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

Bakalářská práce

Interaktivní zobrazení obrazu z pohledu pozorovatele

Plzeň, 2008 Martin Tichota

ABSTRAKT

The name of my bachelor thesis is Interactive image visualization from viewer's sight. The main goal of this work is to transfer viewer's translate motion along with his view angle on the output device. Part of the work is use of specialized device for view angle measuring and basic detection of viewer's position from visual information. Solution can be divided into three separate areas. First is finding viewer's position in XZ plane, allowing me to track his translate motion. Second area cover's seeking the view angle. The third is interaction of the previous two areas and also displaying the result. The result could be used for promotional purposes, as well as a basis for further research in this area.

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

V Plzni dne 12. 5. 2008 Martin Tichota .....

OBSAH

1. ÚVOD

Tématem této práce je nalezení způsobu, jak interaktivně přenést translační pohyb pozorovatele spolu s natočením jeho pohledu na výstupní zařízení pomocí počítače. Hlavním úkolem je tedy nalezení uživatelovi pozice v rovinně *XZ*, změření jeho pohledového úhlu a vykreslení scény v souladu se získanými daty. Pro tyto účely jsem měl využít speciální zařízení pro odměřování pohledového úhlu a webovou kameru pro analýzu visuální informace. Jako výstupní zařízení byly nakonec použity brýle s dvěma LCD mini-obrazovkami. Celá práce v sobě tak spojuje řešení z oblastí, jako jsou počítačová grafika nebo počítačové vidění.

Analýza zadaného problému vedla na komplexní řešení. Je to dáno tím, že celý problém vyžadoval použití mnoha různých technologií a zařízení. Jejich vzájemná kooperace nebyla triviální, protože každé zařízení vyžadovalo specifický přístup k jeho správě a interpretaci výsledků. Taktéž bylo nutné vyrobit pomůcky umožňující realizaci celé práce. Řešení lze rozdělit na jednotlivé celky, odpovídající vždy jednomu zařízení, případně technologiím tomuto zařízení příbuzným. Toto rozdělení pak charakterizuje strukturu celé mojí práce.

Kapitola 2 popisuje některé matematické konstrukce použité při vytváření práce. V kapitole 3 uvádím programové vybavení, které jsem využil při správě jednotlivých zařízení a implementaci aplikace V kapitole 4 popisuji jednotlivá zařízení uvedená v zadání. Praktická část je popsána v kapitole 5, detailní rozdělení uvádím v jejím úvodu. V jednotlivých sekcích vždy nejprve popisuji způsob, jakým s daným zařízením pracuji, poté jak využívám jeho výstupu a jak tento výstup přispívá k celkovému výsledku. V této kapitole je také popsána demonstrační aplikace využitá k ověření mého řešení. Na konec této kapitoly uvádím některá omezení související s mým řešením.

Důvodem pro zadání této práce bylo ukázat, jakým způsobem lze přenést uživatelův fyzický pohyb z reálného světa do virtuálního. Výsledek by šel využít například k propagačním účelům naší školy. Také by moje práce mohla posloužit jako odrazový můstek při další vědecké činnosti v této oblasti.

2. UŽITÁ MATEMATIKA

V této části popíši některé použité matematické konstrukce, související hlavně s oblastí zobrazení (viz 5.3) a kalibrací snímacího systému (viz 5.1.3).

2.1. Klasické matematické konstrukce

2.1.1. Skalární součin

Skalární součin[1] udává vztah dvou vektorů. Mějme dva vektory **u** = (*u0*, *u1*, *u2*) a **v**= (*v0*, *v1*, *v2*), jejich skalárním součinem rozumíme číslo

$$(\boldsymbol{u}, \boldsymbol{v}) = u0v0 + u1v1 + u2v2.$$

Skalární součin využívám v souvislosti se změnou bázových vektorů.

2.1.2. Změna báze vektorového prostoru

Změna báze vektorového prostoru je proces, při němž jsou bázové vektory vektorového prostoru změněny. Báze $\mathbb{B}$ je množina vektorů takových, že jsou navzájem lineárně nezávislé a všechny ostatní vektory daného vektorového prostoru jsou pak lineární kombinací těchto bázových. Koeficienty těchto lineárních kombinací jsou souřadnice jednotlivých vektorů prostoru v bázi $\mathbb{B}$. Ve své práci využívám změnu báze vektorového prostor $\mathbb{R}^2$, proto se v dalším popisu omezím pouze na vektory z tohoto prostoru.

Mějme bod *A*, spojnice počátku a bodu *A* tvoří vektor $\boldsymbol{a}$. Souřadnice tohoto bodu i vektoru jsou vztažené k bázi $\mathbb{B}_1 = \{(1,0), (0,1)\}$. Dále mějme dva na sebe kolmé vektory tvořící bázi $\mathbb{B}_2 = \{(1,1), (1,-1)\}$, vektory tvořící tuto bázi označme $\boldsymbol{b}_1^2$ a $\boldsymbol{b}_2^2$. Pro zjištění souřadnic vektoru $\boldsymbol{a}$ a tudíž i bodu *A* v bázi $\mathbb{B}_2$ provedeme následující kroky

$$\boldsymbol{a'}_x = (\boldsymbol{b}_1^2, \boldsymbol{a}), \boldsymbol{a'}_y = (\boldsymbol{b}_2^2, \boldsymbol{a}),$$

Kde $\boldsymbol{a'}$ je vektor $\boldsymbol{a}$ s novými souřadnicemi a $(\boldsymbol{b}_1^2, \boldsymbol{a})$, respektive $(\boldsymbol{b}_2^2, \boldsymbol{a})$ je skalární součin tak, jak jsem ho popsal v části 2.1.1.

2.2. Aplikovaná matematika

2.2.1. Transformace

Protože má moje práce jako hlavní výstup zobrazení scény pomocí počítače, nevyhnu se použití transformací. Transformace v počítačové grafice jsou matematické operace, pomocí nichž je měněna pozice bodu, případně bodů, z nichž jsou poskládány složitější objekty, tak, aby bylo dosaženo požadovaného výsledného obrazu. Jedno z dělení transformací je na lineární a nelineární. Mezi nelineární patří složitější úpravy obrazu jako

je například morfing[1], v dalším textu se však zaměřím pouze na lineární transformace. Podklady pro tuto sekci jsem čerpal z [2] a [3].

Mezi základní transformace bodu patří translace a rotace, obvykle však potřebujeme transformovat složitější grafická primitiva jako je úsečka, obdélník nebo elipsa a objekty z nich složené. K translaci a rotaci se tak přidá zrcadlení, změna měřítka nebo zkosení. Speciální transformací je pak projekce, které převede objekt z n-rozměrného prostoru do (n-1)-rozměrného prostoru. Ta se využívá především při přenosu trojrozměrných objektů na dvourozměrný monitor počítače.

Transformace lze samozřejmě provádět v běžném kartézském souřadném systém, zde se však naráží na určité komplikace a nejednotnost, nehomogenitu. Z důvodu zjednodušení zápisu se proto pro využití transformací rozšiřuje běžný souřadný systém o homogenní souřadnici, což umožňuje pracovat se všemi body prostoru, včetně těch v nekonečnu, stejně. Díky tomu lze všechny transformace zapsat pomocí jedné matice a skládání transformací pomocí násobení matic. Bod P z prostoru $\mathbb{R}^2$ rozšířený o homogenní souřadnici tak má tvar

$$P = \begin{bmatrix} x \\ y \\ w \end{bmatrix},$$

kde w je ona homogenní souřadnice. Kartézské souřadnice $[X, Y]$ bodu P jsou

$$X = \frac{x}{w}, Y = \frac{y}{w},$$

pro $w \neq 0$. Pokud by bylo $w = 0$, bod P by byl takzvaný nevlastní bod roviny, což si lze představit právě jako bod v nekonečnu. Transformací, vyjádřenou maticí $\boldsymbol{T}$, bodu P bychom tedy získali bod

$$P' = \begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \boldsymbol{T} \times P = \boldsymbol{T} \times \begin{bmatrix} x \\ y \\ w \end{bmatrix}.$$

Při vysvětlování jednotlivých transformací se budu držet tohoto značení.

Translace je posunutí bodu o nějakou vzdálenost určitým směrem. Matice této transformace má tvar

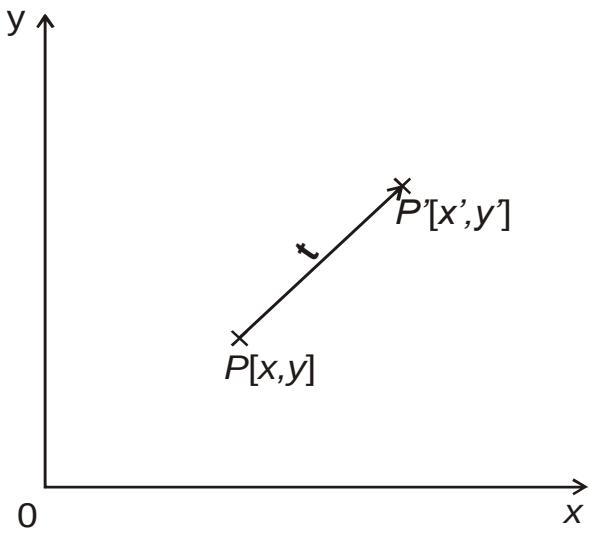
$$\boldsymbol{T} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

pro dvourozměrný prostor,

$$\boldsymbol{T} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

[1] Plynulá změna jednoho obrázku v jiný

pro třírozměrný prostor. Směr a velikost posunutí jsou určené vektorem $\boldsymbol{t} = (t_x, t_y)$, respektive $\boldsymbol{t} = (t_x, t_y, t_z)$.



Obr. 1 Translace bodu *P* o vektor $\boldsymbol{t}$

Rotace je otočení bodu kolem jiného bodu (v rovině) nebo kolem osy (v prostoru) o úhel φ. Matice otočení bodu kolem počátku souřadného systému 0 má tvar

$$\boldsymbol{R} = \begin{pmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

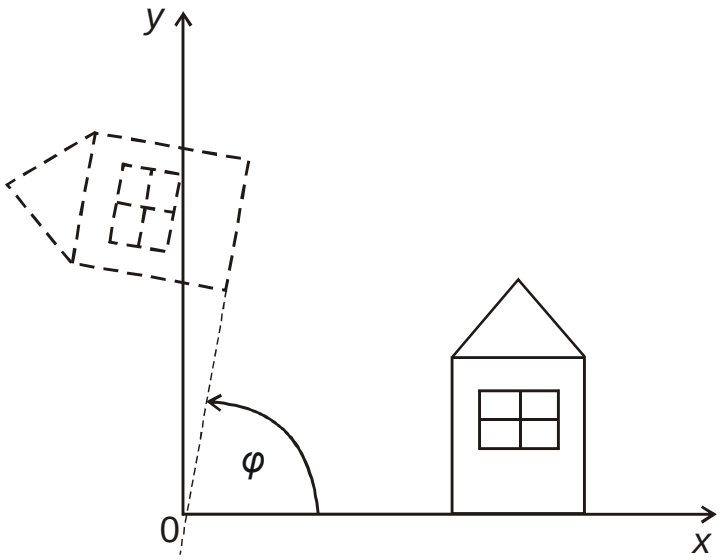
Otočením o kladný úhel φ rozumíme rotaci proti směru hodinových ručiček. Pokud bychom chtěli rotovat kolem obecného bodu, je situace složitější. Museli bychom použít tři transformace. Translaci pro posunutí tohoto bodu do počátku, rotaci pro otočení kolem počátku a poté posunutí zpět do obecného bodu.

Matice otočení kolem jednotlivých souřadných os v prostoru mají tvar

$$\boldsymbol{R_x} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi & 0 \\ 0 & \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

$$\boldsymbol{R_y} = \begin{pmatrix} \cos\varphi & 0 & \sin\varphi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\varphi & 0 & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

$$\boldsymbol{R_z} = \begin{pmatrix} \cos\varphi & -\sin\varphi & 0 & 0 \\ \sin\varphi & \cos\varphi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$



Obr. 2 Rotace kolem počátku o úhel φ

Změna měřítka neboli scaling je transformace provádějící zmenšení nebo zvětšení ve směru souřadných os. Velikost změny měřítka v jednotlivých osách je určena vektorem $\boldsymbol{s} = (s_x, s_y)$, respektive $\boldsymbol{s} = (s_x, s_y, s_z)$. Maticový zápis této transformace ve dvourozměrném prostoru vypadá následovně

$$\boldsymbol{S} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

v třírozměrném pak

$$\boldsymbol{S} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Projekce neboli promítání se, jak už sem uvedl na začátku této podsekce, využívá pro převedení objektů, s kterými interně pracujeme jako s trojrozměrnými, na dvourozměrný monitor počítače. Projekci lze opět zapsat pomocí jedné jediné matice a to díky využití homogenních souřadnic. Projekce se dělí na rovnoběžnou a perspektivní, přičemž druhá jmenovaná se používá častěji.

