České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra počítačové grafiky a interakce

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Bc. Petr Beránek

Studijní program: Otevřená informatika
Obor: Počítačová grafika a interakce

Název tématu: Systém pro automatické snímání pomocí 3D laser scanneru a fotoaparátu

Pokyny pro vypracování:

Prostudujte API dodávané se skenerem Minolta Vivid 9i, které umožňuje programově ovládat tento skener. Dále nastudujte SDK/API fotoaparátů cannon a otočného stolečku s kontrolní jednotkou MARS2. Navrhněte a ověřte možnost pořízení 3D modelu pomocí 3D scanneru a detailní textury pomocí fotoparátu, která bude automaticky registrována na model. Vytvořte program, který bude tento proces provádět automaticky, výsledkem bude 3D mračno bodů, pozice kamer a 3D texturovaný model.
Systém ověřte na předmětech konzultovaných s vedoucím práce. Ověřte vhodnost aktuálního osvětlení a případně navrhněte, jak jej zlepšit.

Seznam odborné literatury:

[1] Pavel Píša, Uživatelský manuál k jednotce MARS 2:
http://cmp.felk.cvut.cz/~pisa/mars/mars_man_cz.html

[2] Multiple View Geometry in Computer Vision, Second Edition. Richard Hartley and Andrew Zisserman, Cambridge University Press, March 2004.

Vedoucí: Ing. David Sedláček, Ph.D.

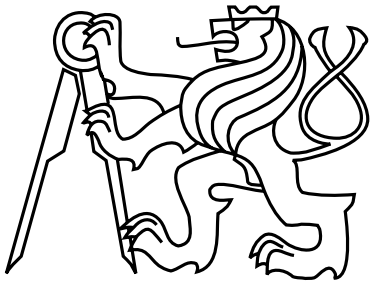
Platnost zadání: do konce zimního semestru 2018/2019

prof. Ing. Jiří Žára, CSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 3.4.2017

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačové grafiky a interakce



Diplomová práce

Systém pro automatické snímání pomocí 3D laser scanneru a fotoaparátu

Petr Beránek

Vedoucí práce: Ing. David Sedláček, Ph.D.

Studijní program: Otevřená informatika, Magisterský

Obor: Počítačová grafika a interakce

18. května 2017

Poděkování

Zde bych chtěl poděkovat vedoucímu práce, Ing. Davidu Sedláčkovi, Ph.D., za konzultace a cenné rady. Dále bych chtěl poděkovat své rodině za podporu během celého mého studia a především v době psaní této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Kladně dne 18. 5. 2017 .....

Abstract

The goal of this thesis is to analyze SDKs for 3D scanner VIVID 9i, camera Canon EOS 100D and control unit MARS 2 and use them to implement application that allows scanning of user defined object using this hardware. Scanner is used to obtain geometrical information, camera is for color information and MARS 2 with turntable is to enable easy gathering of data from more directions. Thesis contains problem analysis, design and implementation of application. Important part is the testing of light's influence on data collected by scanner. The result is the application which allows user to create 3D model of scanned object and export it in bundle or PLY format.

Keywords: 3D scanner, user assisted 3D reconstruction, texture extraction, computer vision, computer graphics

Abstrakt

Cílem této práce bylo nastudovat SDK dodávaná ke 3D skeneru VIVID 9i, fotoaparátu Canon EOS 100D a řídící jednotce MARS 2 a využít je pro implementaci aplikace, která umožní naskenování uživatelem zvoleného objektu s použitím zmíněných zařízení. Skener umožňuje získání geometrické informace, fotoaparát získání barevné informace a otočný stolek s jednotkou MARS 2 je zde pro zjednodušení skenování z více stran objektu. Práce samotná obsahuje především analýzu problému, návrh řešení a implementaci aplikace spolu s testováním vlivu osvětlení na data získaná skenerem. Výsledkem této práce je aplikace, jež umožňuje uživateli vytvoření modelu skenovaného objektu v bundle nebo PLY formátu.

Klíčová slova: 3D scanner, uživatelem řízená 3D rekonstrukce, extrakce textur, počítačové vidění, počítačová grafika

Obsah

Seznam obrázků

Kapitola 1

Úvod

V celé této práci se budu zabývat především 3D skenerem VIVID 9i, fotoaparátem Canon EOS 100D a otočným stolkem s řídící jednotkou MARS 2. V následujících kapitolách se dočtete o popisu jednotlivých zařízení, o SDK/API k nim poskytovaným a také o analýze problému, návrhu řešení, implementaci a testování aplikace, která umožní skenování objektů s použitím těchto zařízení.

1.1 Cíl práce

Cílem mé práce je navrhnout a implementovat aplikaci pro semi-automatické naskenování objektu s pomocí 3D skeneru, fotoaparátu a otočného stolku. Výstupem skenovacího procesu bude scéna v některém z běžně používaných formátů, která vznikne složením jednotlivých skenů dohromady. Tato scéna bude obsahovat mračno bodů a model získaný pomocí skeneru. Zároveň tento model bude otexturován pomocí fotografií získaných fotoaparátem. Dále půjde také o ověření vhodnosti osvětlení používaného při skenování a případný návrh zlepšení.

Kapitola 2

Motivace

Tato aplikace je vyvíjena pro konkrétní typ 3D skeneru Konica Minolta VIVID 9i [20]. Rozlišení tohoto skeneru je 640×480 bodů pro hloubkovou mapu.

Pro hloubkovou mapu je toto rozlišení velmi vysoké, ale stejné rozlišení je i pro barevnou informaci. To znamená, že pro každý bod hloubkové mapy získáme pouze jeden pixel barevné informace. Pro mračno bodů to není problém, jelikož bod má pouze jednu specifickou barvu, ale pro model vytvořený na tomto mračně bodů je barevná informace nedostačující, pokud by byla použita jako textura (To by znamenalo, že na jeden trojúhelník modelu by připadly pouze tři pixely.).

K vytváření výsledné scény, která se bude skládat z mračna bodů, 3D modelu, fotografií a kamer, bude použito několik zařízení. 3D skener k získání hloubkové mapy, základní barevné informace a 3D modelu, fotoaparát pro získání detailní barevné informace a otočný stolek, který umožní sběr dat z více stran objektu.

Kapitola 3

Analýza

V této kapitole se budu zabývat především analýzou jednotlivých zařízení, způsobem jejich ovládání, SDK a protokolů pro práci s nimi. Významnou částí je zde i analýza problémů, který potřebujeme vyřešit (Naskenování dat, složení scény.).

3.1 Použitý hardware

3.1.1 3D skener

3D skener, pro který je tato aplikace vyvíjena, je VIVID 9i[20] od firmy Konica Minolta. Je to bezkontaktní 3D skener využívající laserový paprsek a kameru pro získání mračna bodů o rozměrech 640×480 [1], barevné informace o povrchu skenovaného objektu a trojrozměrného modelu. 3D skener má jako výstup SCSI port a pro připojení k počítači je nezbytné použití konvertoru z SCSI na USB[29], neboť většina běžných počítačů nemá v dnešní době snadno přístupný SCSI port.

3.1.2 Fotoaparát

Fotoaparát, pro který je aplikace vyvíjena je EOS 100D[15] od firmy Canon. Je to digitální zrcadlovka s rozlišením 18 megapixelů (rozlišení 5184×3456) a patří mezi menší fotoaparáty. Fotoaparát bude k počítači připojen přímo pomocí USB.

3.1.3 Otočný stolek

Stolek je ovládán pomocí řídící jednotky MARS 2[27] od firmy PiKRON s.r.o.. Ta je určena pro regulaci polohy až tří stejnosměrných motorů s inkrementálními čidly přírůstku polohy[25]. Tato jednotka má USB port umožňující její připojení k počítači a komunikuje se s ní pomocí sériového portu.

[1] Skener funguje tak, že ze spodní části skeneru je vysílán laserový pruh (V originálu označen "laser stripe".). Laserové paprsky se od povrchu skenovaného objektu odrážejí a jsou zachyceny kamerou, které je umístěna nad místem, odkud se paprsky vysílají. Následně je triangulací vypočtena informace o hloubce, je získáno mračno bodů (Ve zbytku práce budou jednotlivé mračna bodů získána pomocí jednoho skenování označována také jako skeny. Termín sken bude občas také využíván pro mračno bodů a 3D model dohromady, aby se předešlo neustálému rozepisování "mračno bodů a 3D model".) a 3D model.

Obrázek 3.1: Fotografie skeneru VIVID 9i.[8]

3.2 Ovládání 3D skeneru

Ke 3D skeneru VIDIV 9i firma Konica Minolta také poskytuje SDK (software development kit), které umožňuje aplikaci, aby se skeneru připojila a ovládala ho. Toto SDK má určitá omezení, ale jelikož je aplikace implementována pro tento konkrétní typ skeneru (VIVID 9i), bude nezbytné přizpůsobit se požadavkům a limitům jejího SDK.

SDK je dodáváno s knihovnami jak pro 32-bitovou verzi systému, tak pro 64-bitovou verzi, ale SDK je v jazyce C++. To znamená, že aplikace bude muset být psána v čistém C++ nebo může být implementována například v .NET[24] frameworku s použitím i dalších jazyků, jako je například C#.

Nejvýraznější omezení kladené tímto SDK je to, že vyžaduje hlavičkový soubor "windows.h"a mnoho dalších, které jsou přes "windows.h"připojeny. To aplikaci omezuje pouze na platformu windows. Lze předpokládat, že SDK pro další platformy není plánováno, neboť

Obrázek 3.2: Fotografie samotného skeneru VIVID 9i bez tripodu.[9]

VIVID 9i je skener z roku 2006 a poslední aktualizace SDK a jeho dokumentace byla v roce 2010.

Kromě toho ještě SDK vyžaduje pro připojení ke skeneru, aby aplikace běžela v administrátorském režimu a měla k dispozici knihovnu wnaspi64.dll v 64-bitové verzi nebo wnaspi32.dll ve 32-bitové verzi, pokud je připojena pomocí USB-SCSI konvertoru.

3.3 Ovládání fotoaparátu

K fotoaparátu EOS 100D firma Canon poskytuje SDK[14], které umožňuje jeho ovládání. Kromě EOS 100D SDK podporuje i mnoho dalších fotoaparátů značky Canon, takže aplikace není limitována na jeden konkrétní typ fotoaparátu. SDK je v jazyce C++ a má 32-bitovou verzi a 64-bitovou verzi. V 64-bitové verzi jsou umožněny pouze základní funkce jako zpracování fotografií a různé konverze formátů. Komunikace s fotoaparátem v 64-bitové verzi není a musí být využita 32-bitová verze SDK. To klade na aplikaci další omezení, tj. musí to být 32-bitová aplikace. Kromě toho SDK vyžaduje, stejně jako SDK pro skener, hlavičkové soubory "windows.h".

3.4 Ovládání otočného stolku

Otočný stolek je ovládán pomocí řídící jednotky MARS 2 připojené k počítači, jak již bylo zmíněno v části 3.1.3. K jednotce MARS 2 není poskytováno žádné SDK, ale je s ní přímo komunikováno přes sériový port pomocí specifického protokolu[25]. To znamená, že bude potřeba naimplementovat vlastní ovladač pro tuto jednotku pro lepší manipulaci s ní. Na rozdíl od předchozích zařízení tedy ovládání otočného stolku nevyžaduje nic, co by mohlo způsobovat větší problémy a neklade na aplikaci žádná omezení.

Obrázek 3.3: Fotografie fotoaparátu Canon EOS 100D.

3.5 Omezení kladená na aplikaci

Podle informací z předchozích odstavců vyplývá, že co se softwarových omezení týče, aplikace bude muset být implementována jako C++ nebo .NET aplikace pouze pro windows a především ještě ve 32-bitové verzi. Z hardwarových požadavků je zde pouze to, že počítač musí mít alespoň tři USB porty pro připojení všech zařízení.

3.6 Model kamery

Jelikož jak skener, tak fotoaparát jsou v podstatě kamery, musíme si zde kameru definovat.

3.6.1 Pinhole kamera

Kamera je zde reprezentována jako takzvaná "pinhole"kamera. Uvažujeme tedy, že kamera nemá žádnou čočku a clona má v sobě nekonečně malý otvor takový, že z koherentního svazku paprsků projde pouze jeden[2].

[2]To znamená, že otvor propustí z každé možného směru nejvýše jeden paprsek.

Model kamery, viz obrázek 3.4, funguje tak, že máme definovaný prostor L v $\mathbb{R}^3$ s ortonormální pravotočivou bází[3] skládající se z vektorů x, y a z a počátkem O. Tento prostor L budeme nazývat lokálním souřadným systémem kamery.

Nyní si definujeme optickou osu o, což je přímka v prostoru L, a bod S v prostoru L, který označíme jako střed kamery. Pro jednoduchost definujme, že optická osa o prochází počátkem O a její směr určuje vektor z. Bod S umístíme do počátku O. Tím jsme získali osu o, která určuje směr pohledu kamery a bod S odpovídá onomu nekonečně malému otvoru v cloně.

Dále zde máme obrazovou rovinu a, což je rovina kolmá na optickou osu o a je ve vzdálenosti f od bodu S ve směru vektoru z. Vzdálenost f budeme označovat za ohniskovou vzdálenost.

Každý přicházející paprsek, který prochází otvorem v cloně S, musí v nějakém bodě protnout[4] obrazovou rovinu a, kde vytváří získaný obraz. Tento obraz, vznikající na a, je převrácený podle os X a Y. Převrácení je pro nás nevýhodné, a proto si uděláme upravený model, viz obrázek 3.5. Zde si můžeme představit ještě jednu obrazovou rovinu b, rovnoběžnou s a a ve vzdálenosti f od středu S, ale v opačném směru, tedy ve směru vektoru $-z$. V tomto modelu nejsou žádné čočky, které by paprsky lámaly, paprsky procházejí středem S a roviny a a b jsou podle S středově souměrné. Z toho vyplývá, že v rovině b bude vznikat stejný obraz, jako v rovině a, ale nebude převrácen podle žádné z os. Od této chvíle budeme za "obrazovou rovinu"označovat pouze rovinu b.

Je dobré si povšimnout, že velikost obrazu v obrazové rovině je přímo úměrná ohniskové vzdálenosti f. Zvětšení f zvětší i obraz. Pokud tedy řekneme, že obraz na ohniskové rovině má konstantní rozměry, změny v ohniskové vzdálenosti změní velikost kónusu paprsků, který bude moci projít středem S a naší oblastí v obrazové rovině.

3.6.2 Matematická reprezentace kamery

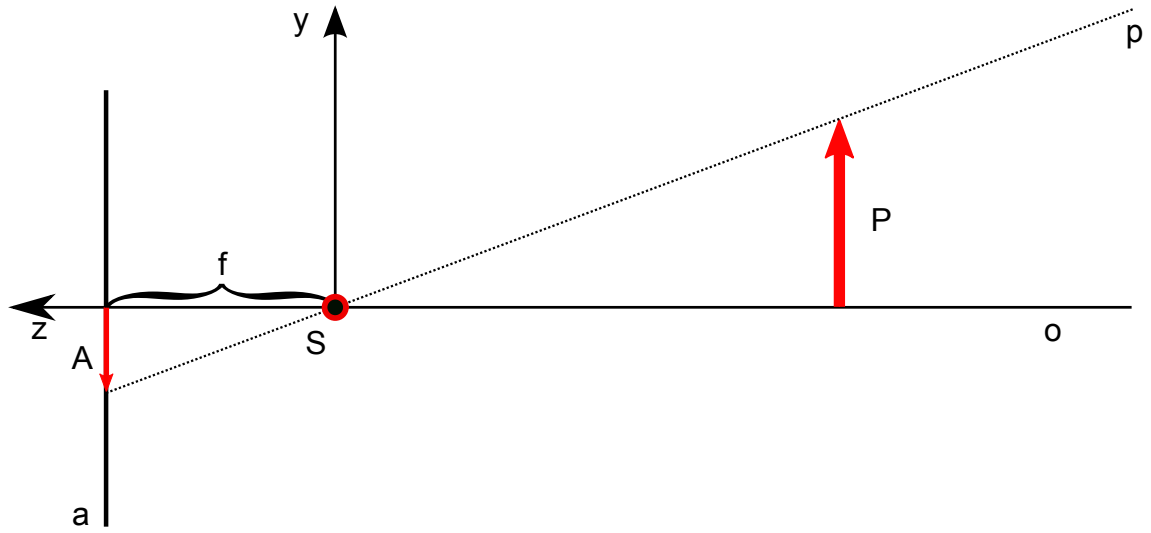
Z bodu X, což je nějaký 3D bod ve světových souřadnicích, potřebujeme získat 2D bod U v souřadnicích fotografie. Pro tuto projekci je nezbytné použít homogenní souřadnice[28] a výsledná projekce lze vyjádřit následující rovnicí.

$$\begin{bmatrix} Ux \\ Uy \\ 1 \end{bmatrix} = P \begin{bmatrix} Xx \\ Xy \\ Xz \\ 1 \end{bmatrix}$$

Ux a Uy jsou souřadnice promítnutého bodu (bod U) vůči levému hornímu okraji fotografie, Xx, Yy, Zz jsou souřadnice promítaného bodu (bod X) vůči světovému souřadnému

[3] Báze se skládá ze tří vektorů, kde všechny tři mají jednotkovou vzdálenost, jsou na sebe kolmé a vektorový součin prvního a druhého vektoru dá třetí vektor. $x \times y = z$

[4] Samozřejmě pokud paprsek není kolmý na optickou osu, v tom případě jí neprotne nikdy, ale takovéto paprsky nepotřebujeme.



Obrázek 3.4: Model kamery typu "pinhole". S - střed kamery, o - optická osa kamery, y - vektor lokálního systému kamery směřující vzhůru, z - vektor lokálního systému kamery, f - ohnisková vzdálenost, a - obrazová rovina, P - zobrazovaný objekt, A - obraz objektu P zobrazený naší kamerou na obraz A v obrazové rovině a. Paprsek p znázorňuje zobrazení bodu, který odpovídá vrcholu objektu P do roviny a.

systému a P je projekční matice 3×4. Matici P je možné rozdělit na matici K a E, kde matici K budeme označovat jako vnitřní parametry kamery[5](Nebo také kalibrační matice kamery.) a matici E budeme označovat jako vnější parametry kamery[6] (póza kamery). Matice K má rozměry 3×3, matice E má rozměry 3×4 a platí, že $P = KE$.

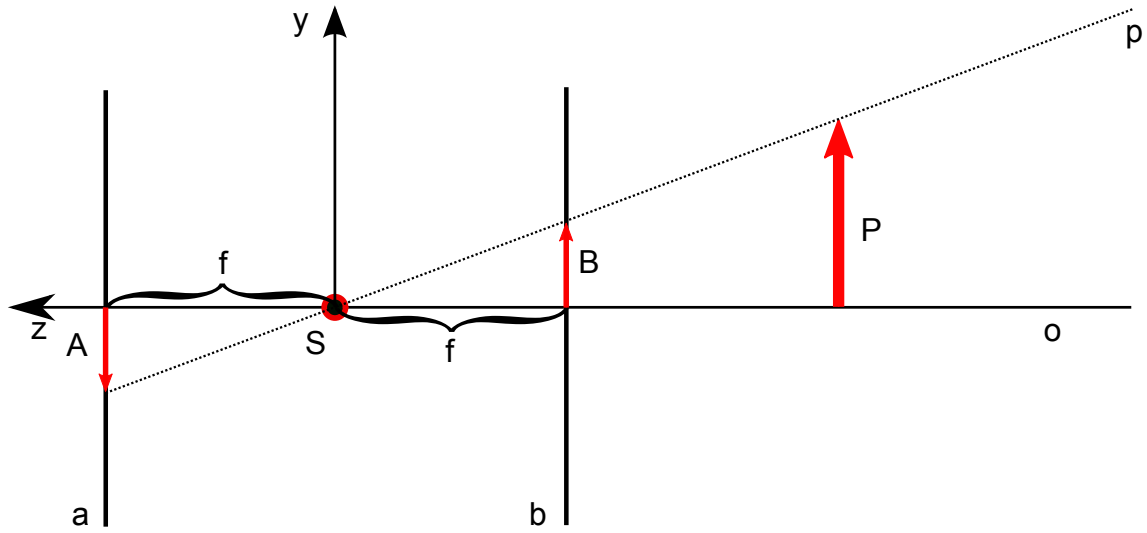
Předpokládejme, že v našem případě nedochází ke zkosení fotografie a můžeme matici K zjednodušit do následujícího tvaru.

$$K = \begin{bmatrix} fx & 0 & Sx \\ 0 & fy & Sy \\ 0 & 0 & 1 \end{bmatrix}$$

V tomto případě parametr fx je ohnisková vzdálenost, která je vyjádřena v jednotkách velikosti pixelu v ose X, obdobně je tomu i pro fy. Pokud jsou pixely čtvercové, tak platí, že $fx = fy$. Hodnoty Sx a Sy v našem případě označují pozici průsečíku optické osy kamery ve fotografii (V naprosté většině případů je to střed fotografie.).

