VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

NÁVRH BYTU POMOCÍ ROZŠÍŘENÉ REALITY

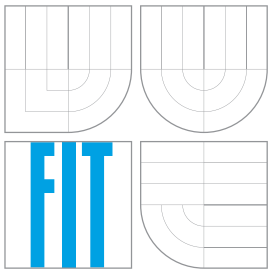
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. DAVID TUŠKA

BRNO 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

NÁVRH BYTU POMOCÍ ROZŠÍŘENÉ REALITY

FLAT DESIGN USING AUGMENTED REALITY

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. DAVID TUŠKA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. VÍTĚZSLAV BERAN

BRNO 2010

Zadání diplomové práce/5416/2009/xtuska01

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2009/2010

Zadání diplomové práce

Řešitel: **Tuška David, Bc.**
Obor: Počítačová grafika a multimédia
Téma: **Návrh bytu pomocí rozšířené reality**
Flat Design Using Augmented Reality
Kategorie: Počítačová grafika

Pokyny:

1. Seznamte se s technikami rozšířené reality a s nástrojem ARToolKit. Prostudujte literaturu týkající se detekce vzorů v obraze a metod pro srovnání obrazu s modelem.
2. Navrhněte systém, který s využitím ARToolKitu detekuje vodicí značku v obraze a nalezne pozici uživatele v modelu bytu. Pozici uživatele využije k vykreslení modelu do obrazu místnosti.
3. Vytvořte testovací sadu dat. Sadu oanotujte a připravte pro testování systému.
4. Navrhněte a implementujte experimentální aplikaci, která bude s pomocí virtuálních brýlí demonstrovat navržený systém.
5. Proveďte experimenty zkoumající přesnost a stabilitu navrženého systému. Experimenty pečlivě zdokumentujte a okomentujte.
6. Diskutujte omezení a nedostatky navrženého řešení a navrhněte případná vylepšení. Vytvořte plakát reprezentující Vaše řešení.

Literatura:

- Hlaváč, V.: Zpracování signálů a obrazů, Praha, ČVUT, 2005.
- Rafael C. Gonzalez, Richard E. Woods: Digital image processing, Prentice-Hall, 2002.
- Galbiati, L. J.: Machine vision and digital image processing fundamental, Prentice-Hall, 1990.
- Dále dle pokynu vedoucího.

Při obhajobě semestrální části diplomového projektu je požadováno:

- Body 1, 2 a 3.

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

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

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

Vedoucí: **Beran Vítězslav, Ing.**, UPGM FIT VUT
Datum zadání: 21. září 2009
Datum odevzdání: 26. května 2010

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

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Diplomová práce se zabývá využitím rozšířené reality pro návrh dispozic bytu. Cílem práce je vytvořit systém, který na základě vodících značek umístěných na stěnách bytu zjistí pozici uživatele v modelu bytu. Pozici uživatele poté použije k ovládání kamery ve 3D modelovacím programu Google SketchUp. Systém je složen ze serverové a klientské části. Serverová část systému zpracovává vstupní video, ve kterém detekuje vodící značky a pomocí nadetekovaných značek vypočítává pozici uživatele. Klientská část systému je reprezentována zásuvným modulem (pluginem) pro 3D modelovací program Google SketchUp. Klient se serverové části dotazuje na pozici uživatele v modelu bytu a následně podle této pozice nastavuje kameru v programu Google SketchUp. Dále se klient dotazuje na jednotlivé snímky z videa, které zobrazuje na pozadí Google SketchUp a tím vytváří obraz reálné scény rozšířené o virtuální objekty.

Klíčová slova

Počítačová grafika, rozšířená realita, Google SketchUp, ARToolkit, ARTag, detekce vzorů, vodící značka, HMD

Abstract

This master's thesis is dealing with the usage of augmented reality for a layout of a flat proposition. The aim of this work is to create a system which detects markers on the walls using ARToolKit library. The position of the used in flat is determined by using the markers. Then the system disposes the user's position for operate with a camera in a 3D modelling program called Google SkretchUp. The system is consisted of a server and a client part. The server one copes with the incoming video in which it detects the markers and evaluates the position of user by the help of the markers. The client part of the system is represented by a plugin module for 3D modelling program Google SkretchUp. A client is asking the server for the positions of the user in the flat model and consequently he set the camera in the program Google SkretchUp according to these positions. The client is also asking for the individual pictures from the video which he shows against the background of Google SketchUp and thanks to this, he creates a view of a real scene extended of virtual objects.

Keywords

Computer graphics, augmented reality, Google SketchUp, ARToolkit, ARTag, mark detection, marker, HMD

Citace

David Tuška: Návrh bytu pomocí rozšířené reality, diplomová práce, Brno, FIT VUT v Brně, 2010

Návrh bytu pomocí rozšířené reality

Prohlášení

Prohlašuji, že jsem tuto semestrální práci vypracoval samostatně pod vedením pana Ing. Vítězslava Berana. Uvedl jsem všechny literáarní prameny a publikace, ze kterých jsem čerpal.

.....
David Tuška
25. května 2010

Poděkování

Rád bych poděkoval Ing. Vítězslavu Beranovi za odborné vedení, účelné připomínky a konzultace.

Obsah

Kapitola 1

Úvod

Při návrhu bytu a kupování nového nábytku si nejčastěji, kromě otázky ceny, klademe otázku: „Bude se nám nová skříň hodit do již vybaveného pokoje?“. A nebo „Je vhodnější umístit sedací soupravu do levého či pravého koutu nově zařizovaného pokoje?“

Má diplomová práce by měla pomoci nalézt odpovědi na tyto otázky. Ulehčit navrhování bytů a díky rozšířené realitě zobrazovat virtuální a ještě neexistující kusy nábytku přímo uprostřed vašeho domova na místě, kam byste je chtěli umístit.

V současné době existuje mnoho komerčních i nekomerčních aplikací pro návrh kuchyní, bytů či zahrad. Jako reprezentaci komerčních můžeme uvést například: uvRoom Arranger, „TurboFLOORPLAN Dům & Interiér & Zahrada“ nebo „KitchenDraw“. Mezi nekomerční patří například: „Sweet Home 3D“ či „IKEA Home Planner“ jako prezentační nástroj produktů firmy IKEA.

Všechny tyto aplikace mohou vytvářet virtuální modely bytů a jejich vybavení. Komerční nástroje obsahují velké množství modelů jednotlivých zařizovacích předmětů. A umožňují zobrazovat 3D model těchto bytů a virtuálních prohlídek na monitoru počítače. Ale žádný z těchto nástrojů neposkytuje možnost využití rozšířené reality k prezentaci a návrhu nového bytu a jeho vybavení. Díky rozšířené realitě je možno zobrazovat virtuální modely v reálné velikosti na místě, kde by měl později stát fyzický model nábytku. A tím uživateli poskytnout reálnější představu o novém uspořádání bytu.

Navrhovaný systém musí být schopen modely nábytku nejen zobrazovat, ale musí umožňovat do místnosti nové modely přidávat a následně je i editovat. Při realizaci systému máme dvě možnosti. První možností je implementovat vlastní editor 3D objektu, druhou možností je použít již některý existující editor. Jelikož aplikací pro 3D modelování existuje velké množství, je zbytečné se snažit implementovat vlastní editor.

Při prozkoumání existujících 3D modelovacích programů jsem se rozhodl, že vytvořený systém v rámci této diplomové práce bude spolupracovat s 3D modelovacím programem Google SketchUp. V tomto programu bude uživatel vytvářet model bytu a modely jednotlivých zařizovacích předmětů.

Aby nebylo nutné implementovat vlastní renderer pro modely aplikace Google SketchUp, rozhodl jsem se použít daný program i pro zobrazování těchto objektů. Cílem této práce tedy bude vytvořit zásuvný modul (plugin) do aplikace Google SketchUp, který bude ovládat kameru v tomto programu. Kamera v modelu se bude nastavovat na stejnou pozici jako je uživatelova pozice v bytě.

K určení pozice uživatele v bytě bude použito vodicích značek, které budou umístěny na stěnách místnosti. Tyto značky budou detekovány s využitím knihovny ARToolKit. Pomocí této knihovny také vypočteme relativní pozici mezi kamerou a vodící značkou.

V první kapitole s názvem „Počítačová grafika“ se seznámíme se základními principy algoritmů, které se používají v rozšířené realitě založené na zpracování obrazu. Jako představitele segmetačních metod pro zpracování obrazu si představíme algoritmy prahovaní a detekce hran.

Druhá kapitola s názvem „Rozšířená realita“ nám v první části vysvětluje, co je to rozšířená realita. Seznamuje nás s principy rozšířené reality, možnostmi realizace a příklady již existujících aplikací využívající rozšířenou realitu v komerčních i nekomerčních projektech. Druhá část této kapitoly rozebírá základní principy a algoritmy činnosti knihovny ARToolKit. Tato knihovna je používána pro detekci vodících a usnadňuje nám tímto vyvíjení aplikací rozšířené reality.

Kapitola „Návrh systému“ pojednává nejprve obecněji o návrhu aplikace rozšířené reality a různých možnostech implementace. Následně je představena struktura řešení, která je navrhnuta jako klient-server aplikace.

Kapitola „AR Flat Design Server“ nám detailněji popisuje návrh serverové části systému, a to především detekce vodicích značek v obraze a následné zjištění pozice uživatele. V této kapitole je také popsán komunikační protokol mezi serverem a klientem.

Předposlední kapitola „AR SketchUp Plugin“ pojednává o návrhu zásuvného modulu do programu Google SketchUp, který je využíván k vykreslování modelu bytu a vytváření obrazu rozšířené reality.

Poslední kapitola se zabývá dosaženými výsledky a experimenty, které jsem během vytváření aplikace prováděl.

Kapitola 2

Počítačová grafika

Počítačová grafika je jednou z nejrychleji se rozvíjejících disciplín oboru informatiky. Můžeme ji rozdělit na dvě základní části. První je vytváření grafických obrazů, virtuálních scén a animací. Tato část je zastoupena v počítačových hrách, v kreslících programech a v našem případě k vykreslení virtuálních objektů do reálné scény. Druhou oblastí, kterou se zabývá počítačová grafika, je zpracování obrazu a počítačové vidění. Výsledky je možné využít při rozpoznávání textu, řízení průmyslových robotů, porovnávání otisků prstů a rovněž pro detekci značek ve videu. Podle kterých dokážeme určit pozici uživatele v modelu místnosti.

Tato kapitola popisuje základní techniky používané v rozšířené realitě, která je založena na zpracování obrazu. V první části této kapitoly je popsáno několik základních segmentačních metod, které jsou v rozšířené realitě použity pro extrakci výrazných bodů z obrazu. Následně je zde popsána jedna metoda shlukování použitá k seskupení nadetekovaných bodů.

2.1 Co je to obraz

„Matematickým modelem obrazu může být spojitá funkce $f(i, j)$ dvou argumentů, souřadnic v rovině. Funkci $f(i, j)$ se obvykle říká obrazová funkce. Hodnotou obrazové funkce je nejčastěji jas (intenzita). Jas je veličina, která souhrnně vyjadřuje vlastnosti obrazového signálu způsobem, který odpovídá jeho vnímání člověkem.“

„Obraz může být v jednodušším případě monochromatický. Je reprezentován jedinou obrazovou funkcí $f(i, j)$. Ve složitějším případě pracujeme s barevným (multispektrálním) obrazem. Každé dvojici plošných souřadnic (i, j) odpovídá vektor hodnot - např. jasů pro jednotlivé barevné složky obrazu.“ [9]

2.2 Segmentace

Abychom mohli umístit virtuální objekty do reálné scény je potřeba nejprve z obrazu získat informace o 3D scéně. Informace poté použijeme k umístění virtuálního objektu na správné místo. Jedním z nejjednodušších způsobů jak získat z obrazu prostorovou informaci je do scény umístit vodicí značky (papírové čtverce s učitým vzorem). Ty jsou poté detekovány a z perspektivního zkreslení značky je vypočtena pozice značky v prostoru. K detekci a rozlišení vodicích značek od pozadí budeme používat segmentační metody.

„Segmentace obrazu je jedním z nejdůležitějších kroků vedoucích k analýze obsahu zpracovávaných obrazových dat. Snahou je rozčlenit obraz do částí, které mají úzkou souvislost

s předměty či oblastmi reálného světa zachyceného na obraze. Výsledkem má být soubor vzájemně se nepřekrývajících oblastí, které buď jednoznačně korespondují s objekty vstupního obrazu, pak jde o kompletní segmentaci, nebo vytvořené segmenty nemusí přímo souhlasit s objekty obrazu, a pak jde o částečnou segmentaci.“ [9]

Jedním z hlavních problémů, které ovlivňují segmentaci, je nejednoznačnost obrazových dat. Data jsou často doprovázena informačním šumem, kterého se snažíme pomocí segmentace zbavit a tím i výrazně redukovat objem zpracovávaných dat.

Prahování

Jednou z nejjednodušších a nejstarších metod segmentace je prahování. Tato metoda je dnes v jednoduchých případech stále používána. Jedním z důvodů je její jednoduchost implementace a také rychlost či možnost paralelizace výpočtů.

Pro svou jednoduchost a rychlost je tato metoda také používána v knihovně ARToolKit (kapitola 3.3), kterou budeme používat při vytváření systému.

Cílem prahování je pro každý bod obrazu (pixelu) přiřadit hodnotu 1, pokud daný bod leží v hledané oblasti, nebo hodnotu 0 ostatním bodům (body pozadí). Prahování je založeno na předpokladu, že hledaná oblast má stejnou nebo velmi podobnou hodnotu jasu. Hledáme tedy body, jež mají jas v předem daném intervalu $\langle a, b\rangle$ 2.1. Druhou možností 2.2 implementace je zvolit si jakýsi práh t. Body, které mají hodnotu větší než zadaný práh, jsou označeny za součást hledané oblasti. Ostatní body jsou označeny jako body pozadí.

Pokud je hodnota prahu pro všechny obrazové body stejná jedná se o globální prahování. O adaptivním prahování, kde se hodnota prahu dynamicky mění, se dozvíme v následující kapitole (kapitola 2.2).

$$g(x,y) = \begin{cases} 1, & \text{pro} f(x,y) \in \langle a,b\rangle \\ 0, & jiank \end{cases} \tag{2.1}$$

nebo

$$g(x,y) = \begin{cases} 1, & \text{pro} f(x,y) \geq t \\ 0, & \text{pro} f(x,y) < t \end{cases} \tag{2.2}$$

„Úspěšnost prahování závisí na znalosti správné hodnoty prahu. Jestliže tuto hodnotu neznáme, je možné pokusit se ji stanovit na základě informací získaných z obrazu, který má být segmentován. Pro obrazy s bimodálním histogramem (histogram se dvěma vrcholy) jasu se např. často doporučuje volit jako hodnotu t prahu hodnotu, v níž histogram dosahuje mezi oběma vrcholy minima.“[1] (Obrázek 2.1)

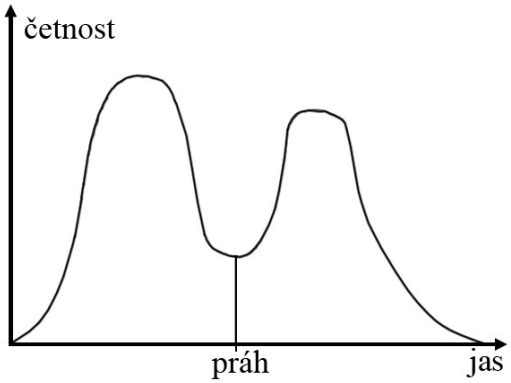
Adaptivní prahování

Hlavní nevýhodou prahování s globálním prahem je dosažení velmi špatných výsledků v případě, že obraz je osvětlen nerovnoměrně.

Zatímco běžné prahování používá globální práh pro všechny pixely, adaptivní prahování mění práh dynamicky nad obrazem, který zpracovává. Tato více propracovaná metoda prahování se může přizpůsobit měnícím se světelným podmínkám v obraze, např. ke které dochází v důsledku prutké změny osvělení nebo díky stínům. [13]

Funkce globálního prahu je stanovena z celého obrazu f:

$$T = T(f) \tag{2.3}$$



Obrázek 2.1: Bimodální histogram jasu

Obrázek 2.2: Vlevo původní obrázek. Vpravo prahování s prahem 128. Převzato z [4]

Kdežto adaptivní prahování využívá lokálního prahu, který je stanoven z části obrázku f_c pro kterou je hodnota prahu počítána:

$$T = T\left(f, f_c\right) \tag{2.4}$$

Jedna z možností jak určit lokální práh je, že obraz f rozdělíme na jednotlivé části obrazu f_c a vypočteme prah nezávisle pro každou část obrazu. Pokud nemůže být práh vypočten pro některou část obrazu (např. celá část obrazu obsahuje pouze jednu barvu), tak práh je stanoven interpolací z okolních částí obrazu. Zpracovávaná část obrazu by měla být tak malá, aby v dané části obrazu bylo osvětlení rovnoměné. Každá část obrazu je poté zpracována s ohledem na lokální práh. [10] [13]

Metoda nazvaná „Chow and Kaneko“ také dělí obraz na menší části, nestanovuje stejný práh pro celou část obrazu, ale pro každý pixel zlášť. Práh pro daný bod obrazu je stanoven pomocí interpolace prahu z okolních částí obrazu. Nevýhoda této medody je velká výpočetní náročnost.