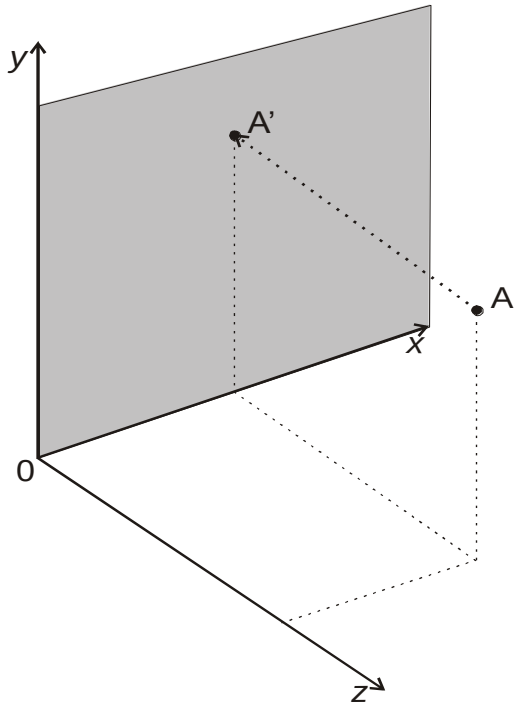
Při **rovnoběžném promítání** jednoduše zapomeneme jednu ze souřadnic, podle toho, do které roviny promítáme. Rovnoběžné se toto promítání jmenuje z toho důvodu, že úsečky, které byly před promítáním rovnoběžné, zůstanou rovnoběžné i po promítnutí. Obrázek Obr. 3 ukazuje rovnoběžné promítnutí bodu A do roviny XY. Maticově lze tuto transformaci zapsat jako

$$\boldsymbol{P_{Rz}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

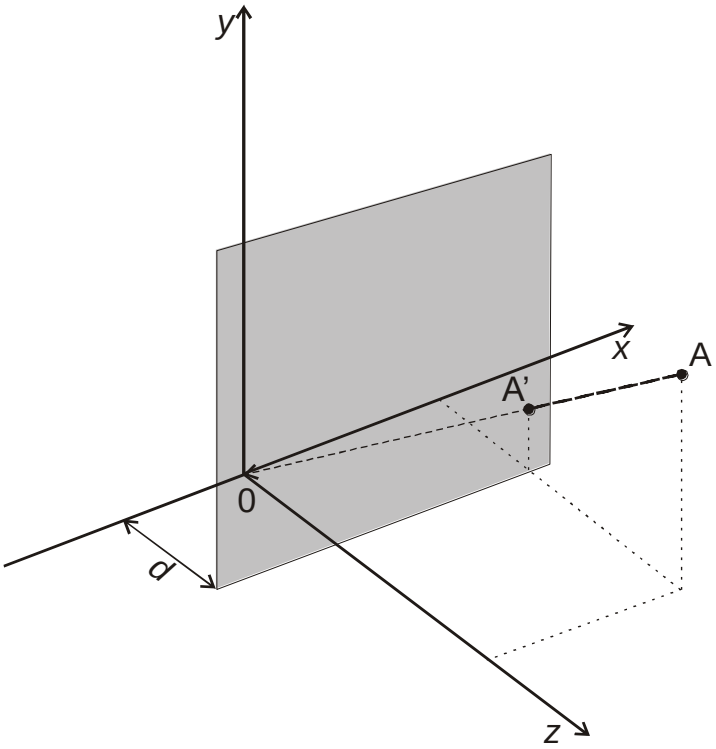
Pokud bychom chtěli promítat do nějaké obecné roviny, je nutné složit více transformací tak, aby osa promítání splynula s osou z.

Při **perspektivní projekci** je pozorovatel umístěn do nějaké konečné vzdálenosti a promítaný bod je poté nalezen v průsečíku průmětny, což je rovina, do které promítáme a spojnice bodu s pozorovatelem. Průmětna se obvykle umísťuje do pevné vzdálenosti d od počátku souřadného, rovnoběžně s rovinnou XY a pozorovatel do počátku souřadnic. Po promítnutí se přímky, které byly původně rovnoběžné s osou z, stanou různoběžkami s jedním průsečíkem. Tento průsečík se označuje jako úběžník. Odtud je vidět, že úběžník osy z, který byl původně v nekonečnu, byl posunut do konečné vzdálenosti. Díky tomu se tato projekci označuje jako jednoúběžníková(viz Obr. 4). Matice této projekce je

$$\boldsymbol{P_P} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & {}^{1}/_{d} & 0 \end{pmatrix}.$$



Obr. 3 Rovnoběžné promítnutí bodu A do roviny XY



Obr. 4 Jednoúběžníkové perspektivní promítnutí bodu A s rovinou průmětny $z = d$

2.2.2. Prahování

Prahování [4] je proces využívaný v počítačové grafice k oddělení relevantních objektů rastrového obrazu od pozadí. Rastrový obraz je rozdělen na jednotlivé pixely, kterým je přidělena nějaká hodnota. V mém případě to byla hodnota jasu jednotlivých pixelů. Před začátkem algoritmu je ještě určen nějaký pevný práh.

Samotný algoritmus vypadá tak, že je celý obraz procházen pixel po pixelu. Hodnota pixelu je vždy porovnána s hodnotou práhu. Pokud je hodnota pixelu menší než hodnota práhu, je tomuto pixelu přiřazena nula. Pokud je vyšší nebo rovna, je mu přiřazena nějaká nenulová hodnota, obvykle jedna. Po průchodu celého obrazu je tedy každý pixel označen jako relevantní nebo jako součást pozadí.

3. PROGRAMOVÉ PROSTŘEDKY

3.1. Direct x

DirectX je sbírka API[2] , pomocí nichž je programátorovi umožněno pracovat s velkým množstvím technických prostředků zaměřených na multimédia. Byl vyvinut firmou Microsoft, proto lze použít pouze na jejích platformách (nejčastěji je to operační systém Windows). Více informací lze nalézt například v [5].

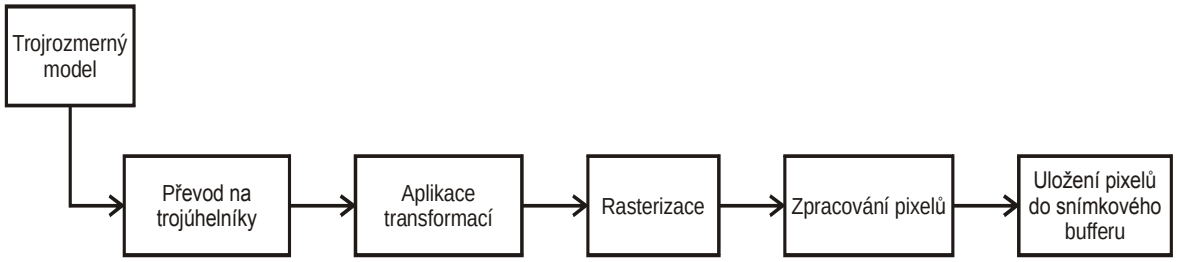
První verze DirectX vznikla současně s operačním systémem Windows 95, což byl nástupce operačního systému MS-DOS. Byl vytvořen z důvodu stability a sjednocení přístupu k jednotlivým zařízením.

Jednotlivá API začínají vždy slovem Direct a pak následuje oblast, ke které poskytují přístup. V mé práci jsem použil Direct3D, DirectInput a DirectShow. Proto se v následujícím textu omezím na popis pouze těchto tří API. Verze DirectX, kterou jsem použil, byla 9.0c.

3.1.1. Direct3D

Direct3D je rozhraní umožňující práci s grafickým hardwarem, především pak jeho využití k zobrazení trojrozměrných modelů. Taktéž umožňuje hardwarovou akceleraci celého procesu zobrazení, případně jeho části. Proces zobrazení je posloupnost algoritmů, které převedou trojrozměrné souřadnice modelu na obrazovku počítače. Jednotlivé kroky tohoto procesu jsou za sebe uspořádány do tzv. pipeline.

Pipeline je obecný optimalizační model pro zpracování dat. Lze si jej zjednodušeně představit jako potrubí, do kterého na jedné straně vkládáme hrubá data a na druhé straně vybíráme zpracovaná. V případě Direct3D odpovídají hrubá data geometrickým souřadnicím modelu, který si přejeme zobrazit. Výstup pipeline je pak obraz modelu na obrazovce počítače. Obr. 5 ukazuje zjednodušené schéma Direct3D 9.0 pipeline.



Obr. 5 Zjednodušená pipeline Direct 3D 9

Na vstupu je 3D model. Tento model je následně rozdělen na trojúhelníky. V další fázi jsou na vrcholy trojúhelníků aplikovány jednotlivé transformace. Vrcholy jsou převedeny ze souřadného systému modelu do souřadného systému světa a kamery a poté je

[2] Z anglického application programming interfaces – aplikační programová rozhraní

na ně aplikována projekce. To vše pomocí maticových transformací tak, jak jsem je popsal v podsekci 2.2.1. V této fázi je také u každého vrcholu zvlášť spočítáno jeho osvětlení. Při rasterizaci jsou nejprve ořezány vrcholy, které jsou mimo zobrazovanou oblast, poté je celý obraz z trojúhelníků převeden na pixely. Barva jednotlivých pixelů je interpolovaná z vrcholů tvořících daný trojúhelník. V další fázi se na jednotlivé pixely aplikují textury, aby mohli být následně uloženy do snímkového bufferu a vykresleny.

Pro podrobnější informace odkazuji na [6], z které jsem čerpal i já.

3.1.2. DirectInput

DirectInput je další rozhraní z kolekce DirectX. Jak jeho název napovídá, zprostředkovává přístup k zařízením, která umožňují uživateli interaktivně ovládat počítač. Hlavním úkolem DirectInput je tedy získávat data z daného vstupního zařízení a předávat je nadřízené aplikaci. Neomezuje se pouze na běžná vstupní zařízení, jako jsou klávesnice a myš, zprostředkovává přístup i k joystickům, volantům a podobným herním ovladačům, včetně podpory funkce force feedback. Umožňuje pracovat i se speciálními zařízeními, jako je například head tracker, který ve své práci používám, a který popisuji v části 4.2. Inicializace těchto vstupních zařízení je však o něco komplikovanější, než je standardní myš a klávesnice, protože nemají v rámci DirectInput předdefinovaný profil a pro správné nastavení je nutná znalost specifikace daného zařízení.

Při práci s DirectInput nejprve musíme nalézt vstupní zařízení, z kterého chceme získávat data. Nalezení zařízení se v terminologii DirectInput nazývá vyjmenování[3]. V podstatě se pomocí určitých omezení hledá vhodné zařízení. Následně se naleznou všechny součásti zařízení, které odpovídají tlačítkům, osám, táhlům a dalším ovládacím prvkům.

Pokud pracujeme s obecným zařízením, které nemá přednastavený profil v rámci DirectInput, je nutné ho nastavit. Nejprve se nastavuje úroveň kooperace, která určuje, jestli bude moci být zařízení sdíleno více aplikacemi systému nebo pouze tou naší. Jako další se musí nastavit formát dat zařízení. Formát dat specifikuje strukturu, která se skládá z jednotlivých *device object*. Určuje velikost této struktury a offsety, na kterých začínají jednotlivé *device object* . Na závěr se nastaví velikost bufferu, do kterého se budou ukládat data ze zařízení, a zařízení se získá[4]. Tím je zajištěno ukládání dat, které lze dále zpracovávat v nadřízené aplikaci.

Pro více informací o DirectInput odkazuji na [7], z které jsem čerpal i já.

[3] Z anglického enumerate
[4] Z anglického acquire

3.1.3. DirectShow

DirectShow je rozhraní umožňující pracovat s multimédii a streamy(proud dat). Jeho hlavní využití je v aplikacích pracujících s videem. Umožňuje přehrávat, zaznamenávat a jinak zpracovávat streamy reprezentující videosekvence nebo zvukové stopy.

Architektura DirectShow je založená na *filtrech*, což jsou samostatné moduly, které vezmou data, které mají na vstupu, zpracují je a pošlou na výstup. Tyto filtry poskládané za sebe pak vytváří pipeline(obdoba viz 3.1.1) aplikace. DirectShow obsahuje množství již předprogramovaných filtrů, stejně tak je ale možné vytvořit si filtry vlastní, uzpůsobené specifickým potřebám aplikace. Po spojení filtrů vznikne takzvaný *filter graph*, který umožňuje pracovat s filtry jako celkem a vytváří tak jádro aplikace.

Hlavním důvodem, proč jsem DirectShow použil ve svém návrhu řešení, byla jeho schopnost zachytit obraz z kamery (viz 4.3) a v pixelové reprezentaci ho předat nadřízené aplikaci. Nevýhodou tohoto rozhraní je, že přístup k jeho metodám jde striktně přes COM objekty. Což je oproti například .NET frameworku[8] programátorsky značně složitější postup. Podrobnější informace o DirectShow lze nálezt v [9].

3.2. Užité programovací jazyky

V této části stručně popíši dva z použitých programovacích jazyků. Tím třetím byl jazyk C++, pro detaily tohoto jazyka pouze odkážu na [10].

3.2.1. C#

C#[11] je moderní objektové orientovaný programovací jazyk. Byl navržen pro maximálně možné využití .NET běhového prostředí. Jako příbuzné lze označit jazyky C++, z kterého si bere hlavně syntaxi, nebo Java, z něhož přebírá některé vlastnosti z oblasti bezpečnosti a zjednodušení. Na jednoduchost je v případě jazyku C# kladen hlavní důraz. Obsahuje proto typovou kontrolu, kontrolu hranic polí nebo automatický garbage collection[5]. Taktéž nedovoluje, narozdíl od C++, přímé přístupy do paměti za pomoci ukazatelů, pouze za předpokladu použití speciální konstrukce. Velký důraz je v případě C# kladen také na přenositelnost kódu.

C# jsem pro svou práci použil hlavně díky jeho jednoduchosti a možnosti využít .NET prostředí.