Matici E můžeme vyjádřit jako složení matice R a vektoru C', kde R je rotační matice 3×3 a C' je trojrozměrný vektor. Vektor C' a jeho jednotlivé složky $C'x$, $C'y$ a $C'z$ označují souřadnice počátku světové souřadné soustavy v kamerových souřadnicích. Můžeme tedy

[5] z anglického "Intrinsic parameters/matrix"
[6] z anglického "Extrinsic parameters/matrix"



Obrázek 3.5: Rozšířený model kamery typu "pinhole"postavený na modelu 3.4. Zde byla přidána obrazová rovina b, na které se vytváří obraz B zobrazovaného objektu P. Oproti obrazu A, obraz B už není převrácen ani podle jedné z os. Díky trojúhelníkové podobnosti a symetrii b a a podle S, A i B se neliší ničím kromě převrácení os.

říci, že k bodu C' existuje ještě bod C takový, že označuje pozici středu kamery ve světových souřadnicích, a platí, že $C' = -RC$.

$$E = \begin{bmatrix} R & -RC \end{bmatrix} = \begin{bmatrix} R & C' \end{bmatrix}$$

Důležité je zmínit, že matice R a tudíž i matice E, nereprezentují transformaci kamery vůči světové souřadné soustavě, ale transformaci světového souřadného systému vůči kameře. To znamená, že bod ve světových souřadnicích vynásobený zleva maticí E bude transformován do systému kamery. Následně se takto transformovaný bod zleva vynásobí maticí K a tím získáme souřadnice promítnutého bodu ve fotografii.

Projekční matici P nakonec získáme složením matic K a R a vektoru C takto.

$$P = \begin{bmatrix} KR & | & -KRC \end{bmatrix}$$

3.7 Obecný popis funkce aplikace

3.7.1 Zjednodušený popis celého procesu

Jak již bylo zmíněno v úvodu 1.1, aplikace má uživateli umožnit naskenovat zvolený objekt. To konkrétně znamená, že aplikace s využitím zařízení zmíněných v části 3.1, tj. s využitím 3D skeneru, fotoaparátu a otočného stolku, musí být schopna obstarat mračno bodů reprezentující skenovaný objekt, jeho 3D model a fotografie, a zároveň toto všechno získat z co nejvíce směrů.

Následně aplikace složí získaná data dohromady do jedné scény. To znamená, že jednotlivé skeny (mračna bodů) a 3D modely, které byly získány postupným skenováním, budou v prostoru umístěny tak, aby na sebe co nejlépe navazovaly(Aby výsledný model a výsledné mračno bodů co nejpřesněji reprezentovaly skenovaný objekt.). Zároveň bude třeba pro každou fotografii získat co nejlepší odhad pozice, kde se v prostoru nacházel fotoaparát, a jaká byla jeho orientace, když danou fotografii pořizoval. To nám umožní, aby textura vytvářená z fotografie správně pasovala na model získaný 3D skenerem.

Ve chvíli, kdy jsou data pořízená v prvním kroku složena dohromady do jedné scény, už nic nebrání v tom, aby byla tato scéna uložena. Lze předpokládat, že uživatel bude chtít dále výslednou scény nějakým způsobem používat, takže bude vhodné pro její uložení zvolit již používaný formát podporovaný dalšími aplikacemi.

3.7.2 Získání dat

Sběr dat o skenovaném objektu bude fungovat na jednoduchém principu "naskenuj objekt", "vyfotografuj objekt", "otoč stolek", "opakuj tolikrát, kolikrát je třeba".

Jak již bylo zmíněno v části 3.1.1, SDK 3D skeneru umožňuje naskenování objektu a následné stažení dat do počítače. SDK umožňuje získat ze skeneru mračno bodů, polygonální 3D model i barevnou informaci. Všechny tyto tři typy dat budeme dále potřebovat pro sestavení výsledné scény. Co se týče barevné informace, tak ta je potřeba především jako barva pro vrcholy z mračna bodů. U mračna bodů nám nízké rozlišení fotografie nevadí, neboť budeme mít jeden pixel na každý bod, což je naprosto dostatečné.

Dále budeme potřebovat získat fotografii skenovaného objektu. K tomu bude využito SDK fotoaparátu, které umožňuje pořízení fotografie a její stažení do počítače. Následně bude třeba otočit stolkem o určitý úhel, což znamená, že musíme vytvořit ovladač pro stolek, který umožní jeho rotaci o konkrétní úhel.

K provedení tohoto procesu potřebujeme znát, kam ukládat získaná data, kolik skenů/fotografií provést a o jaký úhel otáčet stolek.

3.7.3 Pozice a orientace skeneru a fotoaparátu

K tomu, abychom byli schopni složit výslednou scénu, je nezbytné vědět, kde v prostoru byl skener/fotoaparát umístěn vůči otočnému stolku a jaká byla jejich orientace (Dále budou pozice a orientace daného zařízení označovány jako jeho póza.).

To je poměrně problematické. U skenů a 3D modelů máme informaci o tom, kde se v prostoru jednotlivé body nacházejí a jejich pozice jsou v reálných jednotkách, přesněji v milimetrech. Pozice bodů je ovšem v lokálním souřadném systému kamery (3.6). Lokální souřadný systém kamery je definován tak, že kamera je umístěna do počátku tohoto systému, její směr vzhůru je ve směru kladné osy Y a směr pohledu je v záporném směru osy Z. Co

se ale orientace a pozice kamery vůči otočnému stolku týče, jsme schopni pouze udělat odhad vzdálenosti skeneru od skenovaného objektu (Vzdálenost od skenovaného objektu můžeme získat z 3D modelu nebo mračna bodů. Problém ale je, že v nejhorším případě se nepodaří naměřit vůbec žádné body při skenování. To znamená, že s těmito informacemi nelze spolehlivě pracovat.). Co se týče fotografie a fotoaparátu, zde nemáme vůbec žádné informace o jeho póze.

Toto přináší značný problém, neboť bez toho, abychom znali pózy skeneru a fotoaparátu vůči otočnému stolku, nejsme schopni skeny, modely a fotografie složit dohromady do výsledné scény.

3.7.4 Definice světových souřadnic

Jelikož v reálném světě není definováno nic jako světový souřadný systém, musíme si ho zadefinovat. Máme tři zařízení (3D skener, fotoaparát a otočný stolek) v prostoru a bez světového souřadného systému můžeme pracovat jen s pozicemi a orientacemi jednoho zařízení vůči jinému, tj. v jejich lokálních systémech.

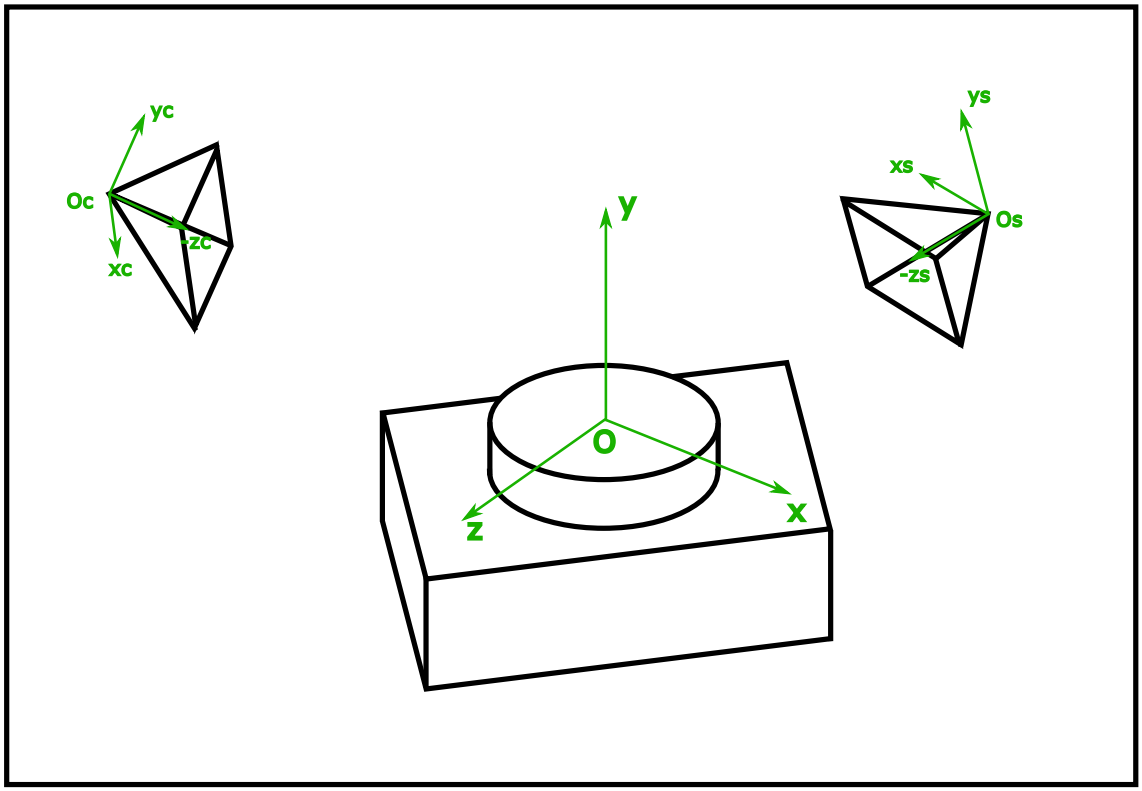
Proto si definujeme světový souřadný systém jako ortonormální trojrozměrný pravotočivý systém. To znamená, že náš souřadný systém bude mít tři bázové vektory, které jsou na sebe navzájem kolmé a všechny vektory mají stejnou délku. Délku pro bázové vektory zvolíme v reálných jednotkách, abychom měli snadno určitelný vztah mezi tímto systémem a reálným světem. Pro přehlednost a jednoduchost definujeme délku každého z bázových vektorů jako jeden metr. V rámci konvencí a pro zjednodušení orientace, označíme bázové vektory jako vektory x, y a z v tomto pořadí a tyto bázové vektory definují osy X, Y a Z, které jsou definovány počátkem našeho systému a jednotlivými bázovými vektory.

Pro zjednodušení skládání skenů, modelů a fotografií dohromady, definujme počátek světového souřadného systému jako bod, kde se nachází průnik plochy otočného stolku s osou, podle které otočný stolek rotuje. Dále definujme, že osa rotace otočného stolku je osa Y našeho světového systému souřadnic.

To znamená, že pokud dokážeme spočítat pózu skeneru a fotoaparátu v tomto souřadném systému, správné umístění jednotlivých skenů, modelů a fotografií bude vyřešeno jednoduchou rotací objektů podle světové osy Y o daný úhel.

3.7.5 Rekonstrukce pózy

Tím se dostáváme k problému, jak jednotlivé pózy vůbec získat. Téma rekonstrukce pózy nepatří mezi triviální problémy počítačového vidění a existují různé metody, jak pózy rekonstruovat. Každá metoda má samozřejmě různou přesnost výsledků a vyžaduje jiná vstupní data.



Obrázek 3.6: Souřadné systémy skeneru, fotoaparátu a otočného stolku (světový souřadný systém). Bod O a vektory x, y a z definují světový souřadný systém. Bod Oc a vektory xc, yc a zc definují lokální souřadný systém fotoaparátu a bod Os s vektory xs, ys a zs definují lokální souřadný systém skeneru. Velikosti bázových vektorů v ilustraci neodpovídají jejich skutečným rozměrům. Vektory světového souřadného systému by měly mít délku jeden metr a vektory obou lokálních systémů by měly měřit jeden milimetr.

Bude třeba rekonstruovat pózu jak pro 3D skener, tak pro fotoaparát a obě dvě pózy musí být vzhledem k otočnému stolku. Nemáme vždy zaručeno, že skener dokáže naměřit nějaké body, proto na ně nemůžeme při rekonstrukci spoléhat.

Věc, kterou spolu mají skener i fotoaparát společné, je fakt, že obě zařízení jsou kamery. Obě zařízení umožňují určitým způsobem zaznamenat barevnou informaci a z toho už můžeme vycházet při rekonstrukci pózy.

Pro rekonstrukci pózy z fotografií existuje řada algoritmů, které se liší podle případu, ve kterém se dají využít a podle toho, kolik informací už známe. Většina těchto algoritmů vychází z toho, že máme ve fotografii zachyceny nějaké významné body v prostoru. Projekce těchto bodů do fotografií budeme označovat jako klíčové body. Tyto klíčové body je možné získat více způsoby. Nejjednodušším způsobem je nechat uživatele, aby tyto body zadal sám. Další možností by bylo využít některý z algoritmů pro automatické hledání klíčových bodů,

což je ovšem pokročilejším tématem počítačového vidění. Proto se spokojíme s tím, že uživatel klíčové body do fotografií zadá ručně. Abychom dokázali získat pózu ve světových souřadnicích, musíme mít nějakou informaci o vztahu kamery a světového souřadnému systému. V tomto případě by bylo nejjednodušší, kdybychom znali pozice klíčových bodů ve světových souřadnicích.

3.7.6 Kalibrační objekt

Nejjednodušším řešením tohoto problému by bylo použití nějakého specifického objektu, který bude mít jednoduchý tvar, jeho velikost bude snadné změřit a bude mít výrazné rysy, na kterých půjde snadno rozpoznat klíčové body. Nejjednodušší a nejlepší objekt pro tento případ by byla krychle. Její tvar je poměrně jednoduchý a běžný (Sehnat objekt ve tvaru krychle by neměl být problém.) a má 8 vrcholů; z nich je možné v danou chvíli vidět 7 naráz, když se zvolí správné natočení. Pro těchto 7 klíčových bodů by bylo poměrně snadné zadat souřadnice ve světových souřadnicích. Pokud bychom umístili objekt na otočný stolek tak, že by jeden z vrcholů ležel přesně na pozici, kde osa rotace stolku protíná jeho plochu, tento vrchol bychom označili souřadnicemi $[0, 0, 0]$. Díky pravoúhlosti krychle by další vrchol, ten nad vrcholem ve středu stolku, byl označen souřadnicemi $[0, A, 0]$, kde A označuje délku hrany krychle. Dalších pět vrcholů bychom označili obdobným způsobem.

3.7.7 Známé parametry kamer

Volba algoritmu pro rekonstrukci pózy záleží také na tom, jaké informace máme o kameře. Obecně platí, čím méně toho známe, tím více bodů budeme k rekonstrukci potřebovat.

Podle části 3.6.2 víme, že kamera je definována vnitřními (kalibrační matice) a vnějšími parametry (póza). Vnější parametry neznáme, ale pokud bychom znali vnitřní parametry, mohlo by to ulehčit naší práci. K rekonstrukci kalibračních matic potřebujeme získat ohniskovou vzdálenost vyjádřenou v pixelech jak v ose X tak v ose Y a posunutí středů kamer (bod kde optická osa kamery protíná obrazovou rovinu) v ose X a ose Y též vyjádřené v pixelech. Z informací o skeneru a fotoaparátu víme, že "shear[7]"parametr je nulový pro obě kamery (Obě osy, i X i Y jsou na sebe kolmé.), takže tento parametr řešit nemusíme.

Co se týče skeneru, u toho je to poměrně jednoduché. U toho je pevně stanovené rozlišení senzoru, tj. 640×480. Co se ohniskové vzdálenosti týče, tak nám stačí z nějakého skenu získat 2 body a z nich ohniskové vzdálenosti dopočítat dokážeme.

Naším cílem je získat ohniskovou vzdálenost vyjádřenou v pixelech. Vyjděme z modelu kamery 3.6, viz obrázek 3.5. Ohnisková vzdálenost je vlastně vzdálenost obrazové roviny kamery. Pokud máme nějaký bod B v prostoru a vytvoříme přímku p takovou, že protíná střed kamery a bod B, tak průsečík přímky p a obrazové roviny reprezentuje pozici daného bodu ve fotografii. Vezměme tedy nějaký sken pořízený skenerem. Najděme ve skenu dva body $A1 = [a1x, a1y, a1z]$ a $A2 = [a2x, a2y, a2z]$ na pozicích $U1 = [u1, v1]$ $U2 = [u2, v2]$. Pozicemi $U1$ a $U2$ je myšlena pozice v hloubkové mapě získané skenerem, pozice $A1$ a $A2$

[7]Shear je zkosení pixelů. Jelikož osy X a Y jsou na sebe kolmé, ke zkosení nedochází.

jsou souřadnice bodů v lokálním souřadném systému kamery. Body volíme tak, aby platilo že $u1! = u2$ a $v1! = v2$. Nyní vytvoříme body $B1$ a $B2$ z bodů $A1$ a $A2$ takovým způsobem, že $B\# = A\#/a\#z$. Tím jsme promítli body $A1$ a $A2$ do jedné roviny, která je rovnoběžná s rovinou XY a tím je rovnoběžná s obrazovou rovinou kamery. V tuto chvíli bychom mohli říci, že pokud je ohnisková vzdálenost kamery rovna jedné, tj. $f = 1$, tak velikost jednoho pixelu v ose X je $dX = (a2x - a1x)/(u2 - u1)$ a obdobné platí i pro velikost v ose Y dY. Tento výpočet vychází z principu podobnosti trojúhelníků, viz obrázek 3.5. My ovšem nechceme jednotkovou ohniskovou vzdálenost, ale jednotkovou velikost pixelu. To vyřešíme snadno. Ohnisková vzdálenost vyjádřená ve pixelech je pro osu X je $fX = f/dX$ a pro osu Y je $fY = f/dY$. K ohniskové vzdálenosti nám už chybí pouze informace o tom, kde optická osa kamery protíná obrazovou rovinu. K tomu nám poslouží rozměry skeneru 640×480 a pozice průniku bude bod $S = [sx, sy]$, kde $sx = 640/2$ a $sy = 480/2$. Z těchto parametrů už dokážeme sestavit kalibrační matici $K1$.

$$K1 = \begin{bmatrix} fX & 0 & sx \\ 0 & fY & sy \\ 0 & 0 & 1 \end{bmatrix}$$

Nyní bychom potřebovali získat kalibrační matici fotoaparátu $K2$. Zde už nebudeme schopni získat parametry takto snadno, jako tomu bylo u skeneru. Co ovšem můžeme využít, je znalost informací o fotoaparátu. Naprostá většina moderních fotoaparátů ukládá do fotografií řadu metadat ve formátu EXIF[16][17]. Z těchto metadat dokážeme získat rozměry fotografie a také ohniskovou vzdálenost v době získání fotografie. Problém s ohniskovou vzdáleností je v tom, že je vyjádřena v reálných jednotkách (obvykle v milimetrech), ale my ji potřebujeme v pixelech. K tomu nám mohou pomoci další parametry ukládané do metadat fotografií. V tomto případě jsou to konkrétně parametry $FocalPlaneXResolution$ a $FocalPlaneYResolution$. Tyto dva parametry nám říkají, kolik pixelů se v dané ose vejde na úsek o velikost $FocalPlaneResolutionUnit$. $FocalPlaneResolutionUnit$ je další z běžných parametrů v metadatech a může nabývat hodnot buď jeden centimetr, nebo jeden palec. V našem případě je to pro daný fotoaparát jeden palec. Když ještě budeme mít šířku W a výšku fotografie H, jejichž získání je triviální, můžeme sestavit matici $K2$ tak, že $fX = f * mmToInch * FocalPlaneXresolution$ a $fY = f * mmToInch * FocalPlaneYresolution$, kde f je ohnisková vzdálenost v milimetrech a $mmToInch$ je konstanta pro převedení milimetrů na palce. Pozici průsečíku optické osy kamery a obrazové roviny vypočteme jako bod $S = [sx, sy]$, kde $sx = W/2$ a $sy = H/2$ a sestavíme výslednou matici:

$$K2 = \begin{bmatrix} fX & 0 & sx \\ 0 & fY & sy \\ 0 & 0 & 1 \end{bmatrix}$$

3.7.8 Algoritmus rekonstrukce pózy

Jak z předchozí části vyplývá, jsme schopni kompletně zrekonstruovat kalibrační matice obou dvou kamer (skeneru a fotoaparátu). To znamená, že jediné, co potřebujeme získat,

jsou vnější parametry, tedy póza. K tomu můžeme použít například algoritmus Perspective-3-Points[8].

