Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Multiuživatelská virtuální realita

Plzeň, 2011 Martin Tichota

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 16.5.2010, Martin Tichota, ...

Poděkování

Rád bych poděkoval Ing. Petru Lobazovi za vedení mé práce, jeho cenné rady a komentáře a také za pozitivní a profesionální přístup.

Abstract

The name of my thesis is Multiuser Virtual Reality and it deals with immersive virtual reality systems. Virtual reality is a concept of artificial environment, which simulates physical presence in real or imaginary worlds. Its main purpose is to enhance user's perception and let him/her experience things that could be difficult or even impossible to accomplish in his/her own reality. Many virtual reality systems have been developed in last 40 years, bringing different levels of immersion to the user. One branch of immersive virtual reality systems is based on tracking of user's position and view direction and updating the virtual scene according to the tracked data.

The goal of this thesis is to inspect existing tools that can be used to create immersive virtual reality based on user's tracking and design and implement such system. In the thesis, original virtual reality system solution is proposed, which enables the user to walk and look around freely and perform virtual walks through various virtual environments. Such a system could then be used for instance as an attractive alternative for architectural visualizations or as an engine for innovative games. The thesis describes the design and implementation in detail together with the theoretical background and review of existing virtual reality tools and systems.

OBSAH

1. Úvod 1
2. Virtuální realita 2
   2.1. Dostupné metody a technologie immersive VR 3
      2.1.1. Head-mounted display 3
      2.1.2. Intertrax 2 HeadTracker 5
      2.1.3. Nintendo Wii Remote 5
      2.1.4. Freetrack 6
      2.1.5. C.A.V.E 7
      2.1.6. Všesměrové pohybové platformy 7
3. Matematický aparát 8
   3.1. Homogenní souřadnice 8
   3.2. Matematický model kamery 9
   3.3. Homografie 11
      3.3.1. Planární homografie 11
      3.3.2. Čtyřbodový algoritmus pro výpočet planární homografie 12
      3.3.3. Rozklad planární homografie 13
4. Návrh systému 14
   4.1. Získání dat o uživatelově pozici a směru pohledu 16
      4.1.1. Snímací zařízení a významné body 16
      4.1.2. Zachycení snímku 18
      4.1.3. Předzpracování snímku 18
      4.1.4. Nalezení středů významných bodů 19
   4.2. Rekonstrukce pohybu 21
      4.2.1. Nalezení korespondencí 21
      4.2.2. Rekonstrukce pohybu z korespondencí 24
      4.2.3. Reprezentace pohybu pro zobrazovací část 35
   4.3. Zobrazení rekonstruovaného pohybu 37
5. Simulace 38
   5.1. Simulace pomocí podpůrných aplikací 38
   5.2. Numerická simulace 39
   5.3. Softwarová simulace 40
      5.3.1. Simulační framework 40

6. Software 43
6.1. VirtualWalkLib 44
6.2. ImageprovidersLib a WebcamLib 47
6.2.1. WebcamLib 52
6.3. Renderery 54
6.4. Client 54
7. Hardware 55
7.1. Uchycení snímacího zařízení 55
7.2. Přídavné infračervené osvětlení 56
7.3. Snímací zařízení 57
7.4. Head mounted display 57
7.5. Významné body 57
8. Omezení systému a možné aplikace 58
8.1. Omezení části pro nalezení uživatelovy pozice a směru pohledu 58
8.1.1. Určení korespondencí 58
8.1.2. Rekonstrukce pohybu 60
8.2. Hardwarová omezení 60
8.2.1. Snímací zařízení 61
8.2.2. HMD 61
8.3. Možné aplikace systému 61
9. Budoucí práce 62
9.1. Zavedení absolutního pozicování 62
9.2. Navržení lepšího hardwaru 63
9.3. Zobrazovací framework 63
9.4. Multiuživatelské prostředí 63
10. Závěr 64
Literatura 65
PŘÍLOHA: UŽIVATELSKÁ PŘÍRUČKA 67
1. Ovládání aplikace 67
2. Překlad a spuštění aplikace 69
3. Hardwarové vybavení 69

1. ÚVOD

Virtuální realita (VR) je pojem odkazující na umělé, „neskutečné“ prostředí, které simuluje prostředí uživatelem považované za skutečné. Cílem tohoto prostředí je obohatit uživatele o nové vjemy, umožnit mu prožít věci, které jsou v jeho realitě obtížně splnitelné, v extrémním případě skutečné prostředí kompletně nahrazuje bez vědomí uživatele. Ačkoliv může být tato otázka chápana jako čistě filozofická, nejčastěji se lze setkat s pojmem virtuální realita v souvislosti s počítačem generovaným prostředím. V nejjednodušším případě je VR chápána jako prostředí zobrazené na obrazovce počítače, v němž se uživatel může pohybovat pomocí běžných vstupních zařízení, jako je klávesnice a myš.

Historie tohoto fenoménu sahá až do počátku padesátých let 20. století, kdy bylo možné poprvé pomocí počítače vykreslit něco alespoň trochu připomínající skutečné prostředí. V roce 1965 pak představil počítačový vědec Ivan Sutherland ve své eseji[1] koncept, který určil směr vývoje virtuální reality a definoval její chápání na další desítky let dopředu. Tento koncept nazval „Ultimate Display“ a obsahoval:

- Virtuální svět, který by vypadal skutečně pro jakéhokoliv pozorovatele. Tento svět by byl pozorován skrz HMD[1], vjem by byl dále obohacený prostorovým zvukem a hmatovými stimuly.
- Počítač, který udržuje virtuální svět v reálném čase.
- Možnost uživatele intuitivně a realisticky manipulovat s předměty, které jsou součástí virtuálního světa.

Možné aplikace tohoto návrhu jsou téměř nevyčerpatelné, od tréninkových simulátorů, přes prostředí pro sociální studie, zábavní průmysl až po science fiction scénáře podobné filmu Matrix. Snaze naplnit tento koncept se poté věnovalo nespočet prací. Největší boom zažily systémy VR zřejmě v 90. letech, kdy dosáhl technologický pokrok takové úrovně, že bylo konečně možné v reálném čase kombinovat realistická zobrazení na kvalitních displejích spolu s dalšími vjemy. I v současné době je však výzkum na poli VR velmi živý a neustále vznikají nové implementace Sutherlandova konceptu, čehož důkazem jsou organizace jako například ImmersiveTech[36]. Tyto organizace podporují, případně sami provádí, aktivní výzkum a vývoj nových VR systému, které obohacují state-of-the-art této problematiky a posouvají evoluci systémů k ideální formě virtuální reality.

Cílem této práce je návrh dalšího systému VR, který novým způsobem následuje Sutherlandův koncept. Ústředním tématem práce je návrh systému, který v reálném čase zjišťuje uživatelovu pozici a směr jeho pohledu a na základě těchto dat pak aktualizuje scénu, která je uživateli promítána pomocí speciálního displeje. Inovativnost tohoto systému pak spočívá v tom, že se uživatel může volně procházet a jeho akční rádius, tedy plocha, po které se může pohybovat, je limitován pouze zdmi místnosti, ve které je systém instalován. Návrh systému také počítá s možným zapojením více uživatelů do virtuálního světa současně. Cílem není ani tak vytvoření komerčního řešení schopného distribuce, ale spíše prozkoumání možností jakými lze tento systém realizovat, návrh konceptu a jeho ověření pomocí cenově dostupného hardwaru. Motivací pro vývoj byla zejména poptávka po vytvoření demonstračního softwaru pro potřeby univerzity, který by ukazoval možné zpracování tématu virtuální reality. Poptávka po podobné aplikaci, umožňující skutečné procházení virtuálních světů, je ale velká i v oblasti stavebního průmyslu a realitních kanceláří. V neposlední řadě to byla také snaha o vylepšení obdobného systému

[1] head mounted display – více viz sekce 2.1.1

vytvořeného v rámci mé bakalářské práci [19], jeho rozšíření a úprava tak, aby umožňoval neomezený pohyb a případné zapojení více uživatelů.

Práce je organizována následovně: Nejprve je v kapitole 2 obšírněji popsána problematika virtuální reality a uvedeny některé z nástrojů pro její tvorbu. Dále je v kapitole 3 uveden matematický aparát použitý při návrhu systému a nutný k jeho pochopení. Následuje detailní popis návrhu systému v kapitole 4, popis simulací použitých při vývoji a ladění v kapitole 5 a softwarová implementace návrhu v kapitole 6. Navržené hardwarové vybavení je součástí kapitoly 7 a v kapitole 8 jsou diskutována omezení systému a jeho možné aplikace.

2. VIRTUÁLNÍ REALITA

Jak bylo nastíněno v Úvodu, v nejjednodušším případě je jako virtuální realita chápán umělý svět na obrazovce počítače, který uživatel prochází a interaguje s ním pomocí klávesnice a myši. V této podobě se VR objevuje především v klasických 3D akčních počítačových hrách. Mnohem častěji se však o VR mluví v souvislosti s použitím netradičních vstupních a výstupních zařízení, které obohacují a umocňují zážitek uživatele. Tato VR je poté označována jako immersive[2] VR. Těžištěm mé práce je vytvoření právě takového systému, díky kterému je uživatel určitým způsobem vtáhnut do virtuálního světa.

Faktory, které se podílí na míře „vtáhnutí“ uživatele lze rozdělit do několika skupin:

- vizuální složka
- další senzorické podněty (sluch, hmat, čich…)
- sledování uživatelovy pozice a směru pohledu
- interakce s prostředím a podpora více uživatelů

Nejvyvinutějším smyslem člověka je zrak a pokud má být dojem z VR opravdu reálný, je nutné, aby bylo zobrazení na vysoké úrovni. Základním předpokladem je vysoce kvalitní, ideálně fotorealistický rendering zobrazované scény. Dalším krokem je použití stereoskopického renderingu, který umožňuje využít vlastnosti lidského oka k tomu, aby uživatel vnímal scénu trojrozměrně, i když je promítána na ploché médium. Dalším možným vylepšením je použití specializovaného hardwaru, který nějakým způsobem odstíní okolní prostředí a uživatel se pak soustředí pouze na virtuální scénu. Mezi takový hardware lze zařadit například head-mounted displeje nebo podstatně komplikovanější CAVE systémy, viz 2.1.

Vizuální složka už je v současné době poměrně dobře zvládnutá, ovšem samotná potřebnou míru vtáhnutí uživatele nezajistí. Dalším stupněm immersion je tedy sledování (trackování) uživatelovy pozice, případně směru pohledu a aktualizace virtuální scény na základě těchto údajů. Trackování by nemělo omezovat uživatele v pohybu a ideálně by o něm neměl vůbec vědět, aby nepůsobilo rušivě. Metody trackování lze rozdělit do skupin podle principu, kterým sledují uživatele, na:

- Mechanické – uživatel je mechanicky připojen k senzorům, které vyhodnocují jeho pozici
- Akustické – využívá mikrofony a ultrazvukové generátory pro určení pozice
- Magnetické – vysílač generuje pomocí cívek magnetické pole a přijímač ho analyzuje

[2] immersive(angl.) = vtahující, pohlcující

- Neinerciální – využívá pasivní magnetické senzory, které jsou relativní k magnetickému poli země
- Inerciální – používá akcelerometry a gyroskopy, integrací výstupů je počítána pozice a orientace
- Optické – využívá světlocitlivých senzorů, nejčastěji kamer a analýzy světelné informace pro určení pozice uživatele, tento přístup používám ve své práci i já
- Jiné – sem lze zařadit například GPS lokaci a další podobné systémy

Mikšovic[20] nabízí dobré shrnutí možností trackování včetně výhod a nevýhod. Já jsem zvolil pro svoji práci trackování optické, vzhledem k tomu, že s ním již mám zkušenosti a je cenově i technologicky dostupné a dosahuje dobré přesnosti.

Míra immersion jde dále zlepšovat zapojením dalších uživatelových smyslů do virtuální reality. Nejjednodušším je zřejmě sluch, v současné době je vytvoření kvalitního prostorového zvuku na vysoké úrovni, jeho propojením s trackováním uživatelovy pozice pak lze zvýšit míru vtažení poměrně výrazně. V současné době intenzivně zkoumané jsou také možnosti haptických zařízení, které umožňují zapojit do virtuální reality uživatelův hmat. Mezi haptická zařízení patří různé joysticky nebo volanty s podporou technologie *force-feedback*[3], v případě immersive VR jsou ale nejčastější hůlky a tyčinky, pomocí kterých může uživatel aktivně zasahovat do scény, případně jsou tyto ovládací prvky vybaveny také *force-feedback* technologií. Technologie, které by umožňovaly věrně zprostředkovat čich a chuť zatím ještě nejsou na dostatečné úrovni, je to tedy výzva do budoucna a jedna z posledních zásadních překážek, jak smazat rozdíl mezi realitou virtuální a skutečnou.

Specifickým hlediskem je podpora více uživatelů v rámci virtuálního světa. Tato věc je poměrně běžná v „tradičních“ virtuálních realitách, jako jsou počítačové hry, kde je umožněna interakce více uživatelů pomocí multiplayerových verzí hry. Ty využívají síťové technologie pro propojení a interakci hráčů mezi sebou. Z hlediska immersion je tato složka poměrně důležitá, protože umožňuje do virtuální reality zanést sociální faktor a obohatit ji tak o společenské interakce.

2.1. DOSTUPNÉ METODY A TECHNOLOGIE IMMERSIVE VR

V této sekci vyjmenuji a popíši různé technologie pro tvorbu immersive VR, které souvisí s mojí prací. Zmíním několik kompletních řešení, včetně jejich možností a omezení, ale také dílčí hardwarové nebo softwarové nástroje, jejichž kombinací lze virtuální realitu vytvořit.

2.1.1. Head-mounted display

Head-mounted display (HMD) je zobrazovací zařízení, které má uživatel nasazené na hlavě. Je tvořené jedním, monokulární, nebo dvěma, binokulární, displeji, které má uživatel před očima a které zobrazují scénu. Výhodou binokulárního HMD je možnost vytvoření stereoskopického obrazu. Pokud je dobře konstruován, zabrání uživatelovi vidět své okolí a umožní mu tak soustředit se pouze na zobrazovanou scénu a zvýší tak míru immersion. Hlavní výhodou HMD je, že je spojený s uživatelem a umožňuje mu tak teoreticky volný pohyb, na rozdíl od různých stereoskopických displejů a promítacích

[3] *force-feedback* = silová zpětná vazba, ovladač reaguje nejčastěji vibracemi na aktuální dění ve virtuální realitě (jízda po nerovném povrchu, odpor prostředí…)

pláten, které jsou fixované na místě. Nevýhody HMD vychází z jeho konstrukce, dotýkají se hlavně ergonomie a pohodlnosti nošení, zdravotních rizik vzhledem k blízkosti displejů k očím atd.

Mezi stále používané a podporované produkty patří například i-glasses od firmy i-O Display Systems, viz Obr. 1, s udávanými specifikacemi[22]

- Rozlišení : 800 x 600
- FOV : 26 stupňů diagonálně
- Barevná hloubka : 24 bitů
- Obnovovací frekvence : 100 Hz
- Cena : 899 $

Obr. 1 i-glasses SVGA. Obrázek je vlastnictvím firmy i-O Display Systems.

Tento produkt už je však na dnešní dobu zastaralý hlavně díky svému nízkému rozlišení a malému FOV. Alternativou tak může být HMD od firmy Cybermind Interactive Nederland s názvem Visette45 SXGA 3D, viz Obr. 2, a specifikacemi[23]

- Rozlišení : 1280 x 1024
- FOV : 45 stupňů diagonálně
- Barevná hloubka : 24 bitů
- Obnovovací frekvence : 60 Hz
- Cena : 12 900 $,

jehož nevýhodou je vysoká cena.

Obr. 2 Visette45 SXGA 3D. Obrázek je vlastnictvím firmy Cybermind Interactive Nederland

2.1.2. Intertrax 2 HeadTracker

Head tracker Intertrax^2 se skládá z citlivých gyroskopů, které měří odchylku ve třech souřadných osách. Tato data je pak možné přes USB rozhraní z trackeru periodicky získávat a používat je v hostitelské aplikaci. Jak už název napovídá, je tento trakcer určen ke snímání orientace hlavy a vzhledem k jeho malé velikosti ho lze přichytit například na HMD. Vzhledem k tomu, že tento tracker používá k měření gyroskopy, je aktuální orientace vždy relativní k předchozímu vzorku. Díky tomu se s přibývající dobou měření akumuluje chyba a po určitém čase už mohou být výsledky měření nepoužitelné. Z vlastních testů, které jsem se zařízením prováděl, se projevilo, že po 1-2 minutách začne být odchylka natolik viditelná, že negativně ovlivňuje uživatelovo vnímání a je nutné tracker restartovat do základní polohy. Ukázka trackeru je na Obr. 3.

Obr. 3 Intertrax^2 headtracker. Obrázek je majetkem firmy Intertrax

2.1.3. Nintendo Wii Remote

Wii Remote nebo také zkráceně Wiimote je netradiční ovladač pro herní konzoly Nintendo Wii. Jeho hlavní výhodou je možnost sledování pohybu a pozice díky vestavěnému tříosému akcelerometru a optickému senzoru citlivému na infračervené světlo. I když je Wiimote původně určené pouze pro konzoly Wii, lze ho připojit i k počítači pomocí rozhraní Bluetooth, čímž se podrobně zabývá [24].

Díky akcelerometru lze zjistit přírůstek rotace v jedné ze tří os vzhledem k předchozímu vzorkování. Integrací přírůstků v časovém intervalu $[t_0, t_1]$ pak získáváme orientaci ovladače vzhledem k poloze v čase t_0. Vestavěný optický senzor sice umožňuje snímat okolí, slouží však pouze jako vstupní zařízení pro zabudovaný procesor, který na snímek ze senzoru aplikuje filtr pro detekci shluků. Z ovladače tedy nelze získat data přímo ze senzoru, ale pouze středy nalezených blobů. Procesor umožňuje sledování až 4 blobů najednou, pokud je však ve scéně blobů víc, programátor nemá absolutně žádnou kontrolu nad tím, které bloby ovladač zrovna zachytil. Optický senzor Wiimote se tedy hodí pro sledování scény s předem známým počtem blobů(≤ 4) a jejich topologií. Výhodou Wiimote je jeho kompaktnost a také použití bezdrátové technologie, což omezuje akční rádius na dosah Bluetooth vysílače, který je u Wiimote kolem 120ti metrů. Náhled ovladače viz Obr. 4, technická data Wii Remote jsou

- Optický senzor
  - Rozlišení: 1024 x 768
  - Snímková frekvence: 100 Hz
  - FOV: 33 stupňů horizontálně, 23 stupňů vertikálně
- Akcelerometr
  - Rozsah: $\pm$ 3g

- Stupně volnosti: 3
- Přesnost: 8 bitů

Obr. 4 Ovladač Wii Remote. Obrázek převzat z http://en.wikipedia.org/wiki/Wii_Remote

2.1.4. Freetrack

Freetrack[25] je jedním z kombinovaných nástrojů pro tvorbu virtuální reality. Využívá webové kamery, která snímá uživatelův obličej, ten má na hlavě nasazen jednoduchý model, viz Obr. 5, díky němuž pak obslužný software umožňuje počítat orientaci a pozici hlavy, je tedy podobný 2.1.2. Pracuje ovšem na jiném principu a navíc dokáže určit nejenom náklon hlavy, ale i její pozici. Základním rozdílem je, že u Freetracku je pozicování absolutní, což eliminuje nevýhody zmíněné u headtrackeru Intertrax. Výpočet pozice a orientace funguje na základě algoritmu POSIT[2], který z předem známé geometrie modelu a 3D pozice významných bodů vypočítá z průmětu těchto významných bodů ve snímku kamery orientaci a pozici modelu.

Obr. 5 Ukázka modelu se třemi významnými body, který má uživatel nasazený na hlavě. Obrázek převzat z [25]

Freetrack je využíván zejména různými počítačovými herními simulátory aut nebo letadel, kde jeho softwarové rozhraní umožňuje ovládat směr pohledu hráče. Jde o freeware open-source projekt, cena modelu se pohybuje v řádech dolarů. Jde tedy o velmi levný a přesto velmi dobře použitelný systém. Díky tomu, že jde o kombinaci několika dílčích prostředků, které jsou dohromady optimalizované, lze ho však velmi obtížně integrovat do jiného systému.

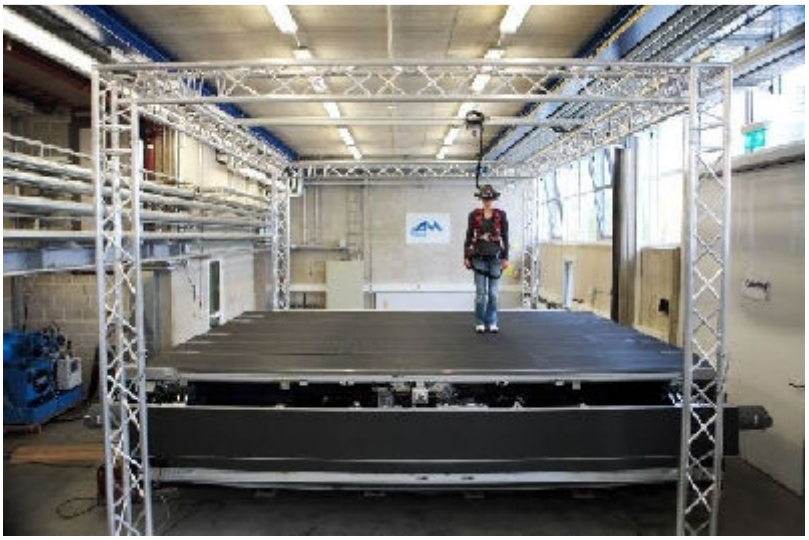
2.1.5. C.A.V.E

CAVE je poměrně komplexní systém vytvářející virtuální prostředí pomocí promítání scény projektory na stěny krychle, v níž se pohybuje uživatel. Ten má na očích speciální brýle, které umožňují spolu s projektory vytvoření stereoskopického obrazu. Dalším prvkem C.A.V.E systému jsou senzory, které sledují pohyb uživatele uvnitř krychle a upravují v závislosti na jeho pozici promítaný obraz. Tyto senzory jsou obvykle elektromagnetické a jsou pevně spojené s kostrou krychle, často jsou také spojeny ještě s optickými senzory, které sledují směr uživatelova pohybu pomocí trackování význačných bodů na uživatelových brýlích. Díky kombinaci těchto technologií je dosaženo vysoké míry immersion, uživatel se může procházet plně 3D prostředím a sledovat vznášející se předměty, obcházet je a zkoumat. Tyto přednosti jsou vyváženy velkou cenou a náročností na realizaci. Navíc je pohyb uživatele omezen rozměrem krychle a především je C.A.V.E systém omezen na jediného uživatele v jeden okamžik, není totiž možné sledovat pohyb a vytvářet stereoskopický obraz pro více uživatelů zároveň.

2.1.6. Všesměrové pohybové platformy

Tyto technologie řeší problém omezenosti akčního pohybového radiusu uživatele obvykle za pomocí nějakého mechanického systému, který na uživatelovu chůzi reaguje stejně velkou, opačně orientovanou silou. Tím je docíleno toho, že uživatel vyvíjí stejný pohyb jako při chůzi, ale díky systému je udržován stále na stejném místě.

Jedno z možných řešení všesměrových pohybových platforem jsou ODT [4]. Podstatou této technologie je rozšíření běžného běžeckého pásu o kolmý směr pohybu, složením těchto pohybů je pak umožněno pokrýt všechny směry v rovině. Uživatel se tedy po pásu pohybuje a ten se pod ním v opačném směru odvíjí. Jednou z konkrétních implementací ODT je například produkt Cyberwalk, viz Obr. 6 německého výzkumného centra AM[26]. Uživatel se na něm může pohybovat rychlostí až $2\,m/s$ a plocha pásu má rozměr $5{,}5 \times 4{,}6$ metrů. Uživatel má na hlavě nasazen HMD a jeho pohyb je trackován mechanicky. Data o jeho pozici jsou použity jednak k aktualizaci zobrazované scény a také ke kompenzaci vychýlení ze středu pásu, což musí být prováděno, aby uživatel z pásu nespadl.



Obr. 6 Všesměrový pás Cyberwalk. Obrázek převzat z [26]

[4] Z anglického omnidirectional treadmill – všesměrový (běžecký) pás

Dalším řešením je projekt americké firmy VirtuSphere, Inc.[27]. Ten problém neomezenosti pohybu řeší umístění uživatele do velké koule, která je usazena v loži osazeném mechanickým systémem pro snímání a řízení rotace koule. Chůzí uživatel kouli roztáčí a mechanický systém snímá směr a rychlost rotace. Tato data jsou opět použita pro aktualizaci zobrazované scény v HMD, které má uživatel na hlavě a také pro řízení rychlosti rotace. Virtusphere je dodávána ve dvou verzích:

- vojenské – slouží pro simulace bojových operací atd.
- zábavní – slouží pro všechny možné zábavní aplikace, jako jsou počítačové hry, atd.

V zábavní verzi má celý systém tyto specifikace:

- výška - 3 metry
- užitná plocha - 10 metrů čtverečních
- váha - 300 kilogramů
- doba potřebná ke složení – 6 člověko-hodin

Všesměrové pohybové platformy cílí na velmi podobné aplikace jako mnou vyvíjený systém. Liší se v několika faktorech, z nichž ten nejzásadnější je rozhodně cena, která se v roce 2009 pohybovala u systému Virtusphere kolem 15-ti tisíc dolarů a také složitost a komplexnost, která znesnadňuje instalaci a manipulaci se systémem. Navíc trpí zejména systém VirtuSphere problémem setrvačnosti. Když se uživatel rychle rozejde a pak náhle zastaví, setrvačnost koule mu podrazí nohy.

3. MATEMATICKÝ APARÁT

V této kapitole bude uvedeno minimální potřebné množství matematické teorie nutné k pochopení některých částí systému. Vzorce a rovnice budou uvedeny bez odvození a důkazů, pro detailní popis bude čtenář vždy odkázán na příslušné zdroje.

Ústředním matematickým aparátem v mé práci je epipolární geometrie, zejména pak určení pozice a orientace kamery z korespondencí mezi dvěma snímky. Proto je obsah a organizace této kapitoly následující: nejprve jsou stručně uvedeny základy projektivní geometrie a homogenních souřadnic, dále je definován použitý matematický model kamery. Následně je definována homografie a její speciální případ-planární homografie, dále jsou popsány její vlastnosti a uveden jeden z algoritmů pro její výpočet a rozklad.

3.1. HOMOGENNÍ SOUŘADNICE

Homogenní souřadnice jsou důležitý konstrukt, který podstatně zjednodušuje mnoho výpočtů souvisejících především s geometrickými transformacemi. Umožňují popsat nejen základní transformace jako je rotace a translace, ale také složitější perspektivní projekce.

Jak uvádí [3], uspořádaná čtveřice čísel $[x, y, z, w]$ představuje homogenní souřadnice bodu A s kartézskými souřadnicemi $[X, Y, Z]$ ve třech rozměrech, platí-li:

$$X = \frac{x}{w}, Y = \frac{y}{w}, Z = \frac{z}{w}$$
$$w \neq 0$$

Prostor, který tvoří homogenní souřadnice, se nazývá projektivní prostor.