3.2.2. Managed C++

Managed C++ [12] je rozšíření běžného C++ o určité struktury, tak, aby mohl využívat prostředky .NET prostředí. Slovo managed v názvu znamená, že je program napsaný v kódu tohoto jazyka řízen .NET virtuálním strojem, který funguje jako

[5] Uvolňování paměti, kterou už aplikace dále nepotřebuje

bezpečností prvek a odstiňuje program od procesoru a dalších prvků hardwaru. Managed C++ má tu výhodu, že může komunikovat jak s vysokoúrovňovými jazyky jako je C#, tak s nízkoúrovňovým obyčejným C++. Díky tomu je často využíván jako most mezi těmito jazyky. Stejně tomu bylo i v mém případě, kde jsem Managed C++ použil jako spojku mezi běžným C++, které jsem použil ke správě webové kamery (více v části 5.1.1), a jazykem C#, v kterém je naprogramován zbytek demonstrační aplikace (viz 5.4).

4. POUŽITÝ HARDWARE

4.1. Head-mounted Display

Jak už z názvu vyplývá, je Head-mounted display (HMD) zobrazovací zařízení nošené na hlavě. Toto je hardware, pomocí kterého budu celou práci zobrazovat.

HMD mohou být monokulární (mají obrazovku pouze pro jedno oko) nebo binokulární (mají obrazovky před oběma očima). Obrazovky jsou samozřejmě zmenšené a mohou být klasické CRT, případně z tekutých krystalů.

Výhodou binokulárních HMD je jejich schopnost vytvářet stereoskopický obraz. Člověk vidí stereoskopicky (trojrozměrně)díky mozku, který skládá obrazy z obou očí přes sebe. Protože jsou oči od sebe vzdálené přibližně šest centimetrů, obraz z každého je mírně odlišný. Aby byla možnost vytvořit iluzi 3D obrazu pomocí HMD, musí být i jimi zobrazovaný obraz pro každé oko trochu odlišný. Toho lze dosáhnout vícero způsoby.

Výrobce grafických karet nVidia, dodával ke svým výrobkům speciální stereo ovladače, které s vybranými HMD a aplikacemi vytvářeli stereo efekt. Tyto ovladače však měli jeden zásadní nedostatek a to byla nemožnost jakkoliv kontrolovat výsledný obraz.

Druhou možností je vytvořit aplikaci, která bude renderovat pro každé oko jiný obraz, tak aby byla vytvořena 3D iluze. K tomu je však třeba znát jak přesně HMD s příchozími daty pracují. Obvykle totiž mají pouze jeden VGA vstup, takže nemůžeme posílat data zvlášť do jedné a druhé obrazovky. Brýle pak pracují v takzvaném interleaved režimu, což znamená, že příchozí data přepínají mezi obrazovkami. Režim interleaved může být řádkový (na levé obrazovce se zobrazují liché řádky a na pravé sudé) nebo obrazovkový.

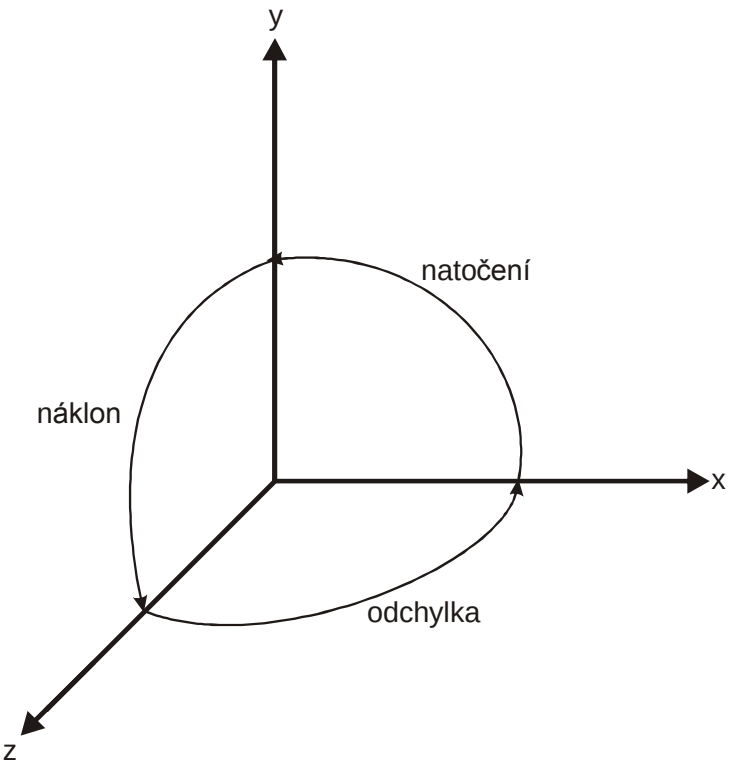
Původně jsem chtěl využít stereoskopického zobrazení i v mé práci. Bohužel hardware, který jsem měl k dispozici, již nebyl výrobcem nadále podporován, takže nebylo možné využít všech jeho možností. HMD, který jsem využil ve své práci k zobrazení, nese název i-Glasses PC/SVGA(viz Obr. 6). Byl vyroben americkou firmou i-O Dislpay Systems. Jako jejich hlavní výhoda je vyzdvihována nízká hmotnost a možnost je plně přizpůsobit tvaru hlavy uživatele. Značnou nevýhodou těchto brýlí, o které se ale nikde nemluví, je téměř nulová podpora pro vývojáře software, kteří by chtěli s těmito brýlemi pracovat. Další výraznou nevýhodou je jejich relativní zastaralost, první i-Glasses PC/SVGA byly vyrobeny někdy v roce 2002 a od té doby se téměř nezměnili.

Obr. 6 i-glasses PC/SVGA

4.2. Head tracker

Head tracker Intertrax[2] je zařízení, snímající pohyb vaší hlavy a umožňující tento pohyb zpracovat pomocí počítače. Skládá se z citlivých gyroskopů, které měří odchylku ve všech třech souřadných osách. Data jsou pak posílána hostitelské aplikaci. Tracker pracuje se třemi termíny pro jednotlivé odchylky:

- Pozitivní odchylka je definována jako rotace hlavy doleva
- Pozitivní náklon je definován jako naklonění hlavy vzhůru
- Pozitivní natočení je definováno jako naklonění hlavy doleva



Obr. 7 Orientace rotací trackeru

Formát dat pro natočení, náklon a odchylku, ve kterém moje aplikace přijímá data od trackeru je 16-ti bitové slovo, kde 16384 odpovídá rotaci o +180° a -16384 rotaci o

−180°. Rozsah vracených čísel je tedy od 0 do -32768 při záporném směru rotace a od 0 do 32768 při kladném směru rotace [13].

Tracker, který mám k dispozici, je od americké firmy Intersense, která se zabývá ve velkém problematikou trackování(sledování) pohybu. Je relativně malý a dá se snadno přichytit head-mounted display. Jediným ovládacím prvkem na něm je tlačítko, kterým se určí současná poloha trackeru jako výchozí (to znamená poloha, ve které uživatel drží hlavu zpříma a dívá se před sebe).

Obr. 8 Intertrax[2] head tracker

4.3. WEBkamera

Webkameru jsem ve své práci využil při zjišťování polohy uživatele. K tomuto účelu by se dala využít v podstatě libovolná kamera, ovšem webová kamera má tu výhodu, že je malá, snadno se připojí k PC a náklady na její pořízení jsou minimální. To je samozřejmě vyváženo horší kvalitou obrazu. Ta je však pro moje potřeby dostačující.

Běžná webkamera obsahuje čočku, snímač obrazu a nějakou podpůrnou elektroniku. Čočka může být fixní nebo pohyblivá, v druhém případě pak může uživatel manuálně ovládat ostření. Snímač obrazu je zařízení, které převede optický obraz na digitální signál. Obvykle sestává z CCD[6] nebo CMOS senzorů. Rozlišení webkamer se v současné době pohybuje od 1.3 do 3.0 megapixelů. Podpůrná elektronika zajišťuje přenos dat ze snímače obrazu do počítače. U webkamer je v naprosté většině případů využit chipset, který obraz do počítače posílá přes USB rozhraní[14].

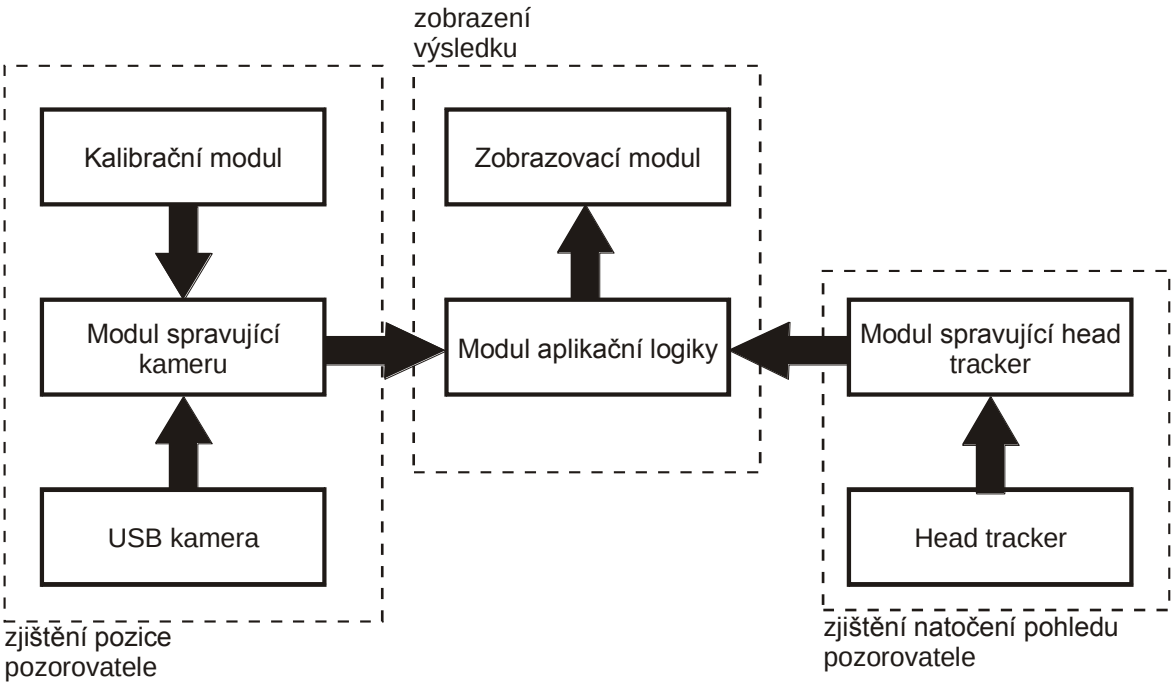
Webkameru, kterou jsem měl k dispozici při vytváření mé práce, byla Webcam Instant od firmy Creative (viz Obr. 9). Jde o standardní webovou kameru, které se k počítači připojuje přes USB port. Snímač obrazu je řešen pomocí CMOS senzorů. Maximální rozlišení obrazu je 352×258 pixelů. Podporuje video formáty I420 a RGB24. Druhý zmíněný využívám i já, z toho důvodu, že je jednoduše využitelný ostatními částmi práce. V tomto formátu je jeden pixel uložen na 24 bitech, kde prvních 8 bitů obsahuje červenou složku, dalších 8 zelenou a posledních 8 modrou složku obrazu. Expozice a vyvážení bílé je u této kamery automatické. Kamera však obsahuje kroužek kolem čočky, kterým se dá manuálně ovládat ostření. Zorné pole kamery je 50 stupňů diagonálně[15].

[6] CCD = charged –coupled device

Obr. 9 Creative Webcam Instant

5. VLASTNÍ PRÁCE

V této části budu popisovat postup, jakým jsem se dobral k řešení, případně prostředky, které jsem k tomu využil. Celé řešení by se dalo rozdělit na následující skupiny: **zjištění pozice pozorovatele, zjištění natočení pohledu pozorovatele, interakce zařízení a technologií - zobrazení výsledku, ověření řešení – demonstrační aplikace**. Každá z těchto oblastí představuje samostatný problém, kterému by mohla být věnována samostatná bakalářská práce. Některé části proto byly zjednodušeny.Rozčlenil jsem jí do jednotlivých modulů, které dohromady vytváří skupiny, tak jak jsem je popsal výše. Obr. 10 demonstruje abstrakci celé práce.



Obr. 10 Rozdělení práce do modulů

5.1. Zjištění pozice pozorovatele

V této části jde o lokalizaci pozice pozorovatele v rovině XZ. Pozici v ose y, tedy uživatelova výška, je brána jako vstup. Uživatel se může pohybovat ve vymezeném prostoru, který je shora snímán kamerou. Hranice vymezeného prostoru jsou dány výškou v jaké je kamera uchycena a FOV[7] kamery. Uživatel má na sobě výrazný referenční bod, pomocí nějž je jednoznačně určena jeho pozice. V každém snímku kamery je poté nutné referenční bod nalézt a tím určit pozici pozorovatele v rovině pohybu XZ.

Se snímkem kamery, což je pravidelný rastr o rozměrech $R_x \times R_y$, pracujeme jako s vlastním souřadným systémem (dále SSK). Jako jednotky tohoto souřadného použijeme pixely[8]. Je ovšem zřejmé, že je obecně odlišný od souřadného systému uživatele (SSU).