Tento algoritmus umožňuje rekonstrukci pózy kamery, pokud máme tři body v prostoru A, B, C, známe jejich 3D souřadnice v systému, v němž chceme rekonstruovat pózu kamery a také máme jejich průměty do obrazové roviny kamery. Dále tento algoritmus vyžaduje znalost kalibrační matice kamery, kterou jsme rekonstruovali v minulém kroku v části 3.7.7 (Kdybychom neznali kalibrační matici kamery, museli bychom použít jiný typ algoritmu. Takové algoritmy fungují obdobně, ale kvůli více neznámým vyžadují více bodů.).

S tímto algoritmem nám bude bohatě stačit, aby uživatel do fotografie zadal tři body, ke každému zadal jeho souřadnice ve světových systému a algoritmus se postará o rekonstrukci samotnou. S využitím kalibračního objektu a jeho vhodným umístěním na otočný stolek, viz část 3.7.4, bude moci uživatel snadno zrekonstruovat pózu kamery. Toto bude nezbytné provést jak pro skener, tak pro fotoaparát.

3.7.9 Nepřesnosti rekonstrukce pózy

Problém u výše zmíněného algoritmu je nepřesnost výsledku. Tato nepřesnost je způsobena tím, že uživatel musí zadat body do fotografie. Zadávání do fotografie samo o sobě znamená nepřesnost, a čím menší je rozlišení fotografie, tím vyšší nepřesnost bude. Pro fotoaparát je rozlišení fotografie 5184×3456, což je docela vysoké rozlišení, ale fotografie získaná skenerem má pouze 640×480. Lze také předpokládat, že uživatel nebude naprosto přesný při zadávání a body budou různě posunuty.

Toto by mohlo znamenat výrazný problém, neboť pro složení skenů a fotografií do finální scény potřebujeme přesnost póz co nejvyšší. Z toho důvodu by bylo vhodné vytvořit nějaký nástroj, který uživateli umožní provádět malé korekce v rotaci a posunutí kamer. To, co nás bude nejvíce trápit z nepřesností, je pozice. I malá nepřesnost v pozici způsobí viditelné díry nebo překryvy v modelu. Její zpřesnění je na druhou stranu poměrně snadné. Co se týče rotace, tak k tomu už uživatel bude potřebovat poměrně dobrou prostorovou představivost, což může být problematické a i přes to může být rotování objektem v prostoru náročné. Bude proto vhodné vymyslet co nejjednodušší nástroj, aby se uživateli co nejvíce usnadnila práce.

3.7.10 Složení výsledné scény

Ve chvíli, kdy máme získána všechna data a zrekonstruované pózy fotoaparátu a skeneru, už nic nebrání tomu, abychom vytvořili výslednou scénu. Ta se bude skládat z mračna bodů, 3D modelu, který je reprezentován sítí trojúhelníků, kamer a fotografií. Kamera zde bude reprezentována svou pózou a kalibrační maticí.

[8] Popis tohoto algoritmu je v [26] kapitoly 7.2 a 7.3

Jelikož jsme si v části 3.7.4 vhodně zvolili světový souřadný systém, bude se nám skládání jednotlivých objektů (skenů, modelů, fotografií) provádět celkem snadno. Modely a mračna bodů ze skeneru jsou v jeho lokálním systému v milimetrech. První věc, kterou tedy provedeme je, že transformujeme mračna i modely tak, aby odpovídaly jednotkám globálního systému, tj. metrům. K tomu využijeme transformaci pro škálování[9] a pozice se vynásobí faktorem 10^{-3}. Jelikož máme pózu skeneru v globálním systému, použijeme tuto pózu jako transformaci[10] a transformujeme jak model, tak mračno bodů z lokálního do globálního souřadného systému. Takto jsme získali vše vyjádřeno v globálním souřadném systému. To znamená, že teď musíme správně natočit mračna i modely tak, jak se otáčel stolek mezi skenováním. Jelikož víme, jaký byl úhel otočení stolku mezi skenováním a souřadnou soustavu jsme zvolili tak, že se stolek otáčí kolem osy Y, stačí nám vytvořit rotaci[11] o daný úhel kolem osy Y. Obdobné provedeme i pro fotoaparát. Zde nám stačí pootočit naškálované pózy fotoaparátu kolem osy Y stejnou rotační maticí, jako byla použita u mračen bodů a modelů.

3.7.11 Texturování 3D modelu

Když máme sestavené všechny geometrické prvky scény, nyní bude třeba aplikovat texturu a výsledný 3D model.

První problém, který bude třeba vyřešit je to, že fotoaparát a skener mohou mít libovolnou pozici ve scéně. Pozice záleží pouze na tom, jak uživatel tyto zařízení umístí v prostoru a na to nemáme žádné pevně dané požadavky. Obecným předpokladem je, že skenovaný objekt je umístěn na otočném stolku a tudíž skener i fotoaparát jsou mimo skenovaný objekt a fotí/skenují jej z venku. Dále lze předpokládat, že fotoaparát i skener jsou umístěny tak, že směry jejich pohledů jsou co nejkolmější na světovou osu Y (Jde především o to, že objekt rotuje kolem osy Y, která v naší scéně určuje směr vzhůru. Pokud by tedy uživatel skenoval/fotil objekt shora/zdola, rotování objektem by bylo naprosto zbytečné, neboť by se výsledky lišily jen pootočením podle optické osy kamery.).

Jelikož tedy máme pózy skeneru a fotoaparátu různé, bude nezbytné určit ke každému modelu, která fotografie byla získána ze směru co nejbližšího směru skenování a tuto fotografii použít pro texturování. To se dá provést poměrně snadno, neboť pozice skeneru a fotoaparátu vlastně rotují kolem osy y v našem modelu (3.7.4). Proto nám stačí promítnout pozice do roviny XZ, spočítat úhly mezi vektory pozic skeneru a všemi vektory pozic fotoaparátů a zvolit ten, jehož poziční vektor svírá s pozičním vektorem skeneru nejmenší úhel.

Ve chvíli, kdy máme určené korespondence mezi skenem (modelem) a fotografií, samotné otexturování je už pouze věcí promítnutí vrcholů modelu pomocí projekční matice fotoaparátu. Jelikož známe vnější (póza) i vnitřní (kalibrační matice) parametry kamery, projekční matici získáme jednoduchým vynásobením kalibrační matice s pózou, viz část 3.6.2. Ve chvíli,

[9] Moderní počítačová grafika[28] část 21.3.4
[10] Moderní počítačová grafika[28] část 21.3
[11] Moderní počítačová grafika[28] část 21.3.2 a 21.3.3

kdy máme i texturovací souřadnice, máme kompletně sestavenou finální scénu a můžeme ji exportovat do nějakého z běžných formátů.

3.7.12 Formát uložení výsledné scény

Jelikož v tomto momentě už bude celá scéna sestavena a bude zbývat pouze její uložení do nějakého formátu, musíme zvolit, jaký formát použijeme. Naneštěstí neexistuje žádný jednotný univerzální formát, který by byl používán všemi aplikacemi a umožňoval reprezentaci všeho, co potřebujeme. Naše scéna obsahuje mračno bodů, texturovaný 3D model reprezentovaný jako síť trojúhelníků a také kamery odpovídající jednotlivým fotografiím (texturám). To jsou velmi specifická data, která chceme ukládat. Problémem je, že většina běžných formátů, které podporují 3D texturované modely (.obj[12], .fbx[11][32]), nepodporují mračna bodů, nemluvě o vlastních datech, jako jsou kamery (Kamery v tomto případě nedělají veliké problémy. Ty by bylo možné nahradit nějakými jinými elementy, nebo by mohly být uloženy do jiného souboru. Problémem je zde především 3D model a zároveň mračna bodů.).

Náš problém by mohl být vyřešen pomocí dvou konkrétních formátů. Formáty bundle[30] a PLY[13]. První z těchto formátů, tj. bundle, je formát používaný programy jako Bundler[31] a Visual SFM[33] pro 3D rekonstrukci scény ze sady fotografií. Problémem je, že tento formát umožňuje pouze reprezentaci mračna bodů a kamer. To by ovšem šlo využít spolu s nějakým formátem pro 3D modely a mít scénu exportovanou do dvou souborů. Druhý z formátů, tj. PLY, je poměrně jednoduchý formát, který umožňuje ukládání bodů, modelů, a jelikož PLY nemá žádnou pevnou specifikaci (uživatel si může libovolně definovat své vlastní elementy), můžeme si v něm definovat a reprezentovat i kamery. Výhodou tohoto formátu je především to, že je možné výslednou scénu otevřít, modifikovat a vykreslit například programem MeshLab[3], což je open-source program pro pokročilé zpracování 3D geometrií a právě umožňuje práci jak s 3D modely, tak i s mračny bodů.

3.8 Algoritmu p3p

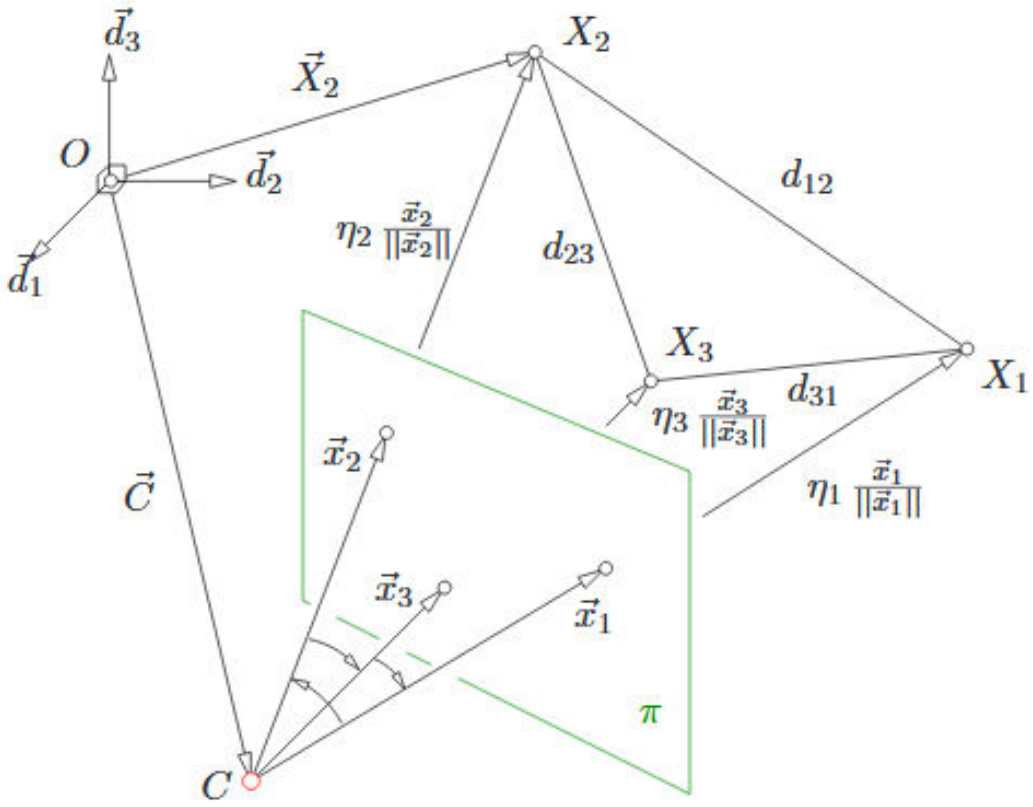
Jak již bylo řečeno v části 3.7.8, algoritmus p3p slouží k rekonstrukci pózy kamery. K tomu je nezbytné, abychom znali kalibrační matici kamery a měli tři body v prostoru, ve kterém pózu hledáme, a také abychom znali pozice průmětů těchto bodů v obrazové rovině kamery.

3.8.1 Definice parametrů

Kamera, jejíž pózu hledáme (dále nazývána jen kamera), má projekční matici P, ve tvaru definovaném v části 3.6.2. P se tudíž skládá z kalibrační matice K, rotační matice kamery R a vektoru pozice kamery C_δ. Z těchto parametrů známe pouze K a hledáme R a C. Světový souřadný systém, ve kterém hledáme pózu kamery, označíme jako prostor δ s ortonormální bází složenou z vektorů d_1, d_2 a d_3. Díky maticím K, R a ohniskové vzdálenosti máme prostorů mnohem více[12].

[12] Tyto prostory jsou dobře popsány v [26] v části 7.1 a v dalších částech této práce se předpokládá znalost těchto prostorů, vztahů mezi nimi a jejich vlastností.

Vstupními parametry jsou tři body v prostoru δ a jsou to body X_1, X_2 a X_3. K těmto bodům máme také zadané jejich pozice v prostoru α(pozice pixelů ve fotografii) a to jsou body x_1, x_2 a x_3, které jsou průměty bodů z δ do obrazové roviny kamery.



Obrázek 3.7: Ilustrace znázorňující základ na kterém pracuje p3p algoritmus. Máme zde světový souřadný systém, ve kterém se vše nachází a je definován počátkem O a vektory d_1, d_2 a d_3. Dále zde máme lokální souřadný systém kamery s počátkem C. Rovina π zde znázorňuje obrazovou rovinu kamery odpovídající rovině b v modelu na obrázku 3.5. Dále zde máme tři body X_1, X_2 a X_3, které jsou vyjádřeny ve světových souřadnicích a jejich projekce do π na body x_1, x_2 a x_3, jejichž pozice známe pouze v této rovině. Hodnoty d_{ij} označují vzdálenosti mezi body X_i a X_j a hodnoty η_i jsou vzdálenosti mezi bodem x_i a X_j. Úhly c_{ij} mezi vektory x_i a x_j jsou v ilustraci znázorněny, ale nejsou popsány.

3.8.2 Úhly a vzdálenosti

Prvním krokem algoritmu je výpočet úhlů mezi paprsky procházejícími středem kamery a našimi body (Paprsek vždy prochází bodem X_i, x_i a středem kamery.). To dokážeme pomocí skalárního součinu mezi vektory x_1, x_2 a x_3. Musíme je ale mít vyjádřené v nějaké ortonormální souřadné soustavě. My je máme vyjádřené v prostoru α. Jejich rozšířením o

hodnotu 1 je vyjádříme v prostoru β, ale ten také není ortonormální. Nejbližší ortonormální prostor je pro nás prostor γ, který získáme transformací vektorů x_i z β pomocí matice K.

$$x_{i_\beta} = \begin{bmatrix} x_{i_\beta} \\ 1 \end{bmatrix}$$

$$x_{i_\gamma} = K^{-1} x_{i_\beta}$$

Nyní když máme vektory x_i vyjádřeny v ortonormálním prostoru γ. Vypočteme kosiny úhlů c_{jk} mezi všemi dvojicemi paprsků skalárním součinem normalizovaných vektorů x_i.

Dalším krokem je vypočtení vzdáleností d_{jk} mezi dvěma body X_j a X_k v prostoru δ, což je triviální problém, jelikož body X_i jsme na vstupu dostali vyjádřené v δ.

3.8.3 Vzdálenost bodů X_i od kamery

Nyní potřebujeme zjistit, jaké jsou vzdálenosti bodů X_i od středu kamery. Tímto získáme trojice parametrů η_i takové, že platí.

$$\eta_i \frac{x_{i_\gamma}}{||x_{i_\gamma}||} = X_{i_\gamma}$$

Také víme, že jelikož X_i jsou vidět ve fotografiích, tak všechny η_i musí být kladná (Jinak by to znamenalo, že dané X_j je za kamerou.).

Z hodnot, které jsme získali dokážeme vyjádřit tři rovnice[13].

$$\begin{aligned} d_{12}^2 &= \eta_1^2 + \eta_2^2 - 2\eta_1\eta_1 c_{12} \\ d_{23}^2 &= \eta_2^2 + \eta_3^2 - 2\eta_2\eta_3 c_{23} \\ d_{31}^2 &= \eta_3^2 + \eta_1^2 - 2\eta_3\eta_1 c_{31} \end{aligned}$$

Tím máme soustavu tří kvadratických rovnic o třech neznámých. Převedeme je na jednu rovnici o jedné neznámé a vypočteme.

Převedení této soustavy není triviální a vede k rovnici čtvrtého stupně. Vyjádření a vyřešení této rovnice je detailně popsáno v [26] v §36 až §43.

3.8.4 Rekonstrukce R a C_δ

Ve chvíli, kdy známe η_1, η_2 a η_3, nám už nic nebrání v tom, abychom zrekonstruovali matici R a pozici kamery C_δ.

[13] Pomocí kosinové věty.

Nejprve si vyjádříme body X_i reprezentované v soustavě δ jako vektory $Y_{i\epsilon}$ v ortonormálním systému kamery (C, ϵ)[14] díky znalosti η_i.

$$Y_{i\epsilon} = \eta_i y_{i\epsilon} = \eta_i \frac{x_{i_\epsilon}}{||x_{i_\epsilon}||} = \eta_i \frac{f x_{i_\gamma}}{||f x_{i_\gamma}||} = \eta_i \frac{x_{i_\gamma}}{||x_{i_\gamma}||}$$

Vztah mezi $X_{i\delta}$ a $Y_{i\epsilon}$ je tedy:

$$\begin{aligned} Y_{1\epsilon} &= R(X_{1\delta} - C_\delta) \\ Y_{2\epsilon} &= R(X_{2\delta} - C_\delta) \\ Y_{3\epsilon} &= R(X_{3\delta} - C_\delta) \end{aligned}$$

Tím dostáváme tři vektorové rovnice v trojrozměrném prostoru, což dává devět rovnic o dvanácti neznámých. Díky podmínkám na ortonormálnost rotační matice R získáváme dalších sedm rovnic.