Alternativní přístup k nalezení lokálního prahu je statisticky zkoumat hodnoty intenzity v okolí každého pixelu. Použitá statistická funkce, která je nejvhodnější, závisí do značné míry na vstupním obrazu.

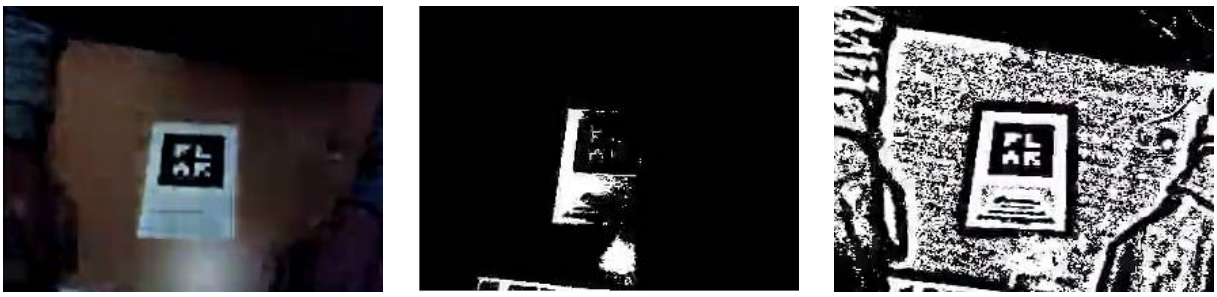
Mezi jednoduché a rychlé funkce patří průměr hodnot intenzity okolí, medián a průměr maximální a minimální hodnoty:

$$T = \frac{min + max}{2} \tag{2.5}$$

Obrázek 2.3: Prahování s optimálním prahem 74. Převzato z [4]

Velikost okolí musí být dostatečně velká, aby pokryla dostatečné množství pixelů z popředí i pozadí, jiank vybraný práh nebude optimální. Na druhou stranu volba okolí, které je příliš velké může být ovlivněno nerovnoměným osvětlením. Tato metoda je méně výpočetně náročná než metoda „Chow a Kaneko“ a přitom dosahuje v některých případech dobrých výsledků. [13]

Na obrázku 2.4c jde pozorovat výhodu použití adaptivního prahování použitého v knihovně FLARToolKit (modifikace knihovny ARToolKit), oproti prahování s globálním prahem 2.4b použitého v původní verzi knihovny ARToolKit (kapitola 3.3).



(a) Původní snímek (b) Globální prahování (c) Adaptivní prahování

Obrázek 2.4: Adaptivní prahování použito v knihovně FLARToolKit. Převzato z [15]

Detekce hran

Hlavní nevýhodou prahování s globálním prahem je dosažení spatných výsledků při nehomogenním osvětlení. Proto knihovna pro rozšířenou realitu ARTag (kapitola 3.5) používá k nalezení vodicích značek metodu založenou na detekci hran.

Hranu v obraze si můžeme představit jako hranici mezi dvěma oblastmi. Můžeme ji také najít na rozhraní světla a stínů. Hrana je vektorová veličina, která je určena velikostí a směrem.

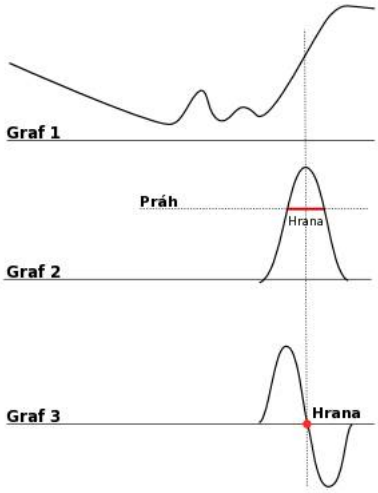
Formálněji se dá definovat hrana jako místo v obraze, kde dochází k velké změně jasové funkce. V tomto místě si můžeme povšimnout, že první derivace funkce jasu má vysokou hodnotu (Obrázek 2.5 Graf 2). Proto je nutné stanovit velikost prahu, který musí derivace v daném bodě přesáhnout, aby byl bod považován za hranu.

Z tohoto vyplývá: „Nejjednoduššími hranovými operátory jsou zjevně derivace $\delta f/\delta x$

a $\delta f/\delta y$, které popisují změnu úrovně jasu ve směru os x a y. Těchto operátorů by bylo možné použít k hledání hran rovnoběžných se souřadnými osami. Při hledání hran obecného směru je zapotřebí vyšetřovat průběh jasu ve směru kolmém na směr potenciální hrany.“ [1] Kvůli jednoduššímu výpočtu se však hrany detekují pouze ve dvou, respektive ve čtyřech směrech.

Při praktické implementaci většinou nepoužíváme pro popis obrazů spojité funkce, ale funkce diskrétní. Snažíme se proto derivace obrazové funkce aproximovat pomocí diferencí realizovaných diskrétní konvolucí. Operátory, které odhadují první derivaci, používají několik masek. Směr gradientu je možno odhadovat hledáním té masky, která odpovídá největší velikosti gradientu.

Druhou možností, jak hledat hrany v obraze, je hledat místo, kde druhá derivace obrazové funkce prochází nulou (Obrázek 2.5 Graf 3). Příkladem je Cannyho hranový detektor.



Obrázek 2.5: Graf 1 Obrazová funkce - Hrana v obraze, Graf 2 První derivace obrazové funkce. Graf 3 - Druhá derivace. Převzato z `http://docs.gimp.org/2.2/cs/filters-edge.html`

Sobelův operátor

Operátorů pro nalezení hran v obraze je celá řada - např. Prewittové, Robinsonův, Kirschův a Sobelův operátor. Jednotlivé operátory se liší především velikostí a koeficienty konvoluční masky. Velikost masky ovlivňuje především reakci na šum v obraze, čím větší konvoluční maska, tím je reakce na šum menší. Koeficenty konvoluční masky ovlivnují reakci na konkrétní vlastnosti obázku např. rychlost růstu gradientu nebo tvar hrany. „Bez znalostí statistických vlastností obrázku nelze předem říci, který z těchto operátorů bude lepší. Velmi často se vhodný operátor vybírá pokusem.“ [9]

Sobelův operátor je většinou definován pomocí dvou konvolučních matic (2.6) velikosti 3x3. Matice G_{x} je použita pro hledání hrany rovnoběžné s vodorovnou osou x, druhá matice G_{y} pro hledání hrany rovnoběžné s osou y. Pokud se na matice podíváme podrobněji, zjistíme, že první matice je pouze otočenou verzí druhé. Teoreticky bychom mohli matici G_{x} postupně otáčet o 45° a tím získat osm matic pro detekci hran v osmi směrech. Pro naše účely nám ale postačí pouze dvě matice 2.6.

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}, G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{2.6}$$

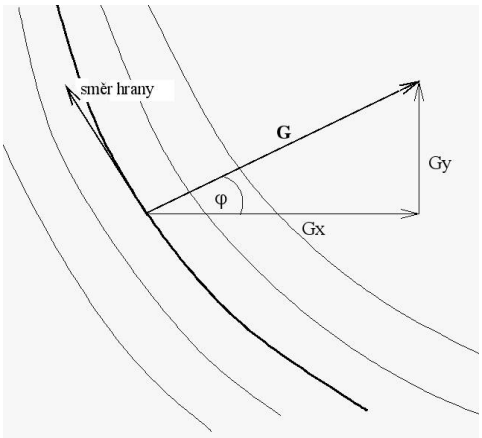
Pokud pomocí konvolučních matic 2.6 vypočteme velikost gradientu ve směru x a y, můžeme poté dle vzorce 2.7 vypočítat velikost gradientu G (Obrázek 2.6). Při implementaci hranového operátoru můžeme výpočet zjednodušit a počítat pouze přibližnou hodnotu gradientu jako součet absolutních hodnot gradientů ve směrech x a y 2.8. Pro porovnávání velikosti gradientu s prahem, který určí zda je bod součástí hrany, nám přibližná přesnost postačí.

$$G = \sqrt{G_x + G_y} \tag{2.7}$$

$$G = |G_x| + |G_y| \tag{2.8}$$

Z obrázku 2.6 můžeme zjistit, že na základě G_x a G_y lze velmi jednoduše vypočítat směr gradientu a tím zároveň směrnici kolmice k samotné hraně.

$$\theta = \arctan \frac{G_x}{G_y} \tag{2.9}$$



Obrázek 2.6: Směr a velikost hrany

Sledování obrysu oblasti

K nalezení obrysu oblasti v obrazu můžeme použít detekci hran a nebo metodu sledování obrysu (angl. Border tracking). Ta je použita v knihovně ARToolKit. Výhodou této metody při použití k nalezení obrysu vodicí značky je, že nám nadetekuje pouze vnější hrany vodicích zanček. Detekce hran najde i hrany, které jsou tvořeny vzorem unitř vodicí zanačky. Ty nám mohou ztěžovat rozhodnutí zda nadetekovaný obrys je či není obrysem vodicí značky a další její zpracování.

Předpokladem použití metody sledování obrysu je, že vstupní obraz je buď binární nebo jednotlivé oblasti jsou označeny. Například pomocí metody „Connected component labelling“ uvedené v kapitole 2.3. Při hledání obrysu můžeme hledat vnitřní a nebo vnější

obrys oblasti. Vnitřní obrys oblasti je vždy součástí oblasti, naopak vnější obrys není nikdy součástí oblasti.

Následující algoritmus definuje sledování obrysu pomocí čtyřokolí i osmiokolí:

1. Prohledávej obrázek z horního levého rohu, dokud nenarazíš na pixel, který je součástí nějaké oblasti. Tento pixel P_0 je prvním bodem obrysu oblasti. Definuj proměnnou dir, která uchovává směr předchozícho posunu podél okraje z předchozího bodu okraje na aktuální bod okraje. Přiřaď:

   (a) $dir = 3$, pokud okraj hledáme v čtyřokolí (Obrázek 2.7a),

   (b) $dir = 3$, pokud okraj hledáme v osmiokolí (Obrázek 2.7b).

2. Prohledej okolí $3x3$ pixely aktuálního bodu okraje proti směru hodinových ručiček. Začni od bodu, který leží od aktuálního bodu ve směru:

   (a) $(dir + 3)\, mod\ 4$, pokud okraj hledáme v čtyřokolí (Obrázek 2.7c),

   (b) $(dir + 7)\, mod\ 8$, pokud okraj hledáme v osmiokolí a aktuální směr dir je sudé číslo (Obrázek 2.7d),

   (c) $(dir + 6)\, mod\ 8$, pokud okraj hledáme v osmiokolí a aktuální směr dir je liché číslo (Obrázek 2.7e).

   První bod, který najdeš se stejnou hodnotou jako aktuální bod, označ jako nový aktuální bod okraje P_{n}. Aktualizuj proměnnou dir.

3. Pokud aktuální bod okraje P_{n} je roven druhému bodu okraje P_1 a předchozí bod okraje $P_{\text{n-1}}$ je roven prvnímu bodu okraje P_0, pak pokračuj krokem 4. Jinak pokračuj krokem 2.

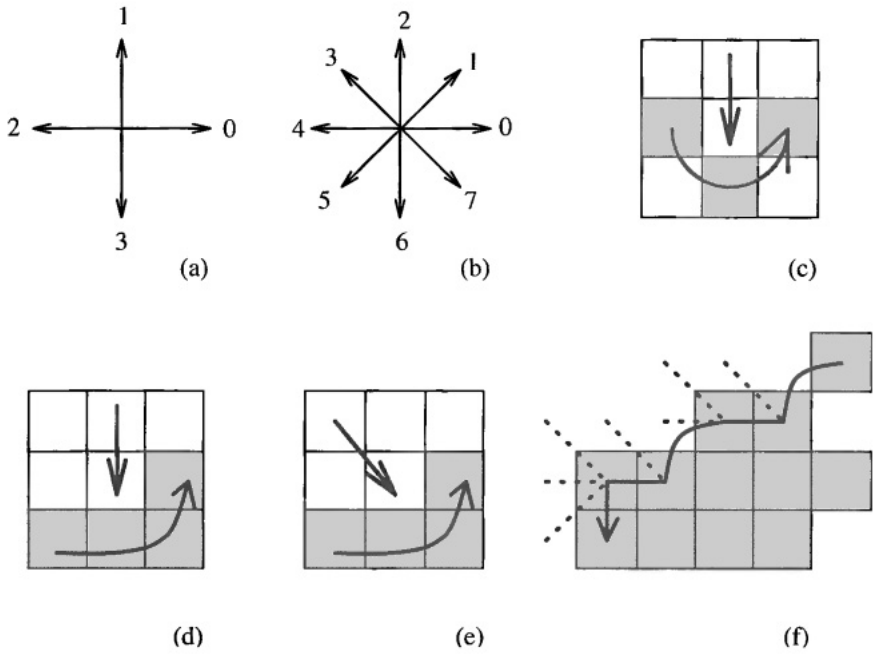
4. Nadetekovaný vnitřní obrys oblasti je reprezentován pomocí pixelů $P_0 \ldots P_{\text{n-2}}$

Tento algoritmus funguje pro všechny oblasti větší než jeden pixel. Algoritmus je schopný naleznout okraje oblasti, ale není schopen naleznout okraje výřezu v dané oblasti. Pokud chceme naleznout okraje výřezu, je potřeba aby sledování obrysu bylo zahájeno pro každý okrajový bod oblasti nebo výřezu, který ještě není součástí žádného obrysu. Sledování dalšího obrysu musí být zahájeno až v okamžik, když sledování předchozího obrysu bylo dokončeno. A může pokračovat ve stejném směru jako hledání prvního pixelu první oblasti.

V případě, že chceme najít vnější obrys oblasti, provedeme hledání vnitřního obrysu za pomocí čtyřokolí. A za body vnější oblasti označíme ty body, které nejsou součásti dané oblasti, ale byly testovány zdali leží uvnitř oblasti.[10]

2.3 Shluková analýza

„Shluková analýza (též clusterová analýza, anglicky cluster analysis) je vícerozměrná statistická metoda, která se používá ke klasifikaci objektů. Slouží k třídění jednotek do skupin (shluků) tak, aby si jednotky náležící do stejné skupiny byly podobnější než objekty ze skupin různých. Shlukovou analýzu je možné provádět jak na množině objektů, z nichž každý musí být popsán prostřednictvím stejného souboru znaků, které má smysl v dané množině sledovat, tak na množině znaků, které jsou charakterizovány prostřednictvím určitého souboru objektů, nositelů těchto znaků.“ [12]



Obrázek 2.7: Sledování vnitříno obrysu. (a) Označení směru u čtyřokolí. (b) Označení směru u osmiokolí. (c) Pořadí prohledávání sousedních pixelů pro čtyřokolí. (d),(e) Pořadí prohledávání sousedních pixelů pro osmiokolí. (f) Hledání okraje oblasti pro osmiokolí (tečkované čáry značí testované body při sledování okolí oblasti). Převzato z [10]

Nyní si popíšeme shlukovací metodu s název „Connected component labelling“, která je použita v knihovně ARToolKit.

Connected component labelling

Connected component labelling (též algoritmus barvení, nebo detekce souvislých oblastí) je metoda, která zpracovává vstupní binární obraz tak, že *souvislé oblasti* označí stejným indexem. Pro každý vstupní pixel označen hodnotou 1 je tedy přiřazena nová hodnota reprezentující unikátní index dané oblasti.

Formální definice *souvislé oblasti* je:
Předpokládejme, že B je binární obrázek a že $B(r,c) = B(r',c') = v$, kde $v = 1$ nebo $v = 0$. Mezi pixely (r,c) a (r',c') vede *spojitá cesta* při zachování hodnoty v pokud existuje posloupnost pixelů $(r,c) = (r_0,c_0),(r_1,c_1),...,(r_n,c_n) = (r',c')$, kde platí, že $B(r_i,c_i) = v$ pro $i = 0,...,n$ a (r_i,c_i) sousedí s (r_{i-1},c_{i-1}) pro $i = 1,...,n$. Posloupnost $(r_0,c_0),(r_1,c_1),...,(r_n,c_n)$ definuje *spojitou cestu* z pixelu (r,c) do (r',c'). *Souvislá oblast* pro hodnotu v je množina pixelů C, mající hodnotu v a mezi každou dvojicí pixelů existuje *spojitá cesta*. [2]

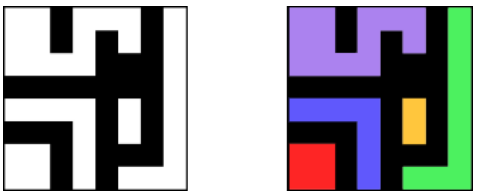
V knihovně ARToolKit (kap. 3.3) je tato metoda používána pro seskupení pixelů reprezentujících vodicí značku ve videu.
Algoritmus pro nalezení souvislých oblastí se dá implementovat rekurzivně nebo dvojprůchodově.