[7] Field Of View – zorné pole
[8] Z anglického picture element

Liší se jak rozměrově, protože jeden pixel obecně nebude odpovídat jednomu centimetru, tak v bázových vektorech. Ty by byly shodné pouze v tom případě, kdy by se nám kameru podařilo upevnit na strop tak, aby byly souřadné osy SSK rovnoběžné s osami SSU. Proto bylo nutné do řešení začlenit kalibraci SSK, tak aby byl zobrazený výstup shodný s pohybem uživatele včetně vzdáleností a vznikla tak věrohodná iluze skutečného pohybu.

Celek tedy funguje tak, že se nejprve zkalibruje souřadný systém kamery pomocí sejmutí snímku s kalibračním štítkem (viz 5.1.5), který definuje orientaci SSU a také jednotkovou velikost. Poté se uživatel pohybuje ve vymezeném prostoru a referenční bod, který má připevněn na hlavě, je snímán kamerou. Data z kamery jsou zpracována tak, aby se nalezl referenční bod. Změna jeho pozice je poté převedena odpovídajícím způsobem na změnu zobrazované scény.

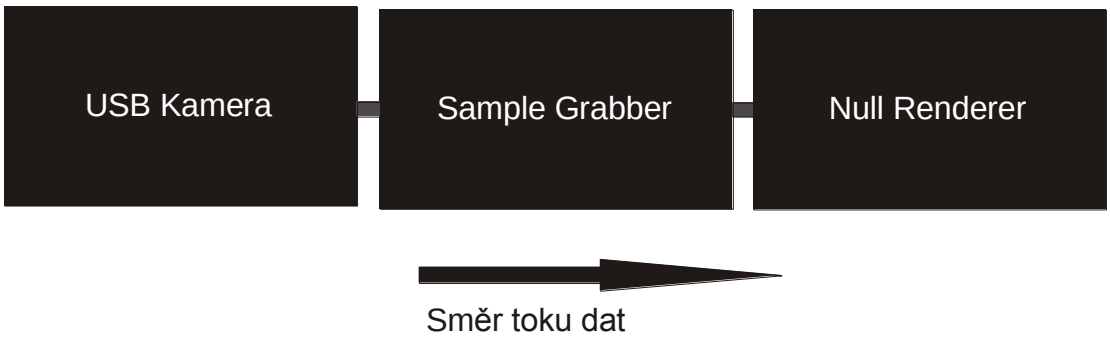
5.1.1. Zpracování výstupu z kamery

K programové správě kamery jsem použil rozhraní DirectShow(viz 3.1.3). Zvolil jsem toto rozhraní z toho důvodu, že je nejpoužívanější při zpracovávání výstupu z kamery. V souvislosti s DirectShow jsem použil části kódu z bakalářské práce jiného studenta[16].

Pro získání dat z kamery jsem provedl následující kroky:

1. Vytvořil jsem instanci Filter Graph Manageru
2. Použil Filter Graph Manager k sestavení grafu

Detailní postup včetně zdrojového kódu zde neuvádím, je popsán v [9], z kterého jsem čerpal i já. V tomto případě bylo nutné sestavit Filter Graph ze tří filtrů. Jednoduché schéma viz Obr. 11.



Obr. 11 Filter Gaph použitý v mojí aplikaci

USB kamera je v rámci DirectShow reprezentována rozhraním *IBaseFilter*. Jde o nejzákladnější typ filtru a poskytuje pouze přístup k datům s kamery. *Sample Grabber* je filtr, který umožňuje příchozí snímky zachytávat a ukládat je do externího bufferu. *NullRenderer* je pak pouze ukončení filter graphu a jeho jedinou funkcí je mazat příchozí snímky, protože ty už nadále nejsou potřeba.

Po spuštění filter graphu tedy *IBaseFilter* začne brát jednotlivé snímky z USB kamery a posílat je dalšímu filtru. Tím je *Sample Grabber*, který snímky ukládá do externího bufferu. Tento buffer pak sdílí modul kamery, který do něj ukládá data a modul spravující kameru, který z něj data vybírá a dále je zpracovává. Abych nemusel složitě řešit synchronizaci mezi oběma moduly, využil jsem ještě jednu možnost *Sample Grabberu*, a tou je uložení pouze jednoho snímku. Aplikační logika si pak vždy, když dokončí výpočty na předchozím snímku, zavolá o další snímek, to má za následek předání řízení kamerovému modulu. Ten spustí filter graph, data z kamery přijdou do SampleGrabberu, který uloží do bufferu jeden snímek a zastaví filter graph. Řízení je opět předáno aplikační logice, ta vybere z bufferu aktuální snímek, zpracuje ho a celý postup se opakuje, dokud uživatel neukončí program.

5.1.2. Nalezení referenčního bodu

V této podsekci se zaměřím pouze na postup, kterým projde jeden snímek od chvíle, kdy je uložen do bufferu kamerovým modulem, do té doby než je nalezena uživatelova pozice. Cílem následujících postupů je nalezení středu referenčního bodu, který má na sobě uživatel uchycen[9]. Ten se v obrazu z kamery jeví jako přibližně kruhová oblast s nejvyšší intenzitou jasu. Tato oblast je také víceméně pravidelná a symetrická. Díky tomu mohu následující vcelku jednoduchý postup detekce použít.

Pro vysvětlení algoritmů, které dále využívám, zavedu následující definice. Mějme obraz *I*, který odpovídá jednomu snímku z kamery. Ten je složen z pixelů $i_{x,y}$, kde

$$x \in \langle 0, R_x - 1\rangle, y \in \langle 0, R_y - 1\rangle,$$

$$x, y \in \mathbb{Z},$$

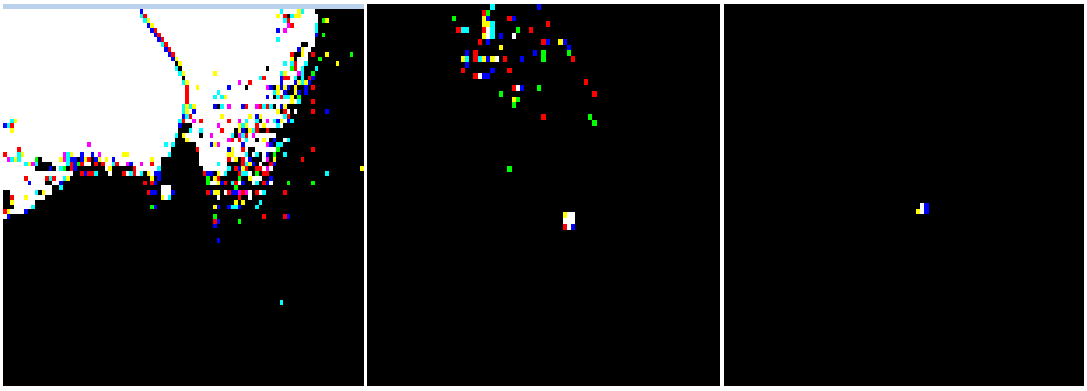
odpovídají pozice příslušného pixelu v obraze. Pixel $i_{x,y}$ obsahuje tři složky, charakterizující jeho barvu. Červenou, zelenou a modrou složku.

Buďme nyní ve stavu, kdy si aplikační logika požádala kamerový modul o nový snímek a ten byl uložen do společného bufferu. Celý obraz je postupně procházen pixel po pixelu a každý z nich je převeden algoritmem převodu z RGB na šeď[17] do šedotónové stupnice. Díky tomu, že pixel je tvořen třemi složkami, kde každá reprezentuje úroveň jednoho barevného kanálu, algoritmus je velice snadné aplikovat. Ve výsledku má pak každý pixel hodnotu od 0 do 255, což odpovídá tomu, „jak moc šedý“ daný pixel je (0 – černá, 255 – bílá). Celý obraz jde pak chápat jako matici $G = (g_{x,y})$. Každý prvek matice pak odpovídá hodnotě z šedotonové stupnice daného pixelu.

Výsledný šedý obraz je dále prahován. Takto prahovaný obraz můžeme opět chápat jako matici $T = (t_{x,y})$., v níž ovšem každý prvek nabývá hodnoty 0 nebo 255, podle toho, zda-li patří k pozadí nebo referenčnímu bodu. Z obrázku Obr. 12c je patrné, že po

[9] Více informací v podsekci 5.1.4

prahování je jediným relevantním objektem v obrazu náš referenční bod. Díky tomu je pak nalezení souřadnic jeho středu poměrně jednoduchou operací.



(a) (b) (c)

Obr. 12 Snímek s referenčním bodem prahovaný s prahem 150(a), totéž s prahem 200(b), totéž s prahem 235(c)

Definujme si množinu R s následujícími vlastnostmi

$$R = \{(i,j): t_{i,j} \in T, t_{i,j} > 0\}$$

Prvky množiny R jsou tedy všechny indexy pixelů z matice T, jejichž hodnota je nenulová, tudíž jsou součástí referenčního bodu.

Definujeme bod S[x, y], který označuje střed referenčního bodu. Jeho souřadnice dostaneme z následujících vztahů:

$$S = \frac{1}{n} \sum_{i=0}^{n} r_i, r_i \in R, \quad (4.1)$$

$$\text{kde } n = |R|.$$

Ze vztahu (4.1) je vidět, že jde o pouhý aritmetický průměr všech pixelů, které tvoří referenční bod. Po provedení všech předchozích kroků se tedy dostaneme do situace, ve které máme v obrazu kamery lokalizován referenční bod a tudíž i uživatelovu pozici. Tímto jsme získali informaci o pozici uživatele v rovině XZ. Díky tomu můžeme nyní sledovat a přenést na obrazovku translační složku jeho pohybu.

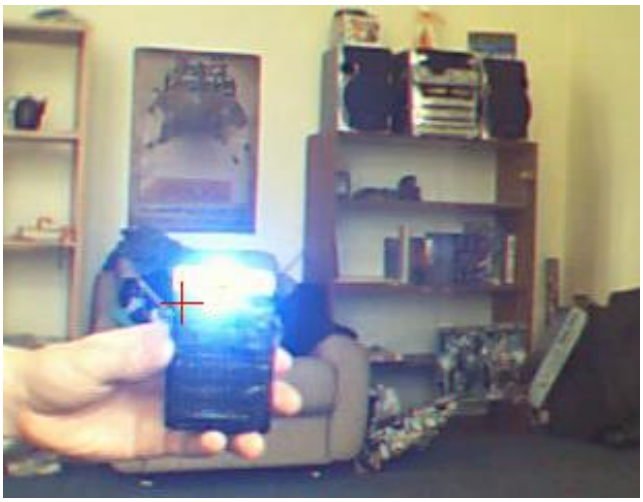
Obr. 13 Zaměřený referenční bod

Jako možnost zvýraznění bodů náležejících prostředí jsem na obraz aplikoval ještě jednoduchou konvoluci[18]. Jako konvoluční masku jsem použil matici

$$\begin{pmatrix} 1 & 2 & 1 \\ 2 & 5 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

a výsledný obraz jsem znovu prahoval. Po několika experimentálních ověřeních jsem ale od konvoluce upustil, protože samotné prahování poskytovalo dostačující výsledky. Konvoluce odstranila šum, který byl však v prahovaném obrazu minimální a také zjemnila okraje oblasti označující referenční bod. Což zpřesnilo nalezení středu referenčního bodu, ovšem výsledné zpřesnění bylo v porovnání se zvýšením náročnosti výpočtu nepodstatné.

Využití konvoluce by ovšem mělo velký význam při použití celé práce v náročných světelných podmínkách jako je přímé sluneční světlo. V takové chvíli totiž dochází k mnohem silnějším odrazům světla, které se kameře jeví stejně jasné jako referenční bod a celý snímek je pak díky tomu silně zašuměn. Následkem toho pak dochází k chybě v určení středu referenčního bodu (viz Obr. 14).



Obr. 14 Odchylka určení středu referenčního bodu za přímého slunečního světla

5.1.3. Kalibrace výstupu z kamery

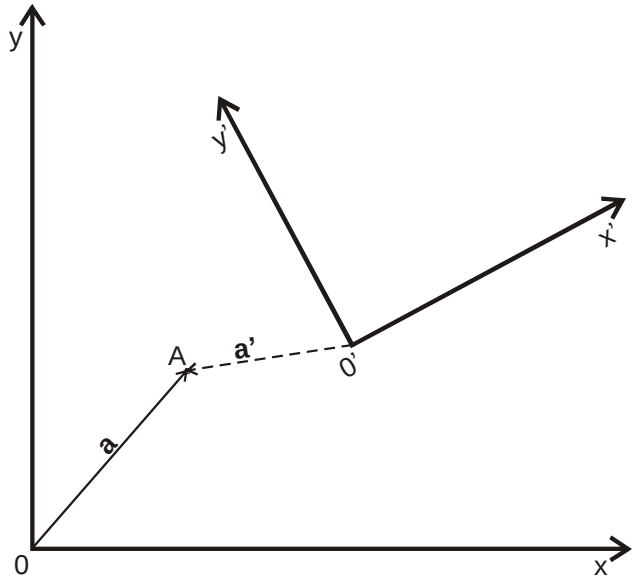
Samotná lokalizace uživatelovi pozice a převod na obrazovku 1:1 nestačí k tomu, aby byla vytvořena věrná iluze pohybu. V tomto případě by odpovídala malá změna pohybu z pohledu uživatele velkému posunu celé scény. Důvodem jsou rozdílné souřadné systémy, systém, v němž se pohybuje uživatel a systém, v němž je zobrazována scéna. Abych tento problém vyřešil, bylo nutné přistoupit ke kalibraci kamerového systému. Pomocí ní bude možné zjistit orientaci a měřítko souřadného systému uživatele a toho využiji k určení pseudo-fyzické pozice uživatele ve virtuální scéně. Jednotka ve scéně bude totiž poté odpovídat jednotce ve skutečném světě uživatele, tudíž i změna polohy uživatele o jeden metr, bude odpovídat změně polohy ve scéně o jeden metr, včetně směru chůze. To vše za předpokladu, že jinak bezrozměrné jednotky, ve kterých se zobrazuje scéna, označíme za metry. Což je poměrně výhodné, protože pak máme velkou kontrolu nad tím, co vlastně zobrazujeme a jestli to vypadá tak jak jsme čekali.