Pro výpočet R nejprve eliminujeme z rovnic C_δ a získáme soustavu:

$$\begin{aligned} Y_{2\epsilon} - Y_{1\epsilon} &= R(X_{2\delta} - X_{1\delta}) \\ Y_{3\epsilon} - Y_{1\epsilon} &= R(X_{3\delta} - X_{1\delta}) \end{aligned}$$

Dále využijeme vlastnost vektorového součinu při změně bází [15], čímž dostaneme:

$$(Y_{2\epsilon} - Y_{1\epsilon}) \times (Y_{3\epsilon} - Y_{1\epsilon}) = R((X_{2\delta} - X_{1\delta}) \times (X_{3\delta} - X_{1\delta}))$$

Z toho dostaneme trojici nezávislých vektorů Z_i vyjádřené v bázích δ a ϵ.

$$\begin{aligned} Z_{2_\epsilon} &= Y_{2\epsilon} - Y_{1\epsilon}, \quad & Z_{2_\delta} &= X_{2\delta} - X_{1\delta} \\ Z_{3_\epsilon} &= Y_{3\epsilon} - Y_{1\epsilon}, \quad & Z_{3_\delta} &= X_{3\delta} - X_{1\delta} \\ Z_{1_\epsilon} &= Z_{2_\epsilon} \times Z_{3_\epsilon}, \quad & Z_{1_\delta} &= Z_{2_\delta} \times Z_{3_\delta} \end{aligned}$$

Rotaci R tedy dokážeme získat tak, že:

$$\begin{bmatrix} Z_{1_\epsilon} & Z_{2_\epsilon} & Z_{3_\epsilon} \end{bmatrix} = R \begin{bmatrix} Z_{1_\delta} & Z_{2_\delta} & Z_{3_\delta} \end{bmatrix}$$

Díky lineární nezávislosti vektorů Z_i a faktu, že rotační matice má plnou hodnost (Nezávislá množina vektorů po rotaci zůstane nezávislá.), můžeme provést úpravu:

$$R = \begin{bmatrix} Z_{1_\epsilon} & Z_{2_\epsilon} & Z_{3_\epsilon} \end{bmatrix} \begin{bmatrix} Z_{1_\delta} & Z_{2_\delta} & Z_{3_\delta} \end{bmatrix}^{-1}$$

Translační vektor C_δ získáme díky znalosti R jako:

$$C_\delta = X_{i\delta} - R^T Y_{i\epsilon}$$

[14] Toto je vlastně lokální systém kamery reprezentovaný ve stejných jednotkách, jako světový souřadný systém a využívá bázi vektorů světového systému transformované pomocí matice R. Zároveň počátek tohoto systému je posunut o vektor C oproti světovému souřadnému systému.

[15] $x_{\beta'} \times y_{\beta'} = \frac{A^{-T}}{|A^{-T}|}(x_\beta \times y_\beta)$

3.9 Rozdělení aplikace na fáze

Z předchozích částí 3.7 a 3.8 vyplývá, že celý proces od získání dat po uložení výsledné scény bude poměrně komplikovaný a bude zahrnovat řadu činností. Proto bude vhodné rozdělit celý proces na menší celky, které od sebe budou co možná nejvíce oddělené, bude je možné provádět odděleně a tyto celky budou co možná nejvíce konzistentní v tom, co v nich uživatel bude provádět.

Jako první samostatný celek bude vhodné zvolit skenování a focení modelu, které budeme označovat za fázi "Získání dat". Na konci celého procesu je sestavení a export scény do konkrétního formátu. Tuto fázi budeme označovat jako "Export scény". To, co z procesu zbývá, je vytváření póz fotoaparátu a skeneru. Tuto část bude vhodné rozdělit na fáze "rekonstrukce pózy"a na "zpřesňování pózy". Rozdělení je vhodné kvůli tomu, že první a druhá část se od sebe velmi liší. V první části uživatel získá hrubý odhad pózy, jehož přesnost je závislá na přesnosti uživatele. Co se druhé fáze týče, tak v té uživatel pouze upravuje dříve získaná data(pózy).

3.9.1 Mezivýsledky

Jelikož celý proces bude rozdělen na jednotlivé fáze, které od sebe budou kompletně odděleny (Kompletním oddělením je myšleno to, že uživatel nebude muset provádět celý proces najednou, bude moci aplikaci mezi jednotlivými fázemi vypnout a bude moci pokračovat jindy v dalších fázích. Samozřejmě zde bude návaznost mezi jednotlivými fázemi, tj. před rekonstrukcí pózy a exportem musí být proveden sběr dat a export může být proveden až po rekonstrukci pózy, etc.). K tomu bude nezbytné, aby byla v každé fázi data, která budou vyžadována v dalších fázích, uložena.

3.9.2 Získání dat

Získání dat je první fáze celého procesu. Cílem této fáze, jak již název napovídá, bude skenování a fotografování objektu. Během tohoto procesu také bude vhodné získat sken a fotografie kalibračního objektu zmíněného v části 3.7.6. Tím se tato fáze stane jedinou fází, která bude vyžadovat použití skeneru, fotoaparátu a otočného stolku. Zároveň tato fáze nevyžaduje příliš veliké zapojení uživatele. Jediné co od něj bude vyžadováno, je zadání parametrů skenování (počet skenů, úhel otočení stolku), položení kalibračního objektu na stolek, jeho naskenování a následně položení objektu, který chce uživatel skenovat a spuštění skenovacího procesu.

Data získaná v této fázi budou především mračna bodů, jejich barevné informace, 3D modely a fotografie. Zde je vidět, že to už bude poměrně veliké množství dat. Proto bude vhodné vytvořit speciální adresářovou strukturu pro logické rozdělení dat na menší celky. Další informace, které v této fázi budou získaný, jsou informace o kalibračním objektu. To bude další mračno bodů, fotografie a 3D model. Dále zde máme ještě informace o tom, kolik skenů má být provedeno a jaké byly úhly mezi jednotlivými skeny.

Jelikož jsou jednotlivé skeny, modely a fotografie uloženy jako samostatné soubory, bude potřeba uchovávat informace o tom, v jakém pořadí byly skeny pořízeny, jaká jsou jména souborů a ještě další informace. Proto bude vhodné vytvořit speciální soubor, který budeme označovat jako "assembly script". Do tohoto souboru se uloží jména všech souborů s mračny bodů, modelů, fotografií, informace o kalibračním objektu, úhly mezi skeny a všechny další informace zmíněné v předchozím odstavci.

Jelikož v této fázi také získáváme informace o kalibračním objektu, můžeme nyní rovnou vytvořit kalibrační matice kamer dokud máme veškeré potřebné informace načtené v paměti. Tyto kalibrační matice mohou být rovněž přidány do assembly scriptu.

3.9.3 Rekonstrukce pózy

Tato fáze sama o sobě bude vyžadovat nejvíce práce uživatele. Bude potřeba načíst fotografie kalibračního objektu. Do těch následně uživatel zadá klíčové body(část 3.7.6) a ke každému z nich doplní pozici ve světových souřadnicích. To bude muset provést jak pro skener, tak pro fotoaparát. To znamená, že uživatel bude muset zadat alespoň tři body do každé fotografie, aby algoritmus mohl pracovat. Pro zvýšení přesnosti bude uživateli umožněno zadat libovolné množství vrcholů. Algoritmus následně proběhne pro každou trojici vrcholů a vybere se ten výsledek, pro který bude zpětná projekce bodů mít co nejmenší odchylku od původních pozic[16]. K tomu, aby bylo možné zrekonstruovat pózy, bude potřeba získat umístění a jména souborů s informacemi o kalibračním objektu (fotografie) a také budeme potřebovat kalibrační matice obou kamer. To vše bude možné získat z "assembly scriptu"vytvořeného při získávání dat.

Výstupem této fáze budou pózy skeneru a fotoaparátu, které se skládají vždy z translačního vektoru a rotační matice. Pózy uložíme do samostatného souboru, který umístíme do naší adresářové struktury vytvořené při získávání dat.

3.9.4 Zpřesnění pózy

Tato fáze bude navazovat na rekonstrukci pózy a sama o sobě vlastně není povinná. Pokud uživatel dokáže pózy dostatečně zrekonstruovat v předchozí fázi, nebude třeba nic zpřesňovat a může jít rovnou na fázi exportování.

Tato fáze bude vyžadovat, aby uživatel vylepšil už existující pózy. Jak již bylo zmíněno v části 3.7.9, bude třeba vytvořit nástroj pro zpřesňování pozice. K tomu by mohlo poměrně dobře posloužit, kdybychom uživateli vykreslili scénu, která bude obsahovat sken kalibračního objektu a základní osy světového souřadného systému (osy X, Y a Z). Následně by uživatel mohl sken ve scéně libovolně posouvat a rotovat s ním. Jako základ pro pozici skenu by byla použita póza získaná v minulé fázi. Uživatel by se snažil zarovnat objekt tak, aby

[16] Tím je myšleno, že se po rekonstrukci pózy sestaví z pózy a kalibrační matice projekční matice a všechny uživatelem zadané body (uživatelem zadané trojrozměrné body) se touto projekční maticí promítnou do fotografie a změří se chyba v projekci jako vzdálenost mezi promítnutým bodem a pozicí, kam tento bod byl ve fotografii zadán uživatelem. Nakonec zvolíme tu pózy, která má nejmenší chyby.

vrchol označený jako střed souřadné soustavy byl ve skutečném středu a ostatní vrcholy by byly na odpovídajících místech. Pokud uživatel bude používat ke kalibraci krychli, jak bylo doporučeno v části 3.7.6, bude zarovnání objektu s osami poměrně jednoduché. Také by bylo možné vykreslovat ve scéně krychli tak, aby usnadnila uživateli přesné zarovnání kalibračního objektu (Uživatel by se snažil umístit sken přímo do dané krychle.).

Co se týče fotoaparátu, s tím bude práce trochu náročnější. Jelikož máme k dispozici pouze fotografii, nemáme moc možností, co by uživatel mohl dělat. Nejjednodušším, a v tomto případě asi nejlepším, řešením bude umožnit uživateli pohled skrze kameru fotoaparátu a zobrazit mu na obrazovce poloprůhlednou fotografii kalibračního objektu. Cílem by následně bylo zarovnat kameru s osami souřadného systému, skenem nebo s vykreslovanou krychlí pro lepší zarovnání skenů.

Když uživatel dokončí tuto práci, uloží nové pózy do souboru, ze kterého byly načteny.

3.9.5 Export scény

V této fázi bude od uživatele vyžadované pouze minimální množství práce. Bude potřeba jen vybrat formát, do kterého chce data exportovat a co vše chce exportovat.

Pro samotné exportování bude potřeba získat pozice a jména souborů, kde jsou uložena mračna bodů, 3D modely a fotografie, počet skenů a rotace mezi nimi a především také pózy skeneru a fotoaparátu, aby bylo možné správné pozicování bodů, modelů, etc. Tato data se snadno získají z "assembly scriptu", ze souboru s pózami a především ze souborů s daty samotnými. Výsledek této fáze bude následně uložen do výstupního souboru, který bude uložen v naší adresářové struktuře.

3.10 Uživatelské rozhraní

Jak je z předchozích částí 3.9.3 a 3.9.4 vidět, celý proces rekonstrukce není úplně jednoduchý, co se práce uživatele týče. Rekonstrukční část, která vyžaduje zadávání bodů, a zpřesňovací část, která bude vyžadovat manipulaci objekty ve 3D scéně, znamenají, že nebude stačit, aby výsledná aplikace byla pouze konzolová. Bude potřeba vytvořit grafické rozhraní (GUI), se kterým by uživatel mohl pracovat.

3.11 VIVID 9i SDK

VIVID SDK je skupina knihoven a hlavičkových souborů dodávaných se skenerem VIVID 9i a poskytuje především dvě základní funkce. První je ovládání skeneru samotného a druhou funkcí je zpracování dat pořízených skenerem. Nás bude zajímat především první funkce, tedy ovládání skeneru.

K SDK jsou dodávány také dva ukázkové projekty(Nadále budou označovány jako "sample"projekty.), kde každý z nich demonstruje několik základních funkcí SDK.

Dokumentace dodávaná s SDK není příliš rozsáhlá a poskytuje spíše základní informace. Proto značná část analýzy vychází především ze zkoumání sample projektů a z pokusů na nich provedených.

3.11.1 CVISensor

Toto je základní třída, jejíž instance reprezentuje skener samotný. Instance se vytváří při připojení ke skeneru a využívá se k volání funkcí s ním spojených.

Poskytuje především funkce pro připojení ke skeneru, připravení skeneru na skenování(funkce "Ready"), zaostření skeneru(funkce "Focus") a ke skenování(funkce "Measure"). Dále poskytuje funkce pro získání a nastavení parametrů skeneru, kalibraci skeneru a řadu funkcí pro získávání dat. Mezi data získávající funkce patří například získání mračna bodů, fotografie, 3D modelu nebo čistých nezpracovaných dat ze skeneru[17].

3.11.1.1 Rutina práce se skenerem

Při volání příkazů, které má skener vykonat se vždy provádí určitá rutina. Nejprve se získá aktuální nastavení skeneru, to se upraví dle potřeby a upravené se zapíše do skeneru. Následně se provede příkaz, který měl být vykonán. Po každé provedené akci se musí kontrolovat, zda funkce vrátila hodnotu $TRUE$[18] nebo $FALSE$ označující úspěch nebo neúspěch funkce. SDK skeneru nevyužívá výjimky, ale právě návratové hodnoty. Pokud se uživatel chce dozvědět, co se stalo při dané akci, musí získat číslo chybové hlášky přes funkci "GetErrCode"na instanci CVISensor.

3.11.1.2 Připojení

Připojení se ke skeneru má trochu odlišnou rutinu od volání běžných příkazů, i kdy i zde probíhá kontrola úspěchu volání funkce stejně. To probíhá zavoláním funkce "CISensorCreate"s parametrem, jaký typ skeneru chceme připojit (To je VIDIV 9i, druhou možností by bylo VIVID 910, který ale nepodporujeme.)

3.11.2 CVISensorPara

Instance této třídy reprezentuje nastavení skeneru. Slouží především k získání informací o skeneru(Instance se používá jako vstupní parametr pro funkci a funkce do ní ukládá skutečné parametry skeneru.) a k nastavení parametrů skeneru. Pro nastavení parametrů skeneru je nejdříve nutné parametry do instance načíst a následně upravit a zapsat, neboť po vytvoření instance obsahuje defaultní hodnoty, které by mohly způsobit špatné nastavení skeneru.

[17] Raw data. Slouží především k tomu, aby byly následně zpracovány uživatelem, nebo pomocí funkcí poskytovaných SDK zmíněných na začátku části 3.11.

[18] Zde se nevyužívají normální logické hodnoty *true* a *false*, ale makra definovaná ve Windows.h.

3.11.3 CVividImportPara

Instance této třídy slouží jako kolekce parametrů, které jsou předávány skeneru, když jsou z něj získávány data do paměti počítače. Využívá se při získávání mračna bodů, 3D modelu i fotografie. Parametry obsahují informace o tom, co vše má být ze skeneru získáno a jak mají být data předzpracována. Například zda mají být vyplněny díry v 3D modelu, zda má být potlačován šum, jakým způsobem má být potlačen a mnoho dalších parametrů.

3.11.4 CVIVertexData

Tato třída slouží jako reprezentace mračna bodů. Uchovává v sobě matici bodů o rozměrech 640×480, přičemž bod $B = [B_x, B_y, B_z]$ na pozici $[B_i, B_j]$ v mřížce leží na přímce procházející středem kamery a bodem $b = [B_i, B_j]$, který leží v obrazové rovině kamery skeneru. Podrobnější informace jsou v části 3.6[19]. Body jsou vyjádřeny v milimetrech a jsou v lokálních souřadnicích kamery skeneru. To znamená, že směr pohledu kamery je poloosa $-Z$ a směr vzhůru je poloosa Y souřadného systému kamery.

3.11.5 CVvdImage

Instance této třídy slouží k reprezentaci barevné informace získané jedním skenováním. Barevná informace může být získána buď v barevné, nebo v černobílé podobě.

3.11.6 CVIObject

Instance této třídy slouží k reprezentaci kolekce 3D modelů získaných skenerem. Objekty jsou pouze kontejnery uchovávající informace o počtu modelů a fotografií, a které samozřejmě také obsahují množiny modelů a fotografií samotných. Objekty této třídy je možné snadno uložit nebo načíst s využitím funkcí poskytovaných třídou CVISensorPolygonFileProcess ve formátu .vvd.

3.11.6.1 CVIModel

Tato třída reprezentuje 3D modely uložené v CVIObject kolekci. Instance CVIModel obsahuje vrcholy dané geometrie a v instanci třídy CVIModelData má uchovány topologické informace(V CVIModelData jsou uloženy plošky pomocí indexů na vrcholy, na kterých jsou vytvořeny.)

3.11.7 CVISensorPolygonFileProcess

Tato třída slouží k práci s 3D modely. Jediné dvě metody, které poskytuje (Kromě metody pro získání poslední chyby, která nastala a konstruktoru.) jsou metody pro uložení modelu do souboru a jeho opětovné načtení ze souboru.

[19] Zjednodušeně řečeno bod na pozici $[B_i, B_j]$ v mřížce odpovídá bodu zachycenému ve fotografii na pixelu $[B_i, B_j]$.

3.12 Canon EDSDK

Canon poskytuje ke svým fotoaparátům SDK (nazývané EDSDK), které umožňuje připojení, ovládání a manipulaci s jejich fotoaparáty. Na rozdíl od SDK skeneru není omezeno pouze na pár konkrétních zařízení.

Obdobně jako je tomu u SDK skeneru, i zde jsou poskytnuty dva sample projekty, které demonstrují základní ovládání fotoaparátu a zpracování fotografií a také poskytují poměrně rozsáhlou dokumentaci celého SDK.

EDSDK má oproti SDK skeneru trochu jiný model ovládání. V SDK skeneru všechny funkce vrátily vláknu, které je zavolalo, řízení až ve chvíli, kdy byla celá práce dokončena. Komunikace aplikace s fotoaparátem přes EDSDK probíhá asynchronně. EDSDK poskytuje kromě funkcí pro připojení k fotoaparátu a nastavení parametrů především sadu asynchronních příkazů.

3.12.1 Komunikace EDSDK s aplikací

Aby byla zajištěna korektní komunikace EDSDK a aplikace, musí aplikace implementovat řadu takzvaných "callback"[20] funkcí, které musí být EDSDK v rámci připojování k fotoaparátu poskytnuty. Základní čtyři callback funkce, které jsou potřeba, jsou "ObjectEventHandler", "PropertyEventHandler", "StatusEventHandler"a "ProgressHandler". ObjectEventHandler funkce je volána tehdy, když má fotoaparát v paměti data, která uživatel chce, což je například pořízená fotografie. Po dokončení získávání fotografie je zavolána tato callback funkce s objektem reprezentujícím pozici fotografie na paměťové kartě fotoaparátu.Další události mohou být například vytvoření nového souboru, smazání souboru, modifikace adresářové struktury a mnoho dalších, souborového systému a paměti se týkajících, událostí. PropertyEventHandler je volána v případě, že dojde ke změně parametru fotoaparátu. Tímto způsobem lze zajistit, že aplikace bude mít vždy aktuální informace o nastavení fotoaparátu. Funkce StateEventHandler je volána v případě, že se změní stav fotoaparátu. To mohou být například chybové hlášky EDSDK, informace, že fotoaparát byl odpojen, nebo další události obdobného rázu.

Co se ProgressHandler týče, tato funkce je volána z EDSDK ve chvíli, kdy probíhá nějaký proces, který zabírá delší čas. Je možné nastavit, zda má být tato funkce volána průběžně s parametrem, kolik procent je z funkce už vykonáno, nebo že má být zavolána až je proces úplně dokončen. Tím je možné zajistit, že se aplikace dozví o dokončení procesu.

3.12.2 Příkazy

Z celé sady příkazů jsou pro nás důležité především příkazy pro získání fotografie a pro její stažení do fotoaparátu.

[20] Funkce, které jsou volány EDSDK z jiného vlákna. Umožňují EDSDK komunikovat s aplikací

3.12.2.1 Připojení k fotoaparátu

Rutina pro připojení k fotoaparátu obsahuje poměrně veliké množství kroků. Nejprve, pokud to už nebylo provedeno dříve, je nezbytné inicializovat SDK voláním "EdsInitializeSDK". Dalším krokem je získání seznamu fotoaparátů a vybráním našeho fotoaparátu pomocí funkcí "EdsGetCameraList", "EdsGetChildCount"a "EdsGetChildAtIndex". Dále následuje získání parametrů fotoaparátu pomocí "EdsGetDeviceInfo", uvolnění listu kamer("EdsRelease"), nastavení callback funkcí zmíněných v části 3.12.1 pomocí "EdsSetObjectEventHandler", "EdsSetPropertyEventHandler"a "EdsSetStateEventHandler".