1	1	0	1	1	1	0	1
1	1	0	1	0	1	0	1
1	1	1	1	0	0	0	1
0	0	0	0	0	0	0	1
1	1	1	1	0	1	0	1
0	0	0	1	0	1	0	1
1	1	0	1	0	0	0	1
1	1	0	1	0	1	1	1

(a) Binární obrázek

1	1	0	1	1	1	0	2
1	1	0	1	0	1	0	2
1	1	1	1	0	0	0	2
0	0	0	0	0	0	0	2
3	3	3	3	0	4	0	2
0	0	0	3	0	4	0	2
5	5	0	3	0	0	0	2
5	5	0	3	0	2	2	2

(b) Označené souvislé oblasti



(c) Binární obrázek a označené souvislé oblasti zobrazeny jako obraz

Obrázek 2.8: Binární obrázek s pěti souvislými oblasti. Převzato z [2]

Kapitola 3

Rozšířená realita

„Rozšířená realita (anglicky augmented reality) představuje mezistupeň mezi realitou skutečnou a realitou virtuální. Rozšířená realita je doplněním obrazu skutečnosti o uměle doplněné obrazce či jiné informace. V současnosti je nejčastějším provedením rozšířené reality zobrazení skutečného obrazu na displeji a jeho doplnění o počítačem dodané informace.“ [11] Jedním z požadavků na rozšířenou realitu je, aby virtuální objekty byly přidávány do scény v reálném čase.

Tato kapitola popisuje možné způsoby realizace rozšířené reality včetně příkladů použití v komerčních i nekomerčních aplikací. Pro jednodušší programování aplikací rozšířené reality je zde uveden popis několika knihoven používaných při vytváření systémů pracujících s rozšířenou realitou.

3.1 Realizace rozšířené reality

Rozšířená realita se dá realizovat dvěma základními způsoby:

- Zobrazáním scény pomocí klasického monitoru
- Použit pro zobrazení speciální zařízení HMD

První způsob realizace se začíná rozširovat daleko rychleji, protože není potřeba žádného speciálního zařízení. Uživatelovi postačí klasický monitor a USB kamera, která bývá nejčastěji umístěna přímo na monitoru. Kamera snímá prostor před monitorem a po zobrazení vytváří efekt kouzelného zrcadla. Místo monitoru může být samozřejmě použito i jiné zařízení například projektor. (Obrázek 3.1)

Druhý způsob je realizován pomocí průhledového náhlavního displeje HMD (Head Mounted Display). Kamera je umístěna na speciálních brýlích a snímá to, co uživatel vidí před sebou. Aplikace video zpracuje, přidá virtuální objekty a zobrazí uživatelovi rožířenou scénu tak jak ji vidí před sebou.

Existují dva typy náhlavních displayů:

- Opticky průhledové (Optical See-Through HMD)
- Video průhledové (Video See-Through HMD)

Opticky průhledové displaye (Obrázek 3.2a) pracují tak, že virtuální objekty jsou zobrazovány na polopropustné zrcadlo, přes které se uživatel dívá.

Obrázek 3.1: ARTag "Magic Mirror". Převzato z [8]

Video průhledové HMD (Obrázek 3.2b) pracují tak, že kamerou nasnímaná scéna je rozšířena o virtuální objekty a uživatelovi je scéna zobrazena pomocí 3D brýlí, které obsahují malou obrazovku umístěnou těsně před očima uživatele.

Opticky průhledové HMD jsou podstatně složitější než video průhledové HMD a vyžadují také podstatně složitější kalibraci. K demonstraci činosti aplikace, která bude v rámci této diplomové práce vyvinuta bude použito video průhledové HMD.

(a) Opticky průhledové HMD

(b) Video průhledové HMD

Obrázek 3.2: Průhledové náhlavní displeje - HMD. Převzato z `http://virtual.lncc.br/ rodrigo/links/AR/node6.html`

3.2 Příklady použití rozšířené reality

Rozšířená relalita velmi rychle získává popularitu, ale je pořád novou technologií. Většina ze současných aplikací je stále ve fázi vývoje univerzit nebo větších firem. Ale najde se již

pár komerčních aplikací. [14]

Aplikace využívající rozšířenou realitu se mohou uplatnit v mnoha oborech při různých příležitostech. A to od herního průmyslu, přes opravu aut až po módní průmysl.

ARQuake

ARQuake je modifikace známé hry Quake využívající rozšířenou realitu pro hraní hry. Uživatel se může procházet v reálném světě a hrát hru Quake proti virtuálním nestvůrám. Jako periferní zařízení pro olvádání hry je voleno zařízení HMD, přenosný počítač, detektor pohybu hlavy a GPS zařízení.[6] Ukázka ovládání hry je na obrázku 3.3 a ukázka ze hry na obrázku 3.4.

Obrázek 3.3: Zařízení pro hraní hry ARQuake. Převzato z [6]

Obrázek 3.4: Hra ARQuake. Převzato z [6]

Reklamní průmysl

V tomto odvětví se objevuje nejvíce komerčních aplikací. Jedná se především o aplikace, které propagují určitý produkt. Na obrázku 3.5 si můžeme prohlédnout aplikaci „iQ Toyota

Augmented Reality“, která zobrazuje 3D model auta Toyota iQ, který si můžete rozložit do poslední součástky doma na Vašem stole.

Obrázek 3.5: iQ Toyota Augmented Reality. Převzato z `http://www.toyota.co.uk/cgi-bin/toyota/bv/frame_start.jsp?id=iQ_reality`

V některých vybraných prodejnách s hračkami (bohužél prozatím ne v české republice) se začalo oběvovat zařízení s názvem „Lego Digital Box Kiosk“ (Obrázek 3.6), které po přiložení krabice s legem před kameru zobrazí zákazníkovi animovaný 3D model stavebnice v jeho rukách.

Obrázek 3.6: Lego Digital Box Kiosk zobrazuje 3D model stavebnice lega. Převzato z `http://www.notcot.com/archives/2009/01/legos_digital_b.php`

3.3 ARToolKit - knihovna pro rozšířenou realitu

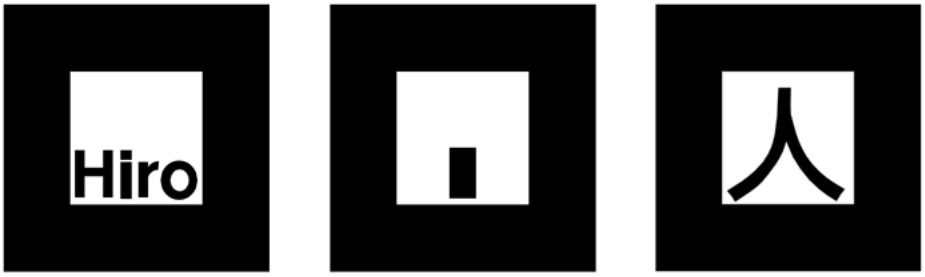
ARToolKit (Augmented Reality Toolkit) je multi-platformní C,C++ knihovna pro snadnější programování aplikací rozšířené reality. Pomocí technik počítačového vidění ARToolKit vypočítává pozici a orientaci kamery relativně vůči vodicí značce tzv. markeru. Z čehož jsme

schopni vypočítat, kde se nachází a kam se dívá uživatel. Toho využijeme k vykreslení virtuálních objektů do snímané scény. Díky neustálému trendu zvyšování výkonnosti osobních počítačů jsme schopni detekovat značky a vykreslovat virtuální objekty v reálném čase.

Knihovna ARToolKit je aktuálně dostupná ve variantách Standard a Professional. Verze ARToolKit Standard v.2.x je dostupná pro nekomerční účely zdarma pod licencí GNU. A momentálně již bohužel není aktivně dále vyvíjena. ARToolKit Professional v.4.x je placená verze určená pro komerční účely. Tato verze má zlepšenou přesnost určování pozice a orientace kamery či podporu pro kamery s vyšším rozlišením a novými video formáty. Dále podporuje markery s optickým kódem, které mají unikátní ID zakódováno v jejech vzoru. Pro tuto semestrální práci budeme využívat knihovnu verze ARToolKit Standard v.2.72.1.

Základní pricip ARToolKitu

Základním principem ARToolKitu je detekce vodicích značek (angl. tracking markers) ve videu. vodicí značky jsou reprezentovány pomocí čtverce s tlustým černým okrajem, který obsahuje určitý vzor. Pokud chceme dosáhnout dobrých výsledků při detekci, tak vzor by měl být asymetrický a neměl by obsahovat příliš mnoho jemných detailů. Obrázek 3.7 zobrazuje příklady vodicích značek.



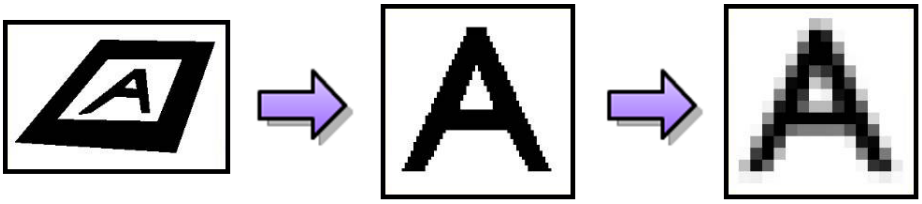
Obrázek 3.7: Ukázka vodicích značek ARToolKitu

Detekce vodicích značek se dá rozdělit na následující kroky:

- Pomocí kamery je snímáno video reálného světa a po jednotlivých snímcích je zpracováváno.
- Nad vstupními snímky je prováděno **prahování** (viz kapitola 2.2) s konstantním prahem, který je předem stanoven v intervalu 0-255. ARToolKit pracuje s konstantním prahem, což může být problém, pokud osvětlení místnosti není konstantní. Výstupem je binární obraz.
- Dalším krokem je **označení souvislých segmentů** pomocí algoritmu conected component labeling (viz kapitola 2.3). Výstupem tohoto kroku je seznam souvislých segmentů.
- Ze seznamu souvislých oblastí jsou metodou **sledování obrysu oblasti** (viz kapitola 2.2) získány obrysy jednotlivých oblastí.

- Z obrysů všech segmentů se vyberou ty obrysy, které je možné reprezentovat pomocí čtyř úseček. Tyto přímky nám ohraničují detekovanou vodicí značku. Výstupem tohoto kroku je čtveřice rovnic definující přímky.
- Na základě rovnic přímek jsou vypočteny rohy značky detekované v obraze.
- Jelikož pohled na vodicí značku je vždy z jiného úhlu a vzdálenosti je potřeba před samotnou identifikací značku normalizovat. Tedy správně natočit a zmenšit velikost na 64x64 pixelů. Normalizace je znázorněna na obrázku 3.8. V tomto kroku se také řeší odstranění zkreslení kamery.
- Normalizovaná vodicí značka se postupně porovnává se všemi zaregistrovanými vzory vodicích značek. Mezi kterými se hledá značka, která je nadetekované vodicí značce nejvíce podobná. Porovnávání probíhá pro natočení vodicí značky o 0°, 90°, 180°a 270°. Z důvodu požadavku na rychlost porovnávání se normalizovaná vodicí značka a jednotlivé vzory zmenšují na velikost 16x16 pixelů.
- Na základě rovnic definujících přímky ohraničující detekovaný vzor se vypočte pozice a orientace vodicí značky. Odkud dále je možno vypočítat pozici kamery relativně vůči vodicí značce.
- Nyní již když máme transformační matici. Vykreslíme virtuální objekt a pomocí transformace jej umístíme na správné místo ve videu.

Postup je znázorněn na obrázku 3.9.



Obrázek 3.8: Normalizace a zmenšení pro detekci vzorů. Převzato z [3]

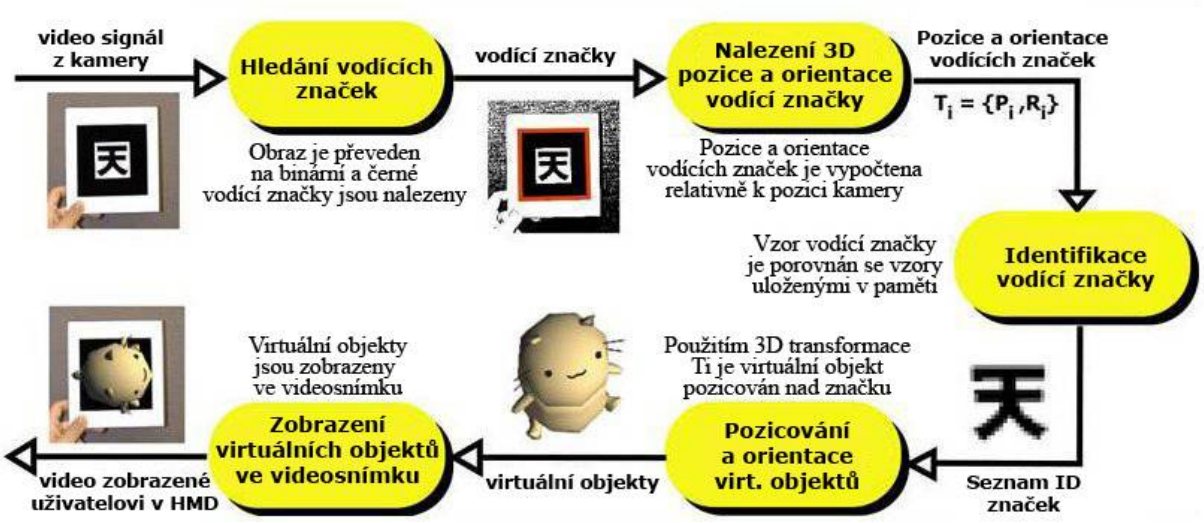
3.4 Modifikace knihovny ARToolkit

Na knihovně ARToolKit Standard je postaveno několik modifikací. Všechny uváděné modifikace jsou distriuované jako „open source“ a pro nekomerční účely zdarma k použití. Licence pro komerční využití modifikací vlastní ARToolWorks, tak jako základní licenci pro ARToolKit Standard při komerčním využití.

ARToolKit Plus

ARToolKitPlus je rozšířená verze knihovny ARToolKit, která zlepšuje některé vlastnosti původní knihovny. Z důvodu změny aplikačního rozhraní na objektově orientovaný návrh není tato knihovna již zpětně kompatibilní s původní knihovnou. V roce 2006 byl vývoj této knihovny ukončen a nahrazen knihovnou „Studierstube Tracker“, která ale již není nabízena zdarma ke stažení.

Hlavní výhody oproti knihovně ARToolKit:



Obrázek 3.9: Princip činosti knihovny ARToolKit. Převzato z [7]

- Objektově orientovaný návrh aplikačního rozhaní (API)
- Detekce až 4096 vodicích značek se zakódovaným ID ve vzoru bez zpomalení výkonu (obrobně jako ARTag viz. kapitola 3.5).
- Nové formáty kamery (RGB565, Gray)
- Podpora různé tloušťky okraje vodicí značky
- Stabilnější sledování vodicí znčky (značka se méně chvěje)
- Mnohé urychlení pro podporu méně výkoných zařízení
- Automatické nastavení prahu pro globální prahování

NyARToolKit

NyARToolKit je portovaná verze ARToolkitu původně do programovacího jazyka Java, následně do C#, C++ a na platformu Android.

FLARToolKit

FLARToolKit je portovaná verze ARToolkitu do programovacího jazyka Action Script 3[1]. Tato verze není založena přímo na původní verzi ARToolkitu, ale na modifikaci NyARToolKit.

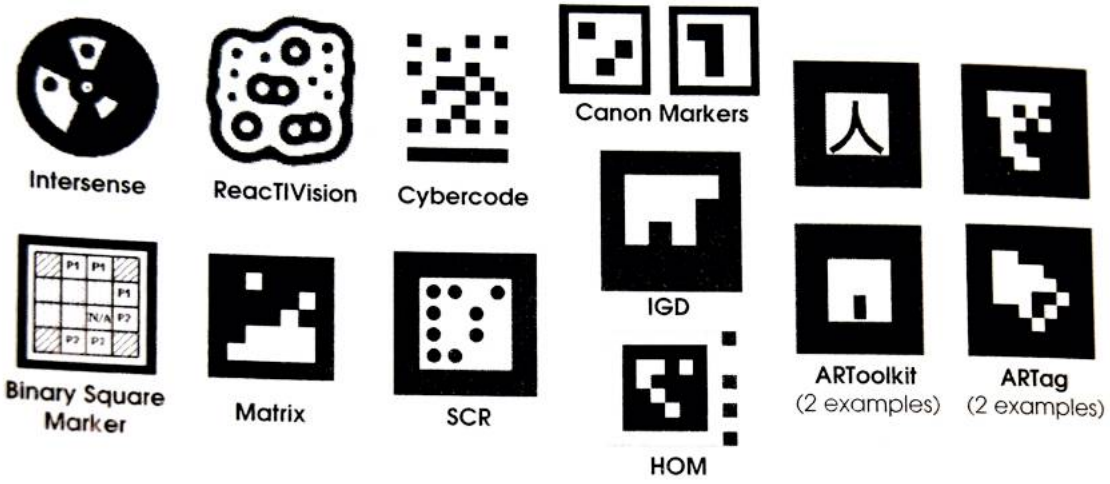
FLARToolKit pro segmentaci vodicích značek nepoužívá globálním prahování, ale adaptivní prahování[2] (Kapitola 2.2). Pro výpočet lokálního prahu je použit filtr pro rozostření (BlurFilter). Hlavní výhodou adaptivního prahování rozdílem je lepší detekce vodicích značek při nekonzistentním osvětlení(Obrázek 2.4), bohužél za cenu snížené výkonosti. [15]

[1] Skriptovací jazyk pro tvorbu webových stránek využívající program Adobe Flash
[2] Adaptivní prahování v FLARToolKit je implementováno od revize 2570.