Kalibrace v podstatě funguje tak, že kamerou sejmeme nějakou maketu uživatelského souřadného systému a ve snímku pak zjistíme jakou má orientaci a měřítko vůči kamerovému souřadnému systému. Na tom, jaký zvolíme postup a materiály při výrobě makety záleží, do jaké míry bude moci být kalibrace automatická. Maketu souřadného systému si lze představit jako nějaké tři výrazné body v rovině, z nichž jeden představuje nulu, druhý bod na ose x a třetí bod na osy y. Vzdálenost mezi nulou a bodem na jedné z os nám pak určuje měřítko. Při kalibraci si pak uživatel podrží tuto maketu ve výšce své hlavy, tak, aby „nula“ ležela přibližně na středu jeho hlavy a „bod y“ ukazoval přímo před něj. Díky tomu pak získám informaci o orientaci uživatele a jeho měřítku. Tato maketa odpovídá kalibračnímu štítku (viz Obr. 15).

Obr. 15 Kalibrační štítek na snímku kamery

Hlavním úkolem je poté nalézt pozici těchto tří bodů v souřadném systému kamery a pomocí nich, respektive vektorů, které z nich vytvoříme, vytvořit převodní vztahy mezi souřadným systémem kamery a uživatele. O lokalizaci bodů se zmiňuji níže, nyní popíšu samotný princip kalibrace a důvod, proč je nezbytné ji použít.

Jsem nyní v situaci, kdy kamera sejmula snímek s referenčním bodem označujícím pozici uživatele. Snímek kamery považuji za souřadný systém $S = \{\mathbf{x}, \mathbf{y}, \mathbf{0}\}$ a nechť je bod A referenčním bodem. Spojnici bodu A a počátku souřadného systému S označím jako vektor $\mathbf{a}$. Tento vektor je vlastně souřadnice bodu A v souřadném systému S. Obecně ale nelze předpokládat, že souřadný systém kamery bude shodný se souřadným systémem pozorovatele[10]. Souřadný systém pozorovatele je reprezentován kalibračním štítkem, jenž tvoří souřadný systém $S' = \{\mathbf{x}', \mathbf{y}', \mathbf{0}'\}$. Spojnice bodu A a počátku souřadného systému S' označím jako vektor $\mathbf{a}'$. Obr. 16 ilustruje výše popsanou situaci.



Obr. 16 Schéma ilustrující nutnost kalibrace

[10] O důvodech se zmiňuji v úvodu sekce 5.1

Z obrázku je zřejmé, že bod A má v souřadném systému S' zcela jiné souřadnice než v systému S, což demonstruje rozdílnost vektorů $\mathbf{a}'$ a $\mathbf{a}$. Pro korektní zobrazení odpovídající pohybům uživatele je proto nutné tyto souřadnice zjistit.

Souřadnice všech bodů, včetně bodu A, jsou ale v souřadném systému kamery. Jinými slovy, báze jejich vektorového prostoru odpovídá vektorům $|\mathbf{x}|$ a $|\mathbf{y}|$. Abych zjistil souřadnice bodu A v souřadném systému S', musel jsem za bázi vektorového prostoru vzít vektory $|\mathbf{x}'|$ a $|\mathbf{y}'|$. Obecný postup převodů mezi vektorovými prostory uvádím v podsekci 2.1.2.

Nyní mám souřadnice bodu A v souřadném systému uživatele, tyto souřadnice mohu chápat opět jako vektor, který označím $\mathbf{p}_r$. Jednotlivé složky tohoto vektoru však stále odpovídají pixelům. Aby se pohyb uživatele věrně převedl na obrazovku, je nutné upravit velikost vektoru $\mathbf{p}_r$ tak, aby odpovídala fyzickým jednotkám. Kalibrační štítek v sobě nese informaci také o měřítku uživatelského souřadného systému, díky pevné vzdálenosti mezi jednotlivými body. Tuto vzdálenost označím jako l.

Sejmutím kamerou zjistím, kolik této vzdálenosti odpovídá pixelů, což lze chápat jako velikost vektoru $\mathbf{x}'$ z obrázku Obr. 16. Následující vztah ukazuje, jakým způsobem získám vektor $\mathbf{p}_f$, jehož velikost je již v metrech.

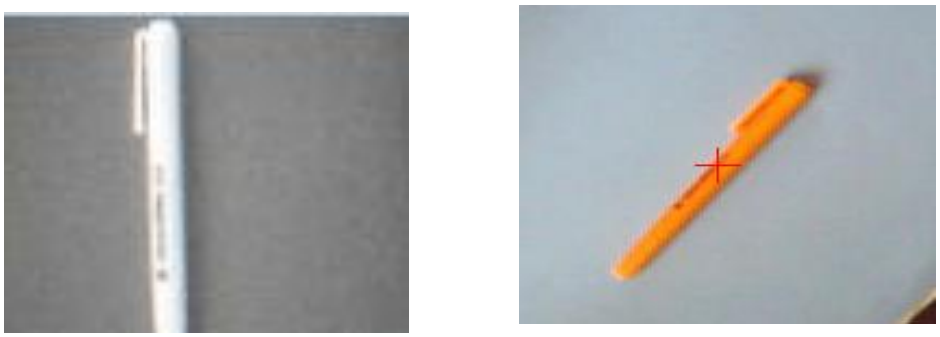
$$\mathbf{p}_f = \frac{l}{\|\mathbf{x}'\|}\,\mathbf{p}_r$$

Při samotné výrobě kalibračního štítku jsem se nejprve ubíral cestou, která se nakonec ukázala jako poměrně špatně použitelná a vydal se cestou odlišnou. Přesto popíšu obě dvě včetně důvodů, proč jsem se rozhodl pro tu druhou.

Prvotní řešení kalibrace by se dalo označit jako pasivní. Materiál použitý na tři význačné body nevyzařoval sám o sobě světlo, měl pouze zvýšenou schopnost odrazivosti. Už z toho vyplývá jasná nevýhoda tohoto postupu. Pokud v místnosti nebude dostatek světla, nebudou mít reflexní body co odrážet a kalibrace bude neúspěšná. I přesto má ale tohle řešení svoje výhody. Tou největší je zřejmě nenáročnost na výrobu a prostředky. Já sám jsem použil čtverečky plastu ze zvýrazňovačů různých barev. Pro jejich rozpoznání jsem postupoval v podstatě stejně jako při hledání referenčního bodu. Celý obraz jsem převedl do šedotónové stupnice, následně ho prahoval a ve výsledku zprůměroval pozice všech pixelů, které nebyly součástí pozadí. Samozřejmě jsem postup musel mírně upravit, protože jsem měl v obraze tři shluky pixelů, které nebyly součástí pozadí, na rozdíl od referenčního bodu, který byl pouze jeden. A navíc aktivně vyzařoval světlo, tudíž nebyl problém ho vhodným prahem oddělit od pozadí.

Abych zvýraznil jednotlivé reflexní body v obraze, při převodu do šedotónové stupnice jsem použil vždy pouze jeden barevný kanál. V červeném kanálu se zvýraznily oranžové čtverečky plastu, v zeleném (překvapivě) žluté. Abych tento efekt podpořil, reflexní body jsem umístil na jednobarevný podklad. První barva na tento podklad která mě napadla, byla obligátní bílá. Protože ale prahuji podle jasu, bílý podklad sám o sobě

odráží dost světla a vytváří tak spoustu šumu, v němž se pak reflexní body ztrácejí. Osvědčilo se mi použít modrý podklad, na kterém se jednotlivé barvy rozpoznávali dobře (viz Obr. 17).



(a) (b)

Obr. 17 příklad snímku s oranžovým fixem na modrém podkladu, pouze červený kanál (a), výsledek rozpoznávání (b)

Další změna oproti postupu při hledání referenčního bodu byla vynucena počtem reflexních bodů. Tu jsem chtěl vyřešit tak, že každý reflexní bod bude z jinak barevného plastu. Obraz se pak třikrát převede do šedotónové stupnice, pokaždé v jiném barevném kanálu. Ideálně by tedy v každém ze tří šedých obrazů měl být nejsvětlejší jeden z reflexních bodů. Bohužel, jednak z důvodů světelných, o kterých jsem se již zmínil, za druhé z důvodu nízkého rozlišení použité kamery, kýženého výsledku jsem nedosáhl.

(a) (b)

Obr. 18 ukázka snímků s oranžovým a zeleným plaste v červeném kanálu(b), totéž, ale v zeleném kanálu(b)

Zkusil jsem tedy použít pro označení nuly žlutého plastu a na x-ový a y-ový bod oranžový. Pokud byl obraz prahován pouze v zeleném kanálu, žlutý plast byl o něco výraznější, takže se dal vyšším prahem oddělit od oranžových. Oranžové jsem pak hledal v červeném kanálu. Obraz jsem ale musel rozdělit na půl a každý z bodů hledat v jedné půlce, „žlutou nulu“ jsem z hledání vyřadil. Kalibrační štítek by ale pak nemohl být natočen libovolným způsobem, protože pak bych nemohl rozhodnout o tom, který z oranžových bodů odpovídá x-ovému bodu, a který y-ovému. Proto jsem od pasivní cesty upustil a nadále se zabýval pouze aktivní. Domnívám se, že pokud by byla použita kamera s dobrou barevnou rozpoznávací schopností a byly upraveny mnou použité algoritmy rozpoznávání, bylo by možné použít pasivní reflexní body. Stále by však bylo toto řešení značně omezeno světelnými podmínkami.

Proto jsem přistoupil k řešení s aktivními význačnými body. Princip jsem chtěl využít stejný jako u pasivního řešení. Tři různě barevné body, každý označující jednu součást souřadného systému. V tomto případě ale tyto body sami světlo aktivně vyzařují. Použil jsem barevné LED diody v barvách červená, zelená a modrá. V této chvíli jsem ale nadále nemohl jednotlivé body rozpoznávat prahováním, protože každá z LED diod září natolik jasně, že se kameře jeví jejich středy jako bílé, takže nelze nalézt práh, kterým by se jednotlivé body oddělily od sebe.

Z tohoto důvodu jsem se tedy rozhodl hledat jednotlivé body přímo ve snímku z kamery, bez toho, abych je převáděl do šedotónové stupnice a prahoval. Jediný krok, který dělám před samotným hledáním je zprůměrování určitého počtu snímku, z důvodu ustálení obrazu a eliminaci případného šumu způsobeného kamerou.

Při samotném hledání bodů ve snímku pak procházím celý obrázek pixel po pixelu a testuji hodnoty v jednotlivých kanálech. Mějme tedy obraz I, složený z pixelů $i_{x,y}$. Pixel $i_{x,y}$ je složen ze tří složek $A_{x,y}$, $B_{x,y}$, $C_{x,y}$, které odpovídají jednotlivým barevným kanálům. Proměnné t_A, $t_{\Delta AB}$, $t_{\Delta AC}$ jsou práh pro kanál $A_{x,y}$, práh pro rozdíl kanálů $A_{x,y}$ a $B_{x,y}$ a práh pro rozdíl kanálů $A_{x,y}$ a $C_{x,y}$. Pokud platí, že

$$A_{x,y} > t_A,$$
$$A_{x,y} - B_{x,y} > t_{\Delta AB},$$
$$A_{x,y} - C_{x,y} > t_{\Delta AC},$$

potom je poloha pixelu přičtena k součtu $P_{kanál\ A}$, která se na konci vydělí počtem takto přičtených pixelů. Tabulka (1) ukazuje konfigurace jednotlivých kanálů.

Kanál A	Kanál B	Kanál C	t_A	$t_{\Delta AB}$	$t_{\Delta AC}$
Červená	Modrá	Zelená	200	60	60
Zelená	Modrá	Červená	160	40	40
Modrá	Červená	Zelená	200	60	60

Tabulka s konfiguracemi kanálů a příslušnými prahy (1)

Tímto vcelku jednoduchým postupem zaměříme poměrně přesně význačné body. I tady však hrají svojí roli světelné podmínky. Přesně opačnou než v případě pasivní metody. Pokud je v místnosti světla příliš, dochází ke stejnému problému jako při hledání referenčního bodu, totiž k zašumění celého obrazu a odchylkám zaměření bodů.

