, čímž bylo představení senzorů mobilních telefonů a jejich využití při programování her s následnou ukázkou, byl tímto úspěšně splněn.

V budoucnu by bylo možné projekt dále vylepšovat. Může se zvětšit herní plocha a implementovat možnost pohybu po okolí. Dále lze zavést možnost hry více hráčů, kteří by po sobě házeli sněhové koule. Absolutním cílem by bylo zveřejnění hry v internetovém obchodu s aplikacemi Google Play.

Seznam použitých zdrojů

[1] Android drží spolu s iOS více než 96% podíl mezi operačními systémy. [online]. *mobilenet.cz* 30. Leden, 2015 [cit. 2015-12-26]. Dostupné z: http://mobilenet.cz/clanky/android-spolu-s-ios-drzi-vice-nez-96-podil-mezi-operacnimi-systemy-18872

[2] BLENDER Timelapse: Low Poly Winter Scene! In: *Youtube* [online]. Zveřejněno 4. 12. 2013 [vid. 2015-6-5]. Dostupné z: https://youtu.be/I7I_SeuM62k

[3] BLENDER Tutorial: Low poly forest assets! In: *Youtube* [online]. Zveřejněno 30. 12. 2013 [vid. 2015-6-5]. Dostupné z: https://youtu.be/rtO9maU709k

[4] Explained: How does VR actually work? [online]. *Wareable.com* 30. březen, 2016 [cit. 2016-03-30]. Dostupné z: http://www.wareable.com/vr/how-does-vr-work-explained

[5] Fovea centralis. [online]. *Wikipedia: the free encyclopedia* San Francisco (CA): Wikimedia Foundation, 2. Dubna, 2016 [citováno 4. 08. 2016]. Dostupné z: https://en.wikipedia.org/wiki/Fovea_centralis

[6] GLSL Tutorial von Lighthouse3D [online]. *OpenGL.org* 2. Září, 2008 [cit. 2016-04-08]. Dostupné z: http://zach.in.tu-clausthal.de/teaching/cg_literatur/glsl_tutorial/

[7] GLSL: An Introduction [online]. *OpenGL.org* 2. Září, 2008 [cit. 2016-04-08]. Dostupné z: http://nehe.gamedev.net/article/glsl_an_introduction/25007/

[8] Gyroscope. [online]. *SensorWiki.org* 26. Duben, 2013 [cit. 2015-12-06]. Dostupné z: http://www.sensorwiki.org/doku.php/sensors/gyroscope

[9] How Lenses for Virtual Reality Headsets Work [online]. *VR Lens Lab* 8. Březen, 2016 [cit. 2016-03-08]. Dostupné z: http://vr-lens-lab.com/lenses-for-virtual-reality-headsets/

[10] Integrované MEMS GYROSKOPY. [online]. *automatizace.hw.cz* 1. Říjen, 2009 [cit. 2015-12-06]. Dostupné z: http://automatizace.hw.cz/integrovane-mems-gyroskopy

[11] Jak funguje elektronický kompas? [online]. *mobilmania.cz* 16. Září, 2009 [cit. 2015-12-26]. Dostupné z: http://navigovat.mobilmania.cz/clanky/jak-funguje-elektronicky-kompas/sc-265-a-1314330

[12] MEMS [online]. *Wikipedia: the free encyclopedia* San Francisco (CA): Wikimedia Foundation, 16. Dubna, 2014 [citováno 6. 12. 2015]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=MEMS&oldid=11860561

[13] New Centimeter-Accurate GPS System Could Transform Virtual Reality and Mobile Devices [online]. *UTNews* 5. Březen, 2015 [cit. 2016-03-09]. Dostupné z: http://news.utexas.edu/2015/05/05/engineers-develop-centimeter-accurate-gps-system

[14] NVIDIA Demonstrates Experimental „Zero Latency" Display Running at 1,700Hz [online]. *RoadToVR* 6. Duben, 2016 [cit. 2016-04-08]. Dostupné z: http://www.roadtovr.com/nvidia-demonstrates-experimental-zero-latency-display-running-at-17000hz/

[15] OpenGL Shading Language [online]. *OpenGL.org* 2. Září, 2008 [cit. 2016-04-08]. Dostupné z: https://www.opengl.org/documentation/glsl/

[16] SMITHWICK, R. Michael., Mayank. VERMA a Leila. MUHTASIB. *Pro OpenGL ES for Android.* New York: Distributed to the book trade worldwide by Springer Science+Business Media, c2012. ISBN 1430240024.