V tuto chvíli už zbývá jen navázat spojení přímo s fotoaparátem (V dokumentaci je spojení s fotoaparátem nazýváno "session".). To umožní volání funkcí na fotoaparátu a že EDSDK bude v aplikaci volat callback funkce spojené s daným fotoaparátem. To probíhá zavoláním funkce "EdsOpenSession"na objekt reprezentující fotoaparát. Následně je nutné nastavit, kam mají být fotografie ukládány (v našem případě je chceme ukládat pouze na disk počítače.) a s tím končí fáze připojování k fotoaparátu. V tuto chvíli už je možné s fotoaparátem pracovat (Pokud všechny výše zmíněné kroky uspěly.).

3.12.2.2 Pořízení fotografie

K získání fotografie fotoaparátem slouží příkaz "kEdsCameraCommand_PressShutterButton", který slouží k simulaci stisknutí tlačítka fotoaparátu pro pořízení fotografie. Tlačítko může mít celkem tři stavy, které jsou "nestisknuté"(Off), "stisknuté napůl"(Halfway) a "stisknuté úplně"(Completely). S těmito stavy se váže právě výše zmíněný příkaz "kEdsCameraCommand_PressShutterButton", se kterým se posílá parametr, na jakou hodnotu tlačítko nastavit. V případě pořízení fotografie tedy nejprve zavoláme tento příkaz s parametrem "kEdsCameraCommand_ShutterButton_Completely"a hned za tím s parametrem "kEdsCameraCommand_ShutterButton_OFF". Tím nasimulujeme jedno stisknutí a uvolnění tlačítka[21].

3.12.2.3 Stažení získané fotografie

Oproti pořízení fotografie je její stažení trochu komplikovanější proces. Stahovací proces musí začínat za "ObjectEventHandler"funkcí, neboť ta dostane jako parametr od EDSDK referenci na umístění fotografie na paměťové kartě fotoaparátu. Z reference na pozici se musí vytvořit objekt "EdsDirectoryItemInfo", který reprezentuje data na určité pozici na kartě. Následně se musí vytvořit datový proud "EdsStreamRef", který je spojen s lokací, kam má být fotografie uložena na disku počítače a je využit při samotném stahování. Jako další krok se musí nastavit pro daný proud právě ona výše zmíněná callback funkce "ProgressHandler", která má být volána v průběhu/po dokončení stahování. Dále následuje zavolání série funkcí "EdsDownload", která zahájí stahování souboru, "EdsDownloadComplete", která musí být zavolána po dokončení stahování a funkce pro uvolnění vstupní reference na pozici fotografie a uvolnění datového proudu.

[21] Kdyby nebyl zavolán příkaz s parametrem "Off", EDSDK i fotoaparát by si stále myslely, že tlačítko má být drženo ve stisknuté poloze, neboť všechny stavy jsou udržovány v poslední nastavené poloze.

3.13 Ovládání jednotky MARS 2

Řídící jednotka MARS 2 k sobě neposkytuje žádné SDK, ale komunikuje se s ní přímo pomocí konkrétního textového protokolu přes sériový port. Všechny možné příkazy je možné najít ve volně dostupné dokumentaci [25].

Pro správnou komunikaci počítače a jednotky MARS 2 je nutné správné nastavení portu. Parametry musí být:

přenosová rychlost: 9600 b/s
počet datových bitů v bajtu: 8
žádná parita
2 stop bity

Celý protokol pro komunikaci s MARS 2 se skládá ze sady textových příkazů, dotazů a potvrzovacích znaků. Zprávy se mohou skládat ze jména, operačního znaku a parametru. Jméno je definováno pro každý příkaz/dotaz, ale operační znak a parametry jsou závislé na tom, jaké jméno bylo zvoleno. Některé zprávy, týkající se konkrétního motoru (MARS 2 dokáže řídit až tři motory najednou.), musí na konci jména mít vloženo písmeno A, B nebo C, aby byla zpráva aplikována vůči správnému motoru. Takovéto zprávy budou mít ve jméně na konci písmeno "m", například "APm"je je dotaz, který říká MARS 2, aby vrátila aktuální polohu motoru "m". Operační znak od sebe odlišuje příkazy ":"a dotazy "?". Parametry mohou být celá nebo desetinná čísla s nebo bez znaménka v určitých rozmezích hodnot, což je specifikováno u každého dotazu/příkazu.

Možných zpráv ke komunikaci s MARS 2 je poskytováno poměrně veliké množství, ale pro naší potřebu bude stačit pouze několik z nich. Důležitým z nich budou: příkaz "GRm", který umožňuje nastavit pro motor "m"relativní pohyb o zadanou vzdálenost, aktuální poloha motoru "m"lze získat pomocí dotazu "APm", příkaz "CLEARm", který vypne motor "m"a nastaví jeho aktuální pozici na hodnotu "0", příkaz "REGMSm", který nastaví maximální povolenou rychlost motoru "m", "RELEASEm", který odpojí regulátory motoru "m"[22] nebo příkaz "READY", který oznámí MARS 2, že až dokončí pohyb všech motorů, má na sériový port zapsat zprávu "R!".

[22] Stolek se zastaví a je umožněno manuální nastavení pozice.

Kapitola 4

Návrh

V této části se nachází návrh celého skenovacího procesu, řešení funkcí, které bude aplikace uživateli poskytovat, způsobu a formátu ukládání dat, algoritmů použitých pro funkci aplikace a také grafického uživatelského rozhraní.

4.1 Ovladač skeneru

SDK skeneru poskytuje, jak bylo popsáno v části 3.11 pouze poměrně základní funkce. Proto bude potřeba naimplementovat ovladač, který umožní lepší kontrolu nad skenerem a bude mít trochu vyšší úroveň abstrakce, než je tomu u SDK.

4.1.1 Obecný popis ovladače

Ovladač bude implementován jako C++ třída, která bude zajišťovat připojení ke skeneru, provedení skenování, získání kalibračních dat a další věci, které vyžadují práci přímo s SDK skeneru.

Dále zde bude přidána řada statických metod pro ukládání a načítání dat získaných skenerem do/ze souborů. Sice některé z těchto funkcí nebudou vyžadovat práci s SDK, ale v rámci udržení určité koherence bude lepší mít vše, co se týká skeneru a jeho SDK pohromadě.

4.1.2 Připojení ke skeneru

První krok, který musí být proveden před samotnou prací se skenerem je připojení k němu. V průběhem připojení půjde především o inicializaci objektu, který bude skener reprezentovat, připojení ke skeneru samotnému a nastavení jeho parametrů. Tato část kódu bude prakticky shodná s připojením ke skeneru, které bylo použito v sample aplikaci přiložené ke SDK neboť zde není mnoho jiných cest.

4.1.3 Příprava skeneru na skenování

Tato funkce zajistí správné nastavení parametrů skeneru pro skenování objektu. Musí být provedena těsně před samotným skenování za podmínek které při skenování budou. Jde především o to, že skener při této operaci testuje, zda dokáže laserový paprsek odražený od objektu zachytit kamerou. Proto je provedeno několik testů [1] a podle nich určí, jak silný laserový paprsek musí použít, aby data získal. Výsledky závisí především na osvětlovacích podmínkách. Pokud je objekt hodně osvětlen, skener bude muset použít laser s mnohem větší silou, ale to také způsobuje že pruh osvícený laserovým paprskem na objektu bude silnější a data naměřená skenerem budou nepřesnější.

Při této fázi mohou nastat problémy, pokud je skenovaný objekt příliš malý. Jde především o to, že pokud se skeneru nepodaří ani na jedné ze vzorkovacích pozic zachytit kamerou odražený laserový paprsek, skener nedokáže provést nastavení parametrů pro skenování a nahlásí chybu. Naneštěstí není v dokumentaci, v hlavičkových souborech a ani v sample aplikaci stanoveno, co se se skenerem stane. Zda si zachová nastavení z posledního skenování a je možné provést skenování, nebo zda skenování po selhání přípravy způsobí, že data získaná následným skenováním budou poškozená. V sample aplikaci je toto řešeno tak, že pokud se nezdařila příprava, je uživateli zobrazena chybová hláška, že má zkontrolovat, jestli laser není blokován nějakou barierou a skenování je přerušeno.

Proto se v této aplikaci budu držet obdobného přístupu.[2] V momentě, kdy se nepodaří získat data dojde k přerušení skenování a nahlášení chyby(sken kalibračního objektu) nebo bude uloženo mračno bodů a 3D model s defaultními hodnotami (skenovaní objektu).

4.1.4 Provedení skenu

V této funkci, jak již název napovídá, půjde o to, aby byly správně nastaveny parametry skenování a byl proveden sken samotný. Po provedení skenu jsou všechna data (barevná informace a hloubková mapa) ze skenování uchována ve skeneru samotném a je možné metodami SDK popsanými v části 3.11.1 získat data v podobě mračna bodů, 3D modelu a bitmapy (barevná informace).

Tato funkce tedy musí být předcházena přípravou skeneru a ta musí uspět. Pokud k přípravě nedojde a je proveden sken, správnost výsledků a ani správnost skenování nelze zaručit.

4.1.5 Získání dat

Důležitou funkcí ovladače bude především získání dat ze skeneru. Zde jde především o získání mračna bodů, 3D modelu a barevné informace. K tomu SDK poskytuje několik metod, které byly popsány v části 3.11.1. Tyto metody berou jako parametr objekt s nastavením, které určuje, jaké vlastnosti budou mít získaná data [3].

[1] Skener provede několik testů laseru na předem daných místech.

[2] Především proto, že není definováno chování skeneru při skenování po selhání přípravy.

[3] Zde jde například o nastavení zda 3D model má mít díry vyplněné trojúhelníky, zda barevná informace má být barevná, nebo černobílá a další.

4.1.6 Ukládání a načítání dat

Dalšími metodami bude ukládání dat do souborů a jejich opětovné načítání. Půjde o ukládání a načítání mračen bodů a 3D modelů. Co se barevné informace týče, ta se bude ukládat společně s mračnem bodů, nebo samostatně ve formátu .png. Bližší informace o formátech uložení dat a důvody k způsobu jejich uložení jsou v částech 4.4.2, 4.4.3 a 4.4.4.

4.1.7 Výpočet ohniskové vzdálenosti z mračna bodů

Další funkce, která bude poskytována ovladačem skeneru je výpočet ohniskové vzdálenosti v pixelech z mračna bodů. Toto bude třeba v rekonstrukci kalibrační matice. Další informace o této funkci najdete v části 3.7.7.

4.2 Ovladač fotoaparátu

Ovladač fotoaparátu bude implementován podobně jako ovladač skeneru. Bude to třída v C++, která umožní připojení k fotoaparátu a poskytne metody pro práci s ním. Jak už ale bylo zmíněno v části 3.12, v EDSDK se se zařízením komunikuje asynchronně pomocí příkazů [4]. Proto bude potřeba naimplementovat jak funkce pro akce, které budeme chtít provádět, tak funkce, které umožní fotoaparátu říci, že byla určitá akce dokončena a pro posílání dat aplikaci.

4.2.1 Procedura focení

Co se fotoaparátu týče, nejdůležitější akce, které uživatel kromě připojení se k fotoaparátu potřebuje, je získání fotografie samotné a její stažení na počítač. V našem případě ale budeme potřebovat, aby aplikace počkala, než bude fotografie pořízena a stažena do počítače, než bude skenování pokračovat. Celá fotící procedura bude částečně podobná proceduře použité v sample aplikaci dodávané s EDSDK. Rozdíl je především v tom, že ji chceme naimplementovat synchronně tak, aby fotící procedura skončila až po dostahování fotografie.

Nejprve budeme muset vyřešit připojení k fotoaparátu. Během toho dochází také k nastavení funkcí, které má EDSDK zavolat, pokud fotoaparát bude něco potřebovat, bude chtít komunikovat, nebo bude posílat data. Tyto funkce jsou tři (HandlePropertyEvent, HandleObjectEvent a HandleStateEvent). Nás bude zajímat především HandleObjectEvent, neboť tato funkce je volána ve chvíli, kdy fotoaparát má připravenou fotografii ke stažení. Zároveň EDSDK předá objekt který obsahuje lokaci, odkud z fotoaparátu má být fotografie stažena.

[4] Aplikace například zavolá metodu SendCommand s příkazem, aby fotoaparát pořídil fotografii. Tato metoda skončí prakticky okamžitě a program pokračuje v běhu. Během toho fotoaparát fotografii pořídí nezávisle na aplikaci.

Procedura focení tedy začne tím, že se zavolají dva příkazy, "ShutterButton_Completely"a "ShutterButton_Off", které způsobí vyfocení objektu a uložení fotografie v paměti fotoaparátu. Pokud oba příkazy uspěly, hlavní vlákno, na kterém běží aplikace začne čekat. Ve chvíli, kdy fotoaparát dokončí fotografování, je z jiného vlákna zavolána funkce "HandleObjectEvent", která dostane objekt reprezentující lokaci, ze které má být fotografie z fotoaparátu stažena. V rámci této funkce dojde k zavolání příkazu pro stažení a nastavení callback funkce, která bude zavolána ve chvíli, kdy je stahování dokončeno (HandleDownloadProgress). Tato funkce nám umožní probuzení hlavního vlákna, aby pokračovalo v práci.

4.2.2 Windows Message Loop

V rámci pokusů provedených na prototypu ovladače jsem narazil na problém. Tento problém spočíval v tom, že v případě, kdy nebyl fotoaparát ovládán sample aplikací, ale jiným programem (Který neobsahovak GUI postavené na základě MFC(Microsot Foundation Class)[22].), EDSDK po vyfocení fotografie nezavolalo žádnou z Handle...Event funkcí. Všechny funkce byly zavolány až ve chvíli, když se aplikace vypnula, znova spustila a došlo k opětovnému nastavení callback funkcí v EDSDK. To ovšem bylo pozdě, neboť fotografie v paměti fotoaparátu už byly smazány.

Řešení se podařilo nalézt hledáním informací po různých internetových stránkách a fórech [4][5]. Problém spočíval v tom, že EDSDK pro svou správnou funkci vyžaduje implementaci takzvané "Windows Message Loop"[21]. Tento konstrukt je vlastně nekonečný cyklus, který odchytí zprávu poslanou systémem windows aplikaci, zpracuje ji a následně ji rozešle dál.

```
MSG msg;
while GetMessage(&msg, NULL, 0, 0)) != 0 do
    TranslateMessage(&msg);
    DispatchMessage(&msg);
end
```

"Message Loop"je využíván u starších C++ aplikací pro windows, pro implementaci reakcí na události vyvolané na GUI elementech. Nevýhodou tohoto konstruktu je, že musí běžet na vlákně, na kterém běží aplikace (Odkud se pracuje s fotoaparátem a EDSDK. Každé vlákno dostává pouze zprávy, které jsou pro něj relevantní), aby zpracoval správné zprávy. To je ovšem problém, neboť je to nekonečný cyklus a my bychom potřebovali, aby cyklus běžel na nějakém vlákně v pozadí.

Tento problém se také podařilo vyřešit. Po řadě pokusů jsem došel k závěru, že postačí, když je "Message Loop"aktivní až od chvíle, kdy jsou zavolány příkazy pro vyfocení fotografie, do chvíle, kdy je stahování dokončeno. V tu chvíli je možné cyklus ukončit a pokračovat v chodu aplikace.

Pomocí "Message Loop"se zároveň vyřešil problém s nutností čekat, než je stahování dokončeno. Funkce pro získání zprávy blokuje až do chvíle, než přijde další zpráva, takže nedochází k aktivnímu čekání.

4.2.3 Další funkce

Co se týče dalších funkcí ovladače, bude vhodné do něj přidat funkce pro získávání ohniskové vzdálenosti(kvůli udržení koherence) z EXIF metadat fotografie.

4.3 Ovladač otočného stolku

Ovladač stolku bude implementován obdobně jako předchozí dva ovladače třídou v C++. Stejně jako ostatní ovladače bude umožňovat připojení aplikace k danému zařízení, v tomto případě k řídící jednotce MARS 2. Dále bude poskytovat sadu metod pro ovládání stolku přes MARS 2. K tomu bude použit protokol komunikace popsaný v části 3.13.

4.3.1 Připojení a komunikace

K připojení k MARS 2 a ke komunikaci s ní je používán sériový port. Práce s ním je možná přímo v čistém C++. Další možností je například použití knihoven poskytovaných .NET frameworkem[24] pro práci se sériovým portem. Kromě výrazného zjednodušení práce se sériovým portem díky .NET wrapperům má .NET framework i další výhody, jako je například implementace GUI pomocí WinForms[23], použití více jazyků, jako je například C++ a C# a další.

4.3.2 Potřebné příkazy pro MARS 2

Abychom mohli pracovat s otočným stolkem, budeme potřebovat sadu příkazů. K tomu ale potřebujeme vědět, co chceme se stolkem provádět. Nám půjde především o to, abychom měli k dispozici funkci, která umožní otočení stolku o určitý úhel v určitém směru a funkce skončí teprve ve chvíli, kdy otáčení stolku skončilo a stolek se zastavil. Blokování chceme především z toho důvodu, aby další skenování nezačalo v průběhu otáčení stolku.

Tuto funkci budeme muset rozdělit do dílčích kroků neboť žádný podobný příkaz(3.13) není podporován, k dispozici máme pouze základní příkazy.

Nejprve bude potřeba poslat příkaz, který stolku řekne, že se má otočit o konkrétní úhel. K tomu využijeme příkaz pro relativní rotaci motoru. Následně bude potřeba počkat až do chvíle, než se motory zastaví. To je možné pomocí příkazu "ready", který jednotce MARS 2 řekne, že až se všechny motory zastaví, má na sériový port zapsat řetězec "R!". K tomu bude potřeba naimplementovat funkci pro čtení dat ze sériového portu, která bude čekat, dokud na sériový port nepřijdou data, ty se přečtou a funkce skončí. Po zastavení motorů v cílové pozici bude nutné motory uvolnit[5]. K tomu se využije příkaz "release". Dalším potřebným příkazem, který bude vhodné využít, je nastavení maximální povolené rychlosti motoru[6]. To je především z bezpečnostních důvodů, ochraně skenovaného objektu a k zamezení chyb.

[5] Důležitým důvodem je hluk, který způsobují pracující motory stolku. Pokud nejsou motory uvolněny, MARS 2 se bude neustále snažit upravovat polohu a bude oscilovat kolem cílové polohy s malými odchylkami.

[6] K tomuto jsem došel po prvních pokusech se stolkem, kdy při defaultní maximální rychlosti, kterou měl nastavenou, což byla konstanta 2000(V dokumentaci není označeno, co tato hodnota konkrétně znamená a zda má nějaké reálné jednotky.). Často se stávalo, že stolek začal oscilovat kolem cílové pozice, odchyloval se

4.4 Soubory, mezivýsledky a jejich formáty

4.4.1 Adresářová struktura

Jak již bylo zmíněno dříve, souborů získaných skenováním a potřebných pro rekonstrukci výsledné scény bude mnoho. Proto bude vhodné je strukturovat a vytvořit pro ně přehlednou adresářovou strukturu. Pro adresářovou strukturu jsem se rozhodl využít strukturu používanou pro výstupy z programu Visual SFM. Tím bude možné mezivýsledky i výstupy uložit do stejné struktury a vše bude organizované. Co se týče PLY výstupu, ten obsahuje pouze jeden soubor s daty případně textury, na které se výstupní soubor odkazuje relativní cestou.

Adreářová struktura má tvar:
path/to/structure/Project_name - Adresář kde jsou všechna data uložena
path/to/structure/Project_name/visualize - Adresář pro uložení fotografií
path/to/structure/Project_name/list.txt - Seznam fotografií [7]
path/to/structure/Project_name/bundle.rd.out - Scéna - výstup Visual SFM

K této struktuře ještě přidáme další adresář kam budeme ukládat mezivýsledky:
path/to/structure/Project_name/raw

4.4.2 Reprezentace skenu - body

Pro reprezentaci mračna bodů jsem se rozhodl využít co nejjednodušší formát a ukládat data jako obyčejný text, který bude přehledný, snadno editovatelný a půjde bezproblémově dále využívat. Pro každý sken bude vždy vytvořen nový souboru. Aby byly skeny odděleny a bylo možné s nimi pracovat samostatně. Data se budou skládat ze tří částí: z hlavičky, dat samotných a patičky.