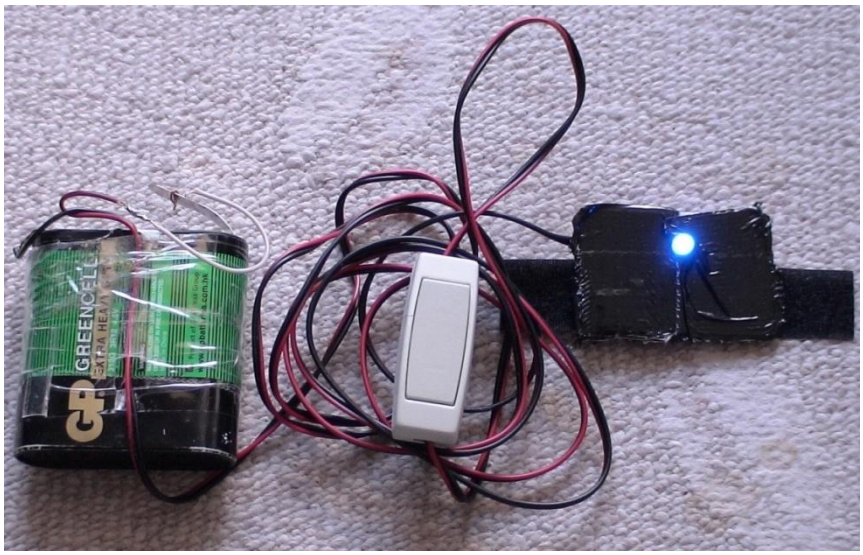
5.1.4. Zařízení pro označení pozice pozorovatele

V této části detailně popíši referenční bod, o kterém jsem se zmiňoval ve své práci již několikrát. Referenční bod slouží pro identifikaci polohy uživatele v obraze kamery. Musí být proto velmi výrazný, aby šel jednoduše lokalizovat. Postup, kterým tento bod hledám jsem popsal výše v podsekci 5.1.2.

Hlavní součástí referenčního bodu je modrá LED dioda, kterou lze ve snímku kamery velmi dobře lokalizovat. Nejprve jsem používal super svítivou LED diodu, které ale až příliš přesvěcovala kameru (viz Obr. 19), což mělo za následek nepřesnost v nalezení jejího středu.

Obr. 19 Obrázek se supersvítivou diodou

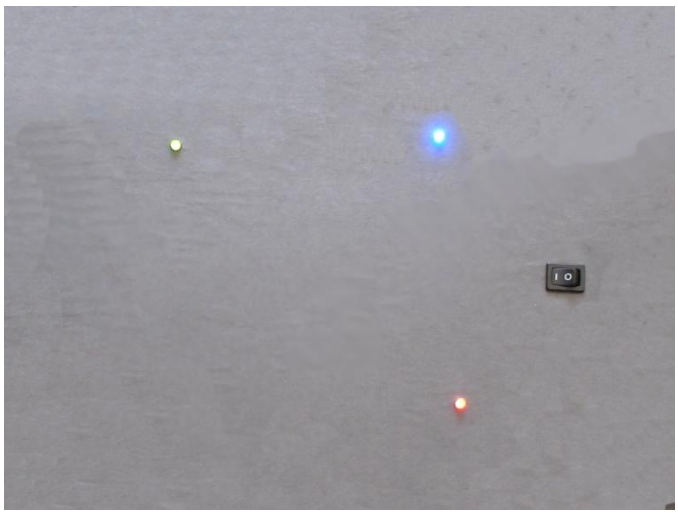
Proto nyní používám obyčejnou LED diodu, která plně dostačuje. Dalšími součástmi referenčního bodu jsou samozřejmě zdroj pro diodu, předřadný odpor, k propojení jsem použil dvojlinkový kabel. Dioda je uchycená v kusu tvrdého plastu, na jehož spodku je suchý zip, kterým se referenční bod přichycuje k uživateli. Protože je kamera umístěná na stropě, bylo nutné umístit referenční bod na uživatelovu hlavu. Druhá část suchého zipu je proto na head trackeru, který má uživatel přibližně v úrovni čela.



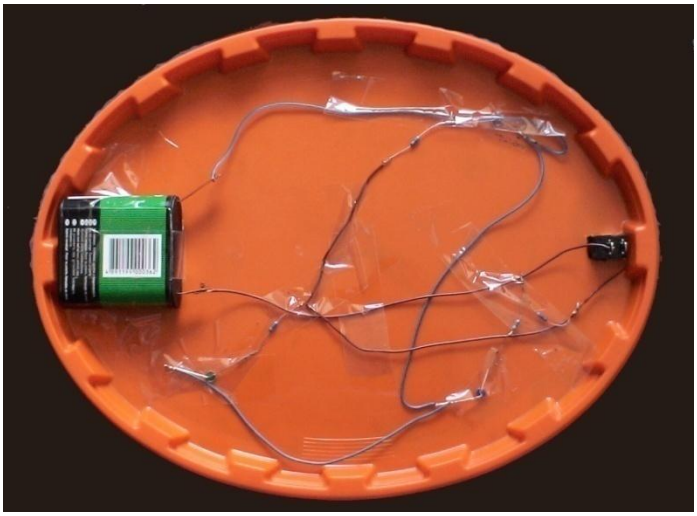
Obr. 20 Referenční bod

5.1.5. Kalibrační štítek

Kalibrační štítek je maketa souřadného systému, v němž se pohybuje uživatel. Slouží pro kalibraci souřadného systému kamery, tak, aby se pohyb uživatele věrně zobrazil včetně odpovídajících vzdáleností a směrů. Jeho konstrukci a dva odlišné směry, kterými jsem se ubíral při jeho výrobě a použití jsem popsal již v podsekci 5.1.3.
V současné chvíli má tedy kalibrační štítek podobu plastového disku, v němž jsou uchyceny, červená, zelená a modrá, svítivé diody. Po zkušenostech s referenčním bodem, jsem automaticky sáhl po těch obyčejných. Pokud bychom je propojili pomyslnou čarou, vytvořily by rovnoramenný pravoúhlý trojúhelník, v němž modrá dioda tvoří vrchol u pravého úhlu a zbylé dvě vrcholy na koncích odvěsen. Délka odvěsny je 15 cm, což je kompromis mezi celkovou velikostí kalibračního štítku a rozpoznatelností z větší výšky. Zelená dioda označuje směr „dopředu“ z pohledu uživatele.



Obr. 21 Obrázek s kalibračním štítkem



Obr. 22 Spodek kalibračního štítku

5.2. Zjištění natočení pohledu pozorovatele

V této části jde prakticky o zjištění směru pohledu uživatele. K tomu je využito zařízení, kterému se říká head tracker. Jde o velmi citlivý mechanismus, který snímá rotace kolem všech třech směrových os. Podrobnější informace v sekci 4.2. Jak už z názvu vyplývá[11], je head tracker určen pro snímání natočení hlavy a tudíž i k zjištění směru pohledu. Rotace hlavy pak musí být vhodně převedeny na rotace kamery ve scéně, aby byla opět zajištěna určitá věrnost zobrazeného obrazu.

5.2.1. Získání dat z head trackeru

Tato část popisuje modul Head tracker. Bude v ní řeč o připojení trackeru k počítači a o způsob získání samotných dat z něj.

Pro práci s head trackerem jsem použil rozhraní DirectInput , které je popsáno v podsekci 3.1.2. Head tracker se nejprve musí najít v připojených zařízeních, poté se musí nastavit formát dat, které tracker poskytuje, a následně se z něj dají data získávat.

O data je vždy požádáno nadřízeným modulem spravujícím head tracker. Řízení je pak předáno modulu head trackeru, který požádá fyzické zařízení o data. Head tracker vrací data v kódu s posunutou nulou, proto vždy od získané hodnoty odčítám konstantu 32 867.

5.2.2. Využití výstupu z head trackeru

Data z trackeru slouží pro rotaci kamery a tím pádem adekvátní změnu pohledu pro uživatele. Jak jsem uvedl v sekci 4.2, data, která head tracker vrací, jsou celá čísla od -32768 do 32768. Kde -16384 značí otočení v dané ose o -180° naopak 16384 otočení o +180°. Rotace však pracují s funkcemi sinus a kosinus, jejichž argumenty musí být radiány, aby rotace proběhla tak jak očekáváme. Proto převádím data z head trackeru na úhly následujícím jednoduchým vzorcem.

$$\varphi = \frac{v * \pi}{16384},$$

kde φ je výsledný úhel a v hodnota vrácená head trackerem.

Před odesláním dat modulu aplikační logiky ještě kontroluji, zda není rozdíl současných a předchozích hodnot velký, což by znamenalo chybu trackeru a neadekvátní skok pohledu.

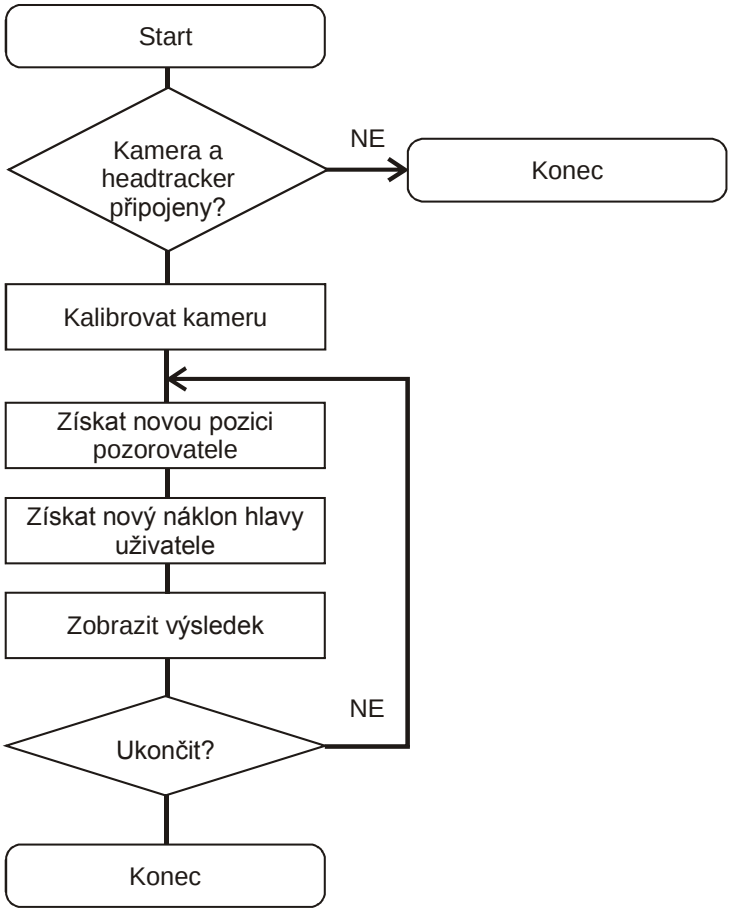
[11] head = anglicky hlava

5.3. Zobrazení výsledku

Tato část zahrnuje nejen samotné vizuální zobrazení na výstupním zařízení, ale také interakci všech použitých technologií a zařízení výše popsaných, bez čehož by zobrazení nebylo možné. Tyto dvě oblasti spolu však natolik souvisí, že jsem se rozhodl spojit je do jedné sekce. Nejprve rozeberu modul aplikační logiky, který je jádrem celého řešení a je to hlavní řídící prvek. V další části pak uvedu, jak se tento modul podílí na zobrazení výsledku a také rozeberu zobrazovací modul.

5.3.1. Interakce použitých zařízení a technologií

Jak jsem uvedl v úplném začátku, zadaný úkol vedl na použití více zařízení a technologií s nimi souvisejícími. Tato dílčí řešení jsem popsal výše a jako jejich abstrakci zavedl různé moduly, odpovídající vykonávané činnosti. Řídícím prvkem celého řešení, jednotící všechny podřízené moduly je modul aplikační logiky. Ten má na starost správnou kooperaci, průběh i zobrazení celého řešení. Celé řešení je tak navrženo jako centrálně řízené. Tento model jsem zvolil z důvodu jednoduchosti implementace v demonstrační aplikace, dále pak pro přehlednost a dobrou kontrolu nad všemi podřízenými prvky. Obr. 23 ukazuje, jakým způsobem modul aplikační logiky pracuje.



Obr. 23 Modul aplikační logiky

Hlavní částí aplikační logiky je tedy smyčka, v níž se pokaždé aktualizuje poloha a náklon hlavy uživatele a tato změna se přenese na výstupní zařízení. Smyčka skončí na příkaz uživatele. Při získávání nové pozice pozorovatele je předáno řízení modulu spravujícím kameru. O tom jak s ním tento modul naloží více v sekci 5.1, respektive v podsekci 5.1.2. Poté co řízení předá zpět aplikační logice, má tato k dispozici aktuální pozici uživatele. Tato pozice je díky kalibraci kamery už ve fyzických jednotkách, v mém případě v metrech. Podobné je to i v případě získání aktuálního náklonu hlavy uživatele. V tomto případě ovšem aplikační logika dostane úhel náklonu v jedné z os.

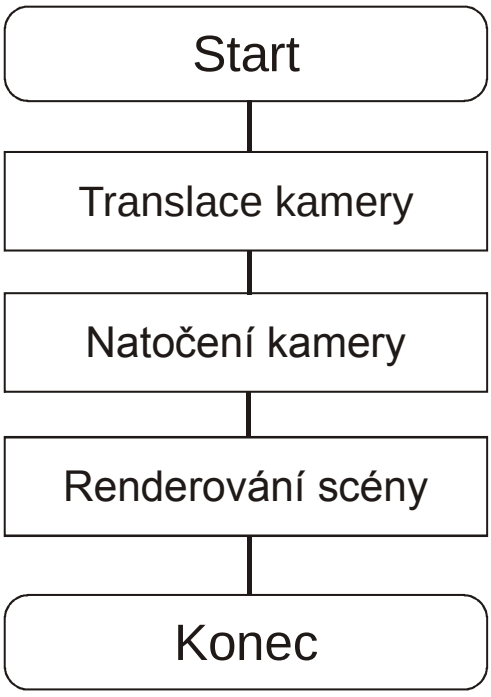
Na základě těchto údajů je upravena pozice kamery, kterou se uživatel dívá do zobrazované scény. Kamera je nejprve posunuta translací na aktuální polohu uživatele a poté otočena kolem jednotlivých os podle natočení hlavy uživatele. Pořadí rotací je důležité, neboť při jiném by výsledek nebyl podle předpokladů. Pokud by P_c byla pozice kamery a Rt_x, Rt_y, Rt_z jednotlivé rotace kolem souřadných os, byla by výsledná pozice kamery

$$P'_c = Rt_x Rt_z Rt_y P_c.$$

5.3.2. Vizualizace výsledku

K zobrazení toho, co poskytne modul aplikační logiky, jsem využil Direct3D. Ten pracuje s termíny, jako je kamera nebo scéna. Kamerou je myšleno pomyslné okénko, kterým se pozorovatel dívá do scény. Scéna je pak nějaké virtuální prostředí s různými objekty a vlastním osvětlením. V mém případě je scéna pouze testovací, tudíž je velice jednoduchá. Obsahuje podlahu, tři zdi a model čajové konvice. Slouží pouze k ověření navrženého řešení.