3.5 Jiné systémy pro rozšířenou realitu

Systémů, které využívají principy počítačového vidění k detekci vodicích značek a výpočtu pozice kamery je hned několik. Příklad několika podobných knihoven a jejich vodicích značek je zobrazen na obrázku 3.10.

Většina z těchto systémů je ve fázi výzkumného vývoje a jejich zdrojové kódy nejsou zveřejněny k volnému použití. Například Binary Squere Marker a Matrix. Ostatní jako Intersense (s kruhovými značky), Canon Markers či SCR a IGD z projektu ARVIKA jsou chráněny autorskými právy a není možno je použít ve svých vlastních aplikacích. [14].



Obrázek 3.10: Systémy pro sledování vodicích značek. Převzato z [14]

ARTag

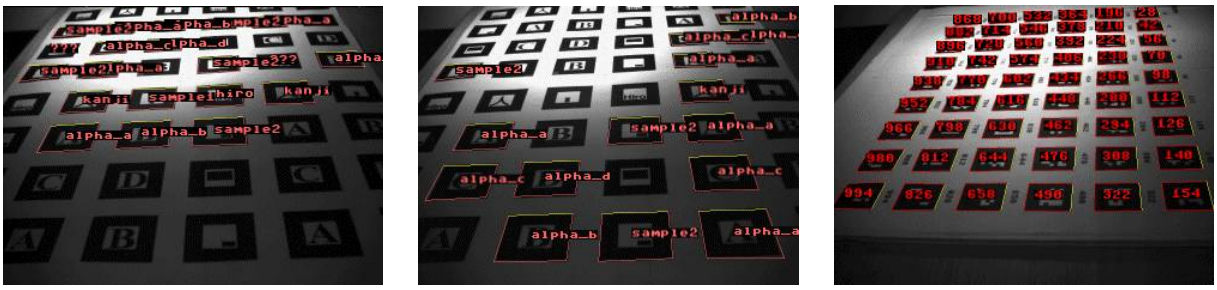
ARTag je knihovna, která byla vyvinuta několik let po vydání původní verze knihovny ARToolkit. Jejím autrem je Mark Fiala. Díky tomu, že byla vyvíjena později, využívá zvýšeného výpočetního výkonu dnešních počítačů a může tedy používat složitější algoritmy pro detekci vodicích značek v reálném čase. Bohužel v okamžik kdy Mark Fiala přestal pracovat v NRC[3] se knihovna přestala dále vyvíjet. Momentálně je možné knihovnu stáhnout pouze pro nekomerční pouze v okamžik, pokud vlastníte knihu „Agmented Reality: A Practical guide“ [14], které pojednává o této knihovně.

ARTag zlepšuje (snižuje) pravděpodobnost chybného nadetekování vodicí značky či pravděpodobnost záměny vodicích značek navzájem. Pro detekci vodicích značek nepoužívá prahování, ale metodu detekci čtyřúhelníků založenou na detekci hran.

Na obrázku 3.11 je vidět detekce vodicích značek při nerovnoměrném osvětlení. První dva obrázky (3.11a a 3.11b) demonstrují jak knihovna ARToolkit při použití metody prahování není schopna rozpoznat všechny vodicí značky současně. Obrázek 3.11c ukazuje jak ARTag rozpoznal všechny vodicí značky. Rozpoznané vodicí značky jsou označeny pomocí ohraničujícího rámečku a ID značky.

Každá vodicí značka ARTagu je čtvercová mřížka s rozměry 10x10 buněk složená z okraje širokého 2 buňky a z vnitřní mřížky o rozměrech 6x6. Okraj vodicí značky může být černý

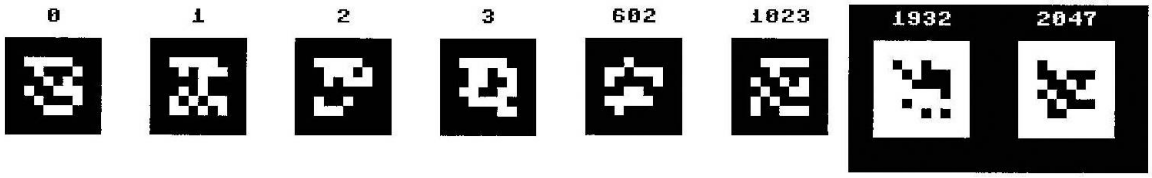
[3]National Research Council of Canada's Institute of Information Technology



(a) ARToolKit (b) ARToolKit (c) ARTag

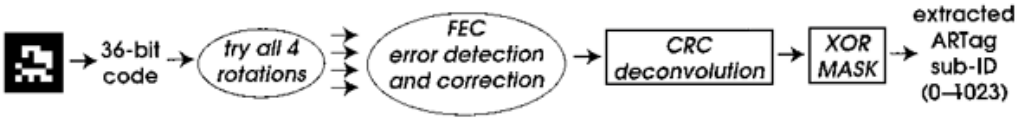
Obrázek 3.11: Porovnání ARTag a ARToolkit při nerovnoměrném osvětlení. Převzato z [8]

na bílém pozadí (ve většině případů) a nebo bílý na černém pozadí. Vnitřní buňky jsou černé nebo bílé a dohromady tvoří vzor značky, který je tvořen tak, aby jednotlivé vzory byly na sebe co nejméně podobné a při otočení vzoru nešlo k záměně za jiný vzor. Ukázka vodicích značek ARTagu s ID, které je reprezentováno pomocí vzoru značky je na obrázku 3.12.



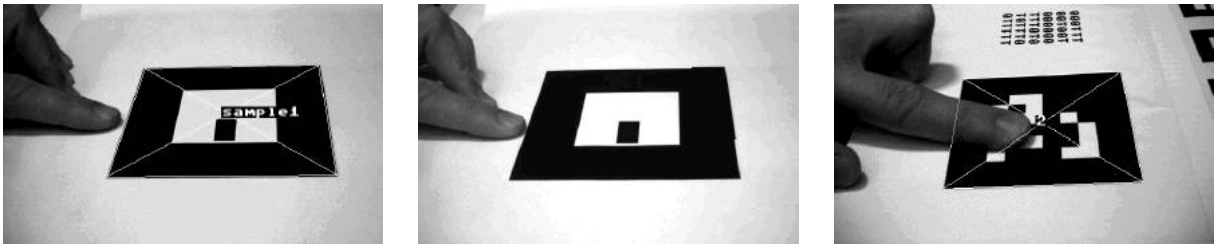
Obrázek 3.12: vodicí značky ARTagu s přiřazeným ID zakódovaným ve vzoru. Převzato z [14]

Každá vnitřní buňka reprezentuje binární hodnotu 1 nebo 0 a dohromady tvoří 36-bitový kód. Tento kód je po nadetekování za pomocí FEC (Forward Error Correction) a 16-bitového kontrolního převeden na 10-bitové ID jednoznačně určující vodicí značku. FEC je metoda používaná k zjištění a opravě chyb vzniklých pri přenosu pomocí doplňkových kontrolních bytů. Jednotlivé části zpracování jsou zobrazeny na obrázku 3.13. Tento krok snižuje pravděpodobnost záměny značek za jiný objekt ve scéně, či záměnu značek navzájem.



Obrázek 3.13: Proces zpracování nadetekované značky pomocí ARTag. Převzato z [14]

Obrázek 3.14a a 3.14b znázorňuje jak je ARToolKit není schopen nadetekovat vodicí značku v okamžik kdy jen malá část vzoru je překryta anebo se jen okraje vzoru dotýká cizí předmět. ARTag díky kódu, který je uložen ve vzoru značky, je schopen rozpoznat vodicí značku i v případě že poměrně velká její část je zakryta. Obrázek 3.14c.



(a) ARToolKit - vzor rozpoznán (b) ARToolKit - vzor nerozpoznán (c) ARTag - vzor rozpoznán

Obrázek 3.14: Porovnání ARTag a ARToolkit při zakritím části značky. Převzato z [8]

Kapitola 4

Google SketchUp

Google SketchUp je program pro tvorbu, úpravu a sdílení 3D modelů. Je vhodný pro návrh budov, nábytku, modelů pro aplikaci Google Earth či zobrazení jakýchkoli vašich nápadů ve 3D. Program je velmi intuitivní a naučit se vytvářet 3D modely většině lidí trvá pouze pár hodin. Ukázka modelu bytu pomocí Google SketchUp je na obrázku 4.1.

Obrázek 4.1: Ukázka modelu bytu v Google SketchUp

V rámci diplomové práce jsem se rozhodl použít Google SketchUp nejprve k tvorbě modelů bytu, nábytku a bytových doplňků. A předpokládal, že výsledná aplikace by poté jednotlivé modely načítala a zobrazovala ve videosekvenci a tím by vytvářela obraz rozšířené reality.

Později se ukázalo, že vyvíjet vlastní aplikaci, která by načítala datové soubory s uloženými modely, následně je zobrazovala a umožňovala jejich pozicování v bytě, by bylo příliž nákladné. A zároveň pro diplomovou práci by byla tato aplikace velmi málo přínosná s po-

rovnáním k její složitost.

Rozhodl jsem se tedy experimentovat s aplikací Google SketchUp nejen pro vytváření modelů, ale i pro jejich samotné zobrazování ve videu. V následujících podkapitolách je popsáno programové rozhraní (API) programu Google SketchUp, které umožňuje tvorbu zásuvných modulů (pluginů) a vytváření vlastních systémů nad touto aplikací.

4.1 Goole SketchUp SDK

Google SketchUp SDK (Software Development Kit) je první ze dvou možností jak programově pracovat s aplikací Google SketchUp. SDK je balíčkem C++ knihoven skládajícím se z knihoven SkpReader a SkpWriter umožňující čtení a zápis datových souborů aplikace Goole SketchUp. Pomocí nichž je možno vystavět vlastní aplikace zobrazující nebo vytvářející modely z Goole SketchUp. Další možností využití těchto knihoven je vytvoření zásuvných modulů pro jiné 3D editory umožňující import a export do datového formátu skp.

4.2 SketchUp Ruby API

Druhou možností jak programově pracovat s aplikací Google SketchUp je SketchUp Ruby API. Jedná se o možnost vytvářet zásuvné moduly napsané v programovacím jazyce Ruby Script. Pomocí nichž je možné vytvářet vlastní kreslící nástroje, generovat složité grafické objekty na jedno klinutí tlačítka myši, ovládat kameru a vytvářet animace.

Na konferenci Google I/O byl dokonce představen koncept arkádové hry Prince IO vytvořené pomocí SketchUp Ruby API. Ukázka je zobrazena na obrázku 4.2.

Obrázek 4.2: Prince IO - koncept hry vytvořené pomocí SketchUp RubyAPI

Ruby Script

Ruby Script je interpretovaný skriptovací programovací jazyk, který plně podporuje objektově orientované programování. Jeho autorem je Yukihiro Matsumoto, který se inspiroval

v programovacích jazycích Perl, Smalltalk, Eiffel, Ada, a Lisp k tomu, aby vytvořil nový programovací jazyk kombinující prvky funcionálního a imperativního programovacího paradigma. Jeho cílem bylo vytvořit programovací jazyk, který bude přirozený, nikoliv jednoduchý. Tak aby šly uplatnit co nejlépe techniky agilního programování. První verze byla zveřejněna v roce 1995. Dnes je Ruby nejvíce používán v Japonsku (země svého původu). Svému rozšíření dlouho dobu bránila absence kvalitní dokumentace v anglickém jazyce.

TCP Sockets v Google SketchUp

Při vytváření zásuvného modulu v rámci mojí diplomové práci bylo potřeba, aby vytvořený systém pracoval s knihovnou ARToolkit - knihovnou pro rozšířenou realitu. Pro komunikaci s touto knihovnou jsem se rozhodl navrhnout systém jako aplikaci typu server-klient. Knihovna ARToolkit bude tvořit server a Google SketchUp se bude k tomu serveru připojovat jako klient.

Při práci se síťovým protokolem TCP/IP nejde ve SketchUp Ruby API standardně použít Ruby knihovna TCPSocket. Knihovna TCPSocket totiž vytváří synchronní spojení a mezi odesláním a přijetím požadavku na server aplikace Google SketchUp nereaguje na uživatelský vstup. Následně po přijetí zprávy ze serveru je potřeba spojení ukončit a po zpracování odpovědi a zobrazení jejího výsledku spojení znovu vytvořit. Tento způsob komunikace se serverem je velmi pomalý a výsledný výsledek vypadá, jakoby aplikace Google SketchUp vůbec nereagovala.

Pro síťovou komunikaci má SketchUp Ruby API svoji vlastní třídu SKsocket využívající TCP/IP sockety. Tato třída je velmi limitována svým využitím a neexistuje k ní žádná dokumentace. UML diagram třídy je na obrázku 4.3.

Třída obsahuje pouze čtyři statické metody, takže není dovoleno vytvářet více než jedno aktivní spojení. První metodou je metoda *connect*. Tato metoda vytvoří obousměrné spojení se serverem na specifikovaném portu. Vytvořené spojení je asynchronní a pomocí funkce *add_socket_listener* můžeme zdefinovat odkaz na funkci, která bude zavolána v okamžik přijetí odpovědi od serveru. Funkce *write* odešle požadavek na server a funkce *disconect* zruší spojení se serverem.

Vytvořené TCP/IP spojení pomocí třídy SKsocket je možné ponechat otevřené po celou dobu kominikace mezi serverem a klientem a není tedy potřeba po každé odpovědi server uzavírat a znovu se k němu připojovat. Tím se výrazně komunikace mezi serverem a klientem urychlí.

Obrázek 4.3: UML diagram třídy SKSocket pro síťovou komunikaci v Google Sketchup

Distribuce zásuvných modulů

Před použitím zásuvných modelů je potřeba, aby byly načteny do aplikace Google SketchUp. Nejjednodušší cestou jak načít jednotlivé moduly je umístit soubory Ruby Scriptu

do adresáře Plugins. Standardní umístění se nachází v „C:\Program Files\Google\Google SketchUp [n]\Plugins“. Jednotlivé skripty musí mít příponu .rb nebo .rbs.

Jelikož Ruby Script je interpretovaný jazyk a tudíž není kompilován. Je možné šířit přímo jeho zdrojové soubory (s příponou .rb). V případě, že nechceme šířit zásuvný modul jako open source můžeme použít program „scrambler“, který zdrojový kód zašifruje, tak aby nebyl standardním způsobem člověku čitelný. Přípona zašifrovaných souborů poté musí být .rbs.

Kapitola 5

Návrh systému

Cílem diplomové práce bylo navrhnout systém, který bude za pomocí knihovny ARToolkit experimentovat s rozšířenou realitou při návrhu bytu. Systém jsem navrhl jako zásuvný modul (plugin) do 3D modelovacího programu Google SketchUp. Principem činosti je na základě vodicích značek zjistit pozici uživatele v bytě a podle této pozice nastavovat kameru v programu Google SketchUp. Přičemž na pozadí Google SketchUp bude zobrazováno video z kamery.

Předpokladem systému bude, že vždy bude ve videu minimálně jeden vodící znak v dostatečně dobré kvalitě tak, aby mohl být nadetekován pomocí knihovny ARToolKit.

5.1 Obecná aplikace využívající rozšířenou realitu

Princip většiny aplikací, využívající prvky rozšířené reality, se dá rozdělit na několik základních úkolů. Schématické znázornění těchto úhloh je zobrazeno na obrázku 5.1.

Obrázek 5.1: Schematické znázornění principu činnosti aplikace rozšířené reality

Získání videosignálu z kamery

V každé aplikaci pracující s rozšířenou realitou je nejprve potřeba načítat jednotlivé snímky z kamery. Jelikož většina knihoven pro práci s rozšířenou realitou (např. ARToolkit, ARTag) v sobě již danou funkčnost obsahuje, není potřeba na načítání snímků z kamery používat další externí knihovny. Jako by například mohla být knihovna OpenCV - jedna z nejpoužívanějších knihovnen pro zpracování obrazu a videa.

Detekce výrazných bodů ve videu

Aby bylo možné do jednolivých snímků získaných z kamery vkreslovat virtuální objekty, je potřeba nejprve detekovat některé výrazné body ve snímcích. A nad tyto body poté vhodně umístit virtuální objekty.

Detekce výrazných bodů ve videu může být založena na detekci vodicích značek s předem definovaným tvarem a vzorem.

A nebo na detekci a sledování příznaků z obrazu, jako jsou hrany či izolované body. Mezi tyto metody sledování patří např. metoda SLAM (Simultaneous localization and mapping) nebo PTAM (Parallel tracking and mapping). Jejich hlavním rozílem je, že PTAM nepotřebuje žádnou předchozí informaci o snímaném prostředí. Tím pádem není potřeba žádných vodicích značek, jelikož významné body ve videu jsou detekovány automaticky bez zásahu uživatele. Je potřeba pouze specifická konfigurace daného systému.

Pro náš systém budeme využívat podle zadní diplomové práce způsob první, založený na detekci vodicích značek pomocí knihovny ARToolkit. Při návrhu systému budeme ale postupovat tak, aby detekce uživatelovy pozice šla vždy zaměnit za jinou metodu.

Výpočet pozice a orientace kamery

Po nadetekování výrazných bodů ve videosnímcích máme pouze informaci o 2D umístění daných bodů v rámci snímku. Z těchto bodů je potřeba vypočítat 3D informaci o pozici vodicích značek, vůči kterým poté budou do videa umísťovány virtuální objekty.