Hlavička bude tvořena základními informacemi o skenu a bude ve tvaru:

%% scan data
width: <width>
height: <height>
verts: <max vertex count>

Parametr $< width >$ zde označuje šířku hloubkové mapy a parametr $< height >$ označuje její výšku. Pro náš skener to boudou konstanty 640 a 480. Parametr <max vertex

o desítky stupňů na obě strany a nebyl schopen zastavit v cíli bez tlumení působícího z vnějšku. Občas se také stávalo, že stolek začal rotaci a překročil cílovou pozici. Oproti předchozímu případu ale nezměnil směr rotace, ale začal rotovat čím dál vyšší rychlostí v jednom směru než se nakonec zastavil na pozici, která se od požadované pozice lišila o řadu stupňů (Odchylka byla většinou do 45°.). Toto rotování, prakticky vždy vedlo k vržení skenovaného objektu do vzdálenosti až 5 metrů od stolku.

[7] Fotografie jsou seřazeny tak, jak odpovídají kamerám. Každý záznam je uložen na jednom řádku ve tvaru visualize/photo.jpg

count> označuje počet vrcholů uložených v souboru. V běžné situaci bude mít hodnotu $< width > * < height >$, což je 307200. V případě, že se skeneru nepodařilo naskenovat žádná data, je zbytečné, aby se ukládalo 307200 řádků s defaultními hodnotami a tak bude parametr <max vertex count> nastaven na hodnotu -1. Tím dojde k jasnému označení, že mají být při načítání skenu použity defaultní hodnoty a načítání bude zkráceno.

Dále bude následovat úsek s daty. Ten bude ve tvaru:

$$\begin{array}{cccccc} < px1.x > & < px1.y > & < px1.z > & < px1.r > & < px1.g > & < px1.b > \\ < px2.x > & < px2.y > & < px1.z > & < px2.r > & < px2.g > & < px2.b > \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ < pxX.x > & < pxX.y > & < px1.z > & < pxX.r > & < pxX.g > & < pxX.b > \end{array}$$

Data jsou tedy uložena tak, že jeden bod je na jednom řádku. Ukládání probíhá postupně po řádcích, tj. máme-li na začátku hloubkovou mapu o velikosti 640×480, nejprve se uloží 480 hodnot prvního řádku, poté 480 hodnot druhého řádku a tak dále, dokud se neuloží celá mapa. Kromě souřadnic x, y a z pro každý bod se ukládají ještě barevné informace r, g a b a všechny hodnoty budou uloženy jako desetinná čísla a oddělena mezerou. Barevná informace bude v rozmezí $0 - 255$ a pozice bude uložena v milimetrech přesně tak, jak byla získána ze skeneru.

Za daty následuje patička s dodatečnými informacemi ve tvaru:

verts_reduced: <valid vertex count>
%%end

Paremetr <valid vertex count> zde označuje počet vrcholů, které jsou ve skutečnosti obsaženy v tomto skenu [8]. Tato informace se následně bude hodit při skládání výsledné scény (Nebude nutné tuto informaci počítat iterováním skrz vrcholy skenu a porovnáváním, neboť bude rovnou k dispozici.)

Aby bylo snadné rozpoznat soubor s mračnem bodů od ostatních souborů, bude pro všechny soubory s mračny bodů použita přípona .scn[9] místo .txt.

Pro uložení těchto souborů bude využita složka "raw"v naší adresářové struktuře (4.4.1), abychom oddělili tyto mezivýsledky od ostatních dat.

4.4.3 Reprezentace skenu - barva

Barva, jak bylo zmíněno v minulé část 4.4.2, bude ukládána spolu s mračnem bodů, aby všechna data, která patří k sobě byla uložena vždy pohromadě v jednom souboru. Ve fázi

[8] Žádný ze skenů nemá nikdy všech 640×480 vrcholů. Záleží především na tom, zda je zabírán celý skenovaný objekt nebo jen část, jak veliké jsou rozdíly v hloubce (Skener umí měřit pouze v hloubkách 0.5 metru až 1.5 metru.), jaké jsou světelné podmínky, jaký povrch má skenovaný objekt a mnoho dalších.

[9] Zkrácené slovo "scan".

rekonstrukce pózy (3.9.3) budeme potřebovat pouze fotografii kalibračního objektu. Proto bude vhodné mít barevnou informaci o kalibračním objektu uloženou přímo některém z grafických formátů. Kdybychom barevnou informaci neuložili zvlášť, museli bychom načíst celý sken kalibračního objektu, projít všechny vrcholy a z nich teprve vybrat barevnou informaci. Další výhoda použití nějakého běžného grafického formátu je, že uživatel si bude moci snadno prohlédnout i barevnou informaci ze skeneru.

Proto barevná informace mračna bodů bude pro kalibrační objekt uložena nejenom u bodů, jak bylo popsáno v předchozí části 4.4.2, ale také ve formátu .png[7] pro usnadnění přístupu k barevné informaci a urychlení jejího načítání.

4.4.4 Reprezentace skenu - model

Ukládání a načítání 3D modelu nebude způsobovat nejmenší problém. SDK skeneru přímo poskytuje metody pro ukládání 3D modelu do souboru a také metody pro načítání modelů z těchto souborů. Ukládá se přímo objekt CVIObject (3.11.6) se všemi daty, které obsahuje. Přípona těchto souborů je .vvd[10].

Všechny modely budou ukládány do složky "raw"v naší adresářové struktuře.

4.4.5 Reprezentace fotografie

Ukládání fotografií nebude problém, neboť EDSDK(SDK fotoaparátu) při stažení fotografie z paměti fotoaparátu na disk počítače fotografii rovnou ukládá ve formátu .JPG. S takto uloženými fotografiemi už nemusíme provádět žádné úpravy a můžeme je rovnou použít.

Fotografie získané fotoaparátem budou ve většině případů ukládány do složky "visualize"v naší adresářové struktuře. Jediná výjimka bude fotografie kalibračního objektu, která bude uložena ve složce "raw". Ta je totiž jen mezivýsledkem a není použita ve výsledné scéně.

4.4.6 Kalibrační objekt

Pro provedení rekonstrukce pózy a pro její zpřesnění bude potřeba uchovat informace o kalibračním objektu. Zde budeme především potřebovat uložit fotografii získanou fotoaparátem, barevnou informaci pořízenou skenerem (4.4.3) a mračno bodů získané skenerem.

Všechna tato data budou uložena ve složce "raw"a budou mít specifická jména pro jejich snadné odlišení od ostatních dat. Tyto soubory budou:

calib_camera.jpg - Barevná informace získaná fotoaparátem.
calib_scanner.scn - Mračno bodů získané skenerem.
calib_scanner.png - Barevná informace získaná skenerem.

[10] Toto je přípona používaná pro 3D modely uložené programy od firmy Konica Minolta.

4.4.7 Skenovaný objekt

O skenovaném objektu bude potřeba uložit řadu informací. Jedna skenovací procedura[11] bude obsahovat získání mračna bodů, 3D modelu a pořízení fotografie. Tato procedura bude provedena tolikrát, kolikrát uživatel chce. Proto definujme, jak budou jednotlivé mračna bodů, 3D modely a fotografie pojmenovávány.

scan#.scn - Mračno bodů
poly#.vvd - 3D model
photo#.jpg - Fotografie

Znak # v tomto případě označuje číslo skenovací procedury, ve které byly data pořízeny. Indexování začíná od hodnoty 0.

4.4.8 Reprezentace pózy

Co se týče pózy fotoaparátu a skeneru, ze bude potřeba reprezentovat translaci kamer ve světových souřadnicích (vůči otočnému stolku v metrech) a jejich rotaci. Pózy budou uloženy v souboru s názvem "poses", který bude uložen ve složce "raw"spolu s ostatními mezivýsledky. Jako příponu bude soubor mít .ass(zkráceno ze slova "assembly"). Tato přípona bude použita i pro "assembly script"(3.9.2 a 4.4.10). Struktura souboru bude:

$$
\begin{array}{ccc}
\%\%scannertranslation & & \\
< scan.T.x > & < scan.T.y > & < scan.T.z > \\
\%\%scannerrotation & & \\
< scan.R.00 > & < scan.R.01 > & < scan.R.02 > \\
< scan.R.10 > & < scan.R.11 > & < scan.R.12 > \\
< scan.R.20 > & < scan.R.21 > & < scan.R.22 > \\
\%\%cameratranslation & & \\
< cam.T.x > & < cam.T.y > & < cam.T.z > \\
\%\%camerarotation & & \\
< cam.R.00 > & < cam.R.01 > & < cam.R.02 > \\
< cam.R.10 > & < cam.R.11 > & < cam.R.12 > \\
< cam.R.20 > & < cam.R.21 > & < cam.R.22 >
\end{array}
$$

Co se týče parametrů, T zde označuje translační vektor kamery a R označuje rotační matici kamery. Co se indexování R týče, $R.IJ$ zde značí prvek matice R na řádku I ve sloupci J. Jednotlivé hodnoty na řádku v souboru budou desetinná čísla a budou odděleny mezerou.

4.4.9 Reprezentace konstantních parametrů fotoaparátu

Tento soubor bude obsahovat informace o fotoaparátu, který je využíván pro skenování. Jde především o tom, že získání některých parametrů z metadat fotografií (EXIF) může být problematické (Některé starší fotoaparáty nemusejí ukládat některé potřebné informace, především často chybí tag $FocalPlaneXResolution$ a $FocalPlaneYResolution$.). V takovýchto

[11] Jako skenovací proceduru zde označuji provedení jednoho skenování objektu během získávání dat. Následně dochází k pootočení otočného stolku a probíhá další skenovací procedura.

případech nemůžeme spoléhat na informace uložené ve fotografiích. Jelikož je aplikace vyvíjena pro konkrétní typ fotoaparátu, možností by bylo použít konstanty daného fotoaparátu, ale pro umožnění použití i jiných fotoaparátů bude uložení konstant v souboru bezpečnější.

Soubor bude uložen v adresáři aplikace se jménem "cam.calib". Konkrétní obsah souboru je:

exifW <width>
exifH <height>
exifX <resX>
exifY <rexY>

Parametry <width> a <height> zde označují výšku a šířku fotografií. Parametry <resX> a <rexY> zde označují rozlišení senzoru kamery v pixelech na jednu jednotku délky (jeden palec) v osách X a Y.

4.4.10 Assembly script

Assembly script je prakticky nejdůležitějším souborem pro celé skenování. Tento soubor bude obsahovat parametry zadané uživatelem pro skenování, kalibrační matice a cesty k souborům, jak bylo zmíněno v části 3.9.2.

Konkrétní složení "assembly scriptu"je:

%% calibration photos
<scanner calibration>
<camera calibration photo>

%% scanner calibration matrix
<k00> <k01> <k02>
<k10> <k11> <k12>
<k20> <k21> <k22>

%% camera calibration matrix
<k00> <k01> <k02>
<k10> <k11> <k12>
<k20> <k21> <k22>

%% scanning parameters
<delta angle>
<number of scans>

%% scan files
%% photos
sc <scan01>
po <poly01>
ph <photo01>
sc <scan02>
po <poly02>
ph <photo02>
.
.
sc <scanN>
po <polyN>
ph <photoN>

Na začátku jsou názvy souborů <scanner calibration> a <camera calibration photo>. Parametr <scanner calibration> je pouze název souboru bez přípony a odkazuje jak na mračno bodů získané z kalibračního objektu (defaultně calib_scanner.scn), tak na fotografii kalibračního objektu pořízenou skenerem (defaultně calib_scanner.png). Co se týče parametru <camera calibration photo>, ten obsahuje název fotografie kalibračního objektu pořízené fotoaparátem a má i příponu(defaultně calib_camera.jpg).

Dále následují kalibrační matice pro skener a pro fotoaparát. Obě matice mají rozměr $3x3$ a prvek matice kIJ odkazuje na prvek matice na řádku I a ve sloupci J. Matice jsou ukládány po řádcích jako desetinná čísla a jednotlivé hodnoty na řádku jsou odděleny mezerou.

Následující dva parametry jsou <delta angle>, což je úhel otočení stolku mezi dvěma skenovacími procedurami, a <number of scans> což je celkový počet skenů, který byl proveden.

Od tohoto bodu až do konce souboru následují trojice řádků. Každá trojice popisuje výstupy jedné skenovací procedury a tyto výstupy jsou <scan#>, <poly#> a <photo#>, které obsahují jména souborů s mračny bodů, 3D modely a fotografiemi pro sken s číslem #(indexujeme od 0). Defaultními hodnotami jsou jména definovaná v části 4.4.7. Těchto trojic se v souboru vyskytuje tolik, kolik bylo provedeno skenů (hodnota parametru <number of scans>).

4.5 Výstupy - Visual SFM

Jedním z podporovaných formátů, ve kterém bude možné uložit výslednou složenou scénu bude bundle formát[30] ve stejném tvaru, jako je možné získat z programu Visual SFM[33]. Výstup musí mít tři základní složky. Těmito složkami jsou soubor bundle.rd.out, který obsahuje body, kamery a další parametry scény, list.txt, který obsahuje seznam fotografií (ve formě relativní cesty k nim) použitých ve scéně v pořadí tak, jak odpovídají kamerám ve scéně, a nakonec fotografie samotné.

Sestavení souboru list.txt je poměrně triviální. Na každém řádku souboru je jedna cesta s relativní cestou k jedné fotografii.

Soubor bundle.rd.out má specifickou strukturu, která tu bude postupně rozebrána. Celý soubor je textový a tak je snadné jak jeho čtení, tak jeho úpravy.

```
# Bundle file v0.3
<num_cameras> <num_points>
<camera1>
<camera2>
...
<cameraN>
<point1>
<point2>
...
<pointM>
```

Parametry <num_cameras> a <num_points> jsou počet kamer a bodu ve scéně reprezentované celým číslem oddělené mezerou.

Každá položka <cameraX> reprezentuje jednu kameru ve scéně a je nahrazena pomocí ohniskové vzdálenosti f, dvou parametrů pro „radial distortion“ $k1$ a $k2$, rotační matice R (3×3) a jedním translačním 3D vektorem t. Matice R je v souboru zapsána obdobně jako tomu je u "assembly scriptu". Což znamená, že je zapsána na tři řádky a každý řádek obsahuje tři hodnoty oddělené mezerou. Stejně tak vektor t je zapsán jako tři hodnoty na řádce oddělené

mezerou.

<f> <k1> <k2>
<R>
<t>

Každý bod <pointX> je reprezentován pomoci vektoru *position*, který obsahuje 3D pozici v prostoru, 3D vektoru *color*, který obsahuje tři složky RGB, přičemž každá složka je v rozmezí 0 až 255.

<position>
<color>
<view list>

View list pro každou kameru má tvar takový, že nejprve je na řádce celé číslo N, udávající v kolika kamerách je daný bod zaměřen a to je následováno N čtveřicemi <camera> <key> <x> <y>, kde první parametr udává index kamery, kde je bod vidět, <key> je index bodu použitého při rekonstrukci a <x> <y> jsou pozice pixelu, na kterém je bod vidět ve fotografii. Jelikož tyto informace nemáme (toto je využito při structure-from-motion rekonstrukci, kterou Bundler i Visual SFM provádějí), celý <view list> bude nahrazen jedním číslem 0.

4.6 Výstupy - PLY

Ply formát je poměrně jednoduchý formát pro reprezentaci geometrií a topologií v prostoru, jak již bylo zmíněno v části 3.7.12. Formát nemá pevně definováno, jaké prvky musí obsahovat a de facto je na tvůrci, jaké prvky si zadefinuje. Problém s tímto je v tom, že pokud tyto prvky mají být rozpoznány nějakou aplikací, musí mít určitou specifickou formu. Proto jsem se rozhodl vycházet z formátu používaného programem MeshLab[3]. Výhodou je, že tento program umožňuje práci s mračny bodů, 3D modely a dalšími prvky a ke všemu je open-source, takže ho může používat prakticky kdokoliv.

Celá scéna je v tomto formátu reprezentována jedním jediným souborem s příponou .ply, který obsahuje skoro všechna data (všechna kromě textury, ty jsou v samostatných souborech). Struktura výstupního souboru je rozdělená do hlavičkové části a do dat samotných.

Hlavička souboru na prvním řádku obsahuje řetězec "ply"označující, že se jedná o .ply soubor. Další řádky označující variantu formátu začínají na klíčové slovo "format"a jsou následovány nějakým klíčovým slovem jako je "ascii", "binary_big_endian", etc. a na konci řádku je číslo verze. Jelikož my budeme výsledek ukládat v textovém formátu, použijeme "format ascii 1.0". Za definicemi formátu se nachází seznam textur použitých ve scéně jako "comment TextureFile XXX"kde na pozici "XXX"je cesta k fotografii/obrázku. MeshLab podporuje textury ve formátu .jpg i .png což pro naše potřeby stačí.

V další části hlavičky následují definice elementů s jejich počtem. Už ve chvíli zápisu definice elementu tedy musíme přesně vědět, kolik jich budeme chtít zapisovat. Po definici

elementů začíná část s daty samotnými. Část s daty (pro ascii formu) vypadá tak, že na každém řádku je jeden element a všechny jeho prvky jsou hodnotu po hodnotě zapsány na daný řádek. Oddělovačem prvků je v tomto případě mezera.

4.6.1 Mračno bodů

Nejdůležitějším elementem našeho výstupu bude vrchol, dále označovaný jako "vertex". Vertexy budou použity jak pro reprezentaci celého mračna bodů, tak pro vrcholy 3D modelu. Každý vertex je definován svou 3D pozicí, barevnou informací ve formě RGB a my si k němu ještě přidáme informaci o tom, ke kolikátému mračnu bodů/3D modelu patří.

Výsledná hlavička potom bude vypadat.

element vertex #
property float x
property float y
property float z
property uchar red
property uchar green
property uchar blue
property uchar scan

Parametr # zde označuje počet vrcholů, který bude zapsán v datové části.

4.6.2 3D model

3D model zde bude reprezentován jako trojúhelníková síť. Musíme tedy zadefinovat element "face", který reprezentuje jednu plošku modelu. Program MeshLab podporuje pouze trojúhelníkové plošky, i když formát by umožňoval tvary s libovolným počtem vrcholů. Každá ploška se tedy skládá ze seznamu indexů vrcholů, seznamu texturovacích souřadnic a následně indexu odkazujícího na texturu použitou pro danou plošku.

Definice vypadá takto:

element face #
property list uchar int vertex_indices
property list uchar float texcoords
property int texnumber

Obdobně jako u vrcholů i zde parametr # zde označuje počet elementů tohoto typu, které ve scéně budou. Listy v tomto formátu fungují tak, že jako první hodnota je počet prvků listu, v tomto případě to bude u indexů vertexů číslo 3 a pro texturovací souřadnice 6. Za tímto číslem následují samotné prvky. V případě seznamu "vertex_indices"jsou to celá čísla (int) a u seznamu "texcoords"je to desetinné číslo (float). Indexování textur je od nuly.

4.6.3 Kamery

Jelikož kamery nejsou programem MeshLab podporovány, vytvoříme si vlastní element, který bude sloužit k naší reprezentaci kamer. Jde nám především o uložení pózy kamery a ohniskové vzdálenosti v pixelech. Ostatní parametry už dopočítat dokážeme.

Definice kamery bude:

element camera #
property float ox
property float oy
property float oz
property float R00
property float R01
property float R02
property float R10
property float R11
property float R12
property float R20
property float R21
property float R22
property float foc

Stejně jako u předchozích, i zde # označuje počet těchto elementů ve scéně.