Projektivní souřadnice mají mnoho užitečných vlastností jako je jednotná reprezentace kuželoseček, snadný výpočet průsečíků přímky, apod. Pro moji práci je

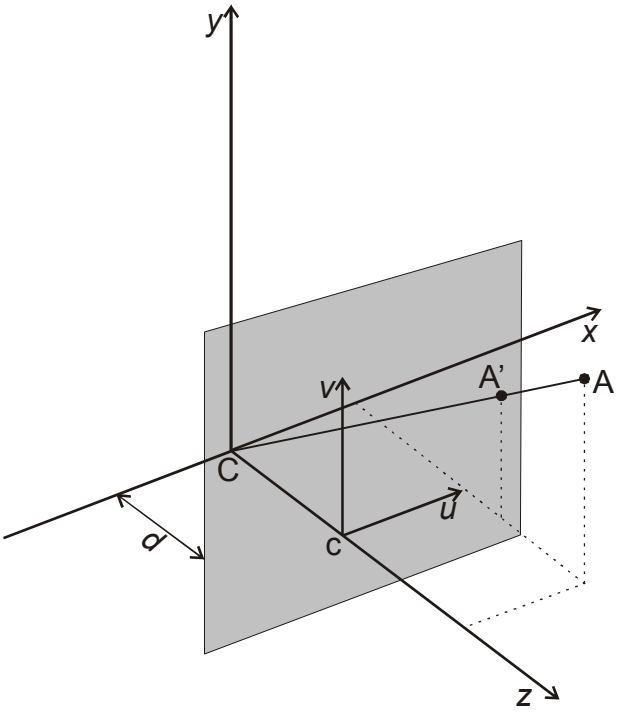
podstatná především možnost jednotné reprezentace geometrických transformací. Bod $A_P = [x, y, z, w]^T$ lze transformovat pomocí jediné transformační matice $T_{4\times4}$, která vznikne složením dílčích transformací. Translaci bodu A_P do bodu $A'_P = [x', y', z', w']^T$ o vektor $[a, b, c]^T$ pak lze zapsat jako

$$A'_P = \underbrace{\begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{T} A_P$$

Další transformace, s kterými lze pomocí homogenních souřadnic pracovat jsou projektivní transformace, které budou popsány v dalších částech.

3.2. MATEMATICKÝ MODEL KAMERY

Model kamery použitý v mé práci je standardní pinhole model, viz Obr. 7, který popisuje transformaci 3D bodu z prostoru na obrazovou rovinu, která reprezentuje snímek kamery. Informace v této podkapitole byly převzaty z [5]. Na Obr. 7 je pro jednoduchost souřadný systém kamery zarovnán se souřadným systém světa, optický střed kamery C je umístěn v počátku souřadného systému, optická osa kamery je totožná s osou z. Ve vzdálenosti d od počátku, je kolmo na osu z umístěna obrazová rovina/průmětna kamery, jejíž souřadný systém (u, v) je rovnoběžný se souřadným systémem kamery. Průmětu bodu A na průmětně odpovídá průsečík paprsku (CA) s průmětnou a je označen A'.



Obr. 7 Pinhole model kamery

Matematicky lze tuto transformaci popsat pro promítaný bod $A = \left[a_x, a_y, a_z\right]^T$ a jeho obraz $A' = \left[a'_x, a'_y, a'_z\right]^T$ bez použití projektivních souřadnic s pomocí podobnosti trojúhelníků jako

$$a'_x = \frac{d a_x}{a_z}$$

$$a'_y = \frac{d a_y}{a_z}$$

$$a'_z = d$$

Využitím projektivních souřadnic můžeme projekci popsat pomocí jedné matice a vyjádřit jí tak jako lineární transformaci. Promítaný bod A i jeho obraz A' rozšíříme o složku a_w, resp. a'_w, a samotnou transformaci pak popíšeme pomocí projekční matice P, která promítaný bod $A = [a_x, a_y, a_z, a_w]^T$ promítne do bodu $A' = [a'_x, a'_y, a'_z, a'_w]^T$ jako

$$\begin{bmatrix} a'_x \\ a'_y \\ a'_z \\ a'_w \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}}_{P} \begin{bmatrix} a_x \\ a_y \\ a_z \\ a_w \end{bmatrix} \tag{3.1}$$

Je zřejmé, že po převedení bodu A do euklidovského prostoru získáme stejný výsledek jako bez použití projektivních souřadnic.

Pro další výpočty bude vhodné uvést ještě jeden možný zápis projekce. Situace zůstává stejná jako na Obr. 7 Pinhole model kamery, bez ztráty obecnosti položíme vzdálenost $d = 1$. Promítaný bod je reprezentován stále v souřadném systému kamery jako $A = [a_x, a_y, a_z, a_w]^T$, promítnutý bod ale nyní budeme reprezentovat v souřadném systému průmětny (u, v) jako $A' = [a'_u, a'_v, a'_w]^T$. V tomto tvaru je zachycena podstata projektivní transformace prováděné kamerou, tedy transformace bodu z 3D do 2D prostoru. Projekční matice P_0 má pak tvar

$$\begin{bmatrix} a'_u \\ a'_v \\ a'_w \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{P_0} \begin{bmatrix} a_x \\ a_y \\ a_z \\ a_w \end{bmatrix} \tag{3.2}$$

V předchozích odstavcích byl model kamery popsán v základní pozici zarovnaný se souřadným systémem světa. V obecném případě je projekční matice P složena ze tří různých matic. První je takzvaná matice vnitřních parametrů kamery

$$K = \begin{bmatrix} \alpha_u & \gamma & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

která popisuje vlastnosti kamery a hlavně umožňuje převod bodu A' ze souřadného systému průmětny (u, v) do souřadného systému kamery, který je na Obr. 7 totožný se souřadným systémem světa. Koeficienty α_u a α_v určují zvětšení kamery ve směru u resp. v, způsobené optickým i vzorkovacím systémem. Bod určený souřadnicemi (u_0, v_0) je průnik optické osy kamery s průmětnou a obvykle se nachází uprostřed snímku. Koeficient γ odpovídá zkreslení a ve většině případů je nulový. Pokud je tato matice pro danou kameru známa, říkáme, že je matice kalibrovaná. Hledání této matice se nazývá kalibrace kamery a metody jejího výpočtu lze nalézt například v [4].

Druhá matice, která popisuje projekci obecné kamery je takzvaná matice vnějších parametrů D. Ta popisuje natočení a posunutí kamery vzhledem k souřadnému systému světa a má tvar

$$D = \begin{bmatrix} R & T \\ 0_3^T & 1 \end{bmatrix}$$

kde R je submatice velikosti 3×3 popisující rotaci a matice T je 3×1 translační matice. Matice D umožňuje transformaci promítaného bodu ze souřadného systému světa do souřadného systému kamery.

Třetí maticí určující výslednou projekční matici P je již zmíněná 3×4 matice P_0 z rovnice (3.2), matice P má tedy tvar

$$P = KP_0D \tag{3.3}$$

a popisuje kompletní transformaci bodu ze souřadného systému světa do souřadného systému průmětny.

3.3. HOMOGRAFIE

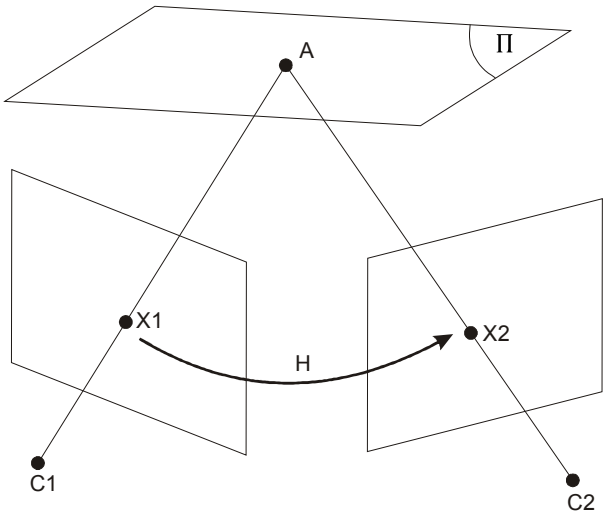
Dle [5] je homografie transformace, která zachovává linearitu, a podprostory jsou pomocí ní mapovány na podprostory stejné dimenze. Může být popsána jako $(n+1) \times (n+1)$ nesingulární matice H, která transformuje X na jeho obraz X' jako

$$X' \simeq HX$$

kde n je dimenze prostoru. Příkladem homografie může být například projekční matice P z rovnice (3.1), která transformuje bod z prostoru P^3 opět do prostoru P^3. V další části popíši speciální případ homografie pro planární scény, ve zkratce uvedu postup pro její nalezení a na závěr popíši možnosti jejího rozkladu.

3.3.1. Planární homografie

V této části budu uvažovat speciální případ, kdy transformovaný bod A leží na obecné rovině Π v prostoru. Tento bod je následně promítnut dvěma různými kamerami do bodů x_1 a x_2, jak je vidět na Obr. 8. Planární homografie pak popisuje transformaci mezi těmito průměty.



Obr. 8 Bod A je první kamerou, se středem promítáníC_1, promítnut do bodu X_1 a druhou kamerou, se středem promítání C_2, promítnut do bodu X_2. Planární homografie H popisuje transformaci mezi body X_1 a X_2.

Jak uvádí [6], lze tuto transformaci odvodit z transformace mezi souřadnými systémy obou kamer, která je složena z rotační matice R a translační matice T. Pro

normálový vektor N roviny Π v souřadném systému první kamery a vzdálenost d mezi rovinou Π a středem promítání první kamery C_1 má pak homografie H tvar

$$H = R + \frac{1}{d} T N^T \tag{3.4}$$

Planární homografie indukovaná rovinou Π pro obrazy (průměty) X_1 a X_2 má pak tvar

$$X_2 \sim H X_1$$

kde operátor ~ odpovídá „až na násobek“. Pokud homografii známe, lze za pomoci původního průmětu určit jeho korespondenci v druhé průmětně. Pokud naopak známe několik průmětů v první průmětně, včetně jejich korespondencí v druhé, lze určit homografii, která popisuje transformaci mezi první a druhou kamerou. Tu lze poté rozložit na jednotlivé strukturní i pohybové parametry znázorněné v rovnici (3.4).

3.3.2. Čtyřbodový algoritmus pro výpočet planární homografie

V této části popíši možnost výpočtu planární homografie z korespondencí [5] ve snímcích. V rámci této podsekce budu používat následující značení:

- X_j je průmět bodu v prvním snímku kamery náležící k j-té korespondenci
- Y_j je průmět bodu v druhém snímku kamery náležící k j-té korespondenci
- $X_j, Y_j \in \mathbb{R}^3$, z-tová souřadnice je ale vždy rovna 1
- celkový počet korespondencí je označen jako n

Jak ukazuje [6], pro výpočet planární homografie postačí už čtyři korespondence tvořené body, z nichž žádné tři nejsou kolineární. Z těchto čtyř korespondencí lze získat prvotní odhad homografie H_L, která se od přesné homografie H liší o násobek λ, tedy

$$H_L = \lambda H \tag{3.5}$$

Matici H_L získáme tak, že nejprve sestrojíme matici $\chi = [a_1, a_2, \dots, a_n]^T \in \mathbb{R}^{3n \times 9}$ z korespondencí X_j a Y_j, kde $a_j = X_j \otimes \widehat{Y}_j$. Operátor $\otimes$ označuje Kroneckerův produkt a $\widehat{Y}_j$ označuje antisymetrickou matici vytvořenou ze složek bodu Y_j. Dále najdeme vektor $H_L^S \in \mathbb{R}^9$ řešením

$$\chi H_L^S = 0$$

Pomocí SVD rozkladu matice χ získáme

$$\chi = U_\chi \Sigma_\chi V_\chi^T$$

a položíme H_L^S rovno devátému sloupci matice V_χ. Vektor H_L^S pak už pouze po sloupcích rozbalíme do matice H_L.

Díky tvaru matice H_L je násobek λ roven druhému vlastnímu číslu matice H_L, tedy

$$\lambda = \sigma_2(H_L)$$

Díky této znalosti je možné matice H_L normalizovat na matici H tak, že

$$H = \frac{H_L}{\sigma_2(H_L)}$$

Odtud získáme matici ve tvaru

[5] Dvojice (X, Y), kde X je průmět bodu ve snímku první kamery a Y je průmět toho samého bodu ve snímku druhé kamery.

$$H = \pm\left(R + \frac{1}{d}TN^T\right)$$

Pro získání jednoho unikátního řešení, využijeme pozitivního omezení hloubky, takže pro j-tou korespondenci (X_j, Y_j) musí platit, že

$$(Y_j)^T HX_j > 0, \forall j = 1, \dots, n$$

3.3.3. Rozklad planární homografie

Často potřebujeme znát jednotlivé pohybové a strukturní parametry, z kterých je planární homografie složená. Rozklad planární homografie z rovnice (3.4) tedy vede k získání parametrů $\left\{R, \frac{1}{d}T, N\right\}$, kde

- R je 3×3 rotační matice, popisující rotaci mezi první a druhou kamerou
- T je 3×1 translační vektor, popisující translaci mezi první a druhou kamerou
- N je normálový vektor plochy Π, viz podsekci 3.3.1
- d je vzdálenost středu promítání první kamery od plochy Π

Je vhodné podotknout, že vzhledem k tomu, že při promítnutí bodu se ztrácí veškerá informace o hloubce d, je možné získat translační vektor T až na násobek $\frac{1}{d}$.

Pro rozklad je nejprve nutné spočítat SVD rozklad skalárního součinu $H^T H$, jako

$$H^T H = V\Sigma V$$

kde $\Sigma = \mathrm{diag}\{\sigma_1^2, \sigma_2^2, \sigma_3^2\}$ je diagonální matice vlastních čísel. Sloupcové vektory matice V označíme jako v_1, v_2, v_3 a pomocí nich definujeme dva jednotkové vektory

$$u_1 = \frac{\sqrt{1-\sigma_3^2}v_1 + \sqrt{\sigma_1^2 - 1}v_3}{\sqrt{\sigma_1^2 - \sigma_3^2}}$$

$$u_2 = \frac{\sqrt{1-\sigma_3^2}v_1 - \sqrt{\sigma_1^2 - 1}v_3}{\sqrt{\sigma_1^2 - \sigma_3^2}}$$

Pomocí nich definujeme matice

$$U_1 = [v_2, u_1, v_2 \times u_1], \qquad W_1 = [Hv_2, Hu_1, (H \times v_2)Hu_1]$$

$$U_2 = [v_2, u_2, v_2 \times u_2], \qquad W_1 = [Hv_2, Hu_2, (H \times v_2)Hu_1]$$

Tabulka Tab. 1 pak ukazuje čtyři možná řešení rozkladu.

Řešení 1	$R_1 = W_1 {U_1}^T$ $N_1 = v_2 \times u_1$ $\frac{1}{d}T_1 = (H - R_1)N_1$	Řešení 3	$R_3 = R_1$ $N_3 = -N_1$ $\frac{1}{d}T_3 = -\frac{1}{d}T_1$
Řešení 2	$R_2 = W_2 {U_2}^T$ $N_2 = v_2 \times u_2$ $\frac{1}{d}T_2 = (H - R_2)N_2$	Řešení 4	$R_4 = R_2$ $N_4 = -N_2$ $\frac{1}{d}T_4 = -\frac{1}{d}T_2$

Tab. 1 Čtyři možná řešení rozkladu planární homografie. Tabulka převzata z [6].

Ke snížení možného počtu řešení můžeme zavést opět omezení pozitivní hloubky, vyplývajícího z fyzického omezení kamery, která může vidět pouze body, kterou jsou před ní. Za předpokladu, že je osa z totožná s optickou osou kamery, musí být $N^T e_3 = n_3 > 0$.

To nám počet možných řešení redukuje na dvě. Jedno unikátní řešení pak můžeme z těchto dvou vybrat na základě nějaké apriorní informace o pohybu kamery nebo vlastnostech roviny Π.

4. NÁVRH SYSTÉMU

Problém řešený navrhovaným systémem je poměrně komplexní a sestává z několika částí, z nichž každá se dotýká odlišné oblasti počítačové vědy. Cílem je vytvořit systém, který bude zjišťovat uživatelovu pozici a směr pohledu a na základě těchto dat bude aktualizovat zobrazovanou virtuální scénu. Z toho vyplývá jeden z požadavků kladených na systém a tou je schopnost provozu v reálném čase. Vzhledem k charakteru práce je taktéž žádoucí, aby náklady na vývoj a provoz systému byly co nejnižší.

Na základě důvodů, které budu postupně uvádět při popisu jednotlivých modulů a jejich implementací níže, jsem dospěl k následujícímu návrhu:

- Uživatel má uprostřed temene hlavy připevněnou kameru, která směřuje kolmo vzhůru
- Strop je osazen významnými body, které jsou kamerou snímány
- Při pohybu uživatele je z rozdílu pozice významných bodů mezi dvěma snímky zrekonstruována translace a rotace kamery
- Tyto transformace jsou interpretovány jako pohyb uživatele a změna směru jeho pohledu, na základě čehož je upravena virtuální scéna, která je uživateli prezentována pomocí HMD.

To sestává z řady dílčích kroků, které při dobrém stupni abstrakce mohou být zcela nezávislé. Bylo tedy vhodné tento problém rozdělit na nezávislé části a ty řešit odděleně, díky čemuž získalo řešení modulární charakter, se všemi výhodami, které tento návrh přináší.

Problém jsem rozdělil na tyto hrubé celky, která se ještě dále rozpadají na dílčí části:

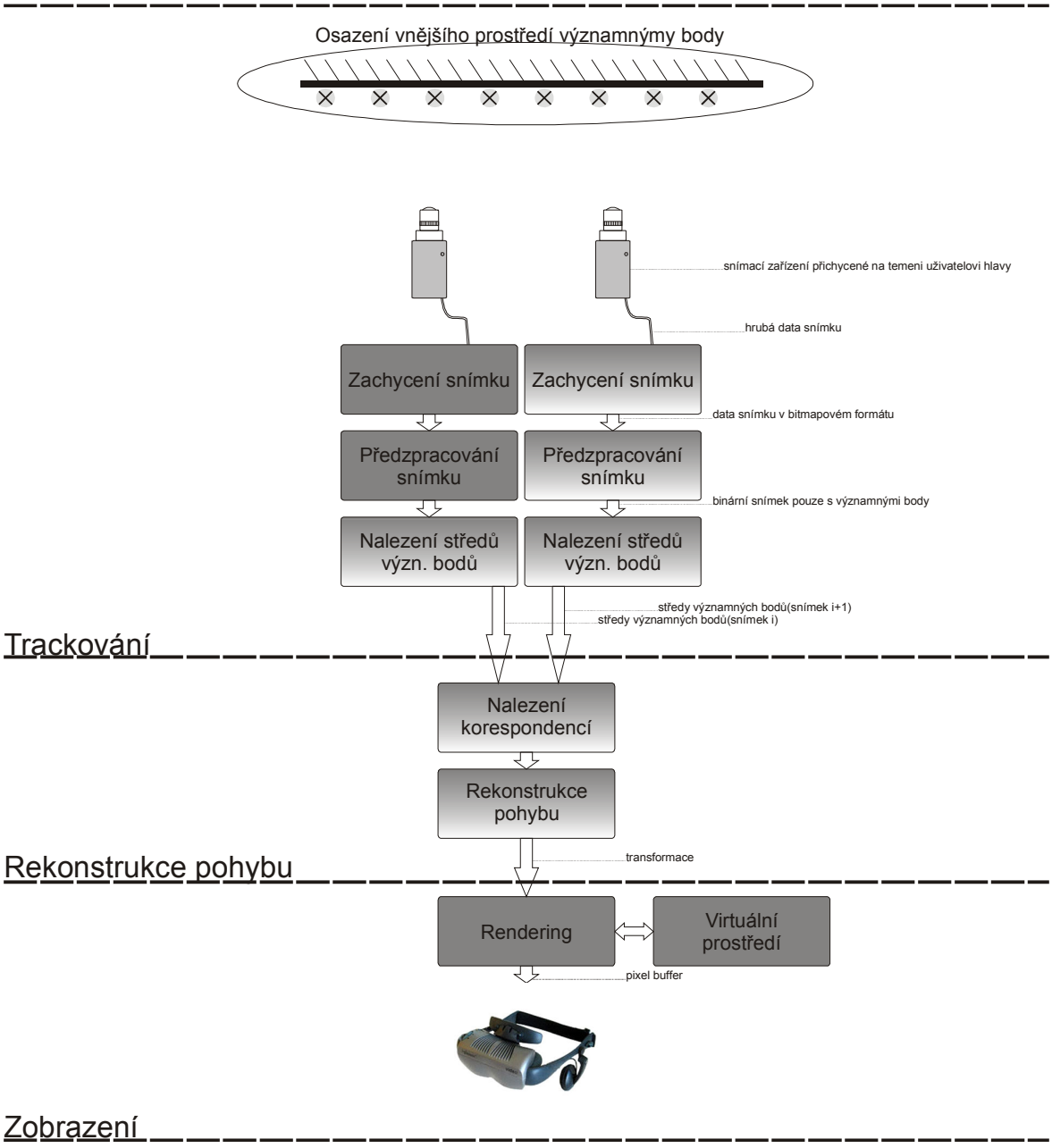
1. Získání dat o uživatelově pozici a směru pohledu
    a. Vliv vnějšího prostředí na měření
    b. Měření
    c. Předzpracování dat pro rekonstrukci
2. Rekonstrukce uživatelova pohybu z naměřených dat
    a. Algoritmus rekonstrukce
    b. Reprezentace
3. Zobrazení uživatelova pohybu
    a. Virtuální scéna
    b. Zobrazení

Ad 1. První celek řeší problém samotného trackování, tedy jakým způsobem zaznamenávat uživatelův pohyb, jakým způsobem a zda vůbec k tomu využít okolního prostředí a jak hrubá data z měření předzpracovat tak, aby následná rekonstrukce byla co nejjednodušší.

Ad 2. Druhý celek řeší numerickou část rekonstrukce. To znamená, jakým způsobem ze vstupních (předzpracovaných) dat rekonstruovat původní uživatelův pohyb a to co nejpřesněji, ideálně bez odchylky od skutečného pohybu. Součástí této části je také jakým způsobem pohyb reprezentovat.

Ad 3. Třetí celek už řeší pouze přenesení zrekonstruovaného pohybu do vizuální podoby a jeho prezentace uživateli ve vhodném virtuálním prostředí.

Pro každý z celků jsem následně navrhl jednotlivé moduly, které by řešily problémy daného celku a opět jsem se snažil tyto moduly dělat co nejvíce nezávislé. Tyto moduly jsou stále ještě určitou mírou abstrakce a neobsahují konkrétní implementace, ta je až na nejnižší úrovni a umožňuje tak testovat více možných řešení pro jednotlivé moduly. Přehled celého systému včetně rozdělení jednotlivých celků do modulů je zobrazeno na Obr. 9. Na obrázku jsou sice dvě kamery, to má však za cíl pouze ilustrovat dva zachycené snímky, pomocí nichž je následně rekonstruován pohyb. Ve skutečnosti je kamera samozřejmě pouze jedna. Pokud dál v textu budu mluvit o dvou kamerách, jsou tím myšleny dva různé snímky, zachycené jednou kamerou v různých pozicích.



Obr. 9 Rozdělení celého systému na jednotlivé celky(čárkovaná čára), jejich rozdělení do modulů(plná čára) a součinost mezi moduly(šipky).

4.1. ZÍSKÁNÍ DAT O UŽIVATELOVĚ POZICI A SMĚRU POHLEDU

Jak jsem uvedl v kapitole 2, existuje mnoho způsobů, jak uživatelovu pozici a směr pohledu měřit. Optický systém trackování byl nakonec vybrán jako nejvhodnější, vzhledem k těmto výhodám:

1. Implicitně netrpí problémem driftu jako inerciální trackery.
2. Neomezuje uživatele v pohybu.
3. Obecně má nízkou cenu, a přesto může dosahovat vysoké přesnosti.
4. Při vhodné konfiguraci umožňuje více stupňů volnosti i s jedním snímačem.

Optické trackování však trpí také některými nedostatky jako:

1. Problémy se zákryty a viditelností
2. Výpočetní náročností
3. Vyžaduje konkrétní podmínky prostředí, jako je osvětlení a objekty, vůči kterým je trackování vztahováno

Je dobré podotknout, že nevýhody 2 a 3 na sobě těsně závisí. Čím horší jsou podmínky prostředí, tím větší je obvykle i výpočetní náročnost celého systému, protože k získání potřebné informace je nutné snímek silněji filtrovat, transformovat a jinak zpracovávat. Z toho důvodu byl už do prvního návrhu systému zapracován předpoklad osazení vnějšího prostředí významnými body, které budou mít vlastnosti usnadňující jejich následnou segmentaci a tedy i výsledné trackování. Výstupem této fáze jsou pouze středy významných bodů, čemuž je uzpůsobeno předzpracování zachyceného snímku.

4.1.1. Snímací zařízení a významné body

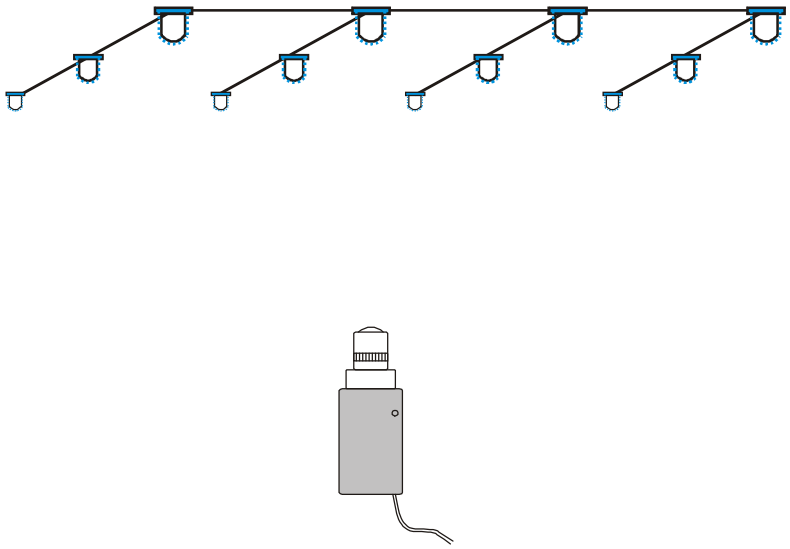
Přestože jsem se snažil každý z modulů navrhnout nezávisle, snímací zařízení je nevyhnutelně závislé na snímaných významných bodech, proto tyto dva moduly popíši společně. Kvůli výpočetnímu modelu popsaném v 4.2 může být rozmístění významných bodů náhodné, je však nutné, aby v každém snímku byly viditelné minimálně čtyři body.

VERZE 1 – AKTIVNÍ VÝZNAMNÉ BODY

Aby referenční body byly snadno segmentovatelné, bylo pro jejich zhotovení využito obyčejných LED modré barvy. Z těchto LED bylo nutné vytvořit síť, která by byla přichycena ke stropu a vytvořila tak operační prostor pro systém, jak ukazuje Obr. 10. Návrh pro tuto síť byly postupně tyto:

- Propojení LED kabelem a jeho přichycení na strop
  - Málo pevná síť – snadné poškození
- Připevnění LED na textilní síť a vedení kabelu po této síti
  - Pevnější konstrukce, snazší upevnění na strop
  - Špatně dostupný materiál, snadno se zamotá
- Vytvoření sítě ze zahrádkářského pletiva, které by zároveň fungovalo jako vodič – diody propojené přímo dráty z pletiva
  - Pevná konstrukce, dostupný materiál, relativně snadné vytvoření velké sítě
  - Velmi obtížné vytvoření obvodu – při prostřihání pletivo ztrácí svoji pevnost, velké riziko zkratu
- Osazení panelu z plexiskla LED a jejich propojení kabelem
  - Pevná konstrukce, snadno lze přichytit na strop

Na základě poslední varianty pak vznikl prototyp tvořený dohromady 24 diodami.