Modul aplikační logiky poskytne zobrazovacímu modulu otočení kamery a její pozici v rovině XZ. Zobrazovací modul pak provede kroky viz Obr. 24. Zobrazovací modul pracuje ve smyčce, tudíž se tyto kroky provádí opakovaně, dokud se uživatel nerozhodne skončit.



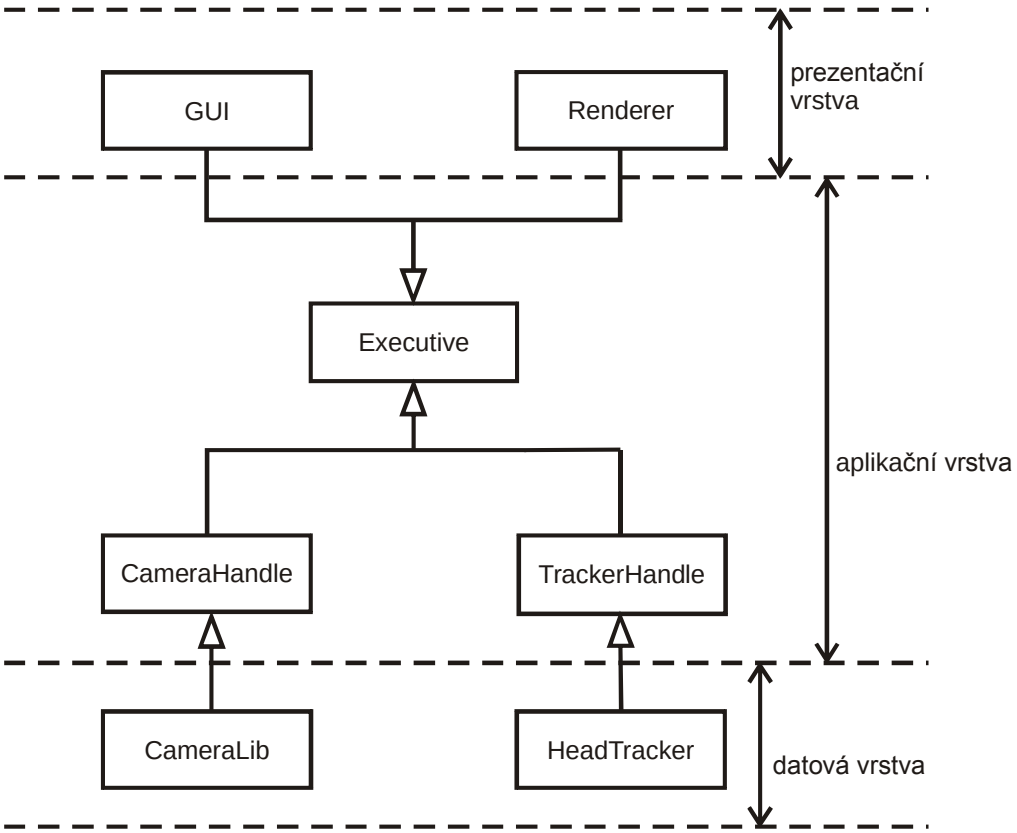
Obr. 24 Schéma zobrazovacího modulu

5.4. Demonstrační aplikace

Pro ověření správnosti řešení bylo nutné vytvořit demonstrační aplikaci, které využije všechny výše zdokumentované postupy a umožní tak převést teoretické řešení do praxe. Celá aplikace byla napsána v jazyce C# na platformě .NET, s výjimkou modulu webkamery, který bylo nutné napsat v jazyce C++, protože DirectShow není pod .NETem implementován.

5.4.1. Architektura aplikace

Díky faktu, že je aplikace napsaná v čistě objektovém jazyce, je rozdělena do jednotlivých tříd, které tvoří vrstvy. Architektura aplikace se liší od návrhu řešení tak jak je uveden na začátku kapitoly 5 v tom smyslu, že se třídy nerovnají modulům a i v samotné rozdělení lze nalézt určité rozdíly. Obr. 25 ukazuje schematický návrh aplikace. Tento návrh je zjednodušený, jsou na něm jen ty nejdůležitější třídy a modul GUI je zástupný symbol pro veškeré třídy obsluhující uživatelský vstup.



Obr. 25 Zjednodušená architektura demo aplikace

Z obrázku 26 je vidět, že aplikace dodržuje zásady třívrstvé architektury. Třída *Executive* je hlavní výkonná třída, která má na starost řízení celé aplikace, a přes kterou jde komunikace všech ostatních součástí aplikace. Propojuje v sobě výstupy z částí pro zjištění pohledu pozorovatele a zjištění natočení pohledu pozorovatele a tyto posílá třídě *Renderer,* která na jejich základě provede vykreslení scény. Zároveň zachycuje všechny vstupy od uživatele, které jí zprostředkují třídy grafického uživatelského rozhraní[12].

CameraLib ve skutečnosti není samostatná třída. Protože je DirectShow přístupný pouze přes jazyk C++, bylo nutné vytvořit knihovnu pracující s DirectShow, která se připojila k mé C# aplikaci. Tou knihovnou je právě *CameraLib*.

CameraHandle je třída, která spravuje samotnou web kameru. Využívá knihovny *CameraLib* k získání jednotlivých snímků. Tyto snímky dále zpracovává tak, jak bylo popsáno v podsekci 5.1.2. Taktéž zajišťuje kalibraci kamerového souřadného systému, tak jak bylo popsáno v podsekci 5.1.3.

HeadTracker je třída poskytující přímý přístup k head trackeru. Má na starost jeho správnou inicializaci a poskytování dat nadřízeným třídám. Head tracker zpřístupňuje přes rozhraní DirectInput. Jednotlivé vykonávané činnosti této třídy popisuje podsekce 5.2.1.

TrackerHandle spravuje head tracker, obdobně jako *CameraHandle* spravuje web kameru. Provádí všechny činnosti nutné ke správné interpretaci dat z trackeru a na

[12] GUI = graphical user interface – grafické uživatelské rozhraní

požádání poskytuje tyto data nadřízené *Executive*. Prováděné kroky jsou popsané v podsekci 5.2.2.

Renderer je třída zajišťující samotné vykreslení scény. Od třídy *Executive* dostane informace o pozici pozorovatele a natočení jeho pohledu, tyto převede na adekvátní transformace kamery a zobrazí. Podrobnosti o zobrazení v podsekci 5.3.2.

GUI je balík tříd umožňující uživateli ovládat a nastavovat aplikaci. Uživatelovi příkazy nezpracovává, pouze je pošle nadřízené třídě *Executive*. Jednotlivé možnosti nastavení a ovládání jsou detailně popsány v příloze A.

5.5. Omezení

Celá práce má samozřejmě svoje omezení. Její hlavní oblast využití je především pro demonstrační. V tom stavu v jakém je, by ji zřejmě nebylo možné použít pro nějakou vědeckou případně komerční činnost, jakou by bylo například spojení této práce s nějakou počítačovou hrou. Je to dáno omezeními samotných použitých zařízení a technologií a limitovanými zdroji pro vývoj.

Technologická omezení by se dala rozdělit do tří kategorií:

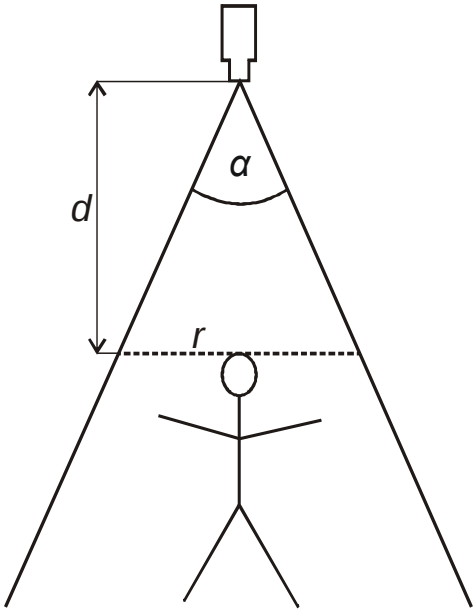
1. Omezení spojená s nalezením pozice pozorovatele
2. Omezení spojená se zjištěním natočení pohledu pozorovatele
3. Omezení spojená se zobrazením

5.5.2. Omezení spojená s nalezením pozice pozorovatele

Nejcitelnější limitace v této oblasti je relativně malý prostor, v kterém se pozorovatel může pohybovat. Tento prostor je vymezen vzdáleností referenčního bodu, tedy hlavy pozorovatele a zorným úhlem kamery (viz Obr. 26). Pokud d bude vzdálenost ref. bodu od kamery a α zorný úhel kamery, potom strana r čtverce, který vymezuje prostor pro pohyb, bude

$$r = 2 * \tan(\frac{\alpha}{2}) * d.$$

V případě běžné místnosti se stropem ve výšce 250 cm a běžného uživatele o výšce 170 cm bude mít prostor, který je kamera schopná zachytit rozlohu přibližně 40×40 centimetrů, což není mnoho. Zmenšit dopad tohoto omezení lze samozřejmě používáním v místnosti s vysokým stropem, případně snížení uživatele, například posazením na pojízdnou kancelářskou židli.



Obr. 26 Omezení pozorovatelova pohledu zorným polem úhlem

Dalším omezení spadající do této kategorie jsou odvozena od citlivosti celého řešení na míru osvětlení místnosti. Jak v případě kalibrace, tak i v případě hledání referenčního bodu, silné světlo, ať už sluneční nebo umělé, způsobí odrazy, které znemožní lokalizaci význačných bodů. Celá práce proto musí být používána v částečně kalibrovaném prostředí, jako je například místnost bez lesklých materiálů a osvětlená difúzním světlem. Přizpůsobit práci obecným světelným podmínkám by bylo časově velice náročné.

5.5.3. Omezení spojená se zjištěním natočení pohledu pozorovatele

Hlavní omezení při používání head trackeru plynou hlavně z jeho citlivosti. Při testování jsem zjistil, že tracker musí být pevně přichycen na uživatelově hlavě, aby poskytoval smysluplná data. Pokud je držen pouze v ruce, dochází k velkému zkreslení, případně data, která tracker poskytuje, jsou zcela odlišná od očekávaných.

To je způsobeno:

1) přílišnou volností pohybu zápěstí, díky níž nelze udržet otáčení trackeru pouze kolem jedné osy
2) sevřením trackeru prsty, ten, pokud je špatně držen, neposkytuje zpravidla data vůbec žádná.

Občas se mi také stalo, že tracker indikoval změnu polohy, i když ležel v klidu na stole. V takovém případě je nejlepší resetovat pozici trackeru pomocí tlačítka k tomu určenému.

5.5.4. Omezení spojená se zobrazením

Pokud bude k vizualizaci práce použit HMD(viz 4.1), je nutné počítat s dalším omezením pohybu, daného délkou kabelu, kterým se HMD připojuje k počítači. Toto omezení lze eliminovat přichycením počítače k uživateli, samozřejmě za předpokladu, že používaným typem počítače je notebook.

Dalším omezením nebo lépe řečeno nedostatkem, je kvalita zobrazené scény. Na tu sice nebyl v zadání kladen důraz, ovšem pro pocit realističnosti je to jeden z klíčových faktorů. Bohužel už mi na vytvoření vizuálně atraktivní scény nezbyl čas. Pokud by byla ale moje práce dále používána, třída zajišťující zobrazení je navržena tak, aby části týkající se scény mohli být snadno nahrazeny něčím komplexnějším.

6. ZÁVĚR

Práci je plně funkční a splňuje všechny body, tak, jak byly uvedené v zadání. Podařilo se mi skloubit všechny zadané technologie tak, aby poskytovaly smysluplný a do určité míry realistický výstup. Oproti původnímu plánu se mi nepodařilo využít k zobrazení výsledku 3D režim brýlí, což bylo způsobeno tím, že brýle nejsou již nějakou dobu výrobcem podporovány. Přesto si myslím, že i v běžném režimu je výsledek přinejmenším zajímavý. Při vypracovávání jsem se dostal také párkrát do slepé uličky a musel od původního záměru upustit. Jako jeden příklad za všechny uvedu návrh kalibračního štítku, u kterého se ukázala pasivní metoda konstrukce jako poměrně špatně použitelná.

Pokud by tato práce měla být využita pro nějaké reprezentační účely, případně dále rozšiřována, určitě by jí prospěla změna zobrazované scény. Tu jsem pro svoji potřebu vytvořil velice jednoduchou, proto bych navrhoval jí kompletně nahradit něčím sofistikovanějším. Další úpravou by mohlo být vylepšení detekce referenčního bodu pomocí manuální manipulace se závěrkou webkamery.