4.7 Získání dat a kalibrace kamer

Jak již bylo popsáno v část 3.9.2, v této části budou získána všechna potřebná data o skenovaném objektu a o kalibračním objektu.

Bude zde prováděno především vytváření adresářové struktury pro uložení dat, získání dat o kalibračním objektu, získání dat o skenovaném objektu a sestavování assembly scriptu (4.4.10).

4.7.1 Vytvoření adresářové struktury

V této fázi uživatel nejprve vytvoří adresářovou strukturu popsanou v části 4.4.1 na jím zvoleném místě na disku. Všechna další získaná nebo vytvořená data budou uložena v této struktuře.

4.7.2 Sestavování assembly scriptu

K sestavení assembly scriptu je potře mít určité informace. Potřebujeme zde znát kolikrát bude objekt skenován, jaký bude úhel mezi jednotlivými skeny, kalibrační matice 3.7.7 skeneru a fotoaparátu a jména všech souborů s daty, které byly získány v celé této fázi.

Co se týče úhlu a počtu skenů, tyto parametry budou zadány uživatelem a my si je tím pádem můžeme od uživatel vyžádat kdy chceme. Kalibrační matice fotoaparátu a skeneru můžeme sestavit až po provedení skenování a focení kalibračního objektu, neboť potřebujeme alespoň jednu fotografii pro získání aktuální ohniskové vzdálenosti fotoaparátu a alespoň jeden sken pro rekonstrukci ohniskové vzdálenosti skeneru. Co se týče jmen souborů, aplikace bude všechny soubory vytvářet s defaultními jmény, jak bylo definováno v části 4.4, takže na jejich vytvoření není třeba čekat.

Proto nejvýhodnější místo pro vytváření bude chvíle, kdy získáme sken a fotografii kalibračního objektu. Následně totiž zrekonstruujeme kalibrační matice, vezmeme parametry skenování zadané uživatelem a jména souborů použijeme defaultní.

Tímto způsobem zajistíme, že uživatel nezapomene na vytvoření assembly scriptu, všechna data v něm budou aktuální a nebudeme muset v programu uchovávat žádná data mezi jednotlivými akcemi (Museli bychom si pamatovat kalibrační matice nebo parametry pro jejich rekonstrukci až do chvíle, než bychom assembly script sestavovali.)

4.7.3 Skenování kalibračního objektu

Dalším krokem, který uživatel provede je skenování a focení kalibračního objektu. Nejprve dojde k tomu, že se aplikace pokusí připojit ke všem potřebným zařízením, tj. skeneru, fotoaparátu a řídící jednotce MARS 2. Pokud se připojení ke kterémukoliv zařízení nezdaří, celý proces skenování zde končí.

Jsou-li všechna zařízení připojena korektně, aplikace od uživatele získá dodatečné informace (úhel otočení stolku a počet skenů), které budou využity k sestavení assembly scriptu.

Aplikace následně naskenuje kalibrační objekt, stáhne do počítače a uloží mračno bodů s 3D modelem a získá fotografii kalibračního objektu.

Jak již bylo zmíněno v části 3.12.1, zde bude potřeba počkat, až se fotografie dostahuje, aby bylo ověřeno, že byla opravdu stažena a mohla být zkopírována na správné místo v adresářové struktuře. Proto bude třeba, aby metoda pro pořízení a stažení fotografie byla blokující do té doby, než je fotografie kompletně dostahována a uložena na disku v počítači.

Jako další krok bude získána ohnisková vzdálenost fotoaparátu z právě uložené fotografie a ohnisková vzdálenost z uloženého mračna bodů. S ohniskovými vzdálenostmi je možné sestavit kalibrační matice obou zařízení a ve chvíli, kdy máme kalibrační matice, můžeme sestavit a uložit assembly script.

V této chvíli jsou už všechna data o kalibračním objektu uložena v adresářové struktuře ve složce "raw"spolu s nově vzniklým "assembly scriptem", který obsahuje klíčové informace pro sestavení výsledné scény.

Nakonec dojde ke korektnímu odpojení všech zařízení.

4.7.4 Skenovací proces

Skenovací proces bude ze začátku shodný s procesem skenování kalibračního objektu. I v této části se aplikace připojí k jednotlivým zařízením a získá informace o skenovacím procesu od uživatele[12].

Dále už následuje samotný sběr dat. V cyklu se provádí operace "proveď sken", "ulož mračno bodů", "ulož 3D model", "pořiď fotografii"a "otoč stolek". Někdy se může se stát, že z daného úhlu se skeneru nepodaří provést přípravu na skenování a tak nebude možné objekt naskenovat. Kdyby se toto stalo v pokročilé části skenování, kdy je jich už značné množství pořízeno a celý proces se blíží ke konci, znamenalo by to, že všechna získaná data může uživatel vyhodit a musí začít znovu. Proto v případě, že se nepodaří pořídit sken, bude místo mračna bodů uložen soubor označující, že při načítání mají být pro všechny vrcholy použity defaultní hodnoty 4.4.2. Pro 3D model je to obdobné, zde se ale uloží kompletně prázdný soubor [13]

4.8 Rekonstrukce pózy

Rekonstrukce pózy, na rozdíl od získávání dat, bude umožňovat pouze jednu akci a to rekonstrukci pózy.

V rámci této akce uživatel nejprve zvolí, odkud z disku chce načíst data pro rekonstrukci pózy a kam i následně bude póza uložena. Tím dojde k načtení a zobrazení obou fotografií kalibračního objektu (jak fotografie pořízené skenerem, tak fotografie pořízené fotoaparátem). Ve chvíli kdy jsou fotografie načteny, uživatel musí zadat do každé fotografie dostatečné množství bodů a ke každému bodu přiřadit 3D souřadnice (souřadnice které má bod ve světovém souřadném systému (3.7.4) v metrech).

[12] Jistě zde každého napadne, proč využívat parametry zadané uživatelem, když můžeme informace vyčíst z "assembly scriptu". Jedním z důvodů je, že oba procesy budou sdílet společné GUI. V tuto chvíli by mohlo být pro uživatele matoucí, proč je potřeba dané informace pro kalibraci, ale nejsou vyžadovány pro samotné skenování. Další věcí je, že takto je skenování kalibračního objektu kompletně odděleno od skenovacího procesu. Ve chvíli, kdy uživatel má určité nastavení zařízení (orientace zařízení v prostoru), pravděpodobně bude využívat stejné nastavení i pro další skeny. V tu chvíli mu stačí pouze provedení jedné kalibrace, následně provedení skenů X objektů. Na konci pouze vezme jeden použitý assembly script, lehce ho modifikuje pro ostatní pro další scény, pokud využívají jiné parametry a i s fotografiemi/skeny kalib. objektu je rozkopíruje k odpovídajícím datům.

[13] Při testování funkcí jsem došel k objevu, že uložení defaultně(bez toho aby do něj skener uložil data) vytvořeného objektu CVIObject způsobí, že soubor, kam měl být uložen bude prázdný. Při načítání toto nevadí, neboť načítací funkce nahlásí, že selhala a v tu chvíli budeme vědět, že máme použít defaultní objekt. Díky prázdnému souboru uživatel bude vědět, že nedošlo k žádné chybě, neboť tam soubor bude mít, i když bude prázdný.

Ve chvíli, kdy uživatel zadá všechna data, která zadat chtěl, začne samotná rekonstrukce. Dojde k načtení kalibračních matic kamer z "assembly scriptu"a všechna data se předají modifikovanému[14] p3p (3.8) algoritmu, který z bodů pózu odhadne.

Jelikož algoritmus p3p funguje tak, že předpokládá, že kamera hledí ve směru osy Z, osa Y směřuje dolů a osa X vpravo, musí být výsledná rotace kamery ještě vynásobena rotací o 180° kolem osy X, abychom dostali standardní kameru, jako jsme ji definovali v části 3.6. Výsledná póza je pak uložena do souboru s pózami "poses.ass"4.4.8 do adresáře, odkud se načítaly fotografie a kalibrační matice. Důležité je pamatovat si fakt, že rotace kamery je vlastně transformace ze světového souřadného systému do systému kamery (samozřejmě teď nemluvíme o translaci). To například znamená, že pokud máme kameru s rotací R a chceme ve scéně vykreslit objekt reprezentující tuto kameru, musíme použít jako transformaci objektu inverzi matice R.

4.9 Zpřesnění pózy

V této fázi uživatel bude provádět úpravy už existující pózy. V prvním kroku tedy musí uživatel zadat, odkud má být póza načtena a také si vybere, co konkrétně chce zobrazit, zda chce mračno bodů, pro úpravu pózy skeneru, fotografii, pro úpravu pózy fotoaparátu, nebo obě dvě najednou.

Pokud uživatel zvolí množnost načtení mračna bodů, aplikace načte pózu skeneru ze souboru "poses.ass", bude načteno mračno bodů reprezentující kalibrační objekt a mračno se vykreslí. Pokud uživatel zvolí načtení fotografie, aplikace načte fotografii kalibračního objektu pořízenou fotoaparátem, načte pózu fotoaparátu, upraví transformaci kamery ve scéně, na pózu fotoaparátu a vykreslí fotografii na obrazovku. Detailnější informace o vykreslování jsou dostupné v části 4.9.1.

Uživatel bude následně moci modifikovat pózu fotoaparátu pomocí pohybu ve scéně. Pohyb ve scéně bude implementován tak, že pohyb bude realizován pomocí kláves na klávesnici a rotace kamerou pomocí pohybu myši. Modifikace pózy skeneru bude probíhat přidáváním translace v určitém směru(vždy po jedné ose) a rotacemi kolem os světového souřadného systému. Jelikož aplikace bude vyžadovat grafické uživatelské rozhraní, jak bylo zmíněno v části 3.10, tyto transformace budou prováděny pomocí tlačítek v GUI a hodnoty, o které bude provedena rotace nebo translace, bude uživatel moci upravit.

Ve chvíli, kdy je uživatel s výsledky spokojen, bude mu umožněno uložit modifikované pózy skeneru, fotoaparátu nebo obě dvě zpátky do souboru, ze kterého byly načteny.

[14] Algoritmu p3p, jak název napovídá, využívá k výpočtům pouze tři body. Modifikace spočívá v tom, že algoritmus přijme libovolný počet bodů, aplikuje na každou trojici bodů standardní p3p algoritmus a vybere ten výsledek, kde zpětná projekce bodů do fotografií má nejmenší odchylku.

4.9.1 Vykreslování

Jak již bylo v minulém odstavci zmíněno, bude třeba vykreslování mračna bodů, fotografií a dalších objektů. K tomu bude vhodné použít OpenGL[6]. Výhodou OpenGL je poměrně rychlé a efektivní vykreslování grafických primitiv a pro naše potřeby bude víc než dostatečné.

V našem případě půjde především o vykreslování obarvených bodů (mračno bodů), přímek (vykreslení os souřadného systému) a texturovaných trojúhelníků (plocha reprezentující obrazovou rovinu kamery s texturou odpovídající fotografii). Výhodou použití OpenGL bude také knihovna GLM[1], která poskytuje mnoho vektorových a maticových operací. Především vytváření transformačních matic, projekčních matic kamer a mnoho dalších věcí, co se maticových operací týče.

4.10 Export scény

V této fázi budou uživateli umožněny dvě akce, export ve formátu bundle a export ve formátu PLY. Obě akce budou sdílet úvodní načítací část, kde dojde k načtení všech potřebných dat ze souborů do paměti počítače. Lišit se budou pouze v zapisování těchto dat do souboru.

4.10.1 Načítací část

Nejprve je od uživatele získána cesta k adresářové struktuře, odkud mají být data získána a ve chvíli, kdy aplikace má tuto informaci, načítání může začít. Jako první je načten "assembly script"a z něj jsou získány všechny další potřebné informace. Jde především o informace o počtu skenů, otočení stolku mezi skeny, kalibrační matice zařízení a všechny jména souborů, které mají být použity. Následně jsou ze souboru "poses.ass"načteny pózy fotoaparátu a skeneru a jsou vytvořeny transformační matice ze získaných hodnot. Také je vytvořena rotační matice Rp reprezentující inverzi rotace[15] stolku mezi dvěma skeny.

V další části následuje cyklus, ve kterém je vždy načteno mračno bodů a 3D model, jsou vytvořeny objekty reprezentující kamery skeneru a fotoaparátu. Kamery jsou modifikovány pomocí rotační matice Rs tak, že rotační matice kamery R je zprava vynásobena rotační maticí Rs, kde matice Rs je kumulativní rotace. Matice Rs je pro první sken rovna I[16] a v každém cyklu je zprava vynásobena maticí částečné rotace Rp. Tím docílíme toho, že každý sken bude oproti tomu předchozímu otočen přesně o jedno pootočení stolku bez toho, abychom museli v každém i-tém cyklu počítat Rp^i, což by bylo zbytečně výpočetně náročné.

Následně jsou mračno bodů, 3D model, kamera fotoaparátu a kamera skeneru uloženy do seznamů, a tím jsou připraveny na další zpracování. Tento celý cyklus se opakuje tolikrát, kolikrát proběhla skenovací procedura.

[15] Inverze je použita z toho důvodů, že rotační matice kamery je transformace ze světového souřadného systému do lokálního systému kamery(až na translaci), což je inverzní operace než ta, kterou chceme, tj. otočit kameru ve světovém souřadném systému.

[16] Matice identity, neutrální prvek pro rotaci.

Když máme všechna data takto zpracovaná a připravená, může se přejít k samotnému exportování, které se liší pro každý z formátů.

4.10.2 Export bundle

Při exportování do formátu bundle, který byl popsán v části 4.5 je nejprve vytvořen soubor bundle.rd.out, do kterého bude celá scéna uložena.

Dalším krokem při exportování je sestavení hlavičky souboru. K tomu je potřeba znát počet bodů a počet kamer, které ve scéně budou. Počet kamer získáme snadno z počtu provedených skenů a počet vrcholů snadno získáme z informací o mračnu bodů, neboť jsme informaci o tom, kolik validních[17] bodů se v mračnu nachází, ukládali. Tím pádem není třeba procházet celé hloubkové mapy jen kvůli získání této informace.

Dalším krokem je zápis jednotlivých kamer do souboru. K tomu potřebujeme znát ohniskovou vzdálenost v pixelech, rotační matici a translaci kamery definované v části 3.6.2. Parametry týkající se "radial distortion"nastavíme na hodnotu 0, neboť nemáme žádnou informaci o těchto hodnotách. Tyto hodnoty postupně zapíšeme do výstupního souboru přesně tak, jak bylo definováno v části 4.5.

Když máme do souboru zapsány všechny kamery, je načase začít zapisovat jednotlivé body. Pro každý bod každého mračna provedeme to, že pokud je tento bod validní, aplikujeme na něj inverzi transformace kamery (Transformace získané z pózy.). Jelikož transformace kamery E je taková transformace, která bod X ve světových souřadnicích transformuje na bod W, což je bod v lokálním souřadném systému, podle $W = E * X$. Z toho nám vyplývá, že pokud upravíme rovnici, tak dostaneme $X = E^{-1} * W$. Což znamená, že k tomu, abychom z bodu W, který odpovídá našemu bodu z mračna, vytvořili bod X, potřebujeme inverzi k matici E. Jelikož matice E je složena podle 3.6.2 z rotační matice R a translace C, výsledná matice E bude invertovatelná, neboť translační matice i rotační matice mají plnou hodnost a jsou tedy invertovatelné. Je zde potřeba dát pozor na to, v jakých jednotkách pracujeme, neboť body mračna jsou v lokálním systému kamery, který je v milimetrech a světový systém je v metrech. To znamená, že body je nutné před transformacemi ještě přeškálovat(vynásobit) konstantou 10^{-3}. Takto přetransformované body je už možné zapsat přímo do souboru ve tvaru definovaném v části 4.5.

Ve chvíli kdy jsou všechny body zapsané do souboru, zbývá už jen jediná akce a to vytvořit soubor "list.txt"obsahující seznam fotografií v pořadí takovém, v jakém jejich kamery byly zapsány do výstupního souboru bundle.rd.out. Tím celý proces exportování končí a scéna je kompletní.

[17] Body, které skener dokázal změřit.

4.10.3 Export PLY

Export do formátu PLY stejně jako export do formátu bundle vyžaduje, aby proběhlo načítání dat. Kromě toho bude vyžadovat ještě dodatečné informace od uživatele a to konkrétně jaká data mají být do výstupního souboru uložena. Uživatel může zvolit, zda bude do scény ukládáno mračno bodů, 3D modely, kamery a nebo libovolná kombinace z nich.

Pro zjednodušení je zde popsán proces, kde se ukládají všechna možná data. Pokud se nějaká data ukládat nemají, příslušná část je vynechána i z definic, i z datové části.

Jak již bylo zmíněno v části 4.6, PLY formát vyžaduje už v definicích na začátku souboru znalost, kolik elementů daného typu bude v datové části zapsáno. Počet elementů typu "vertex"bude tolik, kolik scéna obsahuje bodů v mračnech plus počet vrcholů v 3D modelech. Počet elementů typu "face"bude součet všech trojúhelníkových plošek všech 3D modelů a počet kamer bude odpovídat počtu kamer, které mají být exportovány.

Při zapisování jednotlivých 3D modelů potřebujeme znát pro každou plošku indexy vrcholů, ze kterých se ploška skládá. Každý model má vrcholy indexované od nuly. Jelikož ale ve výsledné scéně budou všechny vrcholy a plošky pohromadě, bude nutné přepočítat indexy v ploškách tak, aby odpovídaly správným vrcholům.

K tomu využijeme prefixový součet. Pro každý model M_i vypočteme počet vrcholů c_i, které mají všechny mračna bodů a modely M_j takové, že $j < i$, dohromady. Toto lze snadno provést jednou iterací přes všechny mračna bodů a přes modely. Nejprve nastavíme c_0 rovno součtu vrcholů všech mračen bodů. Pro každé $i > 0$ potom platí že $c_i = c_{i-1} + K_{i-1}$, kde K_q je počet vrcholů, které má model s indexem q. Takto budeme mít snadno a efektivně spočtené hodnoty, o které musíme indexy vrcholů pro daný model posunout.

Následně zapíšeme všechny definice tak, jak byly popsány v částech 4.6.1, 4.6.2 a 4.6.3. Samozřejmě pouze zvolíme ty, které se budou ve výsledné scéně vyskytovat. Tím je celá hlavička scény dokončená a na řadě je zapisování dat.

Nejprve jsou zapsány všechny elementy typu "vertex". Vrcholy zapisujeme tak, že nejprve zapíšeme mračna bodů a až za ně jednotlivé 3D modely. Vrcholy nejprve transformujeme stejným způsobem, jako jsme to prováděli při exportování do bundle formátu.

Ve chvíli, kdy budeme zapisovat plošky modelů, budeme u vrcholů muset znát jejich texturovací souřadnice. Jelikož je neefektivní, abychom při zápisu každé plošky počítali texturovací souřadnice pro každý vrchol (Každý vrchol bude v modelu použit vícekrát a tak nemá cenu souřadnice počítat více než jednou.). Proto bude vhodné už při zpracování daného vrcholu vypočítat texturovací souřadnice a uložit si je tak, abychom je později měli při ruce. Texturovací souřadnice vypočteme tak, že sestavíme projekční matici kamery a body promítneme tak, jak je to popsáno v části 3.6.2. Jelikož ne každá fotografie musí zabírat celý objekt, bude nutné oříznout texturovací souřadnice do intervalu $< 0, 1 >$, aby nedocházelo k opakování textury.

Co se týče zápisu plošek, zde je potřeba vyřešit určitý problém. Formát PLY sice podporuje plošky s libovolným počtem vrcholů, ale program MeshLab, pro který má být výsledná scéna stavěna, podporuje pouze trojúhelníky. To je problematické z toho důvodu, že 3D modely pořízené naším skenerem obsahují převážně plošky se čtyřmi vrcholy [18]. Proto bude potřeba v případě čtyř-vrcholové plošky danou plošku rozdělit na dva trojúhelníky. Toto se zároveň musí odrážet i v části, kde počítáme celkový počet plošek.