Obr. 10 Schéma snímacího zařízení s aktivními významnými body

VERZE 2 – PASIVNÍ VÝZNAMNÉ BODY

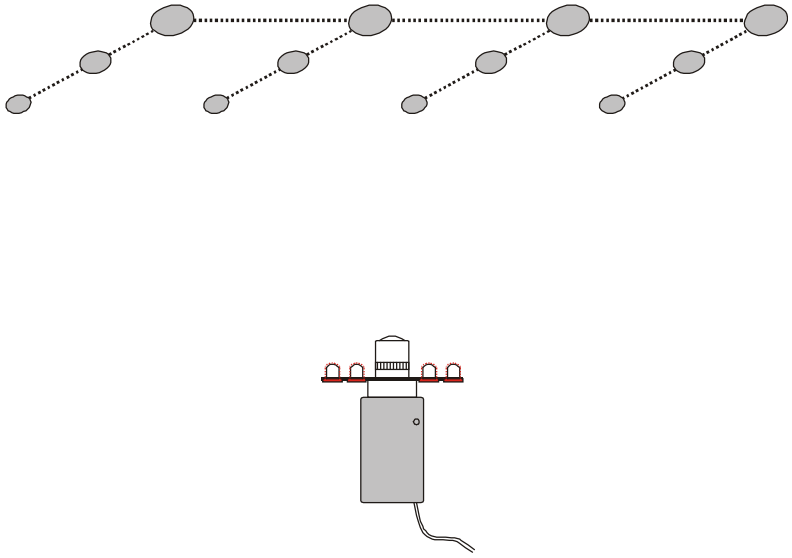
Všechny varianty ve verzi 1 měly jeden zásadní nedostatek, a tím byla složitá výroba sítě referenčních bodů. Pro pokrytí větší plochy by bylo nutné strávit velké množství času propojováním LED, jejich osazováním panelů, atd. Navíc by tato část zbytečně zvyšovala náklady na celý systém. Proto byla představena myšlenka pasivních významných bodů, které by pouze odrážely světlo, a aktivní zdroj by byl spojen s kamerou. Pokrytí stropu významnými body a zvětšování operačního prostoru by pak bylo velmi snadné.

Pro úspěšnou realizaci bylo ale nutné nejprve najít odpověď na tyto otázky:

a) Jaký zdroj světla použít, aby byl dostatečně výkonný a přitom kompaktní?
b) Jak vyrobit významné body, aby měli velkou intenzitu a úhel odrazu?

Při použití obyčejných LED a běžné reflexní folie docházelo k tomu, že se světlo neodráželo pouze od fólie, ale i samotného stropu, což vedlo ke zhoršené segmentaci a chybám v trackování. Navíc trpěl tento systém velkým světelným útlumem vzhledem ke vzdálenosti, takže při větší vzdálenosti kamery od stropu už odražené světlo téměř nebylo možné zachytit.

Řešením, které zodpovídalo snadno obě dvě otázky, se ukázaly být infra-LED. Spolu s tímto zdrojem světla byla použita speciální reflexní folie firmy 3M, která odráží světlo i pod malým úhlem, takže nebyla nutná speciální konstrukce významných bodů. Schéma tohoto řešení lze vidět na Obr. 11, podrobný popis referenčních bodů a úpravy kamery lze najít v kapitole 7.



Obr. 11 Schéma snímacího zařízení s pasivními významnými body a přídavným infračerveným osvětlením

4.1.2. Zachycení snímku

Zachycení snímku je čistě záležitost softwarové implementace, proto bude tento modul diskutován detailně v 6.2.

4.1.3. Předzpracování snímku

Aby bylo možné snadno určit středy významných bodů, je nutné vstupní snímek segmentovat a oddělit významné body od pozadí. Jak bylo již dříve zmíněno, byly významné body spolu se snímacím zařízením konfigurovány tak, aby byla segmentace co nejjednodušší. Díky tomu mohl být tento modul silně zjednodušen a optimalizován, a přispívá tak k celkové výpočetní zátěži minimálně.

PŘEVOD DO ŠEDOTÓNOVÉHO SPEKTRA

Díky výše zmíněné konfiguraci snímacího zařízení a významných bodů je nejcennější informací ve snímku jas jednotlivých pixelů. Zachycený snímek ze snímacího zařízení je ale v RGB reprezentaci, pro získání jasové informace je tedy nutné celý snímek převést do šedotónového spektra. Pokud označíme pixel na pozici (i, j) p^{ij} vstupního RGB snímku a jednotlivé barevné složky tohoto pixelu $\left(p_R^{ij}, p_G^{ij}, p_B^{ij}\right)$, potom jasovou složku g_Y^{ij} pixelu g^{ij} šedotónového snímku získáme jako

$$g_Y^{ij} = 0.21 p_R^{ij} + 0.71 p_G^{ij} + 0.07\, p_B^{ij} \tag{4.1}$$

Jak uvádí Skala[7], v RGB reprezentaci platí, že pro poměr barevných složek

$$r: g: b = 1: 1: 1$$

je poměr jasů

$$1: 4{,}6: 0{,}06$$

Každá ze složek tedy obsahuje nestejnou část jasu, proto má rovnice (8.1) tvar váženého průměru, jehož váhy zajistí správnou transformaci jasové složky a vyvážený kontrast výsledného snímku, který je pro další zpracování důležitý. Jednotlivé váhy byly získány z [8].

BINARIZACE SNÍMKU

Pro oddělení významných bodů od pozadí je použito klasického algoritmu prahování. Vzhledem k předchozímu kroku je snímek převeden do šedotónového spektra a pixel g^{ij} vstupního snímku I má pouze jedinou složku g_Y^{ij}, která obsahuje hodnotu v intervalu $\langle 0{,}255 \rangle$ reprezentující jas. Pro předem zvolený práh T nabývá pak pixel t^{ij} výstupního snímku I_T hodnotu 0 nebo 1 podle následujícího algoritmu

$$t^{ij} = 1 \; pro \; g_Y^{ij} \geq T$$

$$t^{ij} = 0 \; pro \; g_Y^{ij} < T$$

Určení prahu T je pro další zpracování kritické. Ačkoliv existuje řada algoritmů pro automatické určení prahu, jako je například OTSU[9], určení prahu je v mé práci plně manuální. Jak bylo již dříve zmíněno, konfigurace významných bodů spolu se snímacím zařízením zajišťuje velmi dobrý kontrast mezi významnými body a pozadím, je tedy velmi snadné pro uživatele práh ručně nastavit. Navíc mají prahované body přibližně konstantní jas po celou dobu běhu systému, není tedy nutné práh v průběhu měnit. Poslední výhodou manuálního určení prahu oproti automatickým metodám je jeho nulový příspěvek k výpočetní náročnosti.

NORMALIZACE A ODSTRANĚNÍ ŠUMU

Při segmentaci snímku bývá posledním krokem takzvaná normalizace. V této fázi je snímek zbaven chybně segmentovaných pixelů, vzniklých špatně určeným prahem, chybou snímacího zařízení atd. Vzhledem k podobě snímků, které obsahují souvislé přibližně kruhové oblasti významných bodů, má šum charakter izolovaných pixelů. Proto byl pro jeho odstranění zvolen dobře známý algoritmus otevření [8]. Otevření je morfologický operátor, který provede erozi celého snímku, následovanou dilatací. To má za následek vyhlazení obrysů objektů, odstranění malých objektů a přerušení úzkých spojení, což vede k tomu, že je ze snímku odstraněn šum a obrysy významných bodů jsou vyhlazeny, díky čemuž má následující fáze lepší výsledky.

4.1.4. Nalezení středů významných bodů

Vstupem tohoto modulu je segmentovaný snímek, kde všechny pixely s hodnotou 1 náleží významným bodům. Výstupem tohoto modulu, stejně jako kompletního celku pro získání dat o uživatelově pozici a směru pohybu, jsou středy významných bodů. K jejich získání jsem použil několik přístupů, které nyní popíši.

HLEDÁNÍ POMOCÍ HRANOVÉHO OPERÁTORU

Prvním přístupem bylo využití gradientního operátoru prvního řádu pro detekci hran. Jako operátor byl zvolen Sobelovův operátor, jehož konvoluční jádra jsou [8]:

$$G_Y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} \; G_X = \begin{bmatrix} -1 & 0 & -1 \\ -2 & 0 & -2 \\ -1 & 0 & -1 \end{bmatrix}$$

Tento přístup vycházel z předpokladu, že významné body tvoří kruhy v segmentovaném snímku. Jeho cílem bylo získat hrany náležící významným bodů a z nich získat jeho střed. Navržená myšlenka byla následující:

1. Posunout Sobelův operátor po snímku
2. Pokud nalezena hrana pokračovat, jinak zpět na 1
3. Posunout operátor ve směru určeném maximálním gradientem
4. Pokud navštíven výchozí pixel pokračovat, jinak zpět na 3
5. Získány hrany náležící významnému bodu. Pokud zpracován celý snímek – konec, jinak zpět na 1.

Tento přístup byl velmi rychle zamítnut z důvodu silné nestability, pokud nebyla nalezena hrana, celý algoritmus se zacyklil.

CLUSTEROVÁ ANALÝZA

Druhou myšlenkou bylo využít clusterové analýzy pro nalezení středů významných bodů. Nejprve byla vytvořena datová struktura cluster, která obsahuje pole bodů [6], náležejících cluster a střed clusteru, viz Tab. 2. Počet $clusterPoints$ bude označen jako k.

Cluster
• $Point[]\ clusterPoints$ • $Point\ center$

Tab. 2 Datová struktura cluster

Nad touto datovou strukturou pak byly definovány tyto operace:

- $CREATE(Point\ pt)$ – vytvoří nový cluster a bod pt je určen jako střed clusteru.
- $INSERT(Point\ pt)$- která se pokusí vložit bod pt do příslušného clusteru. Vložení je úspěšné, pokud je

$$\|center - pt\| \leq MAX\_RADIUS$$

kde MAX_RADIUS je předem určený práh definující maximální možný poloměr clusteru. Pokud je tato podmínka splněna, je bod pt přidán do pole $clusterPoints$ a střed clusteru $center$ je aktualizován podle následujícího vzorce

$$center = \frac{1}{k}\sum clusterPoints$$

Pokud tato podmínka splněna není, skončí operace $INSERT$ chybovým stavem.

Navržený algoritmus pak projde celý snímek I, kde $I(x,y)$ odpovídá hodnotě pixelu na pozici (x,y) a pixely, resp. jejich souřadnice, náležící významným bodům, rozdělí do clusterů. Po rozdělení pak algoritmus vyřadí ty clustery, které jsou menší než předem nastavený práh $MIN\_CLUSTER\_POINTS$. Algoritmus pracuje následovně:

1. Inicializovat pole `clusters` pro ukládání clusterů
2.
```
for x:=0;x<=IMAGE_WIDTH{
for y:=0;y<=IMAGE_HEIGHT{
      if(I(x,y)>0){
            bool res = false;
            /*Projdi všechny vytvořené clustery a pokus
            se do nich vložit bod*/
            foreach(Cluster c in clusters){
                  res = c.INSERT(Point(x,y));
                  /*Skonči, pokud je vložení úspěšné*/
                  if(res == true){
                  break;
            }
            /*Pokud bod nebyl vložen do žádného clusteru,
            vytvoř nový*/
            if(res == false){
                  clusters.add(CREATE(Point(x,y)));
            }
      }
}
```

[6] Dvojice (x,y) - souřadnice pixelu

}

3. Z pole `clusters` vyřadit ty clustery, pro něž platí $k < MIN\_CLUSTER\_POINTS$

Očekávaná složitost tohoto algoritmu je zhruba $O(N)$, kde N je počet pixelů snímku. Bod číslo 3 je v algoritmu jednak z důvodu odstranění clusterů, které tvoří šum, tedy typicky několik málo pixelů vzniklých špatnou segmentací, ale také pro odstranění clusterů, které tvoří významné body na okraji snímku. Ty nejsou na snímku celé, je tedy zřejmé, že díky výše zmíněnému naivnímu určení středu pomocí aritmetického průměru, by jejich střed nebyl určen korektně a ovlivnilo by to tak další zpracování.

Pro získání shluků tvořících významné body byl nakonec použit algoritmus detekce shluků knihovny FiltersDll[28]. Ten byl zvolen především z důvodu stability, spolehlivosti a optimalizace. Algoritmus nalezne shluky a každý z nich uloží do datové struktury obsahující velké množství informací o shluku, včetně jeho středu. Je také možné nastavit minimální velikost shluku, čehož bylo využito z důvodů zmíněných v předešlém odstavci.

4.2. REKONSTRUKCE POHYBU

Tento celek má za úkol ze středů významných bodů získaných v předchozí fázi rekonstruovat pohyb kamery a ve vhodné reprezentaci ho předat zobrazovacímu celku. Problém rekonstrukce 3D pohybu ze série snímků je v literatuře označován jako *egomotion estimation* a techniky řešení tohoto problému mohou být rozděleny na dva základní přístupy:

- metody pracující s diskrétní časovou reprezentací
- metody pracující s okamžitou-spojitou časovou reprezentací

První zmíněná metoda reprezentuje pohyb jako sérii statických snímků, v nichž je následně analyzována změna pozice sledovaných bodů. Druhá metoda přistupuje k pohybu ve snímcích jako ke spojitému problému a k jeho analýze využívá takzvaných *image velocities*, tedy jakýchsi rychlostí snímku, vzniklých právě díky pohybu. Tuto metodu využívá k určení pohybu například [10] a [11]. Já jsem se však ve své práci použil první metodu a to ze dvou důvodů:

1. Nižší výpočetní náročnost
2. Možnost absolutního pozicování

Vzhledem k vybrané metodě řešení jsou nutné alespoň dva po sobě jdoucí snímky k získání informace o pohybu. Vstupem jsou tedy dvě množiny, které obsahují obrazy středů významných bodů ve dvou snímcích. Mezi těmito dvěma množinami je nejprve nutné najít korespondence, tedy přiřadit středy významných bodů v první množině středům v druhé množině, které odpovídají stejným významným bodům. Následně je na základě těchto korespondencí rekonstruován pohyb kamery. Zobrazovací části jsou poté předány informace o pohybu kamery v podobě translačního vektoru a rotací kolem jednotlivých os. V této podkapitole se tedy věnuji nejprve nalezení korespondencí, poté popíšu různé přístupy, které jsem použil při rekonstrukci pohybu včetně diskuze jejich vlastností, na závěr pak uvedu způsob, jakým je informace o pohybu předána zobrazovací části.

4.2.1. Nalezení korespondencí

Jak bylo řečeno výše, nalezení korespondencí má za úkol nalézt dvojice středů, které odpovídají stejným významným bodům. Formálně zapsáno má první množina tvar

$$S_1 = \{X_1, X_2, \dots, X_n\}$$

kde X_i je dvourozměrný bod

$$X_i = (x_i^1, x_i^2)$$

a n je celkový počet bodů v první množině. Druhá množina má pak tvar

$$S_2 = \{Y_1, Y_2, \dots, Y_m\}$$

kde Y_j je opět dvourozměrný bod

$$Y_j = \left(y_j^1, y_j^2\right)$$

a m je celkový počet bodů ve druhé množině. Obecně může platit, že

$$m \neq n$$

protože vlivem pohybu mezi dvěma snímky může přestat být některý z významných bodů viditelný a jeho střed pak bude pouze v množině S_1. Úkolem je tedy najít takové dvojice $\left(X_i, Y_j\right)$, které odpovídají stejným středům významných bodů.

Mezi body v množinách S_1 a S_2 zřejmě platí následující vztah

$$Y_j = X_i + \Delta \boldsymbol{t}_{ij}$$

$$\forall i = 1, \dots, n; \forall j = 1, \dots, m$$

kde $\Delta \boldsymbol{t}_{ij}$ je dvourozměrný vektor posunutí. Pro určení korespondence jsem určil jednoduchou podmínku pro $\Delta \boldsymbol{t}_{ij}$. Bod Y_q je korespondencí X_p pro takové p a q, pro které platí, že

$$\Delta \boldsymbol{t}_{pq} = \min_{i,j} \Delta \boldsymbol{t}_{ij} \tag{4.2}$$

$$\forall i = 1, \dots, n; \forall j = 1, \dots, m$$

Jiným slovy hledáme takové dva body z množin S_1 a S_2, které jsou si nejblíže. Pouze tato podmínka však pro korektní určení korespondencí nestačí. Vzhledem k tomu, že obecně

$$|S_1| \neq |S_2|$$

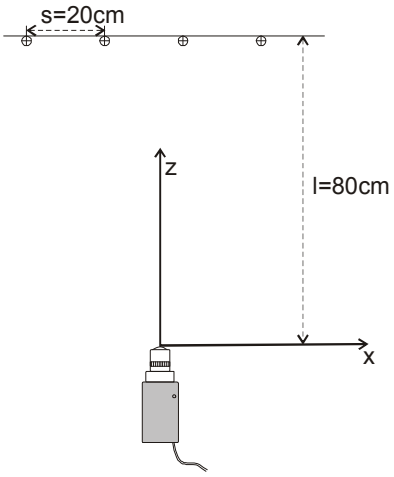
mohl by nastat případ, kdy jeden bod z množiny S_1 bude mít více korespondencí v S_2, což je nepřípustné. Proto je omezena maximální možná vzdálenost korespondujících bodů na c_{max} a k podmínce (4.2) tedy přibývá ještě

$$\left\|\Delta \boldsymbol{t}_{pq}\right\| \leq c_{max}$$

K použití kritéria minimální vzdálenosti mě dovedla následující jednoduchá úvaha, která vycházela z modelové situace reprezentující standardní podmínky při provozu systému. Pomocí ní dokážu, že toto velmi jednoduché kritérium postačí k nalezení korespondence.

Nechť je konfigurace kamery a významných bodů následující:

- vzdálenost l průmětny kamery od stropu je $80cm$
- vzdálenost středů významných bodů s je $20\ cm$
- orientace souřadného systému viz Obr. 12



Obr. 12 Orientace souřadného systému kamery

Pro jednoduchost budu uvažovat průměrnou rychlost translačního pohybu v rámci systému

$$\bar{v} = 5\,km/h \doteq 1.4\,m/s$$

což odpovídá průměrné rychlosti lidské chůze. Pokud budu dále předpokládat, že průměrná snímková frekvence kamery za sekundu je

$$f = 25\,Hz$$

potom dostávám maximální možnou vzdálenost, kterou může kamera urazit v rámci jednoho snímku

$$t_{max} = \frac{\bar{v}}{f} \doteq 5.5\,cm/frame$$

Nyní naleznu souřadnice průmětů dvou náhodně zvolených sousedících středů významných bodů

$$A = [-10, 0, 80, 1]^T \; B = [10, 0, 80, 1]^T$$

podle rovnice (3.3)(8.1) z podkapitoly 3.2 pro první i druhou kameru. Pro jednoduchost uvažuji, že jsou kamery kalibrované, tedy matice K je rovna jednotkové matici

$$K = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

a mohu jí tedy vynechat. Bez újmy na obecnosti mohu uvažovat souřadný systém první kamery zarovnaný se souřadným systémem světa, tedy matice vnějších parametrů první kamery je taktéž rovna jednotkové matici

$$D_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Průměty bodů A a B pro první kameru jsou tedy

$$A_1' = P_0 D_1 A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} -10 \\ 0 \\ 80 \\ 1 \end{bmatrix} = \begin{bmatrix} -10 \\ 0 \\ 80 \end{bmatrix}$$

$$B_1' = P_0 D_1 A = \begin{bmatrix} 10 \\ 0 \\ 80 \end{bmatrix}$$

Druhou kameru jsem posunul o předpokládanou maximální vzdálenost t_{max} ve směru osy x, matice vnějších parametrů druhé kamery má tedy tvar

$$D_2 = \begin{bmatrix} 1 & 0 & 0 & 5.5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

a průměty bodů A a B pro druhou kameru jsou tedy

$$A_2' = P_0 D_2 A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 5.5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -10 \\ 0 \\ 80 \\ 1 \end{bmatrix} = \begin{bmatrix} -4.5 \\ 0 \\ 80 \end{bmatrix}$$

$$B_2' = P_0 D_1 A = \begin{bmatrix} 15.5 \\ 0 \\ 80 \end{bmatrix}$$

Je zřejmé, že korespondence budou určeny korektně jako dvojice

$$(A_1', A_2')$$
$$(B_1', B_2')$$

protože na první pohled platí, že

$$\|A_1' - A_2'\| < \|B_1' - A_2'\|$$

a

$$\|B_1' - B_2'\| < \|A_1' - B_2'\|$$

Pro tyto parametry, které vychází z průměrného provozu systému, vyplývá, že kritérium minimální vzdálenosti bude dostačující pro určení korespondencí. Také je z uvedeného zřejmé, že vzdálenost významných bodů a maximální rychlost pohybu spolu úzce souvisejí. Možná omezení, která tento přístup má, včetně rozboru rotačního pohybu lze najít v sekci 8.1.1.

4.2.2. Rekonstrukce pohybu z korespondencí

Výstupem předchozí fáze jsou dvojice bodů, takzvané korespondence$(X_i, Y_i)_i$, které v sobě nesou informaci o pohybu kamery mezi snímky. Úkolem této části je z nich tuto informaci zpětně získat. V průběhu vývoje systému prošla tato část několika změnami, které měly za úkol zpřesnit výslednou rekonstrukci.

GEOMETRICKÉ ŘEŠENÍ

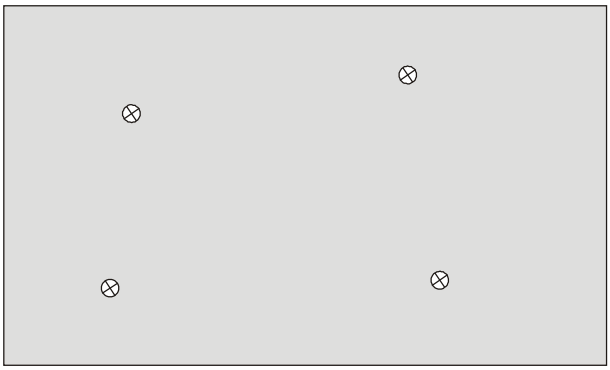
Původní myšlenka vycházela z geometrického pohledu na získané korespondence. Korespondující body lze chápat jako počáteční a koncové body vektorů

$$\boldsymbol{m}_i = Y_i - X_i$$

které reprezentují pohyb kamery a vytváří pohybové pole. Tato myšlenka není rozhodně nová, tyto takzvané motion vektory tvoří základ video kompresních algoritmů, jako například u standardů MPEG. Algoritmy optického toku pracují taktéž s vektory charakterizujícími pohyb kamery. V obou dvou případech jsou však tyto vektory získány z rozdílu po sobě jdoucích snímků a určeny pro každý pixel nebo skupiny pixelů a jsou jinak zpracovávány. Já jsem se pro rekonstrukci pohybu pokusil vektory analyzovat z geometrického hlediska, tedy zkoumat jejich velikosti a orientace a podle toho určit typ pohybu[7], jeho směr a velikost. Tento přístup v podstatě ignoruje matematický model kamery, z čehož plyne řada problémů a omezení, které jsou zmíněné níže. Proto jsem od tohoto řešení velmi záhy upustil, přesto ho zde pro úplnost zmíním. Vycházel jsem z jednoduché simulace, která obsahovala pouze 4 významné body a jejich známé středy,

[7] Translace v rovině kolmé na osu snímání, rotace kolem této osy, atd.

viz Obr. 13. Poté jsem postupně simuloval různé typy pohybů a zkoumal jejich vliv na pohybové pole.



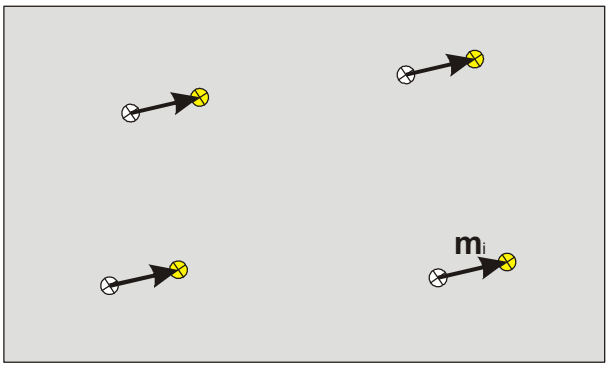
Obr. 13Simulační snímek se čtyřmi významnými body a jejich středy

Nejprve jsem začal s jednoduchým translačním pohybem v rovině kolmé na osu snímání, tato osa směřovala kolmo vzhůru. Jak je patrné z Obr. 14, je pro tento typ pohybu velikost i orientace u všech pohybových vektorů $\boldsymbol{m}_i$ stejná. Pro celkový počet n pohybových vektorů by byl pohyb translační, pokud

$$\left(\frac{\boldsymbol{m}_1}{\|\boldsymbol{m}_1\|}\right)^T \left(\frac{\boldsymbol{m}_i}{\|\boldsymbol{m}_i\|}\right) = 1 \pm \delta \wedge \|\boldsymbol{m}_1\| - \|\boldsymbol{m}_i\| = 0 \pm \delta$$

$$\forall i = 2, \dots, n$$

kde δ značí toleranci, která je do podmínky začleněna kvůli možnému šumu zanesenému z předchozích fází.



Obr. 14 Pole pohybových vektorů při translačním pohybu v rovině kolmé na optickou osu kamery

Pokud se podaří detekovat tento druh pohybu, zjištění směru a velikosti pohybu už je velmi snadné. Je zřejmé, že směr odpovídá přímo některému z vektorů $\boldsymbol{m}_i$, díky možným odchylkám ve směru jednotlivých vektorů je ale vhodné určit jako směr D průměr vektorů

$$D = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{m}_i$$

Velikost pohybu η je pak

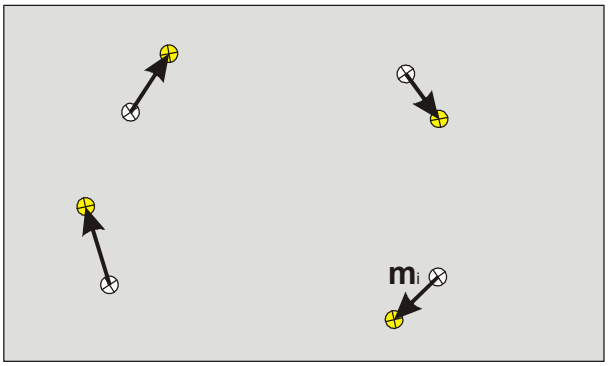
$$\eta = \frac{1}{\lambda}\|D\|$$

kde λ je neznámý koeficient, odpovídající vzdálenosti průmětny kamery od významných bodů. Tento koeficient lze chápat jako parametr určující rychlost pohybu a nastavit ho ručně.