I přes svoji náročnost, nebo možná právě proto, mě práce bavila a vždy, když se mi podařilo přidat další fungující část, měl jsem z toho upřímnou radost. Zábavnost této práce byla také zapříčiněna atraktivitou zadání, kdy jsem musel skloubit technologie a postupy z různých technických oblastí a ty pak přetvořit ve vizuální výstup. Každé vylepšení se pak také odpovídajícím způsobem na tomto výstupu projevilo.

UŽITÁ LITERATURA A ZDROJE

[1] REKTORIS, Karel a spolupracovníci. *Přehled užité matematiky*. Praha: SNTL, 1981.

[2] ŽÁRA, Jiří, BENEŠ, Bedřich, SOCHOR, Jiří, FELKEL, Petr. *Moderní počítačová grafika*. Brno: Computer Press, 2004.

[3] *Centre of Computer Graphics and Data Visualisation* [online], poslední revize 3.1.2008 [cit. 2008-05-11], <http://herakles.zcu.cz/education/ZPG/cviceni.php>

[4] GONZALES, Rafael C. - WOODS, Richard E.. *Thresholding. In digital Iimage Processing*. Pearson Education, 2002.

[5] Microsoft Corporation. *DirectX*[online], < http://msdn.microsoft.com/en-us/xna/aa937781.aspx>

[6] MIRZA, Yahya H. - DA COSTA, Henry. *DirectX 9.0: Introducing the New Managed Direct3D Graphics API in the .NET Framework* [online], <http://msdn.microsoft.com/en-us/magazine/cc164112.aspx>

[7] Microsoft Corporation. *DirectInput* [online], <http://msdn.microsoft.com/en-us/library/bb219802(vs.85).aspx>

[8] Microsoft Corporation. *.NET Framework Developer Centre* [online], <http://msdn.microsoft.com/cs-cz/netframework/default(en-us).aspx >

[9] Microsoft Corporation. *Microsoft Platform SDK* [online], <http://www.microsoft.com/msdownload/platformsdk/sdkupdate/>

[10] STROUSTRUP, Bjarne. *The C++ Programming Language.* Addison-Wesley, 1986.

[11] Microsoft Corporation. *Visual C# Language (C#)* [online], c2008, <http://msdn.microsoft.com/en-us/library/aa287558.aspx>

[12] Microsoft Corporation. *Managed Extensions for C++ Specification* [online], c2008, <http://msdn.microsoft.com/en-us/library/aa712867.aspx >

[13] Virtual i-O. *i-Glasses Developer Kit* [online], <http://www.eserviceinfo.com/downloadsm/30418/I-glasses_S-VGA.html>

[14] Wikipedia contributors. *Webcam* [online], Wikipedia: The Free Encyclopedia, poslední revize 28. 4. 2008 [cit. 2008-05-11], <http://en.wikipedia.org/w/index.php?title=Webcam&oldid=210855544>

[15] Creative. *WebCam Instant* [online], c2003, <http://us.creative.com/products/product.asp?category=722&subcategory=726&product=10410&nav=1>

[16] BYSTŘICKÝ, Václav. *Demonstrační aplikace klíčování videa*. Bakalářská práce, vedoucí Petr Lobaz. Plzeň, 2007.

[17] FANNING, W. David. *Convert RGB Image To Grayscale* [online], c1997, poslední revize 23. 9. 2002 [cit. 2008-05-11], <http://www.dfanning.com/ip_tips/color2gray.html>

[18] WEISSTEIN, Eric W. *Convolution* [online], c1998, <http://mathworld.wolfram.com/Convolution.html>

PŘÍLOHA A

A.1: Uživatelská příručka

Ovládání aplikace je velice jednoduché a uživatelsky příjemné. Je to dáno hlavně minimem možných nastavení, protože aplikace nevyžaduje téměř žádné uživatelské vstupy a pracuje téměř automaticky. V sekci nastavení lze pouze určit výšku uživatele a kalibrovat kameru pro snímání referenčního bodu. Obojí podrobně rozeberu dále. Aplikace si také pamatuje hodnoty z předchozího spuštění. Takže pokud je uživatel i místo používání stejné, stačí vše nastavit pouze jednou a při dalších spuštěních přejít rovnou k samotnému zobrazení.

Před každým spuštěním aplikace zkontrolujte, zda je k počítači připojena kamera a head tracker!

Kroky nutné k nastavení výšky:

1. Spusťte aplikaci.
2. V hlavním menu stiskněte tlačítko **Settings….**
3. Otevře se nové okno. V něm zadejte do textového pole s popiskem **Current user height** svojí výšku v centimetrech.
4. Stiskněte tlačítko **Set** vedle textového pole.
5. Vaše výška byla uložena, pro zavření okna s nastaveními stiskněte tlačítko **Continue….**

Kroky nutné ke kalibraci kamery:

1. Spusťte aplikaci.
2. V hlavním menu stiskněte tlačítko **Settings….**
3. Otevře se nové okno. V něm stiskněte tlačítko **To coordinate space calibration**.
4. Otevře se nové okno kalibrace kamery. Kameru lze kalibrovat automaticky nebo manuálně.

   Automatická kalibrace:

   a. Ujistěte se, že je v záběru kamery kalibrační štítek a jsou viditelná všechna tři světla.
   b. Stiskněte tlačítko **Calibrate** v dolní části obrazovky.
   c. V přibližném středu všech tří světel by se měl objevit nitkový kříž.
   d. Pokud je odchylka jednoho nebo více křížů příliš veliká, případně se zobrazí méně než tři, opakujte bod **b**. Odchylka kříže od středu světla by měla být maximálně 5 milimetrů.
   e. Pokud jsou kříže správně zaměřené, stiskněte tlačítko **Save coordinates and continue….**
   f. Stiskněte tlačítko **Continue… pro** uzavření okna s nastaveními.

Manuální kalibrace:

a. Ujistěte se, že je v záběru kamery kalibrační štítek a jsou viditelná všechna tři světla.
b. V pravém horním rohu obrazovky zaškrtněte políčko **manual control**.
c. Jezdce nahoře a po levé straně se stanou aktivní, stejně jako tři radiové tlačítka po pravé straně. Taktéž se v obrazu s kalibračním štítkem objeví v levém rohu tři čtverce modré, červené a zelené barvy. V této chvíli je však vidět pouze modrý, který zbylé dva překrývá.
d. Nyní pomocí jezdců posuňte modrý čtverec na pozici modrého světla tak, aby střed čtverce korespondoval přibližně se středem světla.
e. Vyberte tlačítko **red light square** ve skupince tlačítek na pravé straně.
f. Opakujte krok **d**, v tuto chvíli ale s červeným čtvercem a červeným světlem.
g. Vyberte tlačítko green light square ve skupince tlačítek na pravé straně.
h. Opakujte krok **d**, v tuto chvíli ale se zeleným čtvercem a zeleným světlem.
i. Pokud jsou všechny tři čtverce na svých místech a na správných světlech (!), stiskněte tlačítko **Calibrate**. Na středech světel se objeví nitkové kříže.
j. Kalibrace byla úspěšně dokončena, stiskněte **Save coordinates and continue….**
k. Stiskněte tlačítko **Continue… pro** uzavření okna s nastaveními.

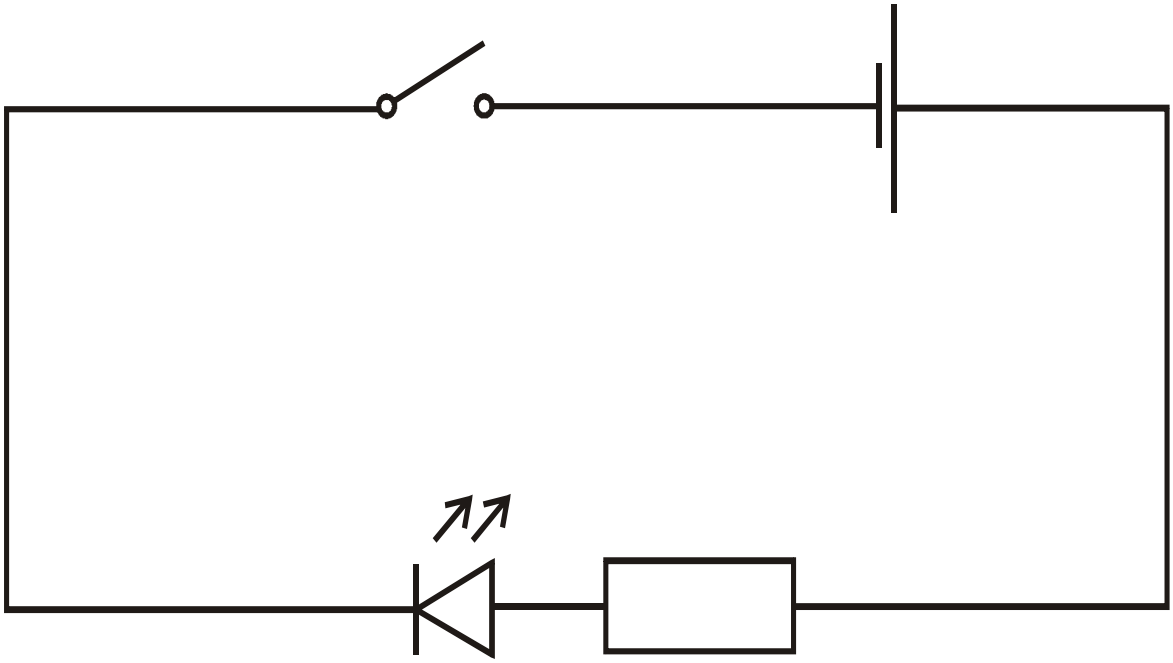
Kroky nutné k provozu aplikace:

1. Spusťte aplikaci.
2. Proveďte nastavení, pokud je to nutné.
3. Připevněte si na hlavu Head tracker a na něj referenční bod.
4. V hlavním menu stiskněte tlačítko **Go!**.
5. Na výstupním zařízení nyní vidíte zobrazenou scénu. Otáčením hlavy a chůzí měníte svojí pozici a pohled na scénu.
6. Pro ukončení stiskněte tlačítko Escape na klávesnici.

PŘÍLOHA B

Schéma zapojení kalibračního štítku a referenčního bodu a výčet použitých součástek.

B.1: Referenční bod



Obr. 1 Schéma referenčního bodu

Použité součástky

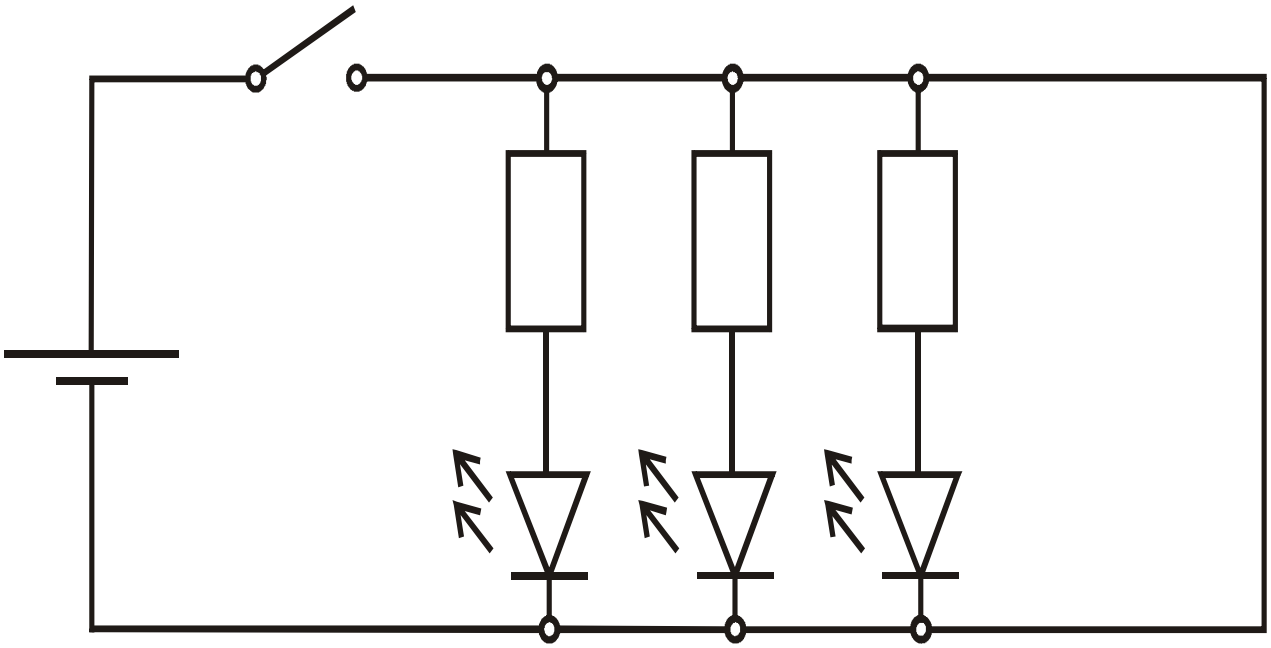
1× modrá LED dioda

1× odpor 300 Ω

1× vypínač

1× baterie 4,5 V

B.2: Kalibrační štítek



Obr. 2 Schéma kalibračního štítku

Použité součástky

1× modrá LED dioda

1× zelená LED dioda

1× červená LED dioda

3× odpor 300 Ω

1× vypínač

1× baterie 4,5 V