Knihovna ARToolkit tento výpočet provádí na základě perspektivní projekce, kdy původně kolmé hrany vodicí značky jsou perspektivní projekcí ovlivněny a již nesvírají pravý úhel. Metody PTAM nejčastěji používají pro výpočet metodu triangulace, kde je ze dvou různých snímků stejných výrazných bodů vypočtena informace o vzdálenosti daného bodu od kamery.

Získání modelu vituálního objektu

V okamžiku, kdy již máme pozici záchytných bodů, potřebujeme získat model virtuálního objektu, který budeme následně zobrazovat. Pro vytvoření modelu můžeme využít jakýkoli 3D modelovací program, který dokáže exportovat model do formátu, který je otevřený a není tedy problém s jeho načtením.

Pro svou jednoduchost jsem pro vytváření 3D modelů zvolil program Google SketchUp. Datový formát .skp, do kterého ukládá svoje modely Google SketchUp, je možné programově číst pomocí C++ knihovny, která je součástí Google SetchUp SDK (kapitola 4.1).

Zobrazení virtuálních objektů ve videu

Jako prostředek pro zobrazení virtuálních objektů ve videu se nejčastěji používá OpenGL. Knihovna ARToolkit poskytuje dvě možnosti, jak využít OpenGL. První možností je, že

o vytvoření grafického okna aplikace se stará přímo ARToolkit. Druhým způsobem využití ARToolkit je, že knihovna vrátí pouze 3D transformační matici reprezentující transformaci mezi markerem a kamerou a o inicializaci grafického okna a o celý process zobrazení se stará programátor.

Při prozkoumání existujících aplikací, které dokáží zobrazovat modely aplikace Google SketchUp jsem zjistil, že jich existuje velké množství. Většina se zaměřuje na fotorealistické zobrazení modelů. Příkladem můžou být aplikace: Render Plus, 3DPaintBrush Indigo Renderer nebo VRay. Některé z těchto rendererů také obsahují malý plugin do aplikace Google SketchUp, který umožňuje aktuální model exportovat rovnou do daného redereru.

Jelikož aplikací, co zobrazují modely vytvořené v aplikaci Google SketchUp existují desítky, rozhodnul jsem se v rámci diplomové práce nevytvářet další podobný renderer, ale vytvořit zásuvný modul do aplikace Google SketchUp. Tento modul bude ovládat kameru a na pozadí Google SketchUp mapovat snímky z videa, a tím vytvářet obraz rozšířené reality.

5.2 Struktura navrženého systému

Systém na návrh bytu pomocí rozšířené reality, který jsem navrhl v této diplomové práci se skládá ze dvou základních modulů. První modul je implementován v C++ s využitím knihovny ARToolkit. Stará o zpracování videa, výpočet relativní pozice mezi vodicí značkou a kamerou.

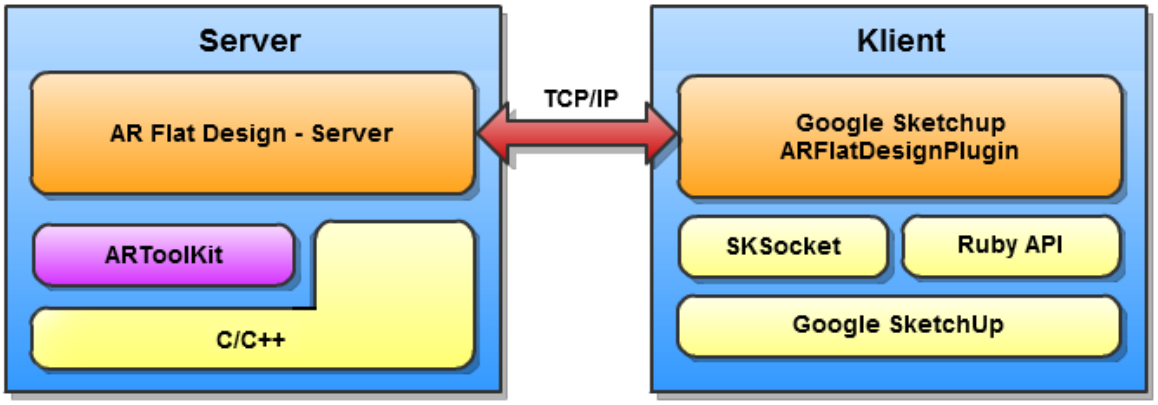
Druhý modul představuje zásuvný modul (plugin) do programu Google SketchUp implementovaný pomocí Ruby Script API. Jeho cílem je dotazovat se v pravidelných intervalech prvního modulu na pozici uživatele v bytě, patřičně nastavovat pozici kamery a mapovat snímky z videa na pozadí tak, aby byl vytvářen obraz rozšířené reality.

Jednotlivé moduly mezi sebou vytvářejí obousměrné TCP/IP spojení a komunikují mezi sebou pomocí vlastního protokolu, který je popsán v kapitole 6.3. První modul při komunikaci reprezentuje server, druhý modul klienta.

Dané řešení je zvoleno z důvodu, že implementace knihovny ARToolKit pro Ruby Script neexistuje. Tento problém by bylo možné vyřešit za pomocí SWIG (Simplified Wrapper and Interface Generator) - open source nástroj používaný k propojení knihoven napsaných v C/C++ se skriptovacími jazyky jako jsou Perl, Python, Ruby, PHP a jiné. Komunikaci pomocí TCP/IP protokolu jsem zvolil z důvodu jednoduchosti, úplného oddělení jednotlivých modulů a transparentnosti jejich komunikace.

Schematické znázornění navrženého systému je zobrazeno na diagramu 5.2.

V následujících dvou kapitolách je detailněji popsán návrh a implementace serverové i klientské části systému pro návrh bytu pomocí rozšířené reality.



Obrázek 5.2: Schematické znázornění navrženého systému

Kapitola 6

AR Flat Design Server

Serverová část daného systému realizuje zpracování video signálu, výpočet uživatelovy pozice v bytě a poskytnutí obrazu z kamery pro zobrazení v Google SketchUp. V této kapitole nejprve popisuji návrh serveru a jeho činnost, kterou vykonává. Poté se zaměřuji na popis komunikačního protokolu, pomocí kterého komunikuje klient se serverem. A v poslední části této kapitoly jsou popsány implementační detaily daného řešení.

6.1 Návrh serveru

Předpokladem při návrhu serveru je, že server je přímo spojen s kamerou, která snímá scénu v bytě. Tím, že zpracováná vstup pouze z jedné kamery, je server omezen. A je předpokládáno, že k serveru se v jeden okamžik připojuje pouze jeden klient. Z praktického hlediska není potřeba, aby tedy server byl konkurentní. Připojení dvou klientů v návrhu není uvažováno.

Zpracování na straně serveru začíná inicializací a vytvořením TCP/IP spojení. Následovně jsou zpracovávány jednotlivé videosnímky z kamery. Pro každý snímek jsou detekovány vodicí značky a vypočtena relativní pozice vůči kameře pomocí knihovny ARToolkit. Jednotlivé videosnímky jsou uloženy a posílány klientovi přímo a nebo jsou mapovány do jednoho velkého obrázku, který je po částech přenášen ke klientovi. Schematické znázornění činnosti serveru je zobrazeno na diagramu aktivit 6.1.

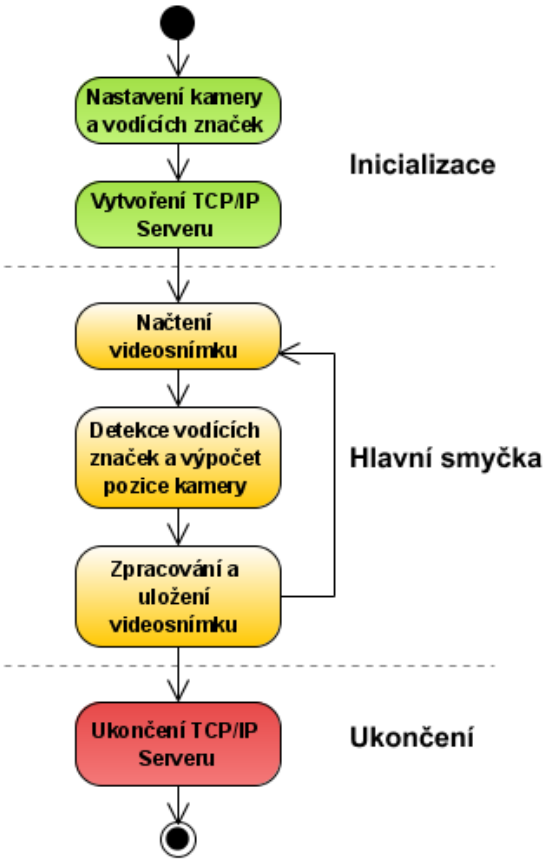
Inicializace serveru

Před vytvořením TCP/IP serveru, který bude komunikovat s klientem, je potřeba provést inicializaci knihovny ARToolkit.

Nejprve je potřeba nastavit parametry kamery. To se děje pomocí dvou souborů. První soubor definuje zkreslení kamery, je binární a můžeme jej získat pomocí nástroje pro kalibraci kamery (viz. dokumentace ARToolKit[1]) anebo použít soubor předdefinovaný, se kterým může ale dojít k určitému zkreslení. Druhý soubor je XML, který definuje rozlišení a datový formát kamery.

Před samotnou detekcí vodicích značek je potřeba knihovně ARToolKit definovat vodicí značky, které má detekovat. vodicí značky se ukládají do datového souboru, kde je uložen vzor značky v rozlišení 64x64 pro natočení 0°, 90°, 180°a 270°. Datové soubory, obsahující definici vzoru značky, je možné vygenerovat pomocí nástroje mk_pattd. Vytváření datových

[1] http://www.hitl.washington.edu/artoolkit/documentation/usercalibration.htm



Obrázek 6.1: Aktivity diagram činnosti serveru

souborů z vodicích značek by mělo probíhat za stejných světelnostních podmínek, za kterých pak budou vodicí značky používány.

Detekce vodicích značek

Při detekci vodicích značek je potřeba nejprve načíst jeden video snímek z kamery. Nad tímto snímkem se pomocí ARToolKitu provede detekce vodicích značek. Detekce k nalezení značek využívá globální prahování (kap. 2.2), kde je potřeba správně nastavit práh. Při inicializaci serveru je tento práh stanoven na experimentálně vyzkoušenou hodnotu.

Jelikož jiné systémy pro detekci vodicích značek, jako je například ARTag, nepotřebují nastavovat globální práh. Otázkou dynamického nastavování a určování prahu z obrázku jsem se nezabýval.

Výpočet pozice uživatele

Za pomoci nadetekovaných vodicích značek a vhodného použití funkcí ARToolKitu můžeme jednoduše získat relativní transformační matici mezi kamerou a vodící značkou. Nejjednodušším způsobem, jak určit pozici kamery, je považovat vodící značku za počátek souřadné soustavy daného systému. V ten okamžik inverzní transformace již definuje pozici kamery vůči počátku souřadné soustavy. Popřípadě je potřeba vynásobit získanou matici translační maticí, která definuje posunutí vodicí značky vůči počátku souřadnic. Výsledná transformační matice se následně bude přenášet na klienta, kde bude dále zpracovávána.

Výpočet pozice uživatele z více vodicích značek

Prozatím je pohyb uživatele omezen pouze na prostor, odkud uživatel uvidí vodící značku. Aby se mohl uživatel volně pohybovat po bytě, je potřeba, aby po bytě bylo rozmístěno více vodicích značek. Poté bude systém omezen tím, že uživatel uvidí minimálně jednu vodící značku.

Nejjednodušším způsobem, jak při použití více vodicích značek určit pozici kamery, je mít u každé vodicí značky uloženou transformaci této značky vůči počátku - označme si tuto transformaci jako T_0. Za pomocí inverzní transformační matice mezi kamerou a vodící zmačkou T_{cminv} jsme schopni podle vzorce 6.1 vypočítat výslednou transformační matici T_{c} udávající transormaci kamery od počátku.

$$T_{\text{c}} = T_0 * T_{\text{cminv}} \tag{6.1}$$

V okamžik, kdy je ve videosekvenci nadetekováno více vodicích značek než jen jedna, je potřeba určit, ze které vodicí značky se bude vypočítávat pozice kamery.

Jedním z možných způsobů, jak určit značku, je využít infomaci s jakou pravděpodobností byla vodící značka správně rozpoznána. Tuto infomaci nám poskytuje knihovna ARToolkit. Nevýhodou je, že pravděpodobnost nám určuje, jak přesně se vodící značka shoduje se vzorem. A neurčuje nám přímo kvalitu a přesnost relativní transformace mezi kamerou a vodící značkou.

Další možností je využít znalosti, že nejpřesnější relativní transformace je získána z vodicí značky, která leží vůči kameře nejkolměji. Úhel, svíraný mezi kamerou a vodící značkou, jde pro jednotlivé osy extrahovat z rotační části transformační matice.

Relativní pozice mezi vodicími značkami

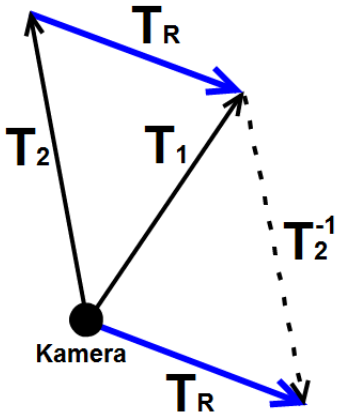
Navržený systém nyní potřebuje u každé vodicí značky mít definovanou transformaci této značky vůči počátku souřadné soustavy. Jelikož se jedná o netriviální úkol, který vyžaduje pečlivé měření a jistou přípravu, snažil jsem se tento nedostatek systému odstanit.

Předpokládejme, že při spuštění serveru bude kamerou snímána pouze jedna vodící značka. Pozici této značky budeme považovat za počátek souřadné soustavy. V okamžik, kdy kamera začne snímat první vodící značku a zároveň některou z dalších vodicích značek, jsme schopni vypočítat relativní pozici mezi vodícími značkami a zároveň tedy pozici druhé vodicí značky. Obdobným způsobem můžeme vypočítat pozice všech dalších vodicích značek v bytě. Podmínkou je pouze to, aby v jeden okamžik byly ve videosekvenci vidět vždy minimálně dvě značky a jedna z nich měla již určenou pozici vůči počátku.

Jelikož jednotlivá měření relativních transformací mohou být velmi nepřesná, nestačí relativní transformace vypočítat pouze jedenkrát, ale je potřeba měření opakovat. Z každého snímku, kde je více vodicích značek než jedna, je možno uchovat relativní transformaci mezi značkami a následně je možné z několika desítek či stovek měření vypočítat průměr.

Aby bylo možné vypočítat relativní pozici mezi dvěmi vodícími značkami T_{R}, je potřeba získat dvě relativní pozice mezi vodícími značkami a kamerou - ty získáme pomocí knihovny ARToolkit. Pokud poté první relativní transformaci T_1 vynásobíme s druhou relativní transformací T_2^{-1}, která bude invertovaná, dostaneme relativní pozici mezi druhou a první vodící značkou. Výpočet je možné zapsat pomocí rovnice 6.2. Na obrázku 6.2 je možno vidět grafické znázornění výpočtu.

$$T_{\text{R}} = T_1 * T_2^{-1} \tag{6.2}$$



Obrázek 6.2: Výpočet relativní pozice mezi vodícími značkami

6.2 Zpracování snímků videosekvence

Kromě detekce vodicích značek a výpočtu pozice uživatele se server stará o ukládání a poskytování snímků z klientovy videosekvence. Pro poskytování dat klientovi byly navrhnuty dva možné přístupy. První - jednodušší přístup udržuje na serveru vždy aktuální snímek z videosekvence a na žádost server tento snímek pošle klientovi. Posílání celého snímku znovu je zbytečně časově náročné, a proto byl navržen druhý způsob, který se snaží přenášet snímek po častech.

Přenášení snímku po částech

Předpokladem při návrhu bylo, že scéna kterou snímáme kamerou je z větší části neměnná a je tedy možné přenášet klientovi jen ty části scény, které se změnily anebo ty, které ještě nebyly přeneseny na klienta.

Jelikož je každý snímek z kamery získán pod jiným úhlem a z jiné vzdálenosti, je potřeba nejprve jednotlivé snímky promítnout do společné obrazové roviny. Toho docílíme transformací obrazu podle matice homografie. Matice homografie se vypočítává na základě znalosti o stejných bodech v obrazu.

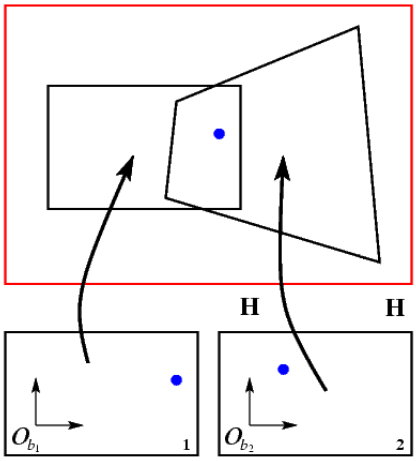
Pro výpočet jsou použity čtyři dvojice bodů obrazu, reprezentující rohové body vodicí značky a k nim odpovídající čtyři body ve společné rovině. Výpočet pozic bodů ve společné rovině je založen na informaci o relativní pozici mezi vodícími značkami.

Princip využití homografie při mapování snímků do společné obrazové roviny je zobrazen na obrázku 6.3.