Dalším typem pohybu, který jsem analyzoval, byl rotační pohyb kolem osy snímání kamery, s kamerou směřující opět kolmo vzhůru. Na Obr. 15 lze vidět vliv tohoto pohybu

na pohybové pole simulačního snímku. Pro detekci tohoto pohybu lze využít různé orientace vektorů $\boldsymbol{m}_i$, pokud tedy

$$\left(\frac{\boldsymbol{m}_1}{\|\boldsymbol{m}_1\|}\right)^T \left(\frac{\boldsymbol{m}_i}{\|\boldsymbol{m}_i\|}\right) < 1 - \delta$$
$$\forall i = 2, \dots, n$$



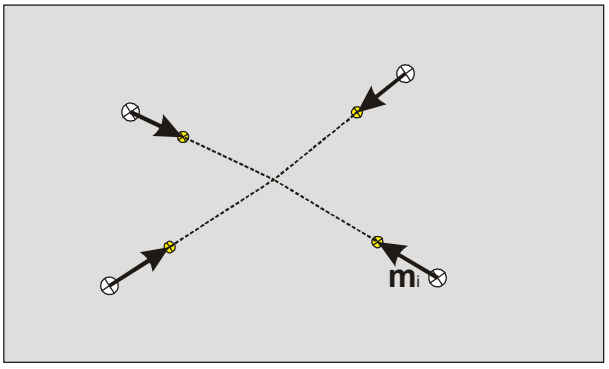
Obr. 15 Pole pohybových vektorů při rotačním pohybu kolem optické osy kamery

je pohyb reprezentovaný vektory $\boldsymbol{m}_i$ rotační kolem optické osy kamery. Velikost pohybu odpovídá v tomto případě úhlu rotace α, který lze zjistit z jakékoliv korespondující dvojice středů významných bodů (X_i, Y_j), respektive z vektorů, které tvoří tyto body a počátek $\boldsymbol{O} = [0,0]^T$. Úhel pak lze snadno zjistit pomocí skalárního součinu jako

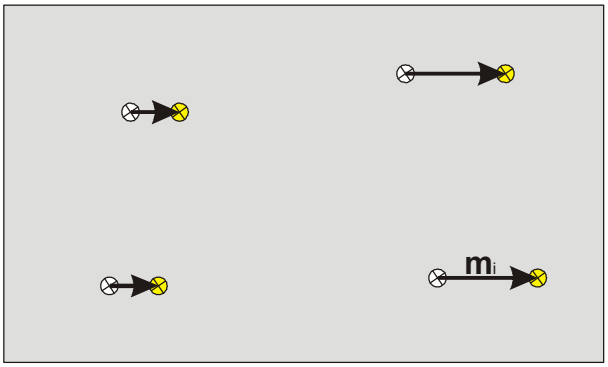
$$\alpha = \cos^{-1}\left[\left(\frac{X_i - \boldsymbol{O}}{\|X_i - \boldsymbol{O}\|}\right)^T \left(\frac{Y_i - \boldsymbol{O}}{\|Y_i - \boldsymbol{O}\|}\right)\right]$$

Pomocí geometrické analýzy pohybových vektorů by šly dobře detekovat také další pohyby jako:

- translační pohyb podél optické osy kamery, viz Obr. 16
  - vektory se protínají v jednom bodě
- rotace kolem os kolmých na optickou osu kamery, viz Obr. 17
  - vektory mají stejný směr, ale různou velikost

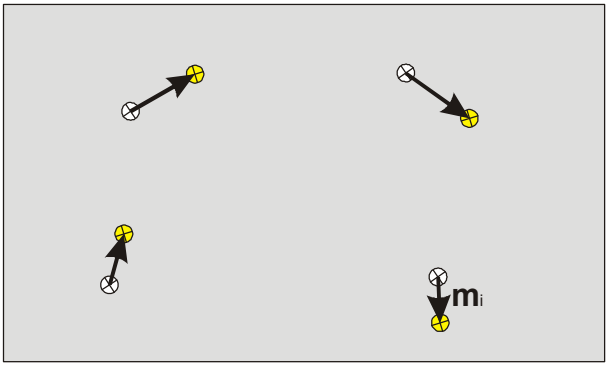


Obr. 16 Pole pohybových vektorů při translačním pohybu podél optické osy kamery



Obr. 17 Pole pohybových vektorů při rotačním pohybu kolem osy kolmé na optickou osu kamery

Pomocí simulací, které jsou zmíněné níže, byla ověřena velmi dobrá stabilita zejména pro translační pohyb v rovině kolmé na optickou osu a rotaci kolem této osy. Přesto má tento přístup jeden zásadní nedostatek, který znemožňuje jeho použití v systému. V obecném případě je totiž pohyb kamery téměř vždy složen z několika různých pohybů v rámci jednoho snímku. Jak ukazuje Obr. 18, je v případě složeného pohybu obtížné detekovat jednotlivé složky a nemožné získat jejich velikost. V případě Obr. 18 by podle výše uvedených podmínek byl pohyb zřejmě detekován jako rotační, výsledný úhel by byl ale naprosto nekorektní a rekonstruovaný pohyb by neodpovídal skutečnému pohybu kamery.



Obr. 18 Pole pohybových vektorů při pohybu složeném z rotace kolem optické osy kamery a translačního pohybu v rovině kolmé na tuto osu

REKONSTRUKCE POMOCÍ LINEÁRNÍHO SYSTÉMU

Po zamítnutí geometrického přístupu byl problém zkoumán z více numerického pohledu. Nejprve bylo nutné přesně specifikovat problém, tedy uvědomit si jaký má být výstup a jaké mám k jeho nalezení vstupy. Jak bylo uvedeno v úvodu této sekce, jsou vstupy dvojice korespondujících bodů $(X_i, Y_i)_i$. Hledám transformační matici 4×4 T_{cam}, která pak popisuje transformaci mezi kamerou prvního a druhého snímku. Při hledání této matice jsem vycházel nejprve ze situace znázorněné na Obr. 19. Kamera po celou dobu směřuje kolmo vzhůru a může vykonávat pouze translační pohyb v rovině kolmé na optickou osu a rotaci kolem této osy. Díky těmto omezením pak bude mít matice T_{cam} tvar

$$T_{cam} = \begin{bmatrix} r_{11} & r_{12} & 0 & \bar{t_x} \\ r_{21} & r_{22} & 0 & \bar{t_y} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Pomocí matematického modelu kamery z podkapitoly 3.2 lze z T_{cam} snadno odvodit, že transformační matice mezi jednotlivými korespondencemi je

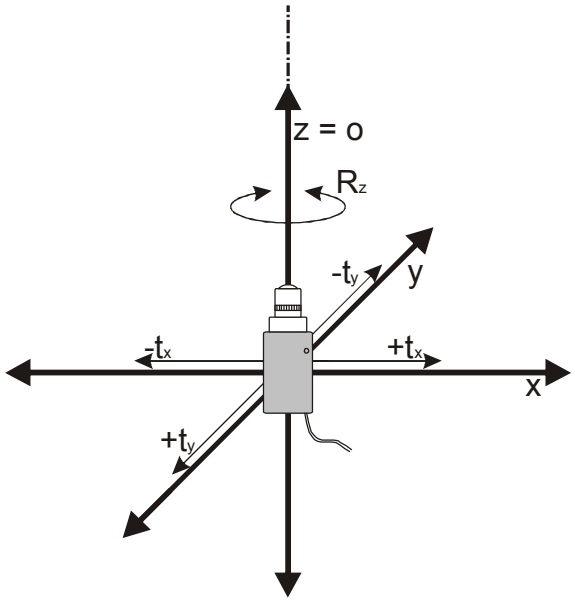
$$T_{cor} = \begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

kde

$$t_x = \frac{1}{\lambda}\bar{t_x}$$

$$t_y = \frac{1}{\lambda}\bar{t_y}$$

a λ je neznámý skalár, který odpovídá vzdálenosti kamery od promítaných bodů. Pokud tedy naleznu transformaci mezi jednotlivými korespondencemi, získám tak zároveň i transformaci mezi kamerami, až na neznámý skalární faktor. Na základě těchto předpokladů pak vzniklo jednoduché řešení, využívající k nalezení matice T_{cor}, respektive jejích prvků, lineární soustavy rovnic.



Obr. 19 Orientace souřadného systému kamery a její stupně volnosti pro rekonstrukci pomocí systému lineárních rovnic. *R* označuje rotaci a *t* translaci, indexy pak označují osy, kterých se transformace týká.

Jak bylo uvedeno v kapitole 3, bude bod $A = [x, y, 1]^T$ transformován pomocí matice T_{cor} do bodu $A' = [x', y', 1]^T$ jako

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

jednotlivé složky bodu A' pak budou mít tvar

$$x' = r_{11}\, x + r_{12} y + t_x$$

$$y' = r_{21}\, x + r_{22} y + t_y$$

Díky dvojicím korespondencí znám bod A, který odpovídá bodu $X_i = [x_i^1, x_i^2, 1]$, i bod A', který odpovídá bodu $Y_i = [y_i^1, y_i^2, 1]$. Mohu tedy sestavit lineární systém rovnic, kde vektor pravé strany budou tvořit složky bodů Y_i a matici soustavy budou tvořit souřadnice bodů X_i. Pro celkový počet korespondencí n bude mít systém tedy tvar

$$
\begin{aligned}
r_{11}\, x_1^1 + r_{12} x_1^2 + t_x &= y_1^1 \\
r_{21}\, x_1^1 + r_{22} x_1^2 + t_y &= y_1^2 \\
r_{11}\, x_2^1 + r_{12} x_2^2 + t_x &= y_2^1 \\
r_{21}\, x_2^1 + r_{22} x_2^2 + t_y &= y_2^2 \\
&\vdots \\
r_{11}\, x_n^1 + r_{12} x_n^2 + t_x &= y_m^1 \\
r_{21}\, x_n^1 + r_{22} x_n^2 + t_y &= y_m^2
\end{aligned}
$$

Pro formální zápis lineárního systému

$$A\boldsymbol{x} = \boldsymbol{b}$$

budou matice systému

$$
A = \begin{bmatrix}
x_1^1 & x_1^2 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & x_1^1 & x_1^2 & 1 \\
x_2^1 & x_2^2 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & x_2^1 & x_2^2 & 1 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
x_n^1 & x_n^2 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & x_n^1 & x_n^2 & 1
\end{bmatrix}
$$

vektor neznámých

$$\boldsymbol{x} = \left[r_{11}, r_{12}, t_x, r_{21}, r_{22}, t_y\right]^T$$

a vektor pravých stran

$$\boldsymbol{b} = [y_1^1, y_1^2, y_2^1, y_2^2, \dots, y_n^1, y_n^2]^T$$

Řešitelnost tohoto systému závisí na počtu korespondencí n:

- Pro $n < 3$ je systém nedourčený a má nekonečně mnoho řešení.
- Pro $n = 3$ má systém jedno unikátní a přesné řešení za předpokladu, že matice A není singulární. V takovém případě lze k řešení systému použít klasické metody, jako je Gaussova eliminace nebo vynásobení celého systému maticí A^{-1}. Matice A je singulární právě tehdy, když jsou body X_1, X_2 a X_3 kolineární, protože sloupce A pak nebudou lineárně nezávislé. Aby byly první tři sloupce

$$
A = \begin{bmatrix}
x_1^1 & x_1^2 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & x_1^1 & x_1^2 & 1 \\
x_2^1 & x_2^2 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & x_2^1 & x_2^2 & 1 \\
x_3^1 & x_3^2 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & x_3^1 & x_3^2 & 1
\end{bmatrix}
$$

lineárně závislé, musí platit, že

$$
\begin{aligned}
a[x_1^1, 0, x_2^1, 0, x_3^1, 0]^T + b[x_1^2, 0, x_2^2, 0, x_3^2, 0]^T + c[1,0,1,0,1,0]^T \\
= [0,0,0,0,0,0]
\end{aligned}
$$

což lze upravit jako

$$
\begin{aligned}
a x_1^1 + b x_1^2 + c &= 0 \\
a x_2^1 + b x_2^2 + c &= 0 \\
a x_3^1 + b x_3^2 + c &= 0
\end{aligned}
$$

Je zřejmé, že toto je splněné přesně v případě, kdy leží body X_1, X_2 a X_3 na jedné přímce.

- Pro $n > 3$ je systém přeurčený, v takovém případě neexistuje unikátní přesné řešení. Tento systém je však možné řešit metodou nejmenších čtverců, která pro systém $A\boldsymbol{x} = \boldsymbol{b}$ naleznene takové řešení $\boldsymbol{x}$, které minimalizuje

$$\min_{\boldsymbol{x}} \|A\boldsymbol{x} - \boldsymbol{b}\|$$

Aby bylo možné nalézt řešení, je tedy nutné, aby $n \geq 3$. Samotná řešitelnost systému však nezajistí správné řešení z hlediska výsledného pohybu. Při skutečném běhu systému je velmi pravděpodobné, že data korespondencí budou zatížena nějakou chybou, způsobenou v některé z předchozích fází. Za předpokladu, že pouze některé korespondence jsou touto chybou zatíženy, bude řešení systému s $n = 3$ touto chybou ovlivněno poměrně významně. Při řešení s $n > 3$ je sice nalezeno pomocí metody čtverců řešení přibližné, je zde však vyšší šance na přesnější výsledek. Díky vlastnostem této metody, která hledá takové řešení $\boldsymbol{x}$, které minimalizuje vzdálenost $A\boldsymbol{x} - \boldsymbol{b}$, je potlačen vliv chybných korespondencí. Obecně lze tedy tvrdit, že čím vyšší bude n, tím správnější bude řešení.

Toto řešení lze ještě dále zjednodušit a zvýšit přesnost výsledku odstraněním redundantní informace. Transformační matice T_{ob}, z které byl sestaven vektor neznámých $\boldsymbol{x}$ obsahuje čtyři koeficienty pro rotaci $r_{11}, r_{12}, r_{21}, r_{22}$. Pokud ale prozkoumám tvar rotační matice

$$R = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

je zřejmé, že

$$r_{11} = r_{22}$$
$$-r_{12} = r_{21}$$

Pokud tedy označím

$$r_x = r_{11} = r_{22} = \cos\alpha$$
$$r_y = -r_{12} = r_{22} = \sin\alpha$$

zjednoduší se vektor neznámých $\boldsymbol{x}$ na

$$\boldsymbol{x} = \left[r_x, r_y, t_x, t_y\right]^T$$

Matice systému A pak bude mít tvar

$$A = \begin{bmatrix} x_1^1 & -x_1^2 & 1 & 0 \\ x_1^2 & x_1^1 & 0 & 1 \\ x_2^1 & -x_2^2 & 1 & 0 \\ x_2^2 & x_2^1 & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_n^1 & -x_n^2 & 1 & 0 \\ x_n^2 & x_n^1 & 0 & 1 \end{bmatrix}$$

Řešitelnost tohoto systému opět závisí na počtu korespondencí n:

- Pro $n < 2$ je systém nedourčený a má nekonečně mnoho řešení
- Pro $n = 2$ má systém jedno unikátní a přesné řešení. Matice A by v tomto případě byla singulární pouze tehdy, pokud by bylo

$$X_1 = X_2$$

- Pro $n > 3$ je systém přeurčený a lze ho opět řešit metodou nejmenších čtverců.

Díky pouze dvěma rotačním koeficientům je zajištěno, že i v případě chyby, bude výsledná transformační matice obsahovat rotaci a ne jiné transformace jako zkosení nebo změnu měřítka, které by vznikly jinými hodnotami u r_{11}, r_{12}, r_{21} a r_{22}.

U obou předchozích metod je vhodné diskutovat ještě jednu věc, a tou je jejich numerická stabilita, která u řešení lineárních systémů přímo souvisí s podmíněností matice A. Jak ukazuje Hartley[12], je v případě podobných výpočtů matice A velmi špatně podmíněná. Hartley[12] sice diskutuje tento problém ve spojitosti s hledáním fundamentální matice z korespondencí, řešený lineární systém je však velmi podobný mému. V zásadě jde o to, že souřadnice typického bodu X_i ve snímku o rozměru 200×200 budou $[100,100,1]$ a budou se tedy lišit o řády. Tato nehomogenita se pak přenese i do matice A a způsobí tak její špatnou podmíněnost a celkovou nestabilitu výpočtu. Poté se malá chyba na vstupu, které se zřejmě nikdy nejde vyhnout vzhledem k nedokonalostem způsobených snímacím zařízením a špatnou segmentací, projeví jako velká chyba v rekonstruovaném pohybu. Velikost této chyby je však velmi subjektivním měřítkem, protože to, co se může z numerického pohledu jevit jako velká chyba, nemusí uživatel vůbec postřehnout. Přesto je vhodné snažit se udělat výpočet co nejstabilnější.

Hartley[12] navrhuje tři jednoduché transformace korespondujících bodů, které výrazně zlepší podmíněnost matice A. Těmito transformacemi jsou:

1. Translace bodů tak, aby počátek souřadného systému byl v těžišti bodů.
2. Translace bodů podle bodu 1 a isotropní změna měřítka[8] tak, aby každý z bodů měl přibližně stejně velké všechny složky. Toho je docíleno takovou změnou měřítka, která zajistí, že průměrná vzdálenost korespondujícího bodu od počátku bude rovna $\sqrt{2}$.
3. Translace bodů podle bodu 1 a anisotropní změna měřítka[9], která má v podstatě stejný efekt jako předchozí úprava, hodí se však v případě, kdy se výrazně liší rozptyl u jednotlivých složek bodů.

Kromě výjimečných situací, kdy kamera snímá okraj pole významných bodů, jsou významné body typicky rovnoměrně rozptýlené v rámci snímku. Obecně lze tedy za jejich těžiště považovat střed snímku. Počátek souřadného systému snímku je v mém případě v levém horním rohu, transformační matice T, která přesune počátek do středu snímku o šířce w a výšce h, bude mít tedy tvar

$$T = \begin{bmatrix} 1 & 0 & -w \\ 0 & 1 & -h \\ 0 & 0 & 1 \end{bmatrix}$$

Jako faktor změny měřítka jsem použil převrácenou hodnotu šířky, respektive výšky snímku, což zaručí, že významné body budou tvořit shluk ve tvaru kruhu okolo počátku pro čtvercové i obdélníkové snímky. Transformační matice S, která toto zajistí, má tvar

$$S = \begin{bmatrix} \frac{1}{w} & 0 & 0 \\ 0 & \frac{1}{h} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Výsledná transformační matice M vznikne složením matice S a T a bude mít tvar

[8] Změna měřítka stejná pro všechny složky
[9] Změna měřítka odlišná pro každou složku

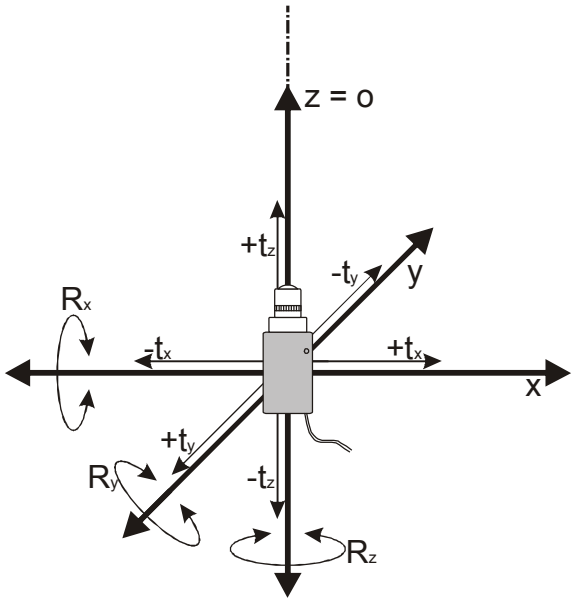
$$M = S \circ T = \begin{bmatrix} \frac{1}{w} & 0 & -1 \\ 0 & \frac{1}{h} & -1 \\ 0 & 0 & 1 \end{bmatrix}$$

Jak prokázali simulace, provedením této transformace se výrazně zlepšila stabilita a také přesnost rekonstrukce.

Výhodou rekonstrukce pohybu pomocí systému lineární rovnic je především jeho jednoduchost a dobrá stabilita, která se projevila při simulacích i reálném testování. Nevýhodou tohoto přístupu je malý počet stupňů volnosti, protože pohyb kamery byl, jak ukazoval Obr. 19, omezen pouze na rotaci kolem optické osy a translační pohyb v rovině kolmé na tuto osu. Proto byl v práci zkoumán následující přístup.

REKONSTRUKCE POHYBU POMOCÍ NALEZENÍ A ROZKLADU HOMOGRAFIE S POUŽITÍM ČTYŘBODOVÉHO ALGORITMU

Vzhledem k tomu, že jsou významné body přichycené ke stropu, sdílí jednu zásadní vlastnost a to, že jsou koplanární. K řešení rekonstrukce pohybu tedy mohlo být využito vlastností planární homografie, která byla popsána v 3.3.1 a vyjadřuje transformaci mezi dvěma snímky bodů, které jsou v jedné rovině. Homografie popisuje libovolnou transformaci mezi snímky, tedy libovolný pohyb, který musela kamera mezi snímky vykonat. Díky tomu je možné rekonstruovat pohyb kamery s šesti stupni volnosti, což odpovídá počtu stupňů volnosti lidské hlavy. Jednotlivé stupně volnosti jsou zobrazené na Obr. 20.



Obr. 20 Orientace souřadného systému kamery a její stupně volnosti pro rekonstrukci pomocí nalezení a rozkladu homografie. *R* označuje rotaci a *t* translaci, indexy pak označují osy, kterých se transformace týká.

Pro nalezení planární homografie je použito čtyřbodového algoritmu popsaného v 3.3.2. Čtyřbodový algoritmus, založený na Direct Linear Transform[13] jak jsem ho zkráceně popsal v 3.3.2. vyžaduje, aby matice χ měla hodnost 8. K tomu stačí přesně čtyři korespondující body, z nichž žádné tři nejsou kolineární. Obvykle je však k dispozici více korespondencí než čtyři a je žádoucí je využít a udělat tak celý výpočet robustnější. Pokud budou korespondence absolutně přesné, bude mít matice χ stále hodnost 8 a bude existovat jedno přesné homogenní řešení. Jak už bylo uvedeno dříve, je však téměř jisté, že korespondence budou zatížené určitou chybou, hodnost matice χ bude vyšší a nebude

existovat přesné řešení. Je tedy nutné řešit minimalizační problém obdobně jako v předchozí podsekci.

Ma[6] navrhuje použít standardní lineární metody nejmenších čtverců pro minimalizaci normy

$$\|\chi H_L^S\|$$

Jak uvádí Dubrofsky[14], je však tento postup nevhodný, protože minimalizuje algebraickou vzdálenost, která je z geometrického hlediska nesmyslná. Dubrofsky[14] pak uvádí několik dalších možných minimalizačních schémat, které lépe odpovídají řešenému problému a přináší tak lepší výsledky.

Ve své základní podobě je čtyřbodový algoritmus také poměrně nestabilní, protože, jak ukazuje Hartley[12], je závislý na počátku souřadného systému korespondencí a měřítku. Tato vlastnost vyplývá z použití SVD pro nalezení řešení přeurčeného systému

$$\chi H_L^S = 0$$

což je přesně popsáno v [13]. Proto je opět vhodné použít normalizaci, která byla popsána v předchozí sekci, tedy:

1. Translace bodů tak, aby počátek souřadného systému byl v těžišti bodů.
2. Translace bodů podle bodu 1 a isotropní změna měřítka tak, aby každý z bodů měl přibližně stejně velké všechny složky. Toho je docíleno takovou změnou měřítka, která zajistí, že průměrná vzdálenost korespondujícího bodu od počátku bude rovna $\sqrt{2}$.
3. Translace bodů podle bodu 1 a anisotropní změna měřítka, která má v podstatě stejný efekt jako předchozí úprava, hodí se však v případě, kdy se výrazně liší rozptyl u jednotlivých složek bodů.

Po normalizaci a použití vhodného minimalizačního schématu se tedy značně sníží odchylka od správného výsledku i pro zašuměná data. Pokud budou ale data obsahovat například špatně určené korespondence, výsledek téměř jistě korektní nebude. Metoda nejmenších čtverců je totiž velmi citlivá na takzvané *outliers*[10], což ukazuje na velmi jednoduchém příkladu proložení bodů přímkou Zulliani[15]. Také ukazuje, že už pro jeden *outlier,* v mém případě tedy jednu špatně určenou korespondenci, metoda nejmenších čtverců selhává a může poskytnout silně nekorektní výsledek.

Pro rozklad homografie na jednotlivé složky je použito postupu popsaného v 3.3.3. Rozklad homografie má čtyři různá řešení, z nichž dvě lze eliminovat omezením pozitivní hloubky. Jak jsem uvedl, je pro vybrání jediného unikátního řešení rozkladu nutná nějaká apriorní znalost o pohybu kamery nebo rovině Π. Vzhledem k tomu, že rovinu Π tvoří strop, bude normálový vektor $\boldsymbol{n}_\Pi$ této roviny ve tvaru

$$\boldsymbol{n}_\Pi = [0,0,1]$$

Jedno z řešení je pak vybráno na základě skalárního součinu normálového vektoru $\boldsymbol{n}_i$ příslušejícího i-tému řešení a vektoru $\boldsymbol{n}_\Pi$. Pokud platí

$$\left(\frac{\boldsymbol{n}_i}{\|\boldsymbol{n}_i\|}\right)^T \boldsymbol{n}_\Pi = 1$$

je i-té řešení prohlášeno za hledaný rozklad homografie. Výsledkem rozkladu je tedy 3×3 rotační matice kamery R a 3×1 vektor posunutí $\boldsymbol{t}$, které popisují pohyb kamery mezi snímky.

[10] V literatuře nelze najít přesnou definici, obecně jde ale o data, která se nějakým způsobem vymykají modelu, který mají správná data – *inliers*

Rekonstrukce pohybu pomocí nalezení homografie čtyřbodovým algoritmem přidává tři stupně volnosti a je tedy univerzálnější pro použití než předchozí rekonstrukce pomocí lineárního systému. Není však příliš robustní, protože pokud se v datech vyskytnou *oultiers,* výpočet založený na metodě nejmenších čtverců selže. Přestože jsou všechny předchozí fáze nastaveny tak, aby byla šance špatně určené korespondence minimalizována, je v zájmu robustnosti celého systému vhodné nalézt řešení, které si dokáže poradit i s daty, ve kterých jsou přítomné *outliers*.

REKONSTRUKCE POHYBU POMOCÍ NALEZENÍ A ROZKLADU HOMOGRAFIE S RANSAC OPTIMALIZACÍ

Jak bylo ukázáno v předchozí podsekci, je homografie vhodná pro rekonstrukci pohybu kamery díky stupňům volnosti, které umožňuje. Výpočet pomocí čtyřbodového algoritmu ale není vhodný z důvodu malé robustnosti. Metoda RANSAC[11] je zřejmě nejrozšířenější metodou pro výpočet homografie z dat zatíženými *outliers*. V zásadě se tento algoritmus skládá z dvou hlavních kroků, které jsou iterativně opakovány[15]:

- Nejprve je vybrána minimální množina vzorků (MMV), tak, že jsou ze vstupní množiny korespondencí náhodně vybrány čtyři, což je minimální počet, pro který lze homografii vypočítat[5]. Následně je homografie vypočítána pouze na základě vzorků z MMV. Tím se tento přístup liší od metody minimálních čtverců, která k výpočtu používá celou množinu.
- V druhém kroku jsou všechny zbylé korespondence, tedy ty, jež nebyly součástí MMV, porovnány, zda jsou konzistentní s vypočítanou homografií a na základě tohoto klasifikovány jako *inliers* nebo *outliers*. Korespondence vyhodnocené jako *inliers* pak vytvoří konsenzní množinu (KM).

Iterativní proces skončí, pokud klesne pravděpodobnost nalezení větší KM pod určitý přednastavený práh. Iterace, která obsahovala největší KM, je pak zvolena a homografie je pak přepočítána na základě všech korespondencí obsažených v KM. Jak uvádí Zulliani[15] je metoda RANSAC schopna nalézt korektní výsledek i v datech obsahujících více než 50% *outliers*.

Rozdělení na *inliers* a *outliers* je klíčovým pro správné fungování algoritmu. Jak uvádí Dubrofsky[14], pro rozdělení je nutné najít práh vzdálenosti, který pro původní bod a bod vytvořený reprojekcí pomocí vypočítané homografie určí s pravděpodobností α, zda je bod *inlier*. Úplné odvození prahu poskytuje například [12]. V implementaci, kterou jsem ve své práci použil[29], je rozdělení na *inliers* a *outliers* řešeno následovně. Pokud je pro korespondenci $(X_i, Y_i)_i$, vypočítanou homografii H a práh thr

$$\|Y_i - HX_i\| > thr$$

je korespondence i považována za *outlier*. Práh thr je nutné nastavit manuálně, pro korespondence měřené v pixelech má podle [29] smysl nastavit thr v intervalu $\langle 1{,}10 \rangle$.

Poté, co je výše popsaným postupem homografie nalezena, je rozložena na jednotlivé složky stejně, jak bylo uvedeno v předchozí podsekci. Nalezení homografie pomocí metody RANSAC má možnou nevýhodu ve větší výpočetní náročnosti, vzhledem k tomu, že jde o iterativní numerickou metodu. Vyšší výpočetní náročnost je však vyvážena velkou robustností výpočtu, díky čemuž byla nakonec v mé práci použita právě tato metoda.