[17] TyphoonLabs' OpenGL Shading Language tutorials [online]. *OpenGL.org* 2. Září, 2008 [cit. 2016-04-08]. Dostupné z: https://www.opengl.org/sdk/docs/tutorials/TyphoonLabs/

[18] Ultrasound to give feel to games [online]. *BBC News* 2. Září, 2008 [cit. 2016-04-08]. Dostupné z: http://news.bbc.co.uk/2/hi/technology/7593444.stm

Seznam obrázků

Slovník pojmů

Drift	Vada virtuální reality, kdy snímání polohy zařízení má konstantní malou chybovost, ale chyba se postupem času sčítá a odchylka od skutečnosti je po čase veliká
HMD	Head Mounted Display – Zařízení, které se připevňuje k hlavě a je schopno zobrazovat virtuální realitu, často je nazýváno též jako VR headset, či jen headset
SDK	Software Development Kit – nástroj k vývoji aplikací
SIMD	Single Instruction, Multiple Data – je způsob paralelního zpracování, kde se najednou zpracovává jedna operace pro více vstupních dat
Subpixel	Každý obrazový pixel se skládá z několika menších barevně svítících plošek
Vertex	V 3D modelech je to místo, kde se stýkají dvě a více hran
VR	Virtuální realita

Seznam příloh na CD

[1] Text této práce jako soubor:
FPF_DP_16_Android senzory_Michael Kocian.pdf

[2] Projekt hry z Frameworku Unity3D verze 5

[3] Zkompilovaná verze hry ve formátu .apk

Příloha A

Kód programu pro simulaci chromatické aberace v jazyce Java.

```
1.  public static void main(String[] args)
2.  {
3.      BufferedImage srcImg = ImageIO.read(new File("D:\\input.png"));
4.                                                  //Načteme vstupní obrázek ze souboru
5.
6.      BufferedImage dstImg = new BufferedImage(srcImg.getWidth(),
7.                               srcImg.getHeight(), BufferedImage.TYPE_3BYTE_BGR);
8.                                        //Vytvoříme si výstupní obrázek stejné velikosti
9.
10.     double centerX = srcImg.getWidth()/2;                    //Určíme si střed v ose x
11.     double centerY = srcImg.getHeight()/2;                   //Určíme si střed v ose y
12.
13.     for (int x = 0; x < dstImg.getWidth(); x++)            //Cyklus pro řádky v obraze
14.     {
15.         for (int y = 0; y < dstImg.getHeight(); y++)             //Cyklus pro sloupce
16.         {
17.             double dx = (x-centerX)*0.05;       //Vypočítáme si vzdálenost od středu,
18.             double dy = (y-centerY)*0.05;            //kterou vynásobíme koeficientem
19.                                                             //roztřepení v obou osách
20.             int rgbcolorR = 0;
21.             int rgbcolorG = 0;
22.             int rgbcolorB = 0;
23.             try                                  //Posunutí jednotlivých barevných složek
24.             {                                            //podle vypočítaných koeficientů
25.                 rgbcolorR = srcImg.getRGB((int) (x), (int) (y));
26.                 rgbcolorG = srcImg.getRGB((int) (x+dx), (int) (y+dy));
27.                 rgbcolorB = srcImg.getRGB((int) (x+2*dx), (int) (y+2*dy));
28.             }
29.             catch (ArrayIndexOutOfBoundsException e) {}
30.
31.             Color r = new Color(rgbcolorR);
32.             Color g = new Color(rgbcolorG);
33.             Color b = new Color(rgbcolorB);
34.             Color finalColor = new Color(r.getRed(),g.getGreen(),b.getBlue());
35.                                       //Vytvoříme novou barvu z požadovaných složek
36.             dstImg.setRGB(x, y, finalColor.getRGB());
37.                                        //Zapíšeme výslednou barvu do nového obrázku
38.         }
39.     }
40.
41.     try {                                            //Zapíšeme výstupní obrázek na disk
42.         ImageIO.write(dstImg, "png", new File("D:\\output.png"));
43.     } catch (IOException ex) {
44.         //Nelze zapisovat
45.     }
46. }
```