Nejprve je na serveru vytvořen jeden dostatečně velký obrázek v paměti - buffer, na který se pomocí homografie postupně mapují jednotlivé snímky z kamery. První vodící značka je mapována na střed obrázku, další vodicí značky jsou mapovány podle relativní pozice vůči první vodící značce. V okamžik, kdy by se již nový snímek mapoval mimo vytvořený buffer je jeho velikost dynamicky zvětšena.

Je předpokládáno, že vodicí značky leží v jedné rovině. V okamžik, kdy by byla nasnímána vodící značka kolmá vůči této rovině, je potřeba vytvořit nový buffer a obdobným způsobem mapovat další stěnu místnosti.

Při mapování snímků do společné obrazové roviny nejsou zpracovávány všechny snímky z kamery, ale jen některé. Kritériem pro výběr daného snímku je především úhel, pod kterým



Obrázek 6.3: Princip použití homografie při promítnutí obrazů do stejné roviny

je vodící značka snímána. Při větším zorném úhlu dochází k velkému zkreslení mapování snímku do společné roviny.

Po namapování každého snímku je proveden test, určující, které části společné obrazové roviny byly změněny. Pro jednoduchost je rovina rozdělena na čtverce a při přenačtení určité oblasti se propočítává, kolik rohů každého čtverce náleží nově přenačítané oblasti. Pokud je toto číslo větší než při předchozí aktualizaci, tak je čtverec označen k přenačtení na straně klienta. Na obrázku 6.4 je možno vidět namapovanou stěnu, která byla přenesena na klienta.

Obrázek 6.4: Namapovaná stěna zobrazená na straně klienta

6.3 Komunikační protokol

V této podkapitole je popsán komunikační protokol, pomocí kterého komunikuje server a klient. Jedná se o stavový protokol, ve kterém na začátku komunikace dojde k inicializaci spojení a poté se klient opakovaně dotazuje serveru na pozici kamery a na videosnímky, které mapuje na pozadí aplikace Google SketchUp. Mezi jednotlivými dotazy nedochází k uzavření a opětovnému návázání spojení.

Základní formát protokolu

Každý dotaz na server či odpověď klientovi je zaobalena do obálky, která má následující tvar:

```
ARSkp VERZE_PROTOKOLU
DOTAZ_CI_ODPOVED
JEDEN_CI_VICE_RADKU
.
```

Obálka vždy začíná hlavičkou, kde je definován komunikační protokol - `ARSkp` a následně verze komunikačního protokolu. Momenálně existuje pouze první verze - `1.0`. Po hlavičce následuje jeden či více řádků s dotazem či odpovědí. Dotazů či odpovědí může být posláno v jednom kroku více než jedno. Na konci je obálka zase uzavřena pomocí jedné tečky na prázdném řádku, aby šlo bezpečně rozpoznat, že je dotaz či odpověď kompletní. Z toho důvodu žádný dotaz či odpověď nesmí začínat tečkou. Jednotlivé řádky jsou odděleny pomocí znaku „LF“ tzn. jsou použity linuxové konce řádků.

Ukázka komunikace mezi klientem a serverem

Následující ukázka demonstruje použití komunikačního protokolu, kde se klient připojí na server. Inicializuje spojení s výběrem mapovací metody „`BACKGROUND`“ a následně požádá server o pozici kamery a o obrázek, jež bude mapován na pozadí Google SketchUp. Následně klient spojení ukončí.

```
K>> ARSkp 1.0
K>> INI BACKGROUND
K>> .

S>> ARSkp 1.0
S>> OK
S>> .

K>> ARSkp 1.0
K>> GET CAMPOS
K>> GET BACKGROUND
K>> .

S>> ARSkp 1.0
S>> CAMPOS:0.99879498;-0.04475892;-0.02013021;0.00000000;-0.04416041;...
S>> IMGBACK:OutImage.bmp
S>> .
```

```
...
K>> ARSkp 1.0
K>> EXIT
K>> .
```

6.4 Implementace

Implementace serveru je provedena v programovacím jazyce C++. K detekci vodicích značek a k výpočtu pozice uživatele v modelu bytu je použita knihovna ARToolKit. Popis této knihovny je popisován v kapitole 3.3. Výběr knihovny byl určen zadáním této diplomové práce.

TCP/IP

Vytvoření TCP/IP serveru je realizováno ve vlastním vlákně programu. Příjem, zpracování a odeslání dat klientovi je tedy nezávislé na zpracování videosignálu a přípravě dat pro klienta. Vytvořením samostatného vlákna pro komunikaci s klientem a pro zpracování dat výrazně přispělo ke zrychlení navrženého systému.

K vytvoření vlákna serveru jsem použil třídu *CWinThread*, která je součástí knihovny *MFC (Microsoft Foundation Class Library)*. Pro sychronizaci procesů a řízení přístupu k datům byla použita třída *CMutex* ze stejné knihovny.

Vytvořený server standardně poslouchá na lokání IP adrese *127.0.01* a portu 8123. Je tedy potřeba, aby dané síťové rozhraní a port nebyly blokovány pomocí firewallu.

Zpracování a uložení videosnímků

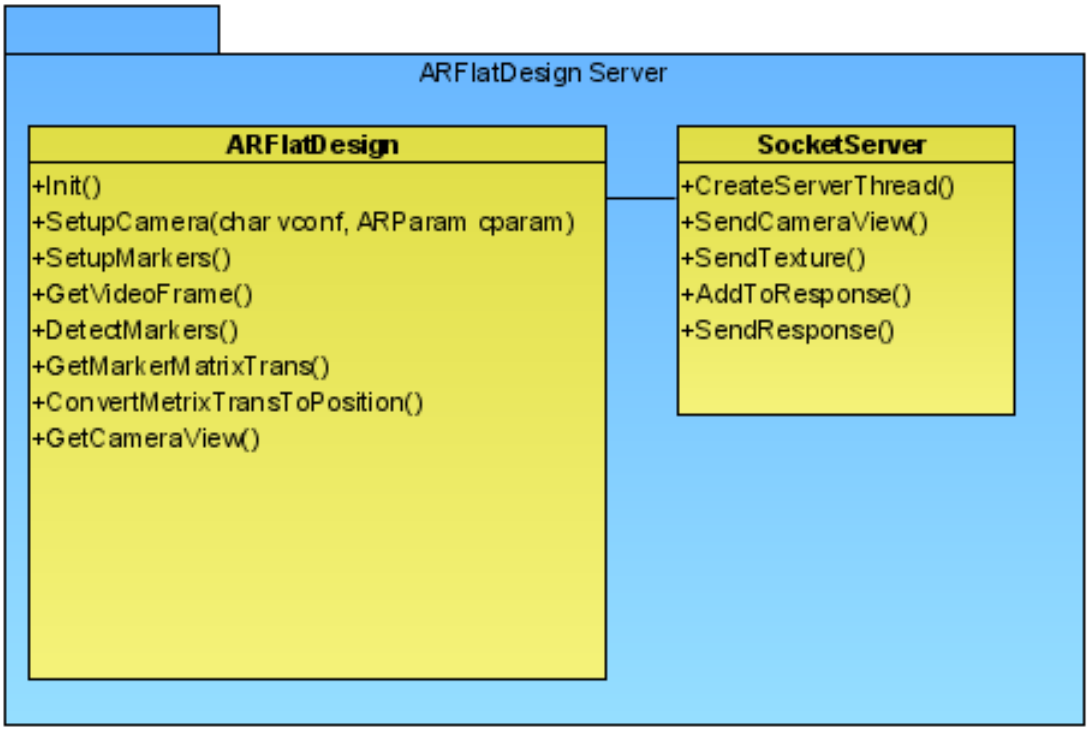
Při komunikaci s klientem nejsou jednotlivé snímky přenášeny pomocí TCP/IP spojení, ale jsou ukládány na disk a klientovi je poslána pouze cesta k tomuto souboru. Snímky jsou ukládány na disk ve formátu BMP pomocí knihovny *EasyBMP* [2].

Při vytváření virtuální stěny a mapování snímků z videosekvence do společné obrazové roviny jsou použity především funkce z knihovny OpenCV[3]. K vypočtení matice homografie je použita funkce *cv::getPerspectiveTransform*, která vypočítává perspektivní transformaci ze čtyř párů odpovídajících si bodů. Samotné mapování do společné roviny, resp. do jednoho obrázku je prováděno pomocí funkce *cv::warpPerspective*.

Návrh tříd

Návrh tříd je možno vidět na UML diagramu tříd, který je znázorněn na obrázku 6.5.

[2] http://easybmp.sourceforge.net
[3] Open Source Computer Vision Libary - http://opencv.willowgarage.com



Obrázek 6.5: UML diagram tříd serveru

Kapitola 7

AR SketchUp Plugin

„AR SketchUp plugin“ je zásuvný modul do 3D modelovacího programu Google SketchUp. V navrženém systému se stará o vizualizaci rozšířené reality. Je koncipován jako klient, který se dotazuje serveru na pozici uživatele v místnosti nebo na snímky z kamery tak, aby vytvářel obraz rozšířené reality pomocí aplikace Google SketchUp.

V této kapitole je představen návrh řešení zásuvného modulu. Je zde vysvětlen princip ovládání kamery a možné způsoby mapování videa na pozadí, včetně rozboru výhod a nevýhod zvoleného řešení. V poslední části této kapitoly jsou popsány implementační detaily řešení pomocí SketchUp Ruby API.

7.1 Návrh

Hlavním cílem klientské části je využití 3D modelovacího prgramu Google SketchUp k zobrazování rozšířené reality. Aby nebylo nutné vyvíjet vlastní zobrazovací engine pro modely zařizovacích předmětů, využil jsem již existujícího 3D editoru. Mimo jiné oblíbenost Google SketchUp může také pomoci k rychlejší distribuci a popularizaci navrženého systému.

Předpokladem k návrhu klientské části je využití aplikačního rozhraní SketchUp Ruby API, pomocí kterého je možné ovládat a přizpůsobovat si program Google SketchUp.

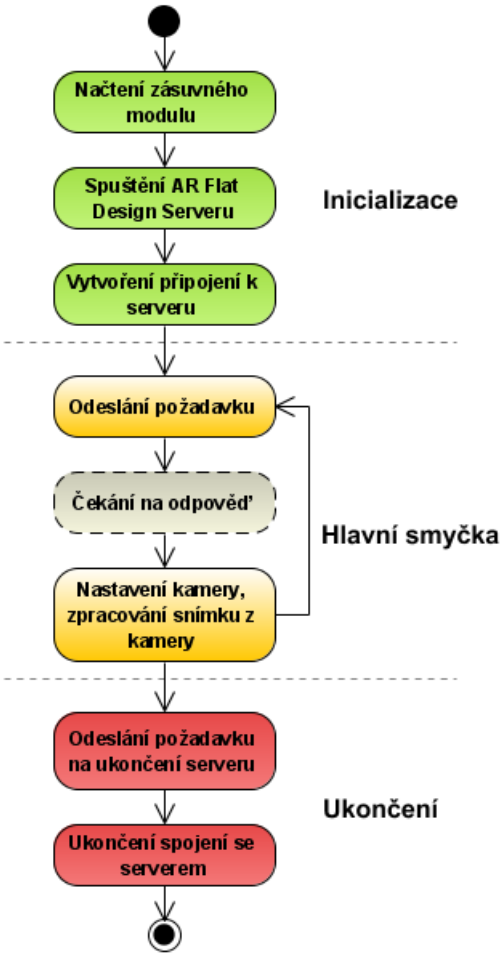
Klientská část systému v návrhu nejprve řeší inicializaci zásuvného modulu a vytvoření připojení k serveru. Následně mezi klientem a serverem probíhá komunikace založená na komunikačním protokolu popsaném v kapitole 6.3. Klient nejprve odešle požadavek na zjištění pozice uživatele v modelu bytu nebo požadavek pro získání smínku z videokamery. Po přijetí odpovědi nastaví správně pozici kamery nebo provede namapování snímku z videokamery na pozadí. Mezi odesláním požadavku a přijetím odpovědi je potřeba, aby program Google SketchUp reagoval na akce vyvolané uživatelským rozhraním jako je např. přidávání, odebírání či posun modelů zařizovacích předmětů.

Schematické znázornění činnosti klientské části systému je zobrazeno na diagramu aktivit 7.1.

Inicializace klienta

Před samotným dotazování serveru je potřeba zásuvný modul načíst do programu Google SketchUp, spustit server a vytvořit spojení se serverem.

Načtení zásuvného modulu se provede automaticky po spuštění aplikace Google SketchUp, jestliže je skript realizující zásuvný modul umístěn do adresáře „Plugins“ aplikace.



Obrázek 7.1: Aktivity diagram činnosti klienta

Po načtení skriptu je nezbytné, aby se do menu nebo na panel nástrojů přidaly volby pro spuštění a ukončení zobrazování rozšířené reality pomocí Google SketchUp.

Při vybrání volby pro spuštění by bylo vhodné, aby se automaticky spustil server a provedla inicializace spojení. Jelikož Ruby Script přímo nepodporuje spuštění externí aplikace na pozadí, je nutné spustit „AR Flat Design Server“ ručně.

Po spuštění serveru a zvolení volby pro spuštění systému se vytvoří TCP/IP spojení se serverem. Na začátku komunikace proběhne inicializace, kde se zvolí způsob mapování videa na pozadí.

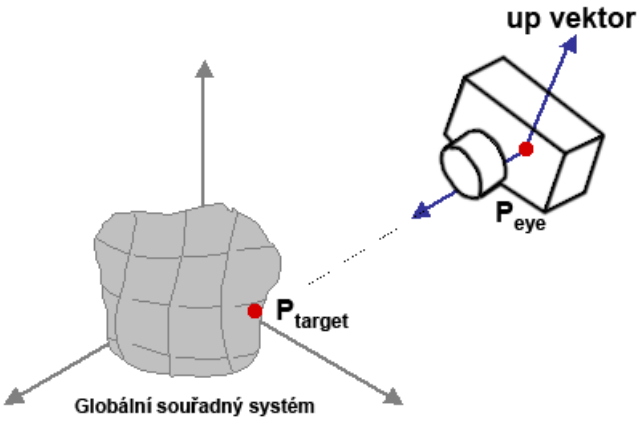
Následuje odesílání jednotlivých dotazů na server a jejich zpracování. Dotazy je možné volit dvěma způsoby. Prvním způsobem je, že po dokončení zpracování odpovědi ze serveru je automaticky na server zaslán další požadavek, přičemž při čekání na odpověď ze serveru může Google SketchUp obsluhovat uživatelské rozhraní aplikace.

Druhou možností je využití aplikačního rozhraní k tvorbě animací. Proces vytváření obrazu rozšířené reality je poté chápán jako jedna dlouhá animace. Google SketchUp v okamžik, kdy není zaneprázdněn, volá „AR SketchUp Plugin“ a dotazuje se jej na další snímek animace. Což je reprezentováno vytvořením dotazu na server a až po zpracování odpovědi je dovoleno vytvořit dotaz na další snímek animace.

7.2 Ovládání kamery

Prvním dotazem, který je odesílán na server, je dotaz na pozici uživatele v modelu bytu, podle které se nastavuje kamera v Google SketchUp.

Odpověď od serveru obsahuje matici 4×4, reprezentující 3D geometrickou transformaci mezi počátkem souřadné soustavy (pozicí první vodicí značky) a kamerou. Transformace je potřeba převést na model kamery Google SketchUp, který je reprezentován pomocí dvou bodů a jednoho vektoru. První bod P_{eye} udává pozici kamery, druhý bod P_{target} udává, kam se kamera dívá a vektor $v_{\vec{\text{up}}}$ určuje natočení kamery. Reprezentace modelu kamery v Google SketchUp je zobrazena na obrázku 7.2.



Obrázek 7.2: Reprezentace modelu kamery v Google SketchUp

Pozice kamery P_{eye} je vypočítána pomocí 3D transformace získané ze serveru. Transformace T je aplikována na bod počátku souřadné soustavy nebo na bod reprezentující pozici první vodicí značky. Výsledkem je přímo hledaný bod P_{eye} reprezentující pozici kamery.

Aby bylo možno vypočítat bod P_{target}, na který je směrován pohled kamery, je potřeba nejdříve vypočítat vektor, určující, kterým směrem se kamera dívá. Nejprve jsou zvoleny tři body, které leží ve stejné rovině jako nadetekovaná vodící značka. Na tyto body je aplikovaná 3D transformace T, která nám body transponuje do roviny kamery. Rozdílem dvou dvojic bodů jsou vypočteny dva vektory ležící v dané rovině. Vypočtením jejich vektorového součinu je získán vektor kolmý k rovině kamery, který reprezentuje vektor určující směr pohledu kamery. Výpočet je možné zapsat pomocí rovnice 7.1. Pro další výpočty převedeme daný vektor na jednotkový (rovnice 7.2).

$$v_{\vec{dir}} = (P_2 - P_1 * T) \times (P_2 * T - P_3 * T) \tag{7.1}$$

$$v_{\hat{dir}} = \frac{v_{\vec{dir}}}{|v_{\vec{dir}}|} \tag{7.2}$$

Výsledný bod P_{target}, určující, kam se kamera dívá, se vypočte pomocí rovnice 7.3. Jednotkový vektor směru pohledu $v_{\hat{dir}}$ je vynásoben vzdáleností nejvzdálenějšího předmětu $d_{\max}$a následně je k němu přičtena pozice kamery P_{eye}. Jako nejvzdálenější předmět v tomto případě považuji plochu v pozadí, na kterou se mapují videosnímky z kamery.

$$P_{\text{target}} = P_{\text{eye}} + \hat{v_{dir}} * d_{\max} \tag{7.3}$$

V neposlední řadě je potřeba určit vektor $\vec{v_{\text{up}}}$, určující natočení kamery. Jeho výpočet byl proveden obdobným způsobem jako výpočet jednotkového vektoru, určujícího směr pohledu. Jediným rozdílem bylo zvolení třech bodů, které určují rovinu. Nyní bylo potřeba vybrat tři body tak, aby definovaly rovinu mezi vodící značkou a kamerou.