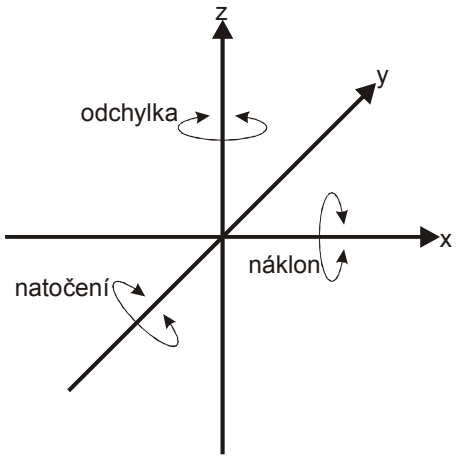
[11] Random Sample Consensus

4.2.3. Reprezentace pohybu pro zobrazovací část

Výstupem části pro rekonstrukci je tedy rotační matice R a vektor translace T. V ideálním případě přesné rekonstrukce by tyto dva objekty byly dostatečné pro reprezentaci pohybu. Pokud je však rekonstruovaný pohyb zatížen nějakou chybou, což je vysoce pravděpodobné, je vhodné hodnoty pohybu testovat, zda chyba není příliš velká a nenaruší tak věrnost virtuální reality. Také může nastat situace, kdy jsou souřadné systémy rekonstrukce a zobrazení různé, je tedy nutné například provést záměnu hodnot pro jednotlivé souřadné osy. Translační vektor obojí umožňuje rovnou, protože díky jeho tvaru

$$T = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

obsahuje hodnoty pro každou osu zvlášť. Bylo by vhodné získat podobnou reprezentaci i pro rotaci. Takovou reprezentaci poskytují například Eulerovy úhly, což jsou tři úhly popisující rotaci kolem jednotlivých os souřadného systému. Ve své práci jsem je označil jako odchylku, náklon a natočení[12] a jejich přiřazení jednotlivým rotacím lze vidět na Obr. 21.



Obr. 21Přiřazení úhlů popisujících rotaci k jednotlivým osám

Z rotační matice však tyto úhly nelze získat přímo, protože jednak je rotační matice R složena z elementárních rotačních matic R_x, R_y, R_z popisujících rotaci kolem jednotlivých os a i tyto matice obsahují úhel nepřímo jako hodnotu kosinu a sinu. Matici R je tedy nutné rozložit, pro rozklad jsem využil algoritmu popsaného v [16]. Jak jsem uvedl, je matice R složená z jednotlivých matic R_x, R_y a R_z. Pokud označím náklon jako θ, natočení ψ a odchylku ϕ, mají tyto matice následující tvar

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix}$$

$$R_y = \begin{bmatrix} \cos\psi & 0 & \sin\psi \\ 0 & 1 & 0 \\ -\sin\psi & 0 & \cos\psi \end{bmatrix}$$

$$R_z = \begin{bmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

složená matice

[12] volný překlad z anglického yaw, pitch, roll

$$R = R_z \circ R_y \circ R_x$$

má pak tedy tvar

$$R = \begin{bmatrix} \cos\psi\cos\phi & -\cos\theta\sin\phi + \sin\theta\sin\psi\cos\phi & \sin\theta\sin\phi + \cos\theta\sin\psi\cos\phi \\ \cos\psi\sin\phi & \cos\theta\sin\phi + \sin\theta\sin\psi\sin\phi & -\sin\theta\cos\phi + \cos\theta\sin\psi\cos\phi \\ -\sin\psi & \sin\theta\cos\psi & \cos\theta\cos\psi \end{bmatrix}$$

Úhel natočení lze získat snadno z třetího řádku a prvního sloupce matice R jako

$$\psi = \sin^{-1}(-R_{3,1})$$

Z třetího řádku a druhého a třetího sloupce získám dvě rovnosti

$$R_{3,2} = \sin\theta\cos\psi$$

$$R_{3,3} = \cos\theta\cos\psi$$

pokud je vydělím, vykrátí se $\cos\psi$ a získám

$$\frac{R_{3,2}}{R_{3,3}} = \frac{\sin\theta}{\cos\theta}$$

což je po zjednodušení

$$\frac{R_{3,2}}{R_{3,3}} = \tan\theta$$

Nyní bych mohl úhel θ získat jako

$$\theta = \tan^{-1}\left(\frac{R_{3,2}}{R_{3,3}}\right)$$

pokud by však byla hodnota $R_{3,3}$ blízká nule, při výpočtu na počítači, který má omezenou přesnost, by mohlo dojít k chybě výpočtu. Podílu se však lze vyhnout použitím funkce $\mathrm{atan2}$, úhel θ pak tedy získám jako

$$\theta = \mathrm{atan2}(R_{3,2}, R_{3,3})$$

Úhel ϕ pak lze získat stejným způsobem, pouze za použití prvního a druhého řádku v prvním sloupci, takže

$$\phi = \mathrm{atan2}(R_{1,1}, R_{2,1})$$

Informace o pohybu jsou pak z modulu rekonstrukce předány do zobrazovacího modulu jako:

- vektor translace $T = [t_x, t_y, t_z]$
- odchylka – ϕ, rotace kolem osy z, odpovídá otáčení hlavy doleva a doprava
- náklon – θ, rotace kolem osy x, odpovídá kývání hlavou dopředu a dozadu
- natočení – ψ, rotace kolem osy y, odpovídá kývání hlavou doleva a doprava

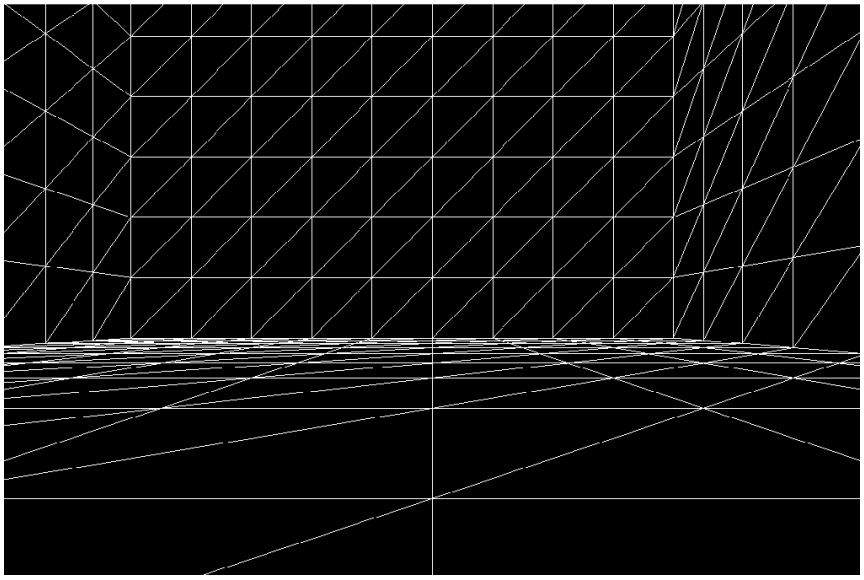
Tyto hodnoty mohou být nyní snadno testovány, zda jsou validní pro zobrazení. V systému jsou tedy nastavené maximální povolené hodnoty, které vychází z fyzických dispozic uživatele a omezení systému popsaných zejména v 8.1.1. Pokud rekonstruované hodnoty tato maxima překročí, nejsou tyto hodnoty při zobrazení použity, protože by narušily celkový dojem z virtuální reality.

Díky tomuto rozkladu mohlo být do systému zavedeno také uzamykání částí rekonstruovaného pohybu. V základním nastavení je pro zobrazení využit kompletní výstup rekonstrukce, další možná nastavení jsou pouze translace a odchylka nebo pouze translace. Tato omezení lze přizpůsobit potřebám konkrétní aplikace.

4.3. ZOBRAZENÍ REKONSTRUOVANÉHO POHYBU

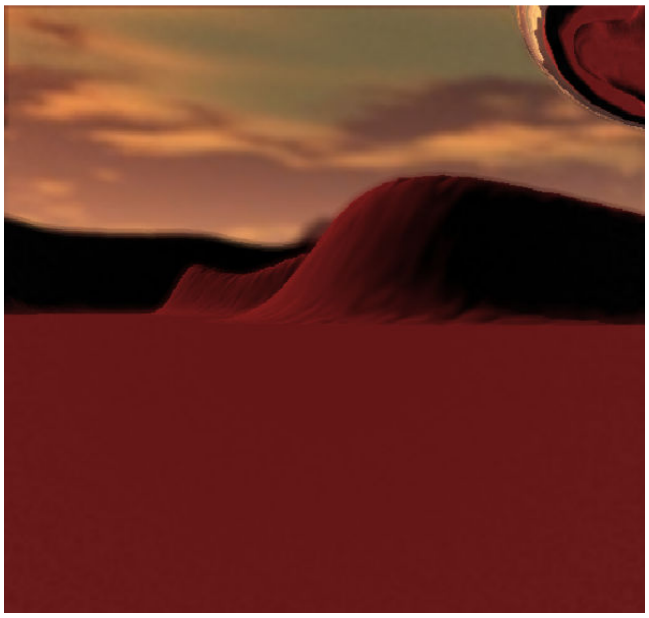
Doposud popsané části jsou nutné pro rekonstrukci pohybu, je to však část zobrazení, která tento proces prezentuje uživateli. Jak už bylo zmíněno v Úvodu, má zobrazení zásadní vliv na míru vtáhnutí uživatele do VR. Kromě kvality samotného zobrazení je to pak především virtuální scéna, její uspořádání a charakter, které se podílí na dojmu uživatele. K návrhu virtuální scény lze použít mnoho různých nástrojů, této problematiky se lehce dotýkám v podkapitole 9.3 předposlední kapitoly. Pro zobrazení bylo použito HMD, jehož technické detaily a popis lze najít v sekci 8.2.2.

V rámci systému byly vytvořeny dva odlišné světy, které umožnily otestovat vlastnosti systému v různých podmínkách. Prvním z nich je simulační prostředí, které je tvořené pouze čtyřmi zdmi a podlahou, tvořenými pouze bílými trojúhelníky, viz Obr. 22. V tomto na prvním pohled strohém prostředí však velmi dobře vyniknou všechny nepřesnosti rekonstrukce pohybu a je tedy vhodné pro hodnocení celkové přesnosti systému. Také lze očekávat, že dojem z rekonstrukce bude podobný i ve scénách zobrazujících uzavřené interiéry.



Obr. 22 Ukázka simulačního prostředí

Druhým světem je pouštní planeta s velkou zrcadlovou koulí uprostřed a malou planetkou obíhající kolem, ukázka na Obr. 23. Toto prostředí naopak reprezentuje exteriér a obsahuje několik pokročilejších grafických efektů, které ho činí vizuálně mnohem atraktivnější než první prostředí.



Obr. 23Ukázka prostředí pouštní krajiny

Samotný proces zobrazení je opět čistě implementační záležitost, věnuji se mu proto v kapitole 6, podkapitola 6.3.

5. SIMULACE

Před samotnou implementací navrženého systému včetně reálného hardwarového vybavení byly provedeny simulace, které měly za úkol ověřit vlastnosti navržených metod a následně posloužit k ladění implementace. Simulace měly v zásadě tři různé podoby:

- Simulace pomocí podpůrných aplikací
- Numerická
- Softwarová

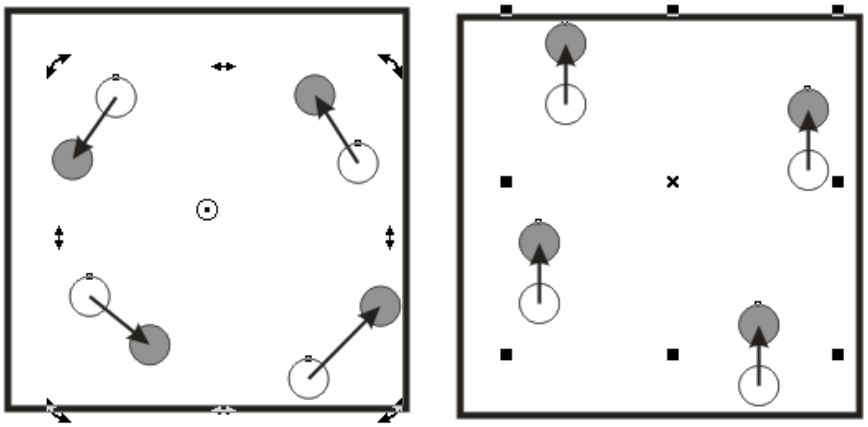
Jednotlivé typy simulací budou nyní v této kapitole popsány včetně některých jejich výstupů.

5.1. SIMULACE POMOCÍ PODPŮRNÝCH APLIKACÍ

Tato simulace stála na samém počátku vývoje systému a měla za úkol ukázat vztahy a fungování při pohybu kamery, která snímá síť významných bodů. K realizaci této simulace bylo využito několika samostatných aplikací jako CorelDraw nebo CVIP Tools, pomocí nichž byly vytvořeny testovací snímky obsahující simulované průměty významných bodů a následně byly zkoumány vztahy mezi transformovanými verzemi těchto snímků. Později byly podpůrné aplikace používány k vytváření testovacích množin, které sloužili jako ladící vstup systému, případně byly použity pro analýzu výstupů systému.

K tomuto účelu nejlépe posloužil CorelDraw, který díky své vektorové reprezentaci zobrazovaných objektů umožňuje provádět přesné geometrické transformace. V tomto programu byl tedy vytvořen čtverec reprezentující snímek. Následně byly do snímku náhodně umístěny čtyři plné kruhy, reprezentující průměty významných bodů. Tyto kruhy pak byly pomocí geometrické transformace duplikovány vzhledem ke středu simulovaného

snímku, čímž byl simulován pohyb kamery. Vzniklé transformované duplikáty byly obarveny a korespondující středy původních a duplikovaných kruhů byly propojeny čarami. Tím vznikly motion vektory tvořící pohybové pole, definice viz 4.2. Orientace a velikost jednotlivých motion vektorů pak byly zkoumány pod různými typy transformací. Ukázka této simulace je na Obr.24.



Obr.24 Simulace pohybu kamery, respektive průmětů významných bodů v rámci snímků. Vlevo je rotace o 30°, vpravo translace o 100 pixelů v záporném směru osy y. Na snímcích lze vidět ovládací prvky transformací programu CorelDraw.

Na základě této simulace pak vznikla především metoda pro rekonstrukci pohybu popsaná v sekci 4.2.2 s označením Geometrické řešení. Pomocí této simulace byly také částečně odvozovány vztahy pro určení korespondencí popsané v 4.2.1 a možná omezení tohoto přístupu, popsaná v 8.1.1.

5.2. NUMERICKÁ SIMULACE

Dalším typem simulace, která byla při vývoji použita, byla numerická simulace. Pomocí této simulace byly ověřovány zejména platnost a vlastnosti matematických modelů použitých v systému, jakými jsou matematický model kamery popsaný v 0, homografie popsané v 3.3 a různých přístupů k rekonstrukci pohybu popsaných 4.2.2, zejména rekonstrukce označené jako Rekonstrukce pohybu pomocí nalezení a rozkladu homografie s použitím čtyřbodového algoritmu. Kromě ověření vlastností výše zmíněných modelů sloužila tato simulace v průběhu vývoje také k verifikaci výstupů ze softwarové implementace systému.

K realizaci numerických simulací bylo využito výpočetního a modelovacího prostředí Matlab[30]. Pro toto prostředí bylo vytvořeno několik krátkých skriptů a funkcí, umožňujících výpočet a použití výše zmíněných modelů. Poté byly vždy tyto funkce aplikovány na sadu testovacích dat a výsledky zkoumány, případně porovnávány s výsledky softwarové implementace pro stejná testovací data.

Hlavní funkcí je

[H,R,T,N] dp_homography(X,Y)

jejíž implementace je v souboru *dp_homography.m*. Ta pro matice X a Y, které obsahují korespondující průměty významných bodů v prvním (X) a druhém (Y) snímku jako řádkové vektory, vypočítá homografii pomocí čtyřbodového algoritmu a provede její rozklad na jednotlivé složky[13]. Pro ověření korektního chování pak byly jako vstup této funkce použity korespondující body vázány jednoduchou známou transformací, například translací v jednom směru nebo rotací o snadno interpretovatelný úhel.

[13] Čtyřbodový algoritmus je popsán v 3.3.2, rozklad homografie je popsán v 3.3.3

K této funkci později přibyl skript *dp_cameraProjection.m*, který obsahuje několik předdefinovaných matic vnějších parametrů kamery a pro významné body v souřadném systému světa provede jejich promítnutí podle pinhole modelu kamery popsaném v 0. Pomocí tohoto skriptu pak mohly být zkoumány vlivy na průměty významných bodů při různých pohybech kamery, jako je například rotace kolem jedné z os kolmých na optickou osu kamery. Spojením tohoto skriptu s funkcí *dp_homography* pak byla také potvrzena schopnost rekonstruovat tyto pohyby pomocí homografie.

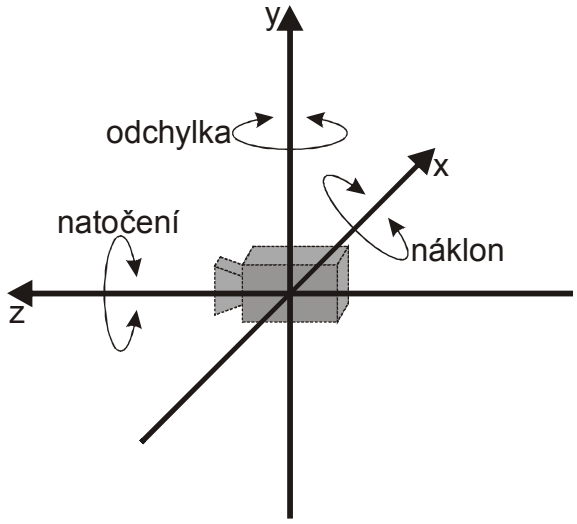
5.3. SOFTWAROVÁ SIMULACE

Numerická simulace hrála významnou roli při vývoji systému, protože umožnila ověření vlastností a jednoduchou verifikaci výsledků softwarové implementace. Nicméně nebyla dostatečně interaktivní a názorná, proto byla vytvořena softwarová simulace, která věrně imituje podmínky reálného provozu systému a je tak nejlepším možným ověřením navrženého řešení bez nutnosti vytvářet hardwarové vybavení a hledat vhodné prostory pro reálné testy. Pomocí tohoto softwaru pak lze provádět mnoho různých simulačních scénářů, testovat různé matematické modely pro rekonstrukci pohybu a zkoumat jejich vlastnosti.

5.3.1. Simulační framework

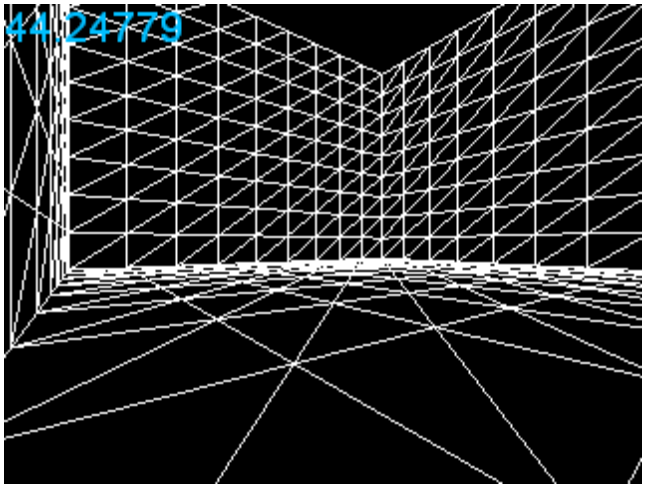
Simulační framework byl vytvořen jako samostatná aplikace, která umožňuje provádět různé simulační scénáře a testovat tak vlastnosti různých metod rekonstrukce, přesnost systému, odezvu, atd. Je implementován stejně jako hlavní aplikace v jazyce C#, pro rendering a ovládání je použito rozhraní DirectX. Před samotným popisem simulace uvedu výčet použitých termínů včetně jejich vysvětlení pro snazší orientaci:

- virtuální kamera (VK) – *Direct3D.device*, které slouží k renderingu scény
- pozice – vektor o třech složkách x, y a z, který určuje pozici VK ve virtuální scéně
- směr pohledu – vektor o třech složkách x, y a z, který určuje směr pohledu VK
- odchylka, náklon, natočení – úhly rotace kolem jednotlivých os. Alternativně mohou tyto úhly vyjadřovat směr pohledu VK, nenulová hodnota značí rotaci kolem příslušné osy. Přiřazení těchto úhlů k jednotlivým osám, stejně jako orientaci souřadného systému vzhledem k virtuální kameře lze vidět na Obr. 25.



Obr. 25Orientace souřadného systému virtuální kamery a přiřazení úhlů rotací k osám souřadného systému

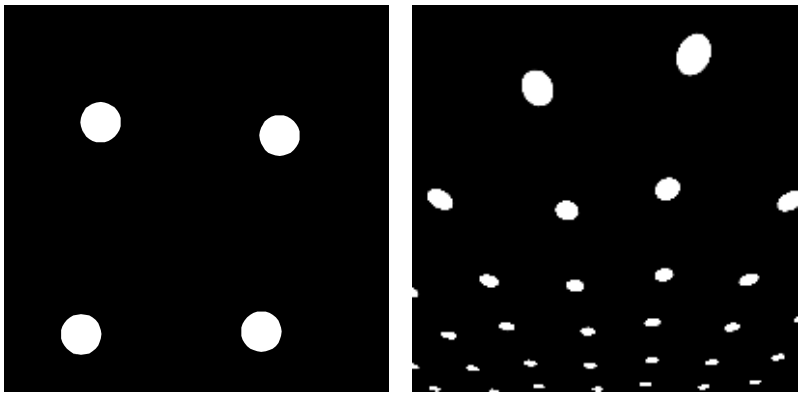
Základem softwarové simulace jsou tři různé virtuální kamery. První je takzvaná simulační virtuální kamera a je součástí jednoduchého enginu, který umožňuje uživateli procházet jednoduché virtuální prostředí. Pozice simulační kamery je ovládána pomocí klávesnice, směr pohledu této kamery pomocí myši, jak je běžné například u 3D počítačových her[14]. Engine simuluje pohyb uživatele po reálné místnosti, simulační VK je zde však čistě z důvodů orientace a snazšího vyhodnocení simulace, protože při reálném běhu aplikace uživatel nevidí místnost, po které se pohybuje díky nasazenému HMD. Virtuální scéna je velmi jednoduchá, jde pouze o krychli bez horní stěny, navíc je celá scéna renderována pouze v režimu drátěné mřížky, jak ukazuje obrázek Obr. 26. Pro simulační účely je však tato podoba velmi výhodná, jak bude zdůvodněno později.



Obr. 26Virtuální scéna simulující místnost, ve které se pohybuje uživatel

Druhou virtuální kamerou tvořící simulaci je takzvaná záznamová virtuální kamera. Ta simuluje záznamové zařízení, které má při reálném běhu systém uživatel na temeni hlavy. Tato VK snímá fiktivní strop, který je tvořen čtverci, na které je namapována textura představující binární obrazy významných bodů, viz Obr. 27.

[14] Odchylka je ovládána pomocí myši, pozice, náklon a natočení jsou ovládány klávesnicí kvůli citlivosti



Obr. 27 Vlevo textura představující významné body, vpravo snímek fiktivního stropu záznamovou VK s náklonem < 90°.

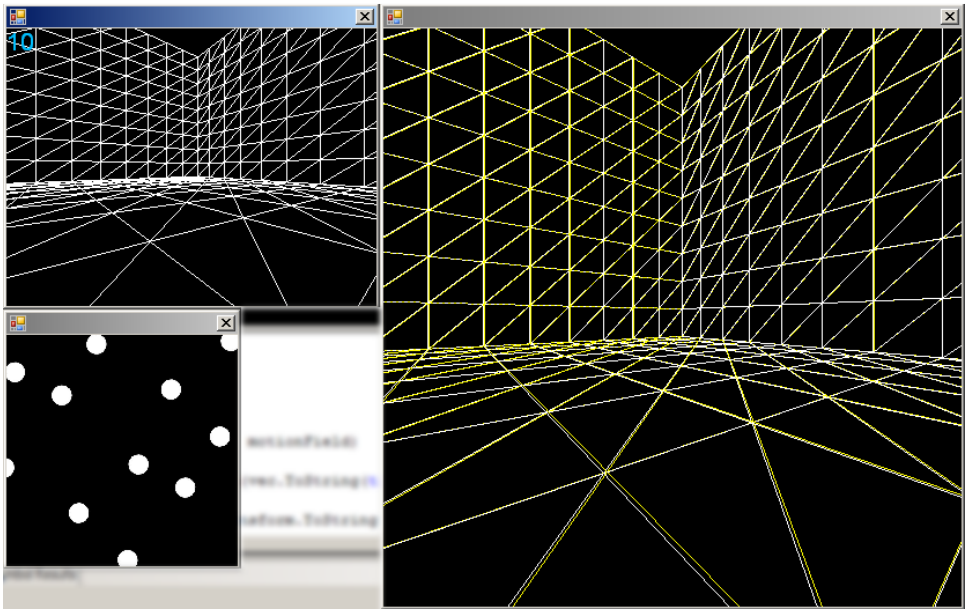
Pozice záznamové VK je shodná s pozicí simulační VK, odchylka i natočení jsou taktéž shodné, pouze náklon záznamové VK je o 90° větší tak, aby v základní poloze směřovala záznamová VK kolmo vzhůru. Samotný záznam významných bodů je realizován pomocí techniky renderingu do textury. Z této textury je pak vytvořena bitmapa, která supluje snímek z webové kamery a je předána spolu s bitmapou z předchozí iterace k rekonstrukci pohybu. Ukázka Uk. 1 ilustruje na výňatku kódu, jak je tento mechanismus implementován v simulačním frameworku. Samotná rekonstrukce je pak prováděna jádrem hlavní aplikace, není tedy součástí simulačního frameworku a bude detailně popsána v kapitole 0.

```
rts.BeginScene(renderSurface);
//…
// rendering stropu s významnými body
//…
rts.EndScene(Filter.None);
frameOld = frameCurr;//snímek z předchozí iterace už není aktuální
//vytvoření bitmapy z textury do které bylo renderováno
GraphicsStream stream = SurfaceLoader.SaveToStream(ImageFileFormat.Bmp,
renderSurface);
frameCurr = new Bitmap(stream);//vytvoření nového aktuálního snímku
//obojí nutné, jinak dochází k memory leaku, což po nějakém čase vyústí v
OutOfMemoryException
stream.Close();
stream.Dispose();
```

Uk. 1Relevantní část kódu ukazující implementaci renderingu do textury

Poslední VK simulačního frameworku je rekonstrukční virtuální kamera, která je simulací toho, co uživatel skutečně vidí v HMD. Renderovaná virtuální scéna je stejná, jako ta, kterou renderuje simulační kamera, pozice a směr pohledu jsou však určeny výstupem z rekonstrukce provedené jádrem hlavní aplikace. Chování rekonstrukčního systému pak lze posoudit porovnáním výstupů simulační a rekonstrukční VK, v ideálním případě budou samozřejmě shodné. Pro ještě názornější porovnání je však do frameworku přidána možnost vyrenderovat scénu nejprve pro pozici a směr pohledu rekonstrukční VK a následně do stejného snímku vyrenderovat odlišnou barvou scénu pro pozici a směr pohledu simulační VK. Díky tomu, že je celá scéna renderována pouze v režimu drátěné mřížky, lze opravdu velmi detailně rozlišit rozdíly mezi simulovaným a rekonstruovaným pohledem. Obrázek Obr. 28 ukazuje náhled celého frameworku včetně překrytí simulačního a rekonstruovaného pohledu. Odchylka simulovaného a rekonstruovaného

pohybu na Obr. 28 odpovídá průměrné odchylce při testování finální verze systému pro rekonstrukci omezenou na translaci a odchylku.