7.3 Mapování videa na pozadí

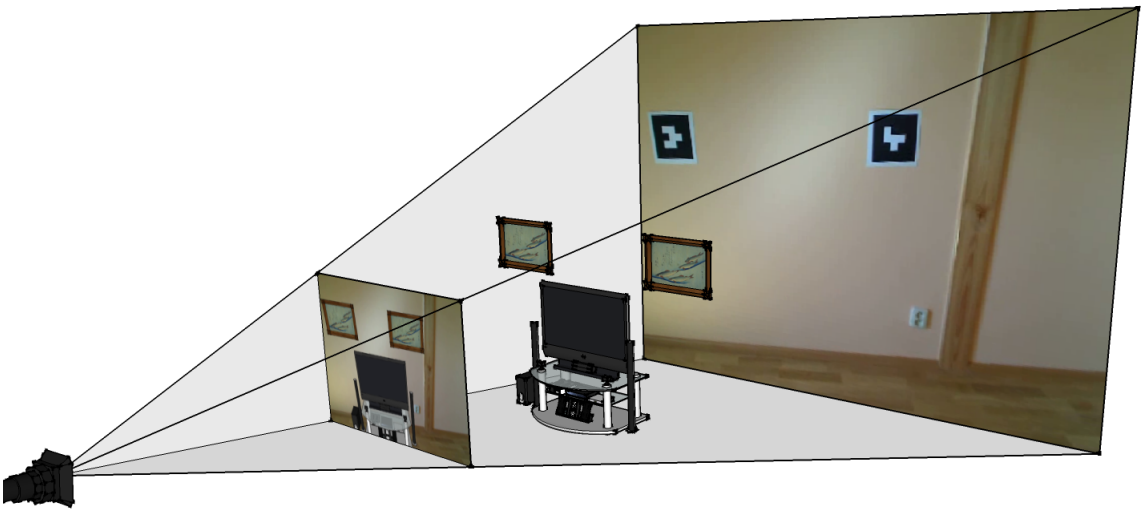
Abychom mohli mluvit o rozšířené realitě vytvářené pomocí Google SketchUp, nestačí pouze nastavovat kameru podle pozice uživatele v místnosti, ale je potřeba do Google SketchUp přenášet snímky z videosekvence a následně je mapovat na pozadí.

Nejjednodušší způsob, jak zobrazit snímky z videosekvence na pozadí, by bylo je umístit na pozadí, které není závislé na zobrazované scéně. Google SketchUp umožňuje nastavovat obrázek na pozadí nebo popředí (tzv. vodoznak) pomocí stylů zobrazení. Ale už neumožňuje tuto vlastnost měnit pomocí aplikačního rozhraní (API).

Musel jsem tedy hledat jiný možný způsob, jak zobrazovat snímky z videosekvence na pozadí. Nalezeným řešením je vkládat snímky do scény tak, aby jejich zobrazení vypadalo stejně, jako kdyby byl obrázek umístěn přímo na pozadí.

Na obrázku 7.3 je zobrazen princip, jakým způsobem je docíleno vytvoření obrazu rozšířené reality. Hluboko ve scéně, kde již nemůže být umístěn žádný objekt, je vložen snímek z videosekvence. Snímek musí být dostatečně velký, aby přesně pokryl celou plochu pozadí programu Google SketchUp. Jako předpoklad jsem uvažoval, že okno programu Google SketchUp bude mít stejný poměr stran jako videosekvence.

Jednotlivé zařizovací předměty jsou poté umístěny uprostřed scény. Výsledný obraz je zobrazen na obrázku 7.3 pomocí snímku, který je umístěn blíže kamery.



Obrázek 7.3: Princip mapování videa na pozadí

Postup mapování obrázku z videosekvence na pozadí scény je následující:

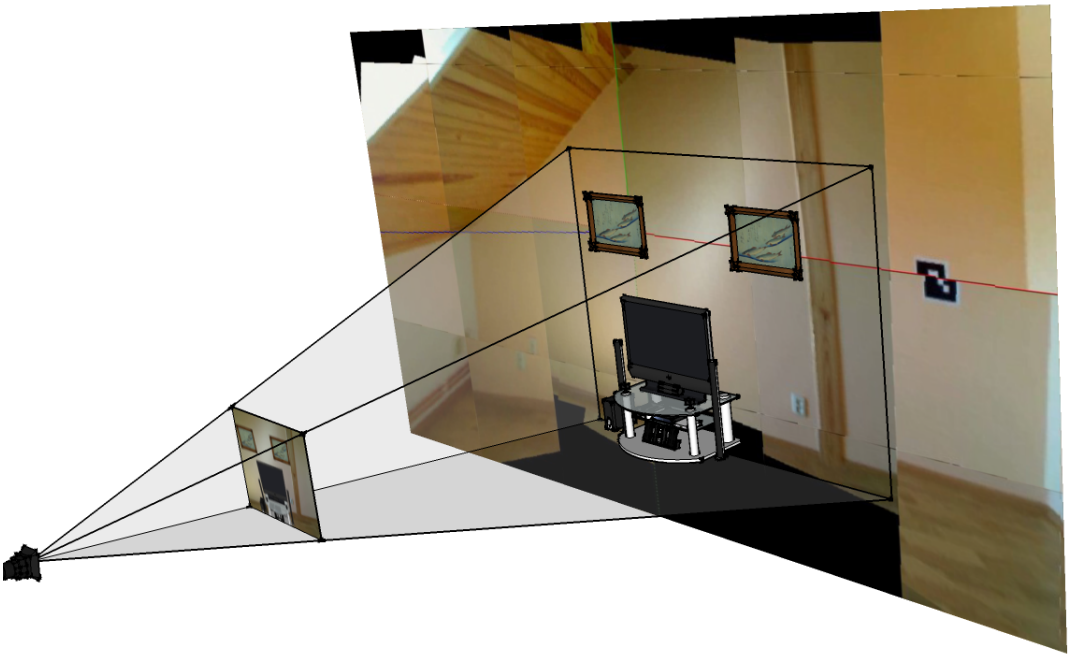
- Snímek je načten ze serveru a je vložen na pozici $(0, 0, 0)$
- Provede se translace snímku, tak aby jeho střed byl na souřadnicích $(0, 0, 0)$
- Za pomoci tří vektorů, reprezentujících jednotlivé osy kamery v prostoru, vytvoříme rotační transformaci, která snímek otočí kolmo proti kameře. Normálový vektor snímku bude opačný k vektoru směru pohledu kamery $\vec{v_{dir}}$.
- Podle úhlu pohledu (FOV - Field Of View) posuneme obrázek do scény tak hluboko, aby byl zobrazen přes celou plochu pozadí programu Google SketchUp.

7.4 Dynamické skládání virtuální stěny

Hlavní nevýhodou mapování snímků na pozadí je rychlost, se kterou se přenáší jednotlivé snímy ze serveru na klienta. Jelikož SketchUp Ruby API nepodporuje žádnou možnost načítání obrázků přímo z paměti, musí být každý obrázek nejprve uložen do souboru na pevný disk a až poté načten pomocí Google SketchUp. Tímto postupem je možno zpracovat na současných počítačích jen velmi málo snímků za sekundu.

Snažil jsem se navrhnout řešení, které by nepřenášelo vždy celý snímek z videosekvence, ale pouze jen část snímku, která byla změněna nebo prozatím nebyla přenesena na klienta.

Serverová část navrženého řešení se stará o zpracovávání jednotlivých snímků z kamery. Jelikož jednotlivé snímky jsou kamerou snímány z více úhlů a vzdáleností, je potřeba jednotlivé snímky mapovat do společné obrazové roviny. Mapování se provádí pomocí homografie do jednoho velkého obrázku. Tento obrázek nám reprezentuje „tapetu“, která reprezentuje povrch stěny místnosti.



Obrázek 7.4: Princip mapování videa na pozadí po částech - vytváření virtuální stěny

Tato tapeta je v aplikaci Google SketchUp narozdíl od mapování snímků na pozadí zobrazována přímo ve scéně na místě, které odpovídá reálné stěně v bytu. Princip mapování a umístění virtuální stěny je zobrazen na obrázku 7.4.

Zobrazování virtuální stěny je realizováno pravidelnými dotazy na server s požadavkem na získání seznamu částí stěny, které jsou označeny k přenačtení. Seznam obsahuje infomace o pozici přenačítané oblasti a obrázku, který má být na dané místo vložen.

7.5 Implementace

Zásuvný modul do aplikace Google SketchUp byl implementován pomocí aplikačního rozhraní Google SketchUp Ruby Script. Modul je koncipován jako klient, který se připojuje k serveru, ze kterého získává a zpracovává data.

Inicializace zásuvného modulu se provede automaticky, jestliže je umístěn ve složce Plugins dané aplikace.

Modifikace uživatelského rozhraní

Po načtení zásuvného modulu je potřeba, aby se do menu přidaly volby pro spuštění a ukončení zobrazování rozšířené reality pomocí Google SketchUp. Modifikace menu je realizována pomocí modulu *Sketchup::UI*, který je součástí Google SketchUp API.

Při zobrazování snímků z videosekvence na pozadí jsou jednotlivé obrázky vkládány přímo do zobrazované scény. Toto řešení má nevýhodu v tom, že uživatel v okamžiku, kdy označuje a manipujuje s předměty, může vybrat i objekty, které tvoří pozadí.

Tento problém se podařilo odstranit pomocí třídy *ARSkpPluginSelectionObserver*, která implementuje rozhraní *Sketchup::SelectionObserver*. Daná třída je volaná vždy, když dojde ke změně vybraných objektů. Následně jsou kotrolovány všechny objekty výběru a ty, jež jsou součástí pozadí, jsou z výběru odstraněny.

TCP/IP spojení

Pro vytvoření spojení mezi klientem a serverem je využita knihovna *SKSocket*, která je součástí aplikačního rozhraní Google SketchUp. Po připojení na server knihovna vytváří obousměrné spojení mezi klientem a serverem. Spojení je zachováváno mezi všemi dotazy na server.

Alternativou k této knihovně je knihovna *TCPSocket*, která je součástí skriptovacího jazyka Ruby Script. Jelikož tato knihovna nevytváří asynchronní spojení mezi serverem a klientem, je pro komunikaci se serverem nevhodná.

Mapování videosnímků na pozadí

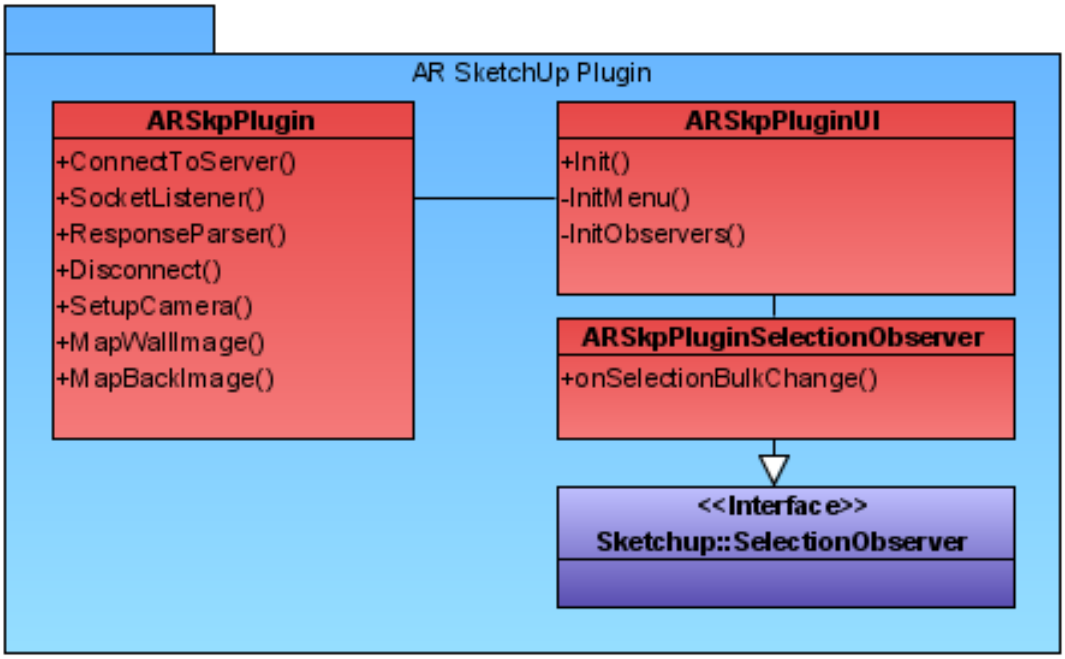
Pro vytvoření obrazu rozšířené reality je potřeba ze serveru získávat jednotlivé snímky z videosekvence a mapovat je na pozadí.

Při řešení této úlohy jsem narazil na dvě úskalí aplikačního rozhraní Google SketchUp. Prvním nedostatkem je chybějící možnost nastavovat obrázek na pozadí pomocí metod aplikačního rozhraní. Přičemž Google SketchUp umožňuje nastavovat vodoznak (obrázek na popředí) či obrázek na pozadí pomocí stylů zobrazení skrze uživatelské rozhraní aplikace. Z toho důvodu jsem se snažil vkládat do scény snímky z kamery tak, aby výsledné zobrazení vypadalo stejně, jako kdyby byl snímek zobrazen na pozadí.

Druhou nevýhodou aplikačního rozhraní je práce s obrázky a texturami. Google SketchUp neumožňuje žádným způsobem přístup k jednotlivým bodům dané textury, ani načítání obrázků přímo z paměti, např. pomocí předání ukazatelu na data v paměti. Jediný způsob, jak mezi serverem a klientem přenášet jednotlivé snímky, je ukládat snímky z videosekvence na pevný disk a poté je znovu načítat.

Návrh tříd

Návrh tříd je možno vidět na UML diagramu tříd, který je znázorněn na obrázku 6.5. Třída *ARSkpPluginUI* se stará o vytvoření potřebných položek v menu. Třída *ARSkpPluginSelectionObserver* realizuje filtraci objektů ve výběru, aby nebylo možné označit objekty, jež jsou součástí pozadí. Připojení na server, nastavování kamery a mapování snímků na pozadí je realizováno ve třídě Třída *ARSkpPlugin*.



Obrázek 7.5: UML diagram tříd klienta

Kapitola 8

Dosažené výsledky a experimenty

Tato kapitola se zabývá zhodnocením výsledků diplomové práce. Nejprve je popsán proces testování navrženého systému, následně dosažené výsledky a neposlední řadě jsou zmíněny poznatky z testování navrženého systému.

8.1 Testování navrženého systému

Cílem diplomové práce bylo navrhnout systém, který bude s pomocí virtuálních brýlí demonstrovat možné použití rozšířené relaity k návrhu bytu. Aby nebylo nutné systém vyvíjet za využití těchto virtuálních brílí byly nasnímány testovací sady dat na kterých se systém vyvýjel a až poté jsem prováděl test systému s virtuálními brílemi

Testovací sady, které byly nastnímány, jsou uloženy v příloze na DVD. Každá testovací sada obsahuje několik video souborů, použité vodicí značky v pdf, rozpoznané značky a v neposlední řadě nákres s rozmístěním vodicích značek v místnosti.

Video soubory byly nasmínámy pomocí webové kamery „Logitech B905“ v rozlišení 800×600.

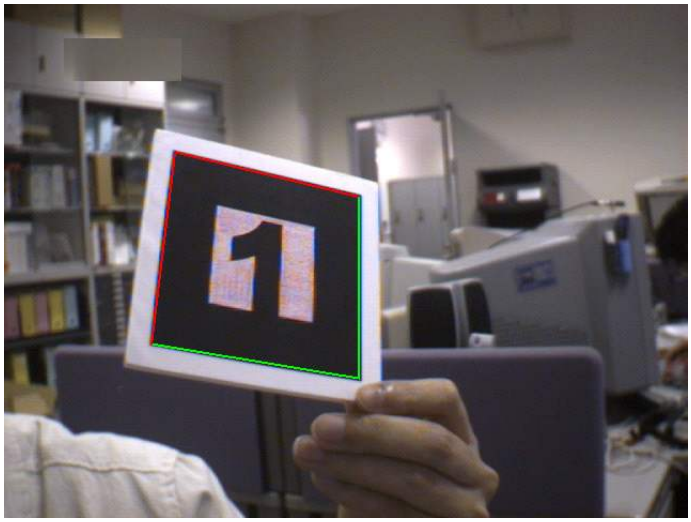
vodicí značky byly vygenerovány pomocí online nástroje ARToolKit Marker Maker[1] ve velikosti 18 cm a uloženy ve formátu PDF.