Obr. 28Náhled všech oken simulačního frameworku. Vlevo nahoře pohled simulační VK, vlevo dole pohled záznamové VK, vpravo pohled rekonstruované VK s překrytím pohledu ze simulační VK, který je vykreslen žlutou barvou

6. SOFTWARE

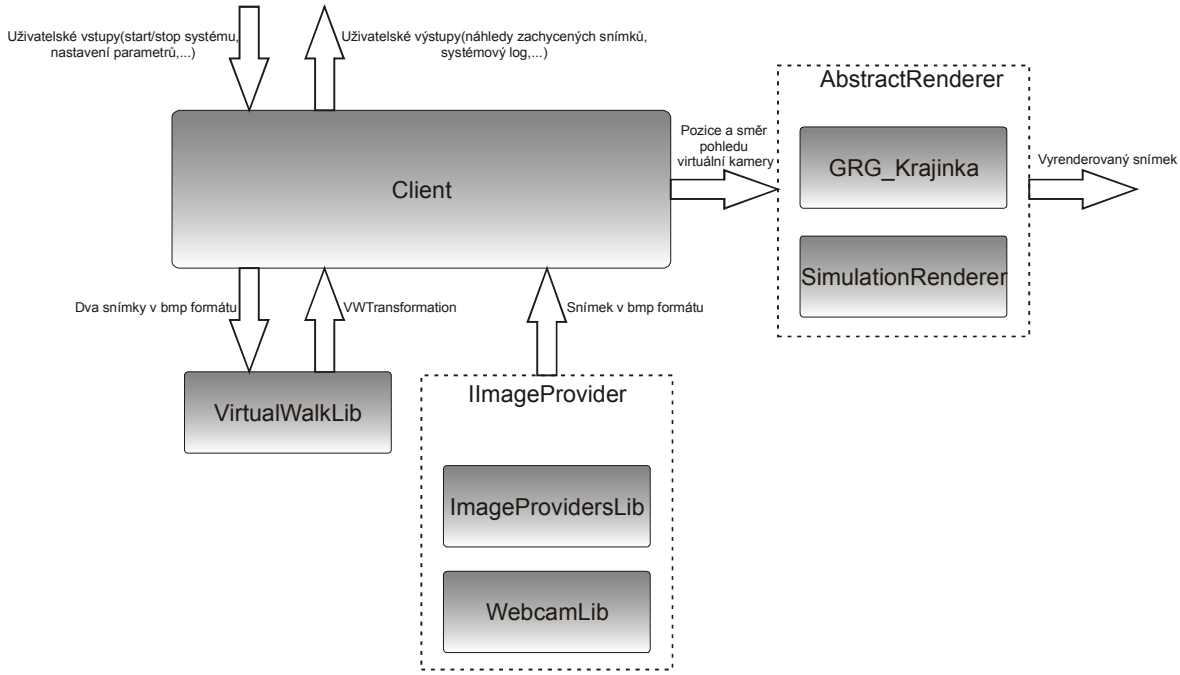
Aby mohl být systém otestován a používán v reálném prostředí, bylo nutné vytvořit jeho softwarovou implementace. Cílem této implementace bylo jednak přenést všechny části návrhu z kapitoly 4 do jazyka počítače, ale také tyto části propojit ve fungující celek a poskytnout uživateli rozhraní, pomocí kterého by systém mohl ovládat. Vzhledem k tomu, že systém není úplně triviální a sestává z mnoha různých částí, bylo vhodné dodržet zásady dobrého softwarového návrhu. Proto byla celá implementace rozdělena do několika nezávislých celků, z nichž každý obsahuje jedno nebo více rozhraní, pomocí nichž ostatní celky s tímto komunikují. Jak ukazuje obrázek Obr. 29, hlavním modulem je Client(dále jen klient), který koordinuje součinnost ostatních modulů a obsahuje uživatelské rozhraní systému[15]. Ostatní moduly mají formu knihoven, jejichž třídy a funkce pak klient kombinuje a řídí tak celý systém. VirtualWalkLib obsahuje veškeré potřebné části nutné k výpočtu rekonstrukce pohybu z dvou snímků, knihovny ImageProvidersLib a WebcamLib umožňují zachycení snímku z různých zdrojů a jsou zpřístupněné přes rozhraní *IImageProvider*. Posledním modulem jsou renderery, které dědí od abstraktní třídy *AbstractRenderer*. Je vhodné podotknout, že díky tomuto modulární návrhu lze jednotlivé komponenty snadno znovu použít v jiné aplikaci, případně některé z nich nahradit. To se typicky týká zejména klienta, který je obvykle velmi specifický pro konkrétní aplikaci.

V aplikaci bylo použito několik různých technologií a programovacích jazyků, stejně jako knihoven třetích stran. Hlavním programovacím jazykem byl C#, část kódu je v jazyce C++, z technologií to byly především DirectX a DirectShow[31]. C# byl zvolen k vytvoření většiny softwarové implementace z několika důvodů. Především je to jeho silně objektově orientovaná povaha, díky níž lze vytvářet velké a přesto přehledné a dobře

[15] V klient-server architektuře by se tedy jednalo o silného (thick) klienta

strukturované aplikace. Navíc mám s OO programováním i jazykem C# velké zkušenosti. Kód, který je pomocí C# vytvořen, je označován jako řízený, to znamená, že není přeložen přímo do strojového kódu, ale do takzvaného mezijazyka. Ten je poté spuštěn v rámci virtuálního stroje, který tento kód překládá do strojového kódu. Virtuální stroj „řídí“ tento kód, to znamená, že automaticky spravuje paměť, provádí obsluhu výjimek atd. To přináší pro programátora další zjednodušení a díky tomu se může plně soustředit na kvalitní návrh aplikace. Řízený kód však s sebou přináší i jednu zásadní nevýhodu a tou je pomalejší běh než u neřízeného kódu. Díky relativní jednoduchosti aplikace však tato nevýhoda neomezila správný chod a zmíněné výhody volby tohoto jazyka nakonec převážily.

Nyní se budu detailněji věnovat jednotlivým modulům z Obr. 29, u každého z nich popíši jeho vnitřní architekturu, použití, kritická místa implementace a podrobněji zmíním použité technologie, knihovny a jazyky použité v popisovaném modulu.



Obr. 29 Zjednodušená vysokoúrovňová architektura aplikace. Šipky mezi moduly naznačují součinnost včetně typu dat v rámci těchto součinností. Čárkované rámečky s popiskem označují rozhraní, přes které s modulem ostatní komunikují.

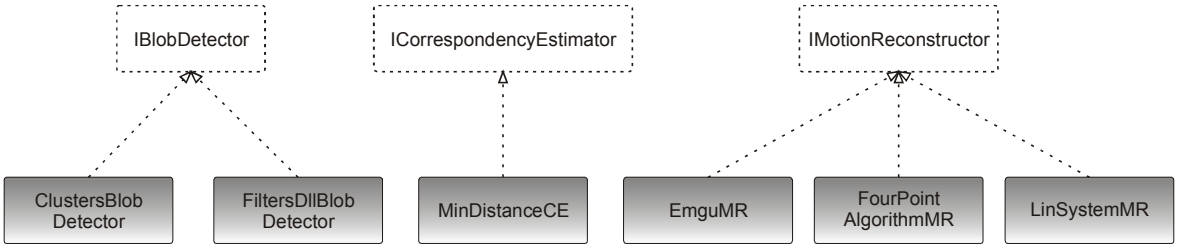
6.1. VIRTUALWALKLIB

Tato knihovna implementuje mechanismy potřebné k rekonstrukci pohybu ze dvou snímků, tak jak byly popsané v 4.1.3, 4.1.4 a 4.2. Knihovna je napsaná kompletně v jazyce C# pro platformu .NET 3.5, až na výjimky jsou všechny datové typy použité v této knihovně ze zdrojových knihoven .NET. VirtualWalkLib je závislá na třech knihovnách třetích stran, jmenovitě EmguCV[32], ze které jsou použity implementace zejména algoritmů počítačového vidění, FiltersDll[28], ze které je použita implementace shlukového detektoru a Math.Net Iridium[33], pomocí které jsou prováděny výpočty lineární algebry.

Cílem bylo vytvořit knihovnu s podobným charakterem jako například právě EmguCV, obsahující třídy, z kterých lze sestavit řešení přizpůsobené konkrétní aplikaci[16]. Spíše než implementovat kompletní pipeline, i když i ta je součástí knihovny, bylo tedy

[16] EmguCV je samozřejmě nesrovnatelně obecnější a rozsáhlejší než VirtualWalkLib, která má velmi specifickou funkci. Odkaz na EmguCV byl uveden pouze jako nastínění zamýšleného stylu.

cílem rozdělit řešení na co nejmenší nezávislé celky, ty implementovat a vytvořit framework, pomocí kterého by bylo možné tyto celky snadno spojovat a přidávat nové/vylepšovat implementaci stávajících. Z toho důvodu vzniklo několik rozhraní obalujících hlavní funkční celky a tato rozhraní pak byla implementována jednotlivými třídami. Tímto je zajištěn unifikovaný přístup k jednotlivým celkům a implementace je částečně odstíněna od programátora sestavujícího pipeline. Přehled jednotlivých rozhraní a tříd, které je implementují, viz Obr. 30.



Obr. 30Přehled rozhraní obalujících hlavní funkcionality. Čárkované obdélníky jsou rozhraní, plné obdélníky jsou implementující třídy, nevyplněná čárkovaná šipka pak vyjadřuje relaci implementuje.

Nyní ve zkratce popíši jednotlivá rozhraní a třídy, které je implementují:

- IBlobDetector

IBlobDetector
ProcessFrame(Bitmap) : List<PointF> *Init()*

  - Rozhraní pro objekty zajišťující detekci shluků ve snímku. Obsahuje metodu, která ve vstupním snímku nalezne shluky a vrátí seznam jejich středů.
  - Implementováno třídami:
    - ClusterBlobDetector – implementace mechanismu popsaného v 4.1.4 s podnadpisem Clusterová analýza
    - FiltersDllBlobDetector – obalující třída pro detektor shluků z knihovny FiltersDll[28]
- ICorrespondencyEstimator

ICorrespondencyEstimator
FindCorrespondencies(List<PointF>,List<PointF>) : List<VWMotionVector>

  - Rozhraní pro objekty zajišťující vytvoření korespondencí. Obsahuje metodu, která mezi dvěma seznamy bodů nalezne korespondence a vrátí seznam vektorů popisujících tyto korespondence.
  - Implementováno třídami:
    - MinDistanceCE – implementace mechanismu popsaného v 4.2.1
- IMotionEstimator

IMotionReconstructor
EstimateMotion(List<VWMotionVector>) : VWTransform

- Rozhraní pro objekty zajišťující rekonstrukci pohybu. Obsahuje metodu, která ze seznamu vektorů popisujících korespondence rekonstruuje pohyb a jeho popis vrátí ve speciální datové struktuře (bude popsána dále).
- Implementováno třídami:
  - EmguMR – wrapper pro funkce knihovny EmguCV[32], které rekonstruují pohyb pomocí mechanismu popsaného v 4.2.2 s podnadpisem Rekonstrukce pohybu pomocí nalezení a rozkladu homografie s RANSAC optimalizací
  - FourPointAlgorithmMR – implementuje mechanismus popsaný v 4.2.2 s podnadpisem Rekonstrukce pohybu pomocí nalezení a rozkladu homografie s použitím čtyřbodového algoritmu
  - LinSystemMR – implementuje mechanismus popsaný v 4.2.2 s podnadpisem Rekonstrukce pomocí lineárního systému

V návratových typech metod některých uvedených rozhraní se objevili specifické datové typy VWTransform a VWMotionVector. Ty jsou také součástí VirtualWalkLib, proto je nyní ve zkratce popíši:

- VWTransform

VWTransformation
yaw : float *pitch : float* *roll : float* *translation : Vector*
FromHomography(Matrix) : VWTransformation

  - Třída popisující transformaci, obsahuje 3 eulerovské úhly odchylky, náklonu a natočení[17] a vektor translace
  - Kromě obvyklých konstruktorů a několika privátních metod obsahuje také statickou metodu pro vytvoření VWTransformation z matice homografie. Implementuje rozklad homografie na jednotlivé části, jak bylo popsáno v 3.3.3 a následně ještě rozloží rotační matice na jednotlivé úhly, jak bylo popsáno v 4.2.3.
- VWMotionVector

VWMotionVector
head : PointF *tail : PointF* *vector : Vector*
Transform(Matrix)

  - Třída reprezentující vektor, který popisuje korespondenci. Seznam objektů tohoto typu tvoří pohybové pole a je použit jako vstup pro rekonstrukci pohybu. Bod head je korespondující bod k bodu tail. Atribut vector je pak klasická vektorová reprezentace.

[17] Volný překlad z anglických termínů yaw – rotace kolem osy y, pitch - rotace kolem osy x, roll – rotace kolem osy z

- Pomocí metody Transform lze na body head a tail aplikovat libovolnou maticovou transformaci.

Dalšími důležitými objekty, které knihovna obsahuje, jsou dvě vlastní výjimky indikující chybové stavy, které mohou při rekonstrukci nastat. Těmito výjimkami jsou:

- FeaturePointsNotFoundException – tato výjimka nastane, pokud jsou po segmentaci snímku nalezeny méně než čtyři shluky. V takovém případě není rekonstrukce možná a je pravděpodobné, že došlo k zákrytu snímacího zařízení, případně uživatel vyšel z oblasti pokryté významnými body a měl by se o tom dozvědět.
- InvalidTransformation – tato výjimka není v knihovně explicitně vyhazována nikde. Vyhození této výjimky by měla zajistit nadřazená aplikace ve chvíli, kdy je zjištěno, že transformace reprezentovaná VWTransformation není validní. Ta není validní v případě, že některý z jejích atributů překročí předem nastavenou povolenou maximální mez, což indikuje chybu ve výpočtu a mohlo by to zásadně ovlivnit uživatelovo vnímání a zničit dojem virtuální reality. Reakce na tuto výjimku by tedy mohla být přerušení výpočtu a doporučení resetu uživatelovi pozice.

VirtualWalkLib kromě implementací jednotlivých objektů obsahuje i kompletní zpracovatelskou pipeline, jejímž vstupem jsou dva snímky ve formátu bmp a výstupem VWTransformation popisující transformaci kamery, která mezi snímky proběhla. Ukázka Uk. 2 nastiňuje jakým způsobem pipeline funguje a ukazuje její modifikovatelnost díky použití rozhraní.

```
private IBlobDetector blobDetector;
private ICorrespondencyEstimator correspondecyEstimator;
private IMotionReconstructor motionReconstructor;
//…
//pParams je struktura, pomocí které lze nastavovat jednotlivé části
//pipeline. Pokud není v pParams příslušná část nastavena, je použita
//defaultní implementace
blobDetector = pParams.BlobDetector == null ?
new FiltersDllBlobDetector() : pParams.BlobDetector;
correspondecyEstimator = pParams.CorrespondecyEstimator == null ?
new MinDistanceCE() : pParams.CorrespondecyEstimator;
motionReconstructor = pParams.MotionReconstructor == null ?
new LinSystemMR() : pParams.MotionReconstructor;
//…
//frame1 a frame2 jsou vstupní snímky
List<VWMotionVector> motionField = ConstructMotionField(frame1, frame2);
return motionReconstructor.EstimateMotion(motionField);
```

Uk. 2 Část kódu provádějící inicializaci pipeline a jednu iteraci rekonstrukce pohybu ze dvou snímků

6.2. IMAGEPROVIDERSLIB A WEBCAMLIB

Knihovna ImageProvidersLib tvoří důležitou část aplikace, protože, jak vyplývá z jejího názvu, obsahuje objekty poskytující snímky. Kromě těchto objektů obsahuje další utility, které se zachycením snímků, jejich úpravou a zpracováním souvisí. Knihovna obsahuje prozatím dva různé druhy objektů poskytujících snímky. Jsou jimi

- ImageLoader – načítá statické snímky, které jsou předem uložené na disku. Do aplikace byl přidán zejména z ladicích a simulačních účelů, protože umožňuje, aby systém pro rekonstrukci pracoval se stále stejnou předpřipravenou sekvencí, díky čemuž i výstupy rekonstrukce budou stejné a je možné je pak snadno

porovnávat/vyhodnocovat. Tímto objektem už se podrobněji nebudu zabývat, protože jeho implementace je poměrně triviální.

- Camera – wrapper pro knihovnu WebcamLib, která umožňuje získat snímky z webové kamery. Knihovna WebcamLib bude níže detailněji popsána.

Oba objekty lze využít v aplikaci samostatně, případně k nim přistoupit přes jednotné rozhraní IImageProvider. Toto rozhraní má následující atributy a metody

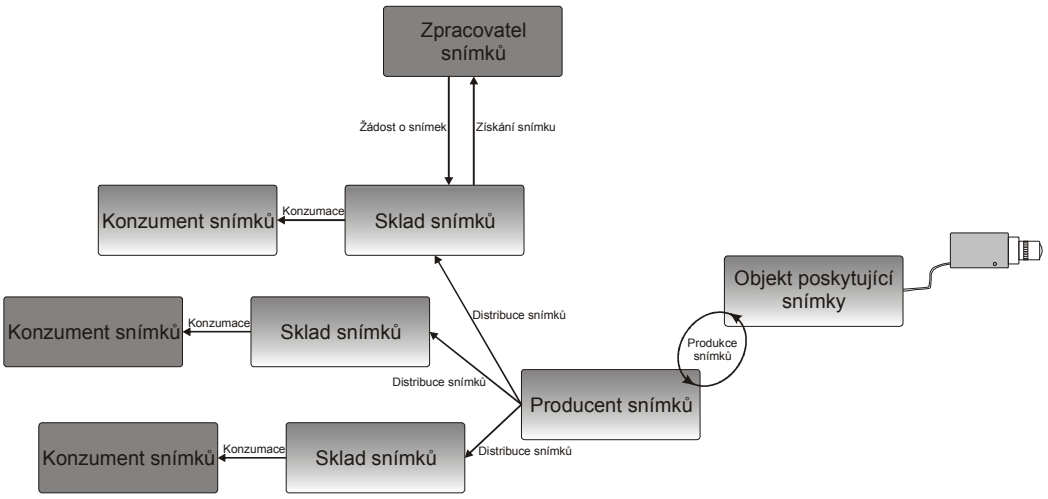
IImageProvider
imageAcquired : bool *imageWidth : int* *imageHeight : int* *imageBufferSize : int* *imageBuffer : byte[]* *bitsPerPixel : int* *imageTag : string*
Init() *Start()* *Stop()* *Pause()*

Kromě zřejmých atributů samotného snímku jako je výška, šířka, počet bitů na pixel atd. má rozhraní atribut *imageBuffer,* což je pole, které je naplněno vždy naposledy zachyceným snímkem. S tím úzce souvisí atribut *imageAcquired*, který má hodnotu true, pokud snímek, který je aktuálně v *imageBufferu* nebyl odtud ještě vyzvednut. Pokud je z *imageBufferu* vyzvednut, musí být atribut *imageAcquired* nastaven na false. Pokud by totiž například snímací zařízení mělo nízkou snímkovou frekvenci v poměru k době výpočtu rekonstrukce a nekontrolovala by se aktuálnost snímku, mohlo by dojít k tomu, že stejný snímek bude zpracován víckrát a výpočet by se tak mohl dostat do nekonzistentního stavu.

Knihovna ImageProvidersLib obsahuje také komplexnější způsob získávání snímků, který částečně vychází ze známého schématu producent-konzument a využívá vlákna. Pracuje s dvěma základními objekty:

- sklad snímků
- producent snímků

Sklad snímků slouží jako buffer, do kterého producent snímků ukládá nový snímek vždy, když ho získá a s každým skladem snímků je asociován konzument snímku, který získá nový snímek automaticky při jeho uložení do skladu. Snímek je ve skladu uložen vždy pouze jeden, pokud je uložen nový, starý je díky tomu přepsán. Snímek může být ze skladu vybrán jakýmkoliv dalším objektem, takzvaným zpracovatelem snímku, který ho zrovna potřebuje, pořád je ale zachován výše zmíněný princip, že jeden snímek může být ze skladu vybrán pouze jednou. Typicky je zpracovatelem klient, který si vyžádá další snímek pro novou iteraci rekonstrukce. Typickým konzumentem snímku je pak například ovládací prvek GUI, který zobrazuje náhledy zachycených snímků. Producent v sobě obsahuje vlákno, které běží nezávisle na hlavním a pokud objekt poskytující snímky (např. webkamera) zachytí nový snímek, tzn. atribut *imageAcquire* má hodnotu true, tento snímek vezme a uloží do všech zaregistrovaných skladů snímků. Schéma popsaného mechanismu lze vidět na obrázku Obr. 31.



Obr. 31Jednotlivé role při zachycení a předávání snímků a součinnosti jednotlivými aktéry.

Pro názornost nyní uvedu třídy implementující sklad snímků a producenta snímků

FrameStore
readerFlag : bool writerFlag : bool
FrameData.get() : Bitmap FrameData.set(Bitmap)

FrameProducer
registeredFrameStores : List<FrameStore> imageProvider : IImageProvider workerThread : Thread
Start(ImageProviderType) Pause() Stop()

Atributy readerFlag a writerFlag u skladu snímků budou vysvětleny níže, metody get a set náleží k property[18] FrameData, jejich funkce je zřejmá. Producent snímků obsahuje metody pro jeho spuštění, zapauzování a zastavení, které víceméně volají stejně pojmenované metody rozhraní IImageProvider. Metoda Start kromě toho spouští i vlákno, které získává snímky z IImageProvider a distribuuje snímky do registrovaných skladů snímku. V C# je nové vlákno vytvořeno pomocí konstruktoru s parametrem, kterým je delegát na metodu, která bude ve vláknu prováděna. Pokud je potřeba, aby i prováděná metoda měla parametr, je možné využít parametrizovaného delegáta, ten však umožňuje pouze jeden parametr, navíc typu Object, takže typová kontrola je na programátorovi. Proto byla vytvořena pomocná třída WorkerThread, která umožňuje flexibilnější parametrizaci metody pracující ve vláknu. Třída WorkerThread má následující podobu

[18] Mechanismus v C#, který má v podstatě stejnou funkci například jako getry a setry v Javě

WorkerThread
imageProvider : IImageProvider *frameStores : List<FrameStore>*
Work()

kde atributy imageProvider a frameStores jsou této třídě předány v konstruktoru. Metoda Work pak provádí veškerou práci a je předána jako parametr při vytváření vlákna. Vytvoření vlákna a jeho spuštění pak demonstruje Uk. 3.

```
//vytvoření instance pomocné třídy s dříve inicializovanými parametry
WorkerThread worker = new WorkerThread(imageProvider,
registeredFrameStores);
//vytvoření nového vlákna s delegátem funkce Work jako parametrem
workerThread = new Thread(new ThreadStart(worker.Work));
//spuštění vlákna = spuštění metody Work
workerThread.Start();
```

Uk. 3Část kódu, který vytváří a spouští nové vlákno

Samotná metoda Work pomocné třídy WorkerThread obsahuje nekonečnou smyčku, která je prováděna, dokud není vlákno zastaveno a zrušeno. V této smyčce se vždy dotazuje objektu poskytujícího snímky, zda je k dispozici nový snímek a pokud ano, vyzvedne z imageBufferu data snímku, vytvoří z nich bitmapu a distribuuje jí do všech registrovaných skladů snímků. Když snímek k dispozici není, vlákno neprovede nic. Pokud by toto bylo implementováno jednoduchou rozhodovací smyčkou, která v případě, že snímek není připraven, pouze pokračuje další iterací smyčky, vedlo by to ke klasickému aktivnímu čekání vlákna. Nevýhodou aktivního čekání je zbytečná konzumace procesorového času, který musí toto vlákno neustále plánovat a snižuje se tak kapacita na jiná vlákna/procesy, které provádí skutečnou práci. Proto jsem v případě, že snímek není k dispozici, implementoval blokující čekání, které uspí vlákno na monitoru[19] objektu poskytujícího snímky. Ten čekající vlákno vzbudí ve chvíli, kdy získá nový snímek, voláním Notify. Pro výňatek z kódu metody Work ilustrující právě popsané mechanismy viz Uk. 4.

```
//nekonečná smyčka, která je přerušena až při zastavení a zrušení vlákna
while (true)
{
	//pokud nemá objekt poskytující snímky k dispozici nový snímek,
	//nelze pokračovat
	if (!imageProvider.ImageAcquired)
	{
		//vstup do kritické sekce, získání výhradního přístupu
		//k objektu poskytujícím snímky
		lock (imageProvider)
		{
			//blokující čekání, vlákno usne na monitoru objektu
			//poskytujícího snímky do doby, než tento získá nový
			//snímek a vlákno vzbudí voláním this.Notify()
			Monitor.Wait(imageProvider);
		}
	}
	//…
	//vyzvednutí snímku z imageBufferu a jeho převod na bitmapu
	//…
	//distribuce snímku do všech registrovaných skladů snímků
	//…
```

[19] Synchronizační primitivum, které obsahuje každý datový typ v C# dědící od základního datového typu Object

```
}
```

Uk. 4Výňatek z kódu metody Work, demonstrující implementaci blokujícího čekání na monitoru objektu poskytujícího snímky

Díky přítomnosti více vláken je nutné řešit jejich synchronizaci. Ta také v podstatě vychází ze schématu producent-konzument a využívá již zmíněné monitory jako synchronizační primitiva. Pro ošetření kritických sekcí je využito mutexu, který je označen v C# klíčovým slovem lock a jeho použití je vidět v Uk. 4. Veškerá synchronizace se odehrává v metodách skladu snímků, přesněji řečeno v get a set metodách property FrameData, které vybírají/ukládají snímky z/do sdílené proměnné frameData typu Bitmap. Synchronizace vychází z [34]:

- Při výběru je nejprve uzamčena kritická sekce získáním výhradního zámku ke skladu snímků. Poté je zjištěno, zda do frameData zrovna není zapisováno. Pokud ano, je volající vlákno uspáno na monitoru skladu snímků, k jeho vzbuzení dojde při ukončení zápisu. Pokud do frameData není zapisováno, je nastavena vlajka čtení z frameData, data jsou přečtena, následně je vlajka čtení zrušena a všechna vlákna, která byla uspána na skladu snímků, jsou probuzena. Část kódu implementující právě popsanou synchronizaci je v Uk. 5.

```
//uzamčení kritické sekce získáním výhradním zámku
lock (this)
{
    //pokud je vlajka writerFlag true, do frameData //zapisováno,
    čtení tedy není možné kvůli konzistenci dat
    if (writerFlag)
    {
          //uspání na monitoru této instance dokud není zápis
          //hotov
          Monitor.Wait(this);
    }
    try
    {
          //signalizace čtení ze sdílené proměnné
          readerFlag = true;
          //signalizace, že snímek již není aktuální
          hasFreshData = false;
          //vyzvednutí dat
          return frameData;
    }
    finally
    {
          //signalizace ukončení čtení
          readerFlag = false;
          //vzbuzení všech uspaných vláken na monitoru této
          //instance
          Monitor.PulseAll(this);
    }
}
```

Uk. 5 Ukázka synchronizace vláken při vyzvednutí dat ze sdílené proměnné frameData

- Ukládání probíhá obdobně jako čtení, nejprve je stejným způsobem uzamčena kritická sekce, poté je zjištěno, zda není ze sdílené proměnné čteno, pokud ano, je provedeno uspání vlákna. Zbytek je taktéž stejný, kromě záměny čtení za zápis. Pro kód provádějící popsaný mechanismus viz Uk. 6.

```
//uzamčení kritické sekce získáním výhradním zámku
lock (this)
{
```

```
        //pokud je vlajka writerFlag true, do frameData je
        //zapisováno, čtení tedy není možné kvůli konzistenci dat
        if (readerFlag)
        {
            //uspání na monitoru této instance dokud není zápis hotov
            Monitor.Wait(this);
        }

        try
        {
              //signalizace čtení ze sdílené proměnné
              writerFlag = true;
              //signalizace, že snímek již není aktuální
              hasFreshData = true;
              //uložení dat a jejich automatické odeslání
              //konzumentovi
              frameData = value;
              frameDataConsumer.FrameData = frameData;
        }
        finally
        {
              //signalizace ukončení čtení
              readerFlag = false;
              //vzbuzení všech uspaných vláken na monitoru této
              //instanci
              Monitor.PulseAll(this);
        }
    }
```

Uk. 6 Ukázka synchronizace vláken při uložení dat do sdílené proměnné frameData

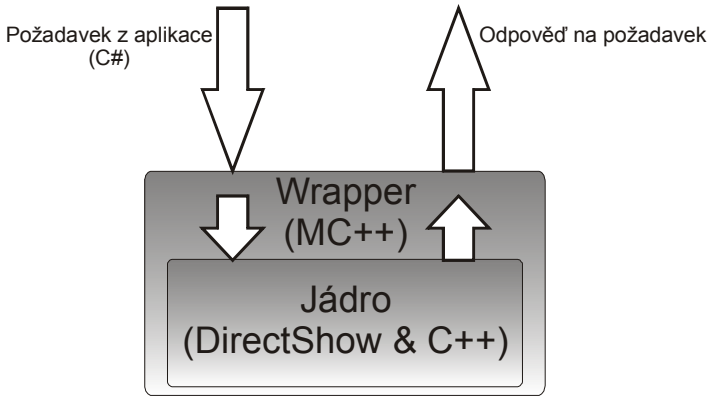
6.2.1. WebcamLib

Tato knihovna umožňuje získávat snímky z libovolného snímacího zařízení, jak už ale z názvu vyplývá, je vytvořená především k získávání snímků z webové kamery. Knihovna v sobě kombinuje několik technologií a programovacích jazyků:

- DirectShow – multimediální framework od firmy Microsoft, který umožňuje pomocí takzvaných filtrů zpracovávat proudy multimediálních dat. Podrobnou dokumentaci k DirectShow lze nalézt v [31]. V knihovně WebcamLib slouží k zachycení snímku z webové kamery.
- MC++ - řízená verze jazyka C++. Přeloží se do řízeného kódu, který je spravován virtuálním strojem, který ho teprve poté překládá do strojového jazyka. Výhody jsou zejména automatická zpráva paměti a možnost interoperovat s dalšími managed jazyky jako je C#. V MC++ je napsán wrapper pro jádro knihovny napsané v C++.
- C++ - neřízená verze jazyka C++, v ní je napsáno jádro knihovny zajišťující získání snímku.