Vytvoření vzorů vodicích značek

Pro každou vodicí značku je potřeba, aby byl vytvořen soubor s nadetekovaným vzorem značky, který je nadetekován za stejných světelnostních podmínek jako nasnímané video soubory. Pro vytvoření těchto souborů je použit program *mk_patt*.

Program mk_patt pro detekci a převod vodicích značek do binárního formátu ARToolKitu je uložen jako příloha na CD. Po jeho spuštění jsme vyzvání k zadání cesty ke konfiguračnímu souboru kamery. A poté se nám otevře okno zobrazené na obrázku 3.8. vodicí značku umístíme do podobných světelných podmínek, ve kterých bude výsledná aplikace spouštěna. Kameru namíříme kolmo nad vodicí značku a pohybujeme s ní do té doby, než se nám kolem vodicí značky nevytvoří zeleno-červený čtverec (Obrázek 3.8). To indikuje, že program mk_patt úspěšně našel vodicí značku. V okamžik, kdy červený roh čtverce je v levém horním rohu a zelený v pravém dolním, klikneme levým tlačítkem myši a zadáme soubor, do kterého se má uložit binární podoba vodicí značky.

[1] http://roarmot.co.nz/ar/



Obrázek 8.1: Program mk_patt pro detekci a převod vodicích značek do binárního formátu. Převzato z [7]

8.2 Dosažené výsledky

V rámci diplomové práce se podařilo vytvořit systém, který využívá 3D modelovací program Google SketchUp pro realizaci obrazu rozšířené reality. Systém je koncipován jako klient-server aplikace. Server zpracovává snímky z videosekvence a detekuje vodicí značky ve videu. Klient se stará o ovládání aplikace Google SketchUp a vytváření obrazu rozšířené reality.

Pozice uživatele v modelu bytu je vypočtena na základě vodicích značek a přenášena ke klientovi, kde je podle ní nastavována kamera programu Google SketchUp. Aby obraz rozšířené reality byl kompletní, je potřeba ještě přenášet do programu Google SketchUp snímky z kamery a mapovat je na pozadí.

Mapování videa na pozadí

První navrženou metodou jak mapovat video na pozadí je přenášet jednotlivé snímky videa ke klientovi a vkládat je do scény tak, aby výsledný obraz vypadal, jako kdyby snímky byly na pozadí Google SketchUp. Ukázka výsledku dané metody je zobrazen na obrázku 8.2. Jelikož Google SketchUp nepodporuje přímé načítání obrázků z paměti, musí tedy být každý obrázek serverem uložen na disk a poté následně znovu přečten klientem. Z tohoto důvodu je rychlost přenášení snímků poměrně pomalá. Při testování aplikace byla naměřena rychlost 4 až 6 snímků za sekundu.

Přenášení obrazu po částech

Jelikož rychlost 4 až 6 snímků za sekundu je pro reálné využití nedostačující, snažil jsem se navrhnout řešení, které nepřenáší mezi serverem a klientem vždy celý videosnímek, ale jen nové či změněné části obrazu. Jednotlivé snímky jsou pomocí matice homografie transformovány do jedné společné roviny (obrázku), která je poté po částech přenášena ke klientovi a mapovaná na virtuální stěnu. Z důvodu nepřesnosti měření vodicích značek pomocí knihovny ARToolKit dochází k odchylkám při skládání virtuální stěny a výsledný obraz ne-

Obrázek 8.2: Ukázka mapování videa na pozadí

vypadá příliš uceleně. Ukázka výstupu této metody je zobrazena na obrázku 8.3. Rychlost navrženého systému je závislá na počtu regiónů, které se přenáší ke klientovi. Při testování aplikace byla naměřena rychlost 4 až 15 snímků za sekundu. Výhodou této metody je jednodušší manipulovatelnost s předměty v programu Google SketchUp. Jelikož virtuální stěna je mapována na stejné místo jako stěna v bytu lze využít možnosti přichytávání objektů k této stěně.

8.3 Přesnost a stabilita navrženého systému

Jelikož je navržený systém závislý na knihovně ARToolKit, která detekuje v obraze vodicí značky je přesnost a stabilita navrženého systému závislá na správné volbě a rozmístění vodicích značek. Dalším výrazným faktorem ovlivňující přesnost navrženého systému je kalibrace kamery.

Kalibrace kamery

Pro správnou činnost knihovny ARToolKit je potřeba provést kalibraci kamery. Výstupem kalibrace kamery je datový soubor „camera_para.dat“, který obsahuje definici zkreslení kamery. Tento soubor je načítán knihovnou při každém spuštění aplikace.

Kalibrace kamery může být prováděna dvoukrokově za využití programů „calib_dist“ a „calib_cparam“ nebo pouze jednokrokově za použití programu „calib_dist“. Dvoukrokové měření je podstatně složitější, ale přináší lepší výsledky při určování pozice vodicí značky v prostoru. Podrobný popis kalibrace kamery je popsán v dokumentaci knihovny ARTool-

Obrázek 8.3: Ukázka mapování videa na virtuální stěnu

Kit[2].

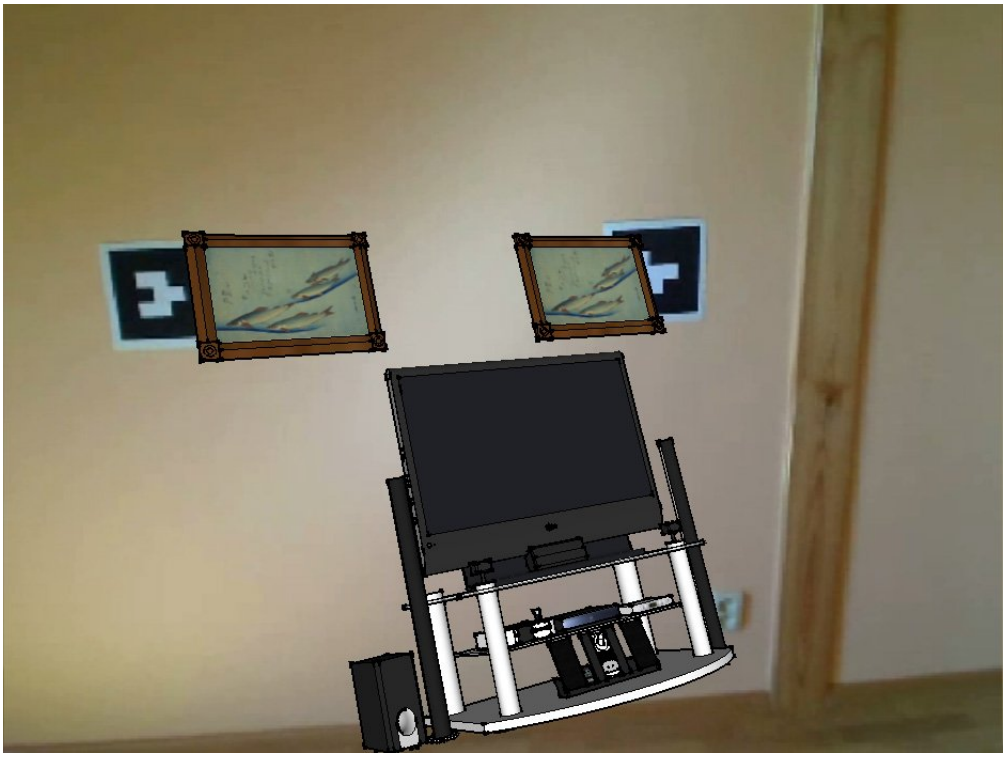
Pokud kamera není správně zkalibrovaná, může docházet k velkým nepřesnostem při určování pozice vodicí značky v prostoru. Obrázek 8.4 zobrazuje možnou chybu při určování pozice vodicí značky, pokud nedošlo ke kalibraci kamery. Na obrázku 8.2 je zobrazena stejná scéna v okamžik, kdy byla kamera zkalibriována.

Výběr a umístění vodicích značek

I když vodicí značky pro knihovnu ARToolKit mohou mít libovolný vzor, je vhodné vybírat vodicí značky, které neobsahují drobné vzory a neobsahují vysoké frekvence v obraze. V průběhu zpracování vodicích značek knihovnou ARToolKit je prováděna normalizace vzoru. Tento proces převádí vzor vodicí značky na matici 16×16. Nejvhodnější vzory tedy jsou ty, které se jednoduše převedou na matici 16×16 a při zmenšení nezmění svoji vizuální podobu. Používané vodicí značky nesmí být rotačně zaměnitelné.

Umístění vodicích značek je potřeba volit tak, aby v obraze byla vždy zobrazena minimálně jedna vodicí značka. Není vhodné umísťovat vodicí značky do jedné přímky. Nerovnoměrným rozmístěním vodicích značek po stěně je v případě, že je kamerou snímáno více vodicích značek výhodnější pro přesnější stanovení 3D pozice uživatele v modelu bytu.

[2] http://www.hitl.washington.edu/artoolkit/documentation/usercalibration.htm



Obrázek 8.4: Ukázka špatného výsledku nezkalibrované kamery

Kapitola 9

Závěr

Cílem této diplomové práce bylo vytvořit systém pro návrh bytu pomocí rozšířené reality. Nejprve byly nastudovány možné způsoby realizace rozšířené reality a knihovny usnadňující práci s rozšířenou realitou. Základní princip těchto knihoven je popsán v kapitole 3. Dle zadání byla pro návrh systému použita knihovna ARToolKit, která ve vstupní videosekvenci detekuje vodicí značky a vypočítává relativní transformační matici mezi těmito značkami a kamerou.

Návrh systému

Při návrhu systému byl použit 3D modelovací program Google SketchUp nejen pro vytváření modelů jednotlivých zařizovacích předmětů, ale i pro samotnou realizaci rozšířené reality. Systém je tedy navržen jako zásuvný modul do aplikace Google SketchUp. O návrhu systému pojednává kapitola 5. Zásuvný modul byl implementován pomocí aplikačního rozhraní SketchUp Ruby Script. Jelikož neexistuje implementace knihovy ARToolKit (ani žádné jiné knihovny) ve skriptovacím jazyku Ruby, navrhl jsem systém jako aplikaci typu klient-server.

Serverová část systému (kapitola 6) se stará o zpracovávání vstupního videa, detekci vodicích značek a výpočet pozice uživatele v modelu bytu. Pomocí vlastního protokolu komunikuje přes lokální síťové rozhraní s klientskou částí systému (kapitola 7). Ta se stará o nastavování pozice a natočení kamery v programu Google SketchUp. Dále klient ze serveru načítá jednotlivé snímky z kamery a mapuje je na pozadí Google SketchUp tak, aby vytvářel obraz rozšířené reality.

Zhodnocení výsledků

V rámci diplomové práce se podařilo vytvořit systém, který využívá 3D modelovací program Google SketchUp pro realizaci obrazu rozšířené reality. Hlavní úskalí navrženého systému je v přenosu obrazové informace na pozadí Google SketchUp. V prvotním návrhu systému, kde docházelo k mapování snímku na pozadí pomocí vkládání obrázku do scény, bylo dosahováno přenosové rychlosti mezi serverem a klientem přibližně jeden až dva snímky za sekundu.

Z tohoto důvodu jsem hledal řešení, které nepřenáší mezi serverem a klientem vždy celý videosnímek, ale jen nové či změněné části obrazu. Rychlost daného řešení je závislá na počtu regionů, které jsou určeny k přenačtení. Při testování se rychlost pohybovala mezi 4 až 15 snímků za sekundu. Nevýhoda tohoto řešení je skládání scény z více videosnímků

„překládáním“, jednotlivých snímků přes sebe a vytvářením virtuální stěny. Z důvodu nepřesnosti měření vodicích značek pomocí knihovny ARToolkit dochází k odchylkám při skládání virtuální stěny a výsledný obraz nevypadá příliš uceleně. Ukázka výstupu této metody je zobrazena na obrázku 8.3.

Úpravou prvního řešení, kdy jsem komunikaci s klientem a zpracování videosekvence umístil do vlastního vlákna programu, došlo k urychlení přenosu až na 4 až 6 snímků za sekundu. Rychlost výsledného řešení příliš neodpovídá pořadavku rozšířené reality, aby virtuální objekty byly do scény vkreslovány v reálném čase.

Hlavní příčinou pomalého řešení je aplikační rozhraní Google SketchUP, protože neobsahuje možnost načítání obrázků přímo z paměti, a tedy každý obrázek musí být nejprve uložen na disk a poté znovu načten pomocí zásuvného modulu. Obrázek není možné ani přenášet pomocí TCP/IP spojení, které je mezi klientem a serverem vytvořeno, protože SketchUp Ruby API neumožňuje přistupovat k jednotlivým bodům obrázku či textuty. Jediná možnost je načítat obrázek ze souboru.

Další vývoj systému

Hlavní nedostatky navrženého systému plynou z chybějící podpory přímého načítání obrázků z paměti pomocí aplikačního rozhraní Google SketchUp. Z tohoto důvodu by bylo vhodné požádat o přidání dané funkcionality do aplikačního rozhraní na oficiálním diskusním fóru[1] v diskusním tématu s názvem „SketchUp Wish List“.

Použitá knihovna ARToolKit Standard v.2.x je již poměrně stará a několik let již není aktivně vyvíjena. Bylo by tedy vhodné použít novější knihovnu pro detekci vodicích značek a nebo použít novější metody používané v rozšířené realitě, které nejsou závislé na detekci vodicích značek. Příkladem může být metoda PTAM (Parallel Tracking and Mapping), která automaticky ve videu detekuje významné body, podle kterých vypočítává pozici uživatele v modelu bytu.

V rámci diplomové práce bylo řešeno pouze zobrazování rozšířené reality. Bylo by tedy vhodné zabývat se uživatelským rozhraním, které by bylo založeno na detekci gest rukou, pomocí kterých by mohl uživatel přemisťovat jednotlivé virtuální předměty v bytě.

[1] http://forums.sketchucation.com

Literatura

[1] Dr. Ing. Eduard Sojka: *Digitální zpracování a analýza obrazů*. VŠB - Technická univerzita Ostrava, 2000, s. 74, 88.

[2] George Stockman, Linda Shapiro: Computer Vision [online]. `http://www.cse.msu.edu/ stockman/Book/2002/Chapters/ch3.pdf`, 2001 [cit. 2009-12-26].

[3] Hirokazu Kato: Inside ARToolKit [online]. `http://www.hitl.washington.edu/artoolkit/Papers/ART02-Tutorial.pdf`, 2002 [cit. 2009-12-28].

[4] James Matthews: Thresholding and Segmentation [online]. `http://www.generation5.org/content/2003/segmentation.asp`, 2004-12-05 [cit. 2009-12-05].

[5] Kolektiv autorů: Sketchup Ruby API Documentation [online]. `http://code.google.com/intl/cs/apis/sketchup/docs/`, 2009 [cit. 2009-12-26].

[6] Kolektiv autorů: About The ARQuake Project [online]. `http://wearables.unisa.edu.au/arquake/` , 22. 5. 2009 [cit. 2010-01-02].

[7] Kolektiv autorů: ARToolkit Documentation [online]. `http://www.hitl.washington.edu/artoolkit/documentation/`, [cit. 2009-12-26].

[8] Mark Fiala: ARTag Rev1: marker Detection [online]. `http://www.artag.net/rev1.html` , 2006 [cit. 2009-12-30].

[9] Milan Šonka, Václav Hlaváč: *Počítačové vidění*. Grada a.s., 1992, s. 23, 70, 97.

[10] Milan Soňka, Václav Hlaváč, Roger Boyle: *Image processing, analysis and machine vision*. Thomson Engineering, 2007, iSBN-10: 0-495-08252-X.

[11] Patrick Zandl: Příští fenomén: rozšířená realita je budoucnost webu [online]. `http://www.lupa.cz/clanky/rozsirena-realita-augmented-reality/` , 22. 5. 2009 [cit. 2010-01-02].

[12] Přispěvatelé Wikipedie: Shluková analýza [online]. `http://cs.wikipedia.org/w/index.php?title=Shlukov613175`, 2009-11-16 [cit. 2009-12-26].

[13] Robert Fisher, Simon Perkins, Ashley Walker, Erik Wolfart: Hypermedia image processing reference: Adaptive Thresholding [online].

http://homepages.inf.ed.ac.uk/rbf/HIPR2/adpthrsh.htm , 2004 [cit. 2010-01-03].

[14] Stephen Cawood, Mark Fiala: *Agmented Reality: A Practical guide*. Pragmatic Bookshelf, 2008, iSBN-10: 1-934356-03-4.

[15] Tomohiko Koyama: Adaptive thresholding experiment (FLARToolKit) [online]. http://blog.jactionscripters.com/2009/05/18/adaptive-thresholding-experiment/ , 18. 5. 2009 [cit. 2010-01-03].

Příloha A

Obsah DVD

Na přiloženém DVD nalezneme zdrojový tvar písemné zprávy, zdrojové soubory testovacích aplikací a také testovací sady dat. Jednotlivé soubory jsou uloženy v následující stromové struktuře:

/bin - zkompilovaná verze aplikace

/data - testovací sady dat

/demo - demonstrační výstupy aplikace

/plakat - plakat k diplomové práci

/prog_doc - programová dokumentace vygenerovaná pomocí programu doxygen

/src - zdrojové kódy navrženého systému

/technicka_zprava

./online - Uložené vybrané online dokumenty, které byly použity při tvorbě technické zprávy

./pdf - technická zprava v pdf pro tisk a pro prohlížení (tisk ma vypnute odkazy v pdf)

./latex - technická zpráva v latex-u