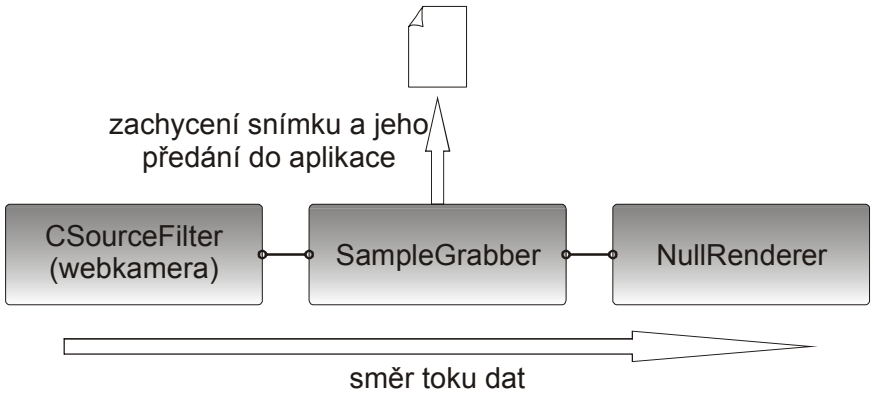
Architektura knihovny, naznačená na Obr. 32, tedy sestává z jádra, které je napsané v jazyce C++ a využívá DirectShow k zachycení snímku z webové kamery. Toto jádro je pak obalené wrapperem v MC++, který umožňuje volat funkce jádra ve zbytku aplikace, která je v C#[20].

[20] Knihovnu jsem vytvořil již v roce 2008 pro potřeby mé bakalářské práce a v průběhu následujících let ji mírně upravil. V současné době již existuje wrapper pro DirectShow přímo v C#, není



Obr. 32 Zjednodušená architektura WebcamLib z pohledu použitých platforem

Jak bylo řečeno výše, pracuje DirectShow s takzvanými filtry, které lze spojovat a vytvářet tím takzvané grafy. Tyto grafy pak lze spouštět, čímž dojde k tomu, že proudy dat začnou téct přes jednotlivé filtry, které je zpracovávají podle své funkce. Úkol WebcamLib je velice jednoduchý, z proudu dat z webové kamery vždy zachytit celý snímek a předat ho do aplikace. K tomuto účelu stačí pouhé tři filtry spojené za sebou. Prvním je zdrojový filtr zprostředkovávající přístup k datům kamery, druhým je takzvaný SampleGrabber, který umožňuje z proudu dat zachytit vzorek, v mém případě tedy snímek. Posledním filtrem je NullRenderer, který příchozí data jednoduše zahodí. Tento filtr je povinný, protože data v grafu musí téct odněkud někam. Obrázek Obr. 33 ukazuje graf s jednotlivými filtry.



Obr. 33 Graf, umožňující zachycení snímku z webové kamery

Přiřazení webové kamery zdrojovému filtru probíhá pomocí vestavěného enumerátoru systémových zařízení, který je součástí DirectShow. Enumerátor vybere první zařízení, které spadá do kategorie vstupních video zařízení a přiřadí ho ke vstupnímu filtru. DirectShow obsahuje defaultní implementaci filtru SampleGrabber, tu jsem ale nahradil vlastní třídou CustomGrabber, která dědí od třídy CTransInPlaceFilter. Třída CustomGrabber má následující parametry a funkce

CustomGrabber
f_cb : imageAcquiredCB
CheckInputType(CMediaType)* *Transform(IMediaSample*)* *SetCallBack(imageAcquiredCB)*

Metoda transform je volána, když do filtru přijde nový snímek, jehož data jsou ve struktuře IMediaSample. V metodě jsou pak z této struktury získány informace o snímku, jako je

tedy nutné kód, který ho využívá psát v C++. Vzhledem k tomu, že knihovnu v této podobě používám již dlouho a je poměrně dobře odladěná, neměl jsem potřebu jí portovat na C# wrapper DirectShow.

šířka, výška a počet bitů na pixel. Odeslání snímku do aplikace se děje přes callback funkci f_cb, jejíž delegát je nastaven pomocí metody SetCallBack.

6.3. RENDERERY

Tyto objekty zajišťují finální vykreslení virtuální scény na základě zrekonstruované pozice a směru pohledu uživatele. S renderery komunikuje klient pomocí abstraktní třídy AbstractRenderer opět kvůli odstínění samotné implementace a snadnému přidání nových rendererů. Abstraktní třída AbstractRenderer obsahuje kromě jiných tyto atributy a metody

AbstractRenderer
RenderFrame() *MainCamera.get() : VirtualCamera* *MainCamera.set(VirtualCamera)*

Metoda RenderFrame vyrenderuje vždy jeden kompletní snímek pro aktuální zrekonstruovanou pozici a směr pohledu. Datový typ VirtualCamera pak slouží právě k přenosu informace o pozici a směru pohledu, z kterého se má renderovat. Informace o směru pohledu jsou reprezentované stejně jako u objektu VWTransformation popsaného v 6.1.

Od třídy AbstractRenderer dědí dva objekty, aplikace tedy obsahuje dva různé renderery. Oba jsou napsané v jazyce C#, liší se však co do použití video rozhraní. Prvním je SimulationRenderer, který ke komunikaci s grafickou kartou používá DirectX. Tento renderer je v podstatě totožný se softwarovým simulačním frameworkem popsaným v 5.3.1. Druhým rendererem je externí aplikace, kterou jsem vypracovával jako semestrální práci pro předmět KIV/GRG[21]. Tento renderer používá rozhraní SlimDX a také obsahuje pokročilé efekty implementované pro procesor grafické karty pomocí jazyka HLSL.

6.4. CLIENT

Klient, jak už bylo řečeno výše, spojuje funkce jednotlivých modulů, obsahuje grafické uživatelské rozhraní, řídí celou aplikaci a je také jediným spustitelným modulem v rámci celé aplikace. Je napsán kompletně v jazyce C# pro platformu .NET 3.5. Obsahuje grafické uživatelské rozhraní (GUI), pomocí kterého lze celý systém ovládat a nastavovat jeho parametry. Hlavní činností klienta je zajišťovat chod celého systému prováděním iterací výpočetní smyčky. Tato smyčka obsahuje volání všech dříve popsaných modulů a tvoří jádro celé aplikace. Důležité části iterace výpočetní smyčky demonstruje výňatek z kódu v Uk. 7.

```
//výpočet provádět pouze ve chvíli kdy sklad snímků obsahuje ještě
//nevyzvednutý snímek
if (frameStore.HasFreshData)
{
      //…
      //vyzvednutí aktuálního snímku ze skladu
      Bitmap fCurrent = frameStore.FrameData;
      frame1 = frame2;
      frame2 = fCurrent;
      frame2.Tag = fCurrent.Tag;

      try
      {
```

```
            //rekonstrukce pohybu ze získaných snímků, objekt userMotion
            //zajistí i aktualizaci virtuální kamery rendereru
            userMotionEngine.UpdateUserMotion(ref frame1, ref frame2);
        }
        catch (VirtualWalkLib.InvalidTransformException e)
        {
            logger.AppendSystemInfo(e.Message);
            renderer.ErrorMessage = "Invalid transform. Reset view.";
        }
        catch (VirtualWalkLib.FeaturePointsNotFoundException e)
        {
            logger.AppendSystemInfo(e.Message);
            renderer.ErrorMessage = "Feature points not visible.";
        }
}
//…
renderer.RenderFrame();
```

Uk. 7 Relevatní část kódu demonstrující jednu iteraci výpočetní smyčky

Nastavování parametrů systému je realizováno pomocí objektu .NET k tomu určeného a tím je objekt Settings. Ten zajistí vytvoření perzistentního úložiště, ve kterém zůstanou nově nastavené hodnoty parametrů i po restartu aplikace, navíc lze nastavení editovat i bez spuštění aplikace, zároveň s perzistentním úložištěm se vygeneruje i xml soubor, s jednotlivými parametry jako uzly.

7. HARDWARE

V předchozích částech textu byly zmiňovány různé části hardwaru nutného k provozu systému. Tento hardware bylo nutné zkombinovat a sestrojit návrh vybavení, které budou využívat jednotliví uživatelé systému. Návrh vybavení vycházel z předpokladů systému, jehož ústředním prvkem je snímací zařízení umístěné na temeni uživatelovy hlavy a směřující kolmo vzhůru. Snímací zařízení pak snímá pasivní významné body na stropě, které jsou osvěcovány přídavným infračerveným osvětlením. Na základě těchto předpokladů pak byla řešena konstrukce následujících komponent:

- uchycení snímacího zařízení
- přídavné infračervené osvětlení

Kromě těchto zkonstruovaných komponent jsou však součástí vybavení i další, již hotové kusy hardwaru, z nichž některé však bylo nutné modifikovat pro potřeby systému. V této kapitole tedy popíšu konstrukční detaily výše zmíněných komponent a dále uvedu výčet dalšího použitého hardwaru včetně případných modifikací. Veškeré zde popsané části vybavení jsou součástí sady pro jednoho uživatele, kromě poslední podkapitoly, ve které jsou stručně zmíněné samotné významné body, a které jsou sdílené všemi uživateli systému.

7.1. UCHYCENÍ SNÍMACÍHO ZAŘÍZENÍ

Snímací zařízení musí být pevně uchyceno na uživatelově hlavě, aby kopírovalo přesně pohyby hlavy a do rekonstrukce pohybu nebyl zanášen šum, způsobený vágním uchycením a nechtěným pohybem snímacího zařízení. Po několika prototypech, jako například přichycení snímacího zařízení na čepici silnou vrstvou lepicí pásky, které byly spíše provizorního charakteru, byla k uchycení využita stavební helma. Původním

záměrem bylo vyvrtat otvory do skořepiny helmy, kterými se provléknou upínací pásky a zafixují tak kameru na helmě. Vzhledem k oblému tvaru helmy však bylo obtížné snímací zařízení umístit tak, aby směřovalo kolmo vzhůru, navíc je samotná helma dost vysoká, což snižovalo vzdálenost snímacího zařízení od stropu. To mělo za následek kromě jiného i zmenšení prostoru snímaného snímacím zařízením a tím pádem menší počet významných bodů v jednom snímku.

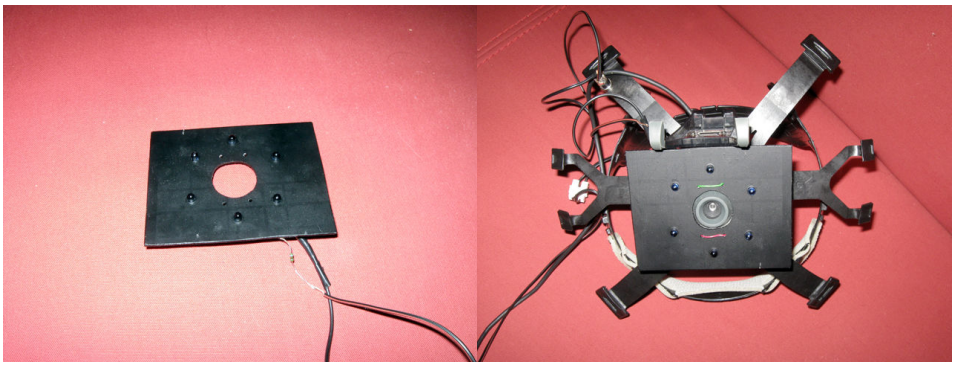
Proto byla nakonec využita pouze síťka, která drží helmu na hlavě. Tato síťka má řadu výhod, je velmi lehká, poskytuje svojí konstrukcí spoustu možností ke snadnému přichycení dalšího vybavení a umožňuje nastavení velikosti obvodu a lze jí tak přizpůsobit různým velikostem hlavy. Snímací zařízení bylo k síťce přichyceno pomocí stahovacího pásku. Kompletní konstrukční řešení lze vidět na Obr. 34.

Obr. 34Síťka s přichycenou kamerou

7.2. PŘÍDAVNÉ INFRAČERVENÉ OSVĚTLENÍ

Přídavné infračervené osvětlení má za úkol osvětlovat pasivní významné body vyrobené z reflexní fólie, které toto světlo odráží a jsou díky tomu výrazné ve snímku ze snímacího zařízení. Konstrukční řešení bylo inspirováno komerčními produkty, jakými jsou například bezpečností kamery s možností nočního vidění. U nich jsou infračervené LED umístěny do kruhu okolo objektivu kamery.

Pro uchycení LED byla vyříznuta destička z tvrdého plastu, do které byl uprostřed vyvrtán otvor pro objektiv snímacího zařízení. Kolem tohoto otvoru bylo vyvrtáno šest menších otvorů, do kterých byly vsazeny infračervené LED. Destičku s LED včetně jejího přichycení na kameru lze vidět na Obr. 35.



Obr. 35Vlevo destička s přídavným infračerveným osvětlením, vpravo destička přichycená na snímací zařízení

7.3. SNÍMACÍ ZAŘÍZENÍ

Jako snímací zařízení byla zvolena díky dobré dostupnosti webkamera Webcam Live! od firmy Creative. Tato webkamera má běžnou konstrukci i parametry. Její průměrná snímková frekvence je okolo 25 snímků za sekundu, rozlišení snímku je 320×240 pixelů. Aby však kamera mohla být použita spolu s infračerveným osvětlením, bylo nutné jí modifikovat. Většina snímacích zařízení obsahující CCD čipy je sice velmi citlivá na infračervené světlo, to je ale velmi často filtrováno, protože způsobuje nechtěné artefakty ve snímcích. Tento filtr je obvykle realizován pomocí tónovaného sklíčka umístěného těsně před čočkou objektivu. Aby mohla být tedy využita citlivost kamery na infračervené světlo, musel být tento filtr demontován, což naštěstí u použité webkamery šlo poměrně snadno. Obr. 36 pak ukazuje výrazný rozdíl mezi snímkem infračervených LED pořízených webkamerou s infračerveným filtrem a snímkem bez tohoto filtru.

Obr. 36Vlevo snímek infračervených LED kamerou s IR filtrem, vpravo ty samé LED ve snímku z kamery bez filtru

7.4. HEAD MOUNTED DISPLAY

Použití HMD v této práci bylo nevyhnutelné. Umožňuje, aby se uživatel volně pohyboval a rozhlížel, ale měl přitom zobrazovanou scénu neustále před očima. Tento faktor a také to, že uživatel díky HMD nevidí okolní prostředí, zvyšuje míru immersion[21]. V práci byl použit HMD i-Glasses PC/SVGA od firmy i-O Dislpay, jehož popis lze nalézt v podkapitole 2.1.1 . Bohužel jde o dost zastaralý model, který trpí řadou neduhů, jakými jsou především velmi špatná ergonomie, nízké rozlišení displejů a jejich malá velikost. Tyto nedostatky naopak míru vtáhnutí snižují, ale v době vývoje byl tento HMD pro mě jediný dostupný.

7.5. VÝZNAMNÉ BODY

Významné body byly vyrobeny ze vzorkového katalogu firmy 3M. Ten obsahoval vzorky reflexních fólií především pro pruhy na oblečení policistů, hasičů a podobně. Z těchto pruhů fólie byly nařezány přibližně čtverce o straně zhruba 1-2 cm. Díky návrhu systému pak mohou být čtverce reflexní folie rozmístěny po stropě libovolně za předpokladu, že bude dodržena minimální vzdálenost mezi nimi, jak je popsáno v 8.1.1.

[21] viz kapitola 2

8. OMEZENÍ SYSTÉMU A MOŽNÉ APLIKACE

V této kapitole budou uvedeny omezení systému včetně jejich zdůvodnění a dopadů na provoz. V poslední podkapitole jsou pak popsány možné reálné aplikace systému.

8.1. OMEZENÍ ČÁSTI PRO NALEZENÍ UŽIVATELOVY POZICE A SMĚRU POHLEDU

8.1.1. Určení korespondencí

V sekci 4.2.1 byla ověřena funkčnost algoritmu pro určení korespondencí na jednoduchém příkladu, který vycházel z běžných podmínek provozu. Bylo by ale vhodné odvodit podmínku, která v obecném případě určí, pro jaké parametry bude použití kritéria minimální vzdálenosti korektní pro určení korespondencí. Pro minimální vzdálenost středů významných bodů v souřadném systému světa s_{min} a maximální změnu polohy průmětů středů významných bodů v rámci jednoho snímku Δm_{max} musí platit

$$\Delta m_{max} < \frac{1}{2} s_{min} \tag{8.1}$$

Pokud bude tato podmínka splněna, korespondence budou **vždy** určeny správně, tato podmínka je tedy postačující.

Vzhledem k tomu, že kamera může kromě translačního pohybu vykonávat i pohyby rotační, může být změna polohy středů průmětů Δm vyvolána různými způsoby, viz Obr. 37. Vzhledem k projekci má každý z pohybů různý vliv na změnu polohy průmětů Δm. Proto nyní ve zkratce popíši omezení pro jednotlivé pohyby, parametry budu uvažovat stejné jako u výše uvedeného příkladu:

- Translační pohyb – jak bylo ukázáno v 4.2.1, odpovídá Δm translačnímu pohybu kamery, Δm_{max} se tedy rovná t_{max}. Pro translační pohyb kamery lze tedy podmínku (8.1) vyjádřit jako
$$\frac{\bar{v}}{f} < \frac{1}{2} s_{min}$$
kde $\bar{v}$ odpovídá průměrné rychlosti v rámci jednoho snímku a f snímkové frekvenci kamery. Pro standardní snímkovou frekvenci $f = 25\,Hz$ a minimální vzdálenost středů $s_{min} = 20\,cm$ je tedy přípustný pohyb kamery v rámci jednoho snímku do $10cm$, což zpětným přepočtem odpovídá průměrné rychlosti $9\,km/h$. Pokud by měla být tato rychlost v rámci jednoho snímku vyvinuta, musela by mít kamera zrychlení $a = 5\,m/s^2$
- Rotace kolem osy z - Δm je v tomto případě přímo úměrné vzdálenosti průmětu od středu promítání. Střed významného bodu $C = [20{,}0{,}80{,}1]^T$ bude promítnut pomocí matice první kamery D_1, která bude opět jednotková, do bodu C_1'. Matice D_2 bude obsahovat rotační podmatici kolem osy z

$$D_2 = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

a promítne bod C do bodu C_2'. Vzhledem k podmínce (8.1) musí platit, že

$$\|C_1' - C_2'\| < \frac{1}{2} s_{min}$$

Z kosinovy věty vyplývá, že

$$\|C_1' - C_2'\| = \sqrt{\|C_1' - \boldsymbol{O}\|^2 + \|C_2' - \boldsymbol{O}\|^2 - 2\cos\gamma\, \|C_1' - \boldsymbol{O}\| \|C_2' - \boldsymbol{O}\|}$$

kde $\boldsymbol{O} = [0,0,1]^T$ značí počátek. Vzhledem k typu pohybu budou zřejmě členy $\|C_1' - \boldsymbol{O}\|$ a $\|C_2' - \boldsymbol{O}\|$ stejné, protože oba průměty budou od středu promítání stejně vzdálené. Pokud je nahradím proměnnou r_P, zjednoduší se celá podmínka na

$$r_P \sqrt{2(1 - \cos\gamma)} < \frac{1}{2} s_{min}$$

Vzdálenost r_P průmětů C_1' od středu promítání lze ale snadno zjistit z promítaného bodu C a jeho složek c_x, c_y, c_z jako

$$r_P = \sqrt{\left(\frac{c_x}{c_z}\right)^2 + \left(\frac{c_y}{c_z}\right)^2}$$

Výsledná podmínka pro rotační pohyb kolem osy z, dávající do vztahu pozici promítaného bodu C a úhel rotace γ, má tvar

$$\sqrt{\left[\left(\frac{c_x}{c_z}\right)^2 + \left(\frac{c_y}{c_z}\right)^2\right] 2(1 - \cos\gamma)} < \frac{1}{2} s_{min}$$

Pro promítaný bod $C = [20,0,80,1]^T$ vyjde řešením této nerovnice pro úhel γ maximální možný úhel rotace kamery γ_{max} v rámci jednoho snímku

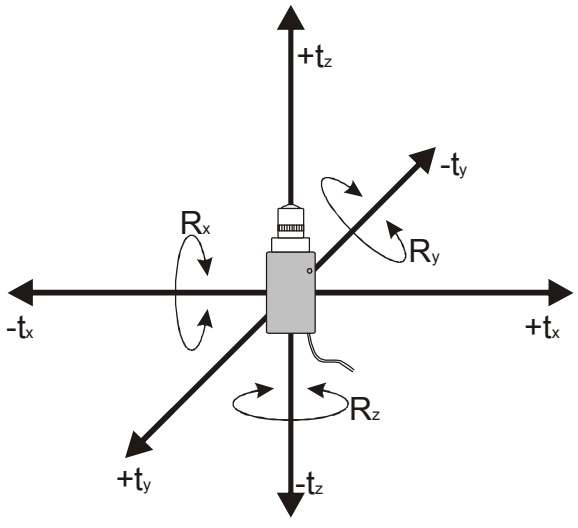
$$\gamma_{max} \doteq 30°$$

což odpovídá úhlové rychlosti $\omega = 13\, rad/s$, tedy zhruba dvěma otáčkám hlavy za sekundu.

- Obdobně lze odvodit omezení pro rotace kolem osy x a y, pro které vychází

$$\alpha = \beta \doteq 7°$$

což odpovídá úhlové rychlosti $\omega = 3\, rad/s$, tedy přibližně jedné otáčce hlavy za sekundu



Obr. 37 Možnosti změny pohybu kamery, t označuje translaci, R označuje rotaci, index značí osu, které se transformace týká

8.1.2. Rekonstrukce pohybu

Reálná implementace systému trpí jedním nedostatkem a tím je horší stabilita rekonstrukce náklonu a natočení[22] a také „citlivost“ této rekonstrukce. Citlivostí je myšleno jednak to, že i malá změna těchto hodnot je poměrně výrazně viditelná pro uživatele, z čehož vyplývá zmíněná nízká stabilita, protože pokud je vstupní změna chybou, uživatel jí velmi snadno zaregistruje. Citlivostí je ale myšlen také fakt, že rekonstrukce těchto hodnot je už sama o sobě mnohem víc náchylná na zanesení šumu než rekonstrukce translačního pohybu a odchylky. Oba dva problémy vychází z charakteru pohybu, který je vykonáván při změně náklonu a natočení.

Šum je do rekonstrukce zanášen především už ve fázi snímání, šumem je v tomto případě myšlen hlavně neúmyslný pohyb snímacího zařízení. Ten může mít například tyto příčiny:

- špatné uchycení snímacího zařízení na hlavě – síťka, na které je snímací zařízení upevněno, může z hlavy sklouzávat díky váze kabelů nebo se může na hlavě drobně kývat kvůli malému utažení apod. Těmto nechtěným pohybům se s vyvinutým prototypem hardwaru nelze zcela vyhnout. Je také zřejmé, že díky tvaru síťky, která kopíruje povrch hlavy, budou mít tyto pohyby stejný charakter, jako když uživatel kývá hlavou a mění se tak hodnoty náklonu a natočení.
- stavba lidského těla – díky gravitace je mnohem pravděpodobnější, že nechtěné pohyby budou mít charakter kývání a naklánění hlavy, než že by se hlava otáčela. To se projeví zejména při chůzi, kdy kývání hlavy téměř nelze vyloučit. V určitých situacích popsaných níže může tento fakt působit rušivě.

Pokud se však k tomuto šumu přidá další chyba způsobená segmentací a samotnou rekonstrukcí může dojít k drobnému chvění obrazu promítaného v HMD. Tato vada je však patrná pouze ve scénách, ve kterých je pozorovatel obklopen uzavřeným prostředím a má tak možnost chvění pozorovat na statických plochách. Ve scénách, které zobrazují venkovní otevřené prostředí, se však tato vada neprojeví. Pokud je však chyba větší, může dojít díky špatné stabilitě rekonstrukce tohoto typu pohybu k rozhození celého zobrazení.

Z tohoto důvodu byla do systému zavedena možnost zamykání rekonstrukce určitých typů pohybů, jak bylo popsáno v 4.2.3. Pokud je tedy systém využíván v aplikacích, jejichž náplní je především chůze, je vhodné zamknout rekonstrukci náklonu a natočení, které by mohli působit rušivě a směr pohledu zjišťovat pouze z odchylky, tedy otáčení hlavy doleva a doprava. Pokud však uživatel dodržuje určitá omezení, jako je pomalé a plynulé rozhlížení, má dostatečně upevněnou síťku se snímacím zařízením atd., je možné systém používat i v režimu volného pohledu.

8.2. HARDWAROVÁ OMEZENÍ

Hardwarová omezení byla částečně nastíněna už v předchozí podkapitole, sekci 8.1.2. Před samotným výčtem je ale vhodné podotknout, že většina hardwarových omezení byla způsobena především vybavením, které bylo v době vývoje systému k dispozici především finančně. Jak bylo zmíněno v úvodu, bylo také cílem vytvořit spíše jakýsi proof-of-concept než komerční produkt schopný prodeje. Lze tedy říci, že hardwarová omezení obecně nesouvisí tolik s kvalitou návrhu systému a lze je ze všech typů omezení odstranit nejsnáze.

[22] Kývání hlavou doprava a doleva, dopředu a dozadu. Více viz 4.2.3.

8.2.1. Snímací zařízení

Použité snímací zařízení, popsané v 7.3, je jedna z levnějších webkamer a také jde již o poměrně zastaralý model. Omezení, které to zanáší do systému, je horní hranice snímkové frekvence 25 snímků za sekundu. Vzhledem k tomu, že je výpočet závislý na snímkové frekvenci kamery, její limit bude zároveň i limit systému. Vyšší snímková frekvence by pak ještě dále snížila omezení určení korespondencí popsané v 8.1.1. Dalším omezením kamery je její rozlišení, které je pouze 320×240, což snižuje přesnost při segmentaci snímků. Na druhou stranu je ale díky nízkému rozlišení segmentace méně výpočetně náročná.

8.2.2. HMD

Použitý head mounted display, popsaný v 7.4, už je také poměrně zastaralý. Jeho nevýhodami je zejména špatná ergonomie, malé displeje a jejich nízké rozlišení. To má vliv především na míru vtáhnutí uživatele, ten totiž díky konstrukci stále vidí okolí, navíc špatná ergonomie po čase silně znepříjemní nošení HMD. Také se mi nepodařilo zprovoznit stereoskopický režim zobrazení, který sice HMD nabízí, ovšem vzhledem k nulové podpoře vývojářů nelze najít žádné SDK nebo manuál k tomuto produktu, který by obsahoval ukázky kódu demonstrující správný postup k zprovoznění stereoskopického režimu.

8.3. MOŽNÉ APLIKACE SYSTÉMU

Aplikace, které by mnou navržený systém mohly využívat, by měly těžit především z možnosti chodit po velkém[23] prostoru. Dobrým příkladem mohou být například virtuální procházky po scénách s architektonickými prvky. Těmi může být například interiér domu, který si může zákazník realitní kanceláře skutečně projít a detailně prohlédnout, aniž by musel dům osobně navštívit. Procházka domem pomocí mého systému umožní zákazníkovi mnohem lepší představu o dispozicích a proporcích interiéru, než pouhé video nebo procházka pomocí klávesnice. Nehledě na to, že tento způsob prezentace je i mnohem přitažlivější a zajímavější.

Mezi další oblasti, ve kterých by bylo možné systém využít, lze určitě zařadit zábavní průmysl. Jednoduchou hrou, demonstrující možnosti systému, by mohl být klon dobře známého Atomic Bombermana. V tom se hráč pohybuje po bludišti, pokládá bomby a s jejich pomocí zabíjí nepřátele. Pravidla této hry by zůstala po integraci systému stejná, pouze by hráč po bludišti skutečně chodil. Ve své multiuživatelské variantě by systém mohl sloužit i pro zkoumání sociálních interakcí v rámci VR.

[23] Velikost prostoru pro chůzi záleží pouze na velikosti místnosti, ve které je systém instalován

9. BUDOUCÍ PRÁCE

V systému je mnoho částí, které poskytují prostor pro další možná vylepšení. Hlavními oblastmi, kterými by se mohli zabývat navazující práce, jsou:

- zavedení absolutního pozicování
- navržení hardwarového vybavení umožňujícího přesnější trackování a snazší manipulaci
- vytvoření frameworku, který by umožňoval snadné přidávání nových virtuálních scén
- integrace v rámci síťového systému pro více uživatelů

Každé z navržených vylepšení nyní stručně popíši včetně nástinu možností řešení. Tyto nástiny jsou pouze okamžitými nápady, které nebyly podrobeny důkladnější analýze a verifikaci správnosti, je nutné je tedy brát jako možné směry, kterými by se hledání řešení mohlo ubírat.

9.1. ZAVEDENÍ ABSOLUTNÍHO POZICOVÁNÍ

Relativní pozicování je zřejmě asi největší současnou nevýhodou systému a uživatelé testující systém ji uváděli jako jediný významnější element rušící dojem z virtuální reality. Problém, který způsobuje relativní pozicování je především ten, že když uživatel vyrazí z jednoho místa, projde nějakou náhodnou trasu a vrátí se na stejné místo, z kterého vyšel, je možné, že pozice a směr pohledu ve virtuální scéně se bude lišit od těchto hodnot ve výchozím místě. To je způsobenu akumulací chyby, která není při relativním pozicování nikterak kompenzována.

Možností jak zavést absolutní pozicování do systému je několik. Jedna z nich by vycházela z matematického modelu kamery popsaného v 0 a mohla by vypadat následovně: Nejprve by byly zjištěny přesné souřadnice viditelných významných bodů v souřadném systému světa při startu systému. To by bylo celkem jednoduše proveditelné pro kalibrovanou kameru, která by byla v poloze kolmé ke stropu a navíc se známou vzdáleností d od kamery ke stropu. Výpočet by probíhal tak, že by se průměty významných bodů převedli ze souřadného systému snímku do souřadného systému světa pomocí inverzních matic figurujících v modelu kameru. Nejprve by byly přenásobením inverzní maticí vnitřních parametrů K^{-1}, která je známá díky kalibrované kameře, transformovány do souřadného systému kamery. Poté by stačilo transformované průměty přenásobit vzdáleností d, čímž by byla získána pozice významných bodů v souřadném systému světa. Toto tvrzení platí, protože díky předpokladu, že kamera směřuje kolmo vzhůru, je matice vnějších parametrů D rovna jednotkové matice, takže i její inverze D^{-1}, kterou se průměty transformují do souřadného systému světa, je rovna jednotkové matici. Vzhledem k tomu, že matice projekce P není čtvercová, nelze vytvořit její inverzi. Jak bylo však uvedeno v 0, efekt jejího přenásobení na promítaný bod je vydělení ostatních složek z souřadnicí. Ta je však v případě, že je kamera v poloze kolmo ke stropu rovna vzdálenosti d.

Poté, co jsou zjištěny souřadnice těchto bodů, jsou uloženy do datové struktury reprezentující mapu významných bodů. Uživatel se pak pohybuje a pohyb je rekonstruován pomocí relativního pozicování popsaného v práci. Pokud se ve snímku objeví nový bod, který ještě není v mapě, je zjištěna opět jeho přesná pozice a je přidán do mapy. Poté, co je do mapy přidáno dostatečné množství bodů, je možné ze známých souřadnic těchto bodů zjistit absolutní pozici kamery v rámci souřadného systému světa z pouhého jednoho snímku, například pomocí [17].

9.2. NAVRŽENÍ LEPŠÍHO HARDWARU

Toto vylepšení se týká zejména systému významných bodů. Ten byl sice v průběhu práce optimalizován, takže vytvoření velké sítě je otázkou pouze nastříhání reflexní fólie a její přichycení na strop, přesto lze v této oblasti navrhnout další vylepšení. Významné body by například mohly být pouze promítány na strop nějakým dostatečně výkonným zařízením vyzařujícím infračervené světlo. Toto zařízení by bylo překryto mřížkou, která by určovala vlastnosti pole významných bodů jako tvar nebo hustota. Předpokladem je, že při vhodné konfiguraci by stačily čtyři takovéto projektory k pokrytí stropu velké místnosti. Výhody takovéhoto řešení jsou zřejmé:

- snadná manipulace – pro vytvoření sítě významných bodů by stačilo pouze vhodně umístit projekční zařízení a spustit ho
- velká kontrola – bylo by velmi snadné kontrolovat tvar, velikost i vzdálenost jednotlivých významných bodů, bylo by taktéž možné jednoduše vytvářet vzory, které by mohli případně posloužit při absolutním pozicování

9.3. ZOBRAZOVACÍ FRAMEWORK

Vzhledem k předpokládaným aplikacím v oblasti architektonických vizualizací a trhu realit by bylo vhodné do softwarové části systému implementovat rozhraní, které umožní snadné přidání nových modelů staveb a virtuálních scén obecně. V tomto směru by bylo vhodné prozkoumat možnosti exportních formátů u softwarových nástrojů používaných k modelování a návrhu interiérů jako je například ArchiCAD. Jedním z možných formátů by mohl být deklarativní jazyk VRML, případně jeho ideový nástupce X3D[18]. Tyto jazyky umožňují popsat kompletně 3D scénu, včetně materiálů, osvětlení atd. a jsou poměrně často mezi exportními formáty profesionálních architektonických softwarů. Dalším formátem pro přenos scény mezi modelovacím programem a mým systémem by mohl být XFile[35] formát společnosti Microsoft, do nějž lze exportovat scénu například v animačním a modelovacím programu Maya, a který lze velmi snadno načíst a renderovat pomocí rozhraní DirectX.

9.4. MULTIUŽIVATELSKÉ PROSTŘEDÍ

Díky návrhu systému, který je koncipován tak, že uživatel je autonomní a zjišťuje si svoji pozici sám, je možné pro jednu síť významných bodů realizovat VR, ve které se bude pohybovat více uživatelů zároveň. Aby však spolu mohli avataři jednotlivých uživatelů ve VR interagovat, bude zřejmě nutné do systému nejprve zavést absolutní pozicování popsané v 9.1, protože určení správné pozice bude v této aplikaci kritické. Při nesprávném určení by do sebe mohli uživatelé v reálném světě vrážet a celý systém by byl velmi obtížně řiditelný. Dále bude nutné navrhnout komunikační protokol, pomocí kterého budou klienti jednotlivých uživatelů komunikovat se serverem udržujícím provoz systému a koordinujícím zobrazení a interakci avatarů.

10. ZÁVĚR

V rámci této práce byl navržen systém virtuální reality, v němž je sledována uživatelova pozice a jeho směr pohledu a na základě toho je aktualizována virtuální scéna, která je prezentována uživateli pomocí takzvaného head mounted displeje. Před samotným návrhem byla provedena rešerše dostupných hardwarových i softwarových nástrojů pro tvorbu virtuální reality, stejně jako kompletních řešení, které souvisely s navrhovaným systémem, a výstupy této rešerše byly zdokumentovány. Matematický model systému byl verifikován pomocí série numerických simulací, pro potřeby ověření vlastností a ladění celého systému byl implementován simulační framework. Po ověření vlastností byl celý systém implementován v podobě knihovny obsahující funkce pro výpočet matematického modelu, aplikace využívající tuto knihovnu a speciálně navrženého hardwaru. Implementace systému byla následně zdokumentována jak z programátorského, tak z uživatelského hlediska. Celý systém byl na závěr testován v reálných podmínkách nezávislým uživatelem.

Systém je schopen v reálném čase rekonstruovat uživatelův pohyb až pro plných šest stupňů s dobrou úrovní realističnosti, což potvrdila zpětná vazba testujících uživatelů. V tomto režimu je však nutné dodržovat určitá omezení stran plynulosti a rychlosti pohybu, jinak se stává rekonstrukce poměrně nestabilní. Pokud je však rekonstrukce omezena pouze na chůzi a otáčení hlavou doleva/doprava, je systém velmi stabilní a poskytuje realistický zážitek. Hlavní nevýhodou systému je jeho relativní pozicování, které snižuje míru realističnosti, protože zpravidla neumožní uživateli vrátit se na stejné místo, z kterého vyšel. Tento problém by měl být řešen v rámci navazujících prací, ideový nástin možného řešení byl navržen již v rámci této práce. Do budoucna by bylo také vhodné vylepšit některé částí hardwarového vybavení pro snazší instalaci systému, vytvořit framework pro jednoduché přidávání nových virtuálních scén nebo integrovat systém v rámci síťové aplikace a zapojit tak do virtuální reality více uživatelů.

Hlavním přínosem práce je především návrh a ověření funkčnosti tohoto návrhu, spíše než samotná implementace systému, která by mohla být dále optimalizována. Práce představuje oproti ostatním podobným systémům jednoduchou a nízkonákladovou variantu. I přesto je však reálné tento systém po navržených úpravách nasadit v praxi, například jako atraktivní variantu virtuální prohlídky, kterou poskytují architektonická studia a realitní kanceláře pro návrhy interiérů.

LITERATURA

[1] SUTHERLAND, Ivan E. The Ultimate Display. *Proceedings of IFIP 65*. 1965, vol. 2, s. 506–508.

[2] DEMENTHON, Daniel F.; DAVIS, Larry S. Model-based object pose in 25 lines of code. *International Journal of Computer Vision*. 1995-06-01, vol. 15, s. 123-141.

[3] ŽÁRA, Jiří, et al. *Moderní počítačová grafika*. Brno : Computer Press, a.s., 2004. 609 s.

[4] ZHANG, Zhengyou. Flexible Camera Calibration By Viewing a Plane From Unknown Orientations. *IICV*. 1999, s. 666-673

[5] QUANQ-TUAN, Luon. Multiple Image Geometry – A Projective Viewpoint. In CHEN, Ch.; WANG, P.S.P. *Handbook of Pattern Recognition and Computer Vision*. [s.l.] : World Scientific Publishing Co. Pte. Ltd., 2005. s. 71-92.

[6] MA, Yi, et al. *An Invitation to 3-D Vision : From Images To Models*. Berkeley : Springer, 2001. 338 s.

[7] SKALA, Václav. *Světlo, barvy a barevné systémy v počítačové grafice*. Praha : Academia, 1993. 130 s. ISBN 80-200-0463-7.

[8] PRATT, William K. *Digital Image Processing*. New York : John Wiley & Sons, Inc., 2001. 738 s.

[9] OTSU, Nobuyuki. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*. 1979, 9, s. 62-66.

[10] QIFA, Ke; KANADE, Takeo. Transforming Camera Geometry to A Virtual Downward-Looking Camera: Robust Ego-Motion Estimation and Ground-Layer Detection. *Computer Vision and Pattern Recognition*. 2003, vol. 1, s. 390-397.

[11] WOELK, Felix; KOCH, Reinhard. Robust Monocular Detection of Independent Motion by a Moving Observer. *Lecture Notes in Computer Science*. 2007, vol. 3417, s. 209-222.

[12] HARTLEY, Richard I. In Defense of the Eight-Point Algorithm. *IEEE Transactions On Pattern Analysis and Machine Intelligence*. 1997, vol. 19, s. 580-593.

[13] HARTLEY, Richard I.; ZISSERMAN, Andrew. *Multiple View Geometry in Computer Vision : Second Edition*. Cambridge : Cambridge University Press, 2004. 672 s.

[14] DUBROFSKY, Elan. *Homography Estimation*. Vancouver, 2009. 32 s. Diplomová práce. The University Of British Columbia.

[15] ZULLIANI, Marco. *RANSAC for Dummies*[online]. 2009[cit. 2011-05-15]. Dostupné z WWW:< http://vision.ece.ucsb.edu/~zuliani/Research/RANSAC/docs/RANSAC4Dummies.pdf >.

[16] SHOEMAKE, Ken. *Graphic Gems IV*. San Diego : Academic Press Professional, Inc., 1994. Euler angle conversion, s. 575.

[17] BÉNALLAL, Mohamed; MEUNIER, Jean. *Computational Science and Its Applications — ICCSA 2003*. Berlin : Springer Berlin / Heidelberg, 2003. A Simple Algorithm for Object Location from a Single Image without Camera Calibration, s. 969-976.

[18] ŽÁRA, Jiří, et al. *Moderní počítačová grafika*. Brno : Computer Press, a.s., 2004. Formáty VRML a X3D, s. 530-531.

[19] TICHOTA, Martin. *Interaktivní zobrazení obrazu z pohledu pozorovatele*. Plzeň, 2008. 42 s. Bakalářská práce. Západočeská univerzita v Plzni.

[20] MIKŠOVIC, Michal. *Displej s pohyblivým pozorovatelem*. Plzeň, 2008. 82 s. Bakalářská práce. Západočeská univerzita v Plzni.

[21] TICHOTA, Martin. *Krajinka*. Plzeň, 2009. 10 s. Semestrální práce. Západočeská univerzita v Plzni.

[22] *I-O Display Systems* [online]. 2009 [cit. 2011-05-14]. I-glasses video 3D Pro: Advanced Eyewear Headset. Dostupné z WWW: <http://www.i-glassesstore.com/ig-hrvpro.html>.

[23] *CyberMind Interactive Nederland* [online]. 2009 [cit. 2011-05-14]. Visette45 SXGA 3D. Dostupné z WWW: <http://www.cybermindnl.com/index.php?Itemid=1&flypage=shop.flypage&option=com_virtuemart&page=shop.product_details&product_id=50>.

[24] *WiiBrew* [online]. 2006 [cit. 2011-05-14]. Dostupné z WWW: <http://wiibrew.org/wiki/Main_Page>.

[25] *Freetrack* [online]. 21/06/2007 [cit. 2011-05-14]. Dostupné z WWW: <http://www.free-track.net/english/>.

[26] *Lehrstuhl für Angewandte Mechanik* [online]. 2009 [cit. 2011-05-15]. Dostupné z WWW: <http://www.amm.mw.tum.de/>.

[27] *Virtusphere* [online]. c2004 [cit. 2011-05-15]. Dostupné z WWW: <http://www.virtusphere.com/index.html>.

[28] *Filters* [online]. 2009 [cit. 2011-05-15]. Dostupné z WWW: <http://filters.sourceforge.net/>.

[29] *OpenCV 2.0 C Reference* [online]. c2009 [cit. 2011-05-15]. Camera Calibration and 3D Reconstruction. Dostupné z WWW: <http://opencv.willowgarage.com/documentation/camera_calibration_and_3d_reconstruction.html>.

[30] *MathWorks* [online]. c1994 [cit. 2011-05-15]. Matlab - The Language Of Technical Computing. Dostupné z WWW: <http://www.mathworks.com/products/matlab/>.

[31] *MSDN* [online]. c2011 [cit. 2011-05-15]. DirectShow. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/dd375454%28v=vs.85%29.aspx>.

[32] *EmguCV* [online]. 2009 [cit. 2011-05-15]. Dostupné z WWW: <http://www.emgu.com/wiki/index.php/Main_Page>. >.

[33] *Math.NET: Iridium* [online]. 2008 [cit. 2011-05-15]. Dostupné z WWW: <http://www.mathdotnet.com/Iridium.aspx>.

[34] *MSDN* [online]. c2011 [cit. 2011-05-15]. Threading Tutorial (C#). Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/aa645740%28v=vs.71%29.aspx>.

[35] *MSDN* [online]. c2011 [cit. 2011-05-15]. X File Format Reference. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/bb173014%28v=vs.85%29.aspx>.

[36] *ImmersiveTech* [online]. c2011 [cit. 2011-05-15]. Dostupné z WWW: <http://www.immersivetech.org/>.

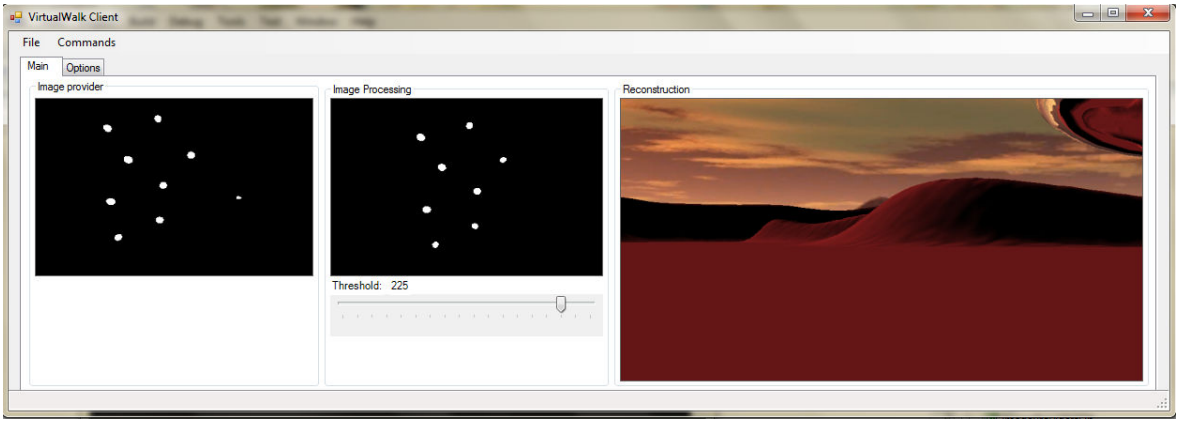
PŘÍLOHA: UŽIVATELSKÁ PŘÍRUČKA

V této příloze bude popsána jednak uživatelská příručka k softwaru, který je součástí celého systému, dále budou uvedeny podmínky nutné pro překlad a instalaci softwaru a také budou uvedena doporučení ohledně užívání hardwarového vybavení.

1. OVLÁDÁNÍ APLIKACE

Aplikaci lze spustit pomocí souboru VirtualWalkClient.exe. Po spuštění se objeví formulář, který slouží k ovládání celého systému a má několik jednoduchých ovládacích prvků. Sestává ze dvou záložek a panelu menu. První záložka, takzvaná hlavní, slouží ke kontrole běhu aplikace a její náhled lze vidět na Obr 1. Záložka obsahuje tyto ovládací prvky:

- Image Provider panel – zobrazuje výstup ze zařízení poskytujícího snímky
- Image Processing panel – obsahuje dva ovládací prvky
  - Threshold posuvník – pomocí něj se nastavuje práh, kterým je vstupní snímek segmentován
  - Processed Image panel – zobrazuje snímek po předzpracování systémem, zde lze kontrolovat správné nastavení prahu
- Reconstruction panel – zobrazuje náhled scény s rekonstrukcí pohybu, tedy to, co uživatel vidí v HMD



Obr. 1 Náhled hlavní záložky

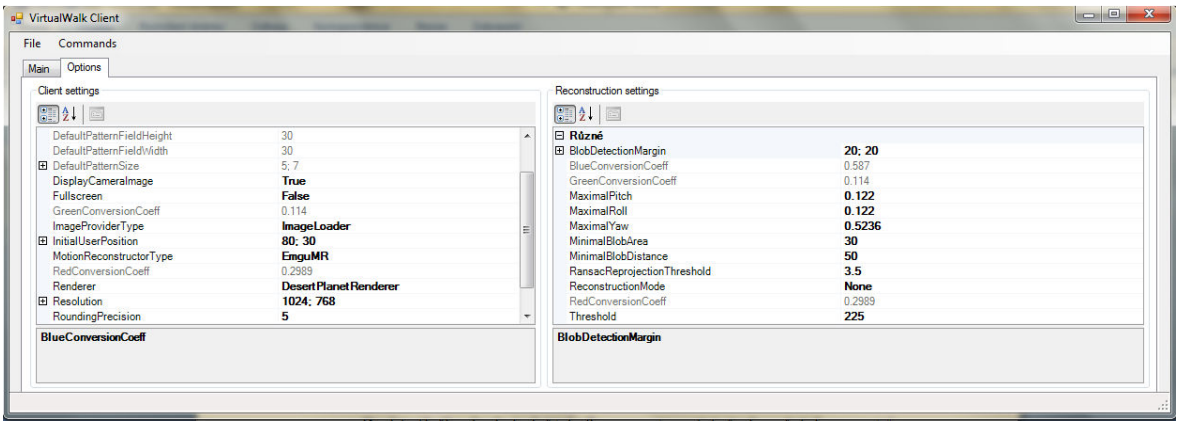
Druhá záložka, viz Obr 2, obsahuje dvě tabulky pro nastavování všech možných parametrů systému, tabulka vlevo slouží pro nastavení parametrů klienta, tabulka vpravo nastavuje parametry pro samotnou rekonstrukci. Parametry pro nastavení klienta:

- Fullscreen – nastavení, zda má renderovaná scéna běžet v okně nebo v režimu přes celou obrazovku
- ImageProviderType – výběr zařízení poskytujícího snímky, na výběr je webkamera nebo objekt, který načítá snímky z disku. Tyto snímky je nutné umístit do adresáře ImageSequence.
- InitialUserPosition – pozice ve virtuální scéně, na které se uživatel objeví po startu systému
- Renderer – výběr zobrazené virtuální scény
- Resolution – nastavení rozlišení zobrazované scény

Parametry pro nastavení rekonstrukce:

- MaximalYaw, MaximalPitch, MaximalRoll – nastavení maximálních přípustných hodnot pro jednotlivé rotace v rámci rekonstrukce

- MinimalBlobArea – nastavení minimální plochy v pixelech, kterou musí mít průmět významného bodu ve snímku. Závisí na fyzické velikosti významných bodů a jejich vzdálenosti od kamery
- MinimalBlobDistance – vzdálenost mezi dvěma nejbližšími významnými body na stropě v centimetrech.
- RansacReprojectionThreshold – nastavení prahu pro metodu RANSAC. Pokud rekonstrukce nefunguje korektně, má smysl měnit toto nastavení v rozmezí 1-10.
- ReconstructionMode – možnost zamknutí rekonstrukce některých typů pohybu, na výběr jsou
    - FullMotion – rekonstrukce všech pohybů
    - TranslationAndYaw – rekonstrukce pouze odchylky a translace
    - TranslationOnly – rekonstrukce pouze translace
    - None – není rekonstruován žádný pohyb
- Threshold – nastavení prahu pro segmentaci vstupních snímků, má stejný efekt jako použití posuvníku v hlavní záložce



Obr A.2 Náhled záložky s nastaveními

Kromě záložek obsahuje formulář také menu, pomocí nějž je celý systém ovládán. V kategorii Commands jsou tyto příkazy:

- Start image provider – spouští objekt poskytující snímky, z pravidla webkameru. Lze spustit též klávesou zkratkou **F1**.
- Stop image provider – zastaví objekt poskytující snímky. Lze zastavit též klávesovou zkratkou **Alt+F1**.
- Start reconstruction – spustí zařízení poskytující snímky a také spustí provádění výpočtu rekonstrukce. Zároveň s tím se otevře další formulář, ve kterém se renderuje scéna s rekonstrukcí pohybu. Tento formulář by měl vidět uživatel v HMD. Lze spustit též klávesovou zkratkou **F5**. Pro přepínání režimu běhu v okně a na celé obrazovce u formuláře zobrazujícím scénu slouží klávesová zkratka **F8**.
- Stop reconstruction – zastaví celý systém. Lze zastavit též klávesovou zkratkou **Alt+F5**.
- Reset user position – nastaví pozici uživatele na výchozí, která je uvedena v tabulce pro nastavení parametrů klienta a směr pohledu uvede do výchozího stavu. Tuto volbu použijte, pokud dojde k chybě v rekonstrukci. Lze provést též klávesovou zkratkou **Ctrl+R**.

2. PŘEKLAD A SPUŠTĚNÍ APLIKACE

Zdrojové kódy aplikace jsou součástí CD, stejně jako soubory umožňující překlad pomocí Microsoft Visual Studia 2008. Pro překlad je nutné mít na cílovém počítači nainstalované tyto komponenty:

- Microsoft DirectX SDK (testováno s verzí Summer 2004)
- SlimDX SDK (testováno s verzí June 2010)
- Microsoft SDK – obsahuje především DirectShow (testováno s verzí 6.0A)
- EmguCV – wrapper pro knihovnu OpenCV (testováno s verzí. 2.1)
- Math.NET Iridium – knihovna pro numerické výpočty (testováno s verzí 2008)
- FiltersDll – knihovna pro zpracování obrázků (testováno s verzí 1.0)

Pro všechny tyto komponenty je nutné mít správně nastavený cesty ve Visual Studiu, poté je možné provést překlad.

Pro spuštění aplikace je nutné, aby na cílovém počítači byly nainstalovány tyto komponenty:

- DirectX 9.0c
- SlimDX
- .NET Framework 3.5

3. HARDWAROVÉ VYBAVENÍ

Aby mohl být systém využíván, je nutné nejprve vytvořit síť z významných bodů na stropě, její velikost pak určuje maximální prostor, po kterém se uživatel může pohybovat. Body by měly mít mezi sebou minimálně 20ti centimetrové mezery a není vhodné z nich vytvářet pravidelnou mřížku, je lepší rozmísťovat je náhodně. Je třeba dát pozor na to, aby v rámci jednoho snímku žádné tři z viditelných bodů nebyly v jedné přímce. Poté je nutné připojit HMD a webkameru k počítači, doporučuje se použít prodlužovací kabely, aby byl zvětšen akční rádius uživatele. Také je nutné připojit napájení k přídavnému infračervenému osvětlení, zdroj musí mít 9V napětí na výstupu spolu se 400mA. Nyní je možné nasadit uživatelovi na hlavu nejprve HMD a poté síťku s webkamerou. Obrázek Obr.3 ukazuje plně vybaveného uživatele.

Obr. 3 Uživatel s nasazeným HMD a síťkou se snímacím zařízením a přídavným IR osvětlením