Na konci zapíšeme jednotlivé kamery. Jejich zápis je triviální a data před zápisem není třeba nijak modifikovat.

4.11 Grafické uživatelské rozhraní

Jak již bylo zmíněno v části 3.10, bude třeba vytvořit grafické uživatelské rozhraní, aby byla uživateli usnadněna práce[19]. Aplikace tedy bude uživateli umožňovat řadu akcí, které budou rozděleny na čtyři základní fáze popsané v části 3.9.

Pro implementaci GUI jsem se rozhodl využít knihovnu WinForms[23]. Tato knihovna je součástí .NET frameworku, který podporuje jazyky C++, C# a další. Zároveň použití .NET frameworku zjednoduší práci s určitými věcmi, jako je například komunikace po sériovém portu zmíněná v části 4.3.1, práce se soubory a další. Zároveň WinForms podporují poměrně snadné zakomponování okna s OpenGL do formuláře a připojení dalších GUI elementů k němu. To využijeme v části 4.9.1, kde se budou při zpřesňování pózy do scény vykreslovat mračna bodů a další objekty pomocí OpenGL.

Co se týče GUI samotného, bude vhodné udělat pro každou fázi procesu jeden vlastní formulář. Tím se zajistí, že každá fáze bude oddělena od ostatních, pohromadě budou jen ty funkce, které k sobě skutečně patří a nedojde k zahlcení uživatele příliš velkým množstvím možností k dispozici. Také bude potřeba přidat formulář pro hlavní menu, které bude sloužit jako rozcestí k jednotlivým fázím.

V rámci zachování určité koherence provádění činností, bude vhodné přidat přístup do formuláře pro zpřesňování pózy až do formuláře pro rekonstrukci pózy, ne do hlavního menu. Prvním důvodem k tomu je fakt, že obě akce jsou si velmi podobné. Obě se týkají pózy skeneru a fotoaparátu v prostoru. Druhým důvodem je fakt, že zpřesňování pózy musí probíhat až po její rekonstrukci. Třetím důvodem je, že zpřesňování je pouze nepovinná fáze, který se takto liší od ostatních fází.

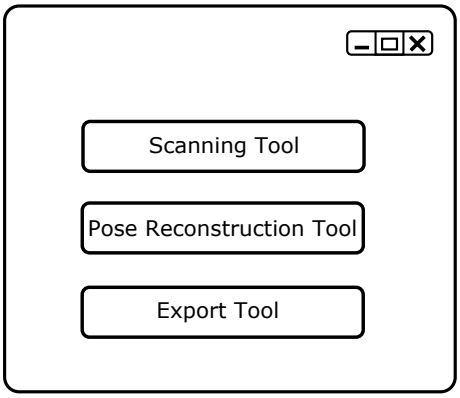
[18] Jelikož je hloubková mapa ve formě pravidelné čtvercové mřížky, každý "čtverec"je reprezentován jednou ploškou. Trojúhelníky vznikají pouze tehdy, pokud se v nějakém ze "čtverců"nepodařilo jeden z vrcholů změřit

[19] V některých částech by práce byla prakticky nemožná, bez použití GUI.

Získání dat, rekonstrukce pózy i export scény je nezbytné provést. V každém z těchto tří kroků jsou vytvářena nová data. Zpřesnění pózy je pouze operace navíc, která modifikuje existující data, a pokud se pózu podařilo zrekonstruovat dostatečně přesně (nebo uživateli daná přesnost stačí), nemusí se tento krok vůbec provádět.

4.11.1 Hlavní menu

Tento formulář bude obsahovat pouze pár prvků, neboť bude sloužit pouze jako rozcestí pro další činnosti. Jelikož formulář pro zpřesňování pózy bude přístupný z formuláře pro rekonstrukci pózy, z hlavního menu budou přístupné pouze tři další formuláře, kterými jsou formuláře pro "získání dat", "rekonstrukce pózy"a "exportování scény".



Obrázek 4.1: Návrh hlavního menu aplikace.

4.11.2 Získání dat

Formulář pro fázi „získání dat“ nemusí být nijak zvlášť složitý. Bude uživateli umožňovat provedení několika akcí, které už byly popsány v části 4.7. První z těchto akcí bude vytvářet adresářovou strukturu, která bude mít konstantní vzhled podle definice v části 4.4.1. K tomu je potřeba pouze jediná informace a to konkrétně umístění, kde má být struktura vytvořena. Pro snížení chybovosti v tomto kroku bude zakázáno vytvářet všechny adresáře kromě uložiště scény a adresáře uvnitř naší struktury. Tím je myšleno, že pokud uživatel zadá cestu .../dir1/dir2/dir3, dir3 bude vytvořen jako hlavní adresář naší adresářové struktury, ale bude vytvořen pouze tehdy, pokud adresáře dir2, dir1 a všechny další adresáře, které je na cestě předcházejí, existují. Tím se zaručí aspoň určitá ochrana proti překlepům. Pokud uživatel zadá velmi dlouhou cestu a hned na začátku udělal překlep, nevytvoří se zbytečně velké množství adresářů na špatném místě .

Ve druhé akci bude proveden jeden sken a bude získána jedna fotografie kalibračního objektu. Sken i fotografie budou uloženy na specifickém místě v adresářové struktuře vytvořené v předchozím kroku, takže první akce musí tuto akci předcházet. Dále bude vytvořen assembly script se všemi informacemi o daném skenování. To znamená, že kromě cesty, kam

ukládat získaná data a assembly script(lze použít stejnou cestu jako pro akci číslo jedna), budeme potřebovat více informací.

Zde bude potřeba ještě získat informaci o tom, kolik skenů má být provedeno, a o jaký úhel se má stolek mezi jednotlivými skeny pootočit. Na počet skenů je jediná podmínka, že bude celé číslo větší než nula a co se úhlu týče, tak ten může nabývat libovolných reálných hodnot různých od nuly.

Další parametry, které budou třeba při vytváření assembly scriptu(parametry fotoaparátu, parametry skeneru, ohniskové vzdálenosti), už poté není těžké získat.

Poslední tj. třetí akce provede samotné skenování a focení kalibračního objektu. K tomu bude vyžadována cesta k adresářové struktuře, kam mají být skeny a fotografie uloženy. Dále bude třeba informace kolik skenů provést a o jaký úhel má být stolek mezi jednotlivými skeny pootočen. Tyto parametry se shodují s parametry z předchozího kroku a bude tedy vhodné je získat stejným způsobem, aby se zredukoval počet UI elementům, kterým bude uživatel vystaven[20]. Kromě toho ale bude potřeba ještě znalost toho, na kterém sériovém portu je MARS 2 se stolkem připojen, aby došlo k připojení ke správnému portu se správným zařízením.

Z toho vyplývá, že formulář bude muset obsahovat alespoň tři tlačítka, každé pro jednu akci, čtyři textová pole pro vstupní parametry (cesta, počet skenů, úhel, sériový port) a nějaký způsob, jak uživateli dát najevo, že akce vyšla, nebo že selhala a proč tomu tak bylo.

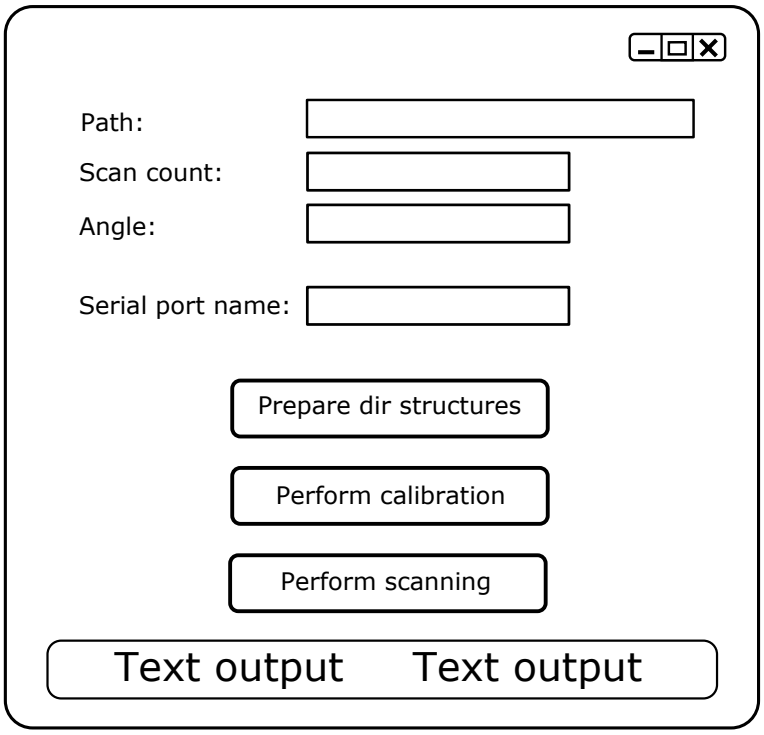
4.11.3 Rekonstrukce pózy

Formulář pro rekonstrukci pózy skeneru a fotoaparátu bude o něco komplikovanější, než formulář pro získání skenu. Tento formulář bude uživateli umožňovat provést rekonstrukci pózy skeneru/fotoaparátu a také bude obsahovat tlačítko pro přechod do formuláře pro "zpřesňování pózy".

Akce, která se v tomto formuláři bude provádět, je tedy rekonstrukce pózy. Konkrétní popis celého postupu byl už popsán v části 4.8. První věc, která je v rekonstrukci pózy potřeba, je cesta k datům uloženým v rámci skenování objektu. Zde je potřeba především assembly script kvůli kalibračním maticím skeneru a fotoaparátu a kvůli cestám k fotografii/skenu kalibračního objektu. Tento problém bude vyřešen obdobně jako u formuláře pro získání dat.

Když už jsou data načtená, bude potřeba pro algoritmus rekonstrukce pózy (p3p 3.8) získat klíčové body ve fotografiích (fotografie získané fotoaparátem a skenerem) a také pozici těchto klíčových bodů ve světových souřadnicích. K tomu bude nezbytné v tomto formuláři zobrazit obě dvě fotografie, jak fotografii získanou skenerem, tak tu získanou fotoaparátem. Dále

[20] Aby uživatel nebyl zahlcen příliš velikým množstvím tlačítek, textových polí, etc.



Obrázek 4.2: Návrh formuláře pro získání dat.

bude třeba uživateli umožnit zadávat body do fotografie kliknutím, což znamená registrovat události vyvolané myší na fotografiích.

Dále bude potřeba přidat nějaký nástroj pro zadávání trojrozměrných souřadnic ke každému zadanému bodu. Tento problém lze snadno vyřešit přidáním možnosti iterovat přes už zadané body a možnosti upravit aktuálně vybraný bod. Aby si uživatel nemusel pamatovat mnoho informací, bude třeba uživateli ukázat, který bod má právě vybraný ve fotografii. Zadání trojrozměrných souřadnic bodu potom bude umožněno skrze tři textová pole a potvrzovací tlačítko, kterým uživatel potvrdí upravení souřadnic.

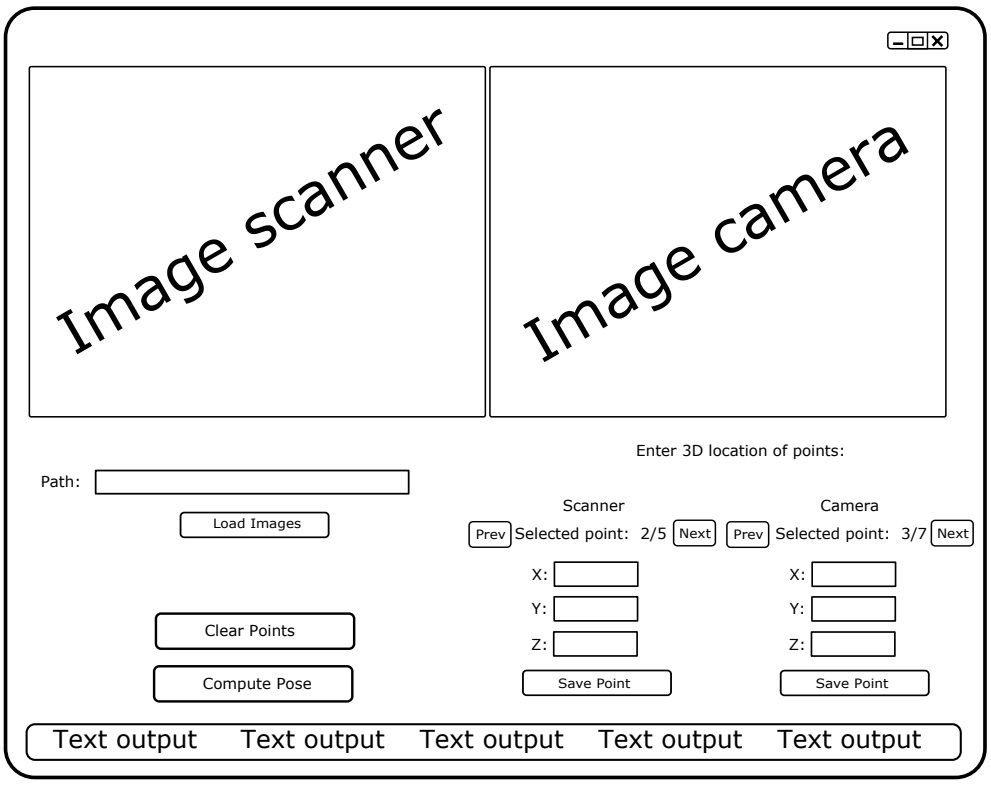
Následně je potřeba vyřešit problém s tím, že máme dvě sady bodů, které chceme editovat. Zde existují dvě možnosti, jak by to šlo vyřešit.

První možností je zdvojit elementy pro zadávání trojrozměrných souřadnic a tím by každá sada bodů měla svůj zadávací panel. Tím by ale došlo ke zdvojení těchto UI elementů. Druhou možností by bylo nechat pouze jeden zadávací panel, ale přidat možnost přepínání mezi editací bodů pro skener nebo pro fotoaparát.

Tato varianta ovšem zase může vést k vyšší chybovosti uživatele. Především, pokud by si před editací bodů nezkontroloval, zda edituje správnou sadu bodů, mohl by si zničit už dříve

zadané souřadnice. Zároveň uživatel zadával body do dvou různých fotografií, ale najednou má pozice zadávat pouze v jednom panelu, což by mohlo vést ke zmatenosti uživatele a inkonzistenci UI. Dalším argumentem pro zdvojení UI elementů je fakt, že když uživatel bude moci iterovat skrz obě sady bodů nezávisle na sobě, může si lehce provést kontrolu, že stejným klíčovým bodům (bodům, které náleží stejnému bodu ve skutečném světě) zadal stejné trojrozměrné souřadnice, což by se při přepínání mezi jednotlivými sadami bodů provádělo špatně. Proto bude lepší zdvojit UI elementy, i když to bude zabírat více místa a budou zde duplicity UI elementů.

Když uživatel bude mít všechny body zadané a všechny body budou mít nastavené souřadnice, už nic nebrání tomu, aby byly spočítány pózy skeneru a fotoaparátu. Dále ještě bude vhodné umožnit uživateli v případě nějaké chyby alespoň začít znovu, takže přidáme tlačítko pro smazání zadaných bodů. Výsledné pózy zrekonstruované v tomto bodě budou uloženy v souboru pose.ass v adresářové struktuře na pozici kam vede cesta zadaná uživatelem, jak již bylo zmíněno v části 4.4.1.



Obrázek 4.3: Návrh formuláře pro rekonstrukci pózy.

4.11.4 Zpřesnění pózy

Formulář pro zpřesnění pózy jsme už popsali v části 4.9. Uživatel musí být schopen zadat cestu, odkud mají být existující pózy načteny. Stejná cesta bude využita pro načtení fotografií

a mračna bodů reprezentujícího kalibrační objekt. Ty budou nezbytné pro modifikace póz. Zadání cesty bude řešeno stejným způsobem, jak u předchozích oken, tj. u získávání dat a rekonstrukce pózy.

Dále bude potřeba panel, v němž se bude vykreslovat celá scéna a k němu bude připojeno ovládání pohybu a rotace kamerou. Ovládání bude standardní pohyb po scéně pomocí šipek a rotace pomocí myši. Pravděpodobně bude vhodné ještě připojit pohyb kamery nahoru a dolů a rotace kolem optické osy kamery (3.6)[21].

Jelikož uživatel bude upravovat kameru právě pomocí tohoto ovládání, je nezbytné umožnit uživateli přímo modifikovat rychlost pohybu kamery po scéně a rychlost otáčení kamery. To umožní uživateli jemné pohyby kamerou i velké změny pozice/rotace, pokud bude chtít. To lze vyřešit dvěma textovými poli a alespoň jedním potvrzovacím tlačítkem.

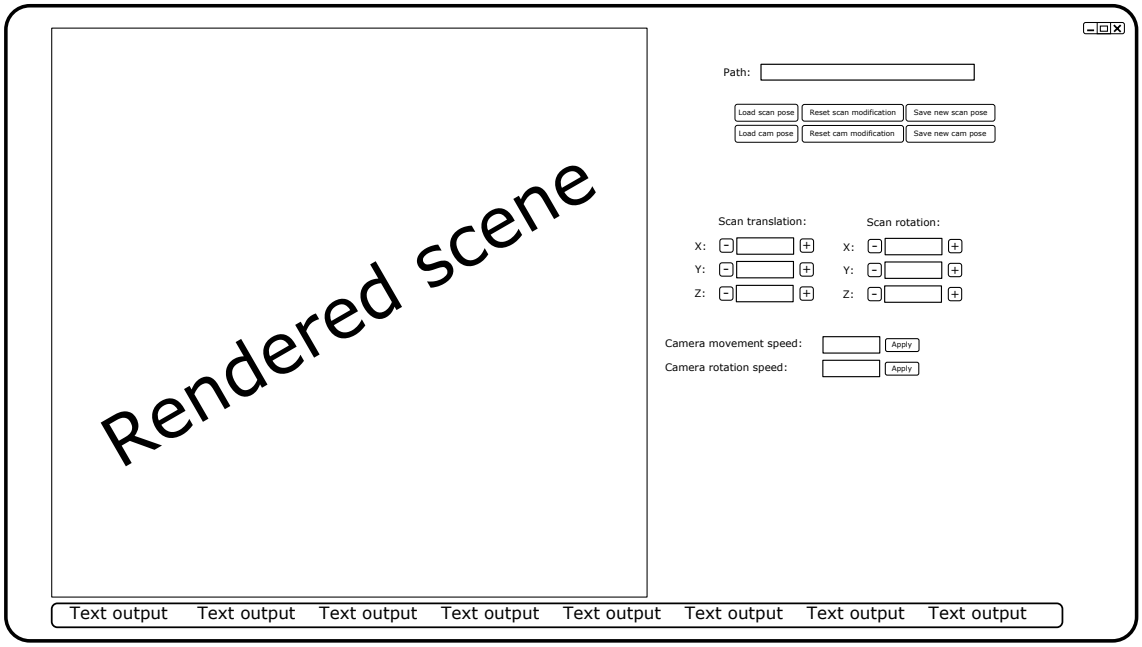
Uživateli také bude muset být umožněno načíst sken, fotografii, nebo obě dvě najednou. Dále bude třeba umožnit uživateli provádět transformace skenu samotného. Zde bude třeba uživateli umožnit pohyb ve třech osách, rotaci kolem tří os a nastavení velikosti posunu/rotace.

Když uživatel zpřesňování dokončí, musí být schopen exportovat novou pózu skeneru nebo fotoaparátu. Jelikož uživatel může chtít upravit pózu pouze jednoho zařízení, bude vhodné umožnit uložení jen jedné pózy bez modifikace té druhé. Zároveň uživatel může chybovat a chtít začít modifikovat pózu znovu od začátku. Začít znovu načítat fotografii nebo sken by bylo zbytečné, takže bude vhodné umožnit zresetovat neuložené modifikace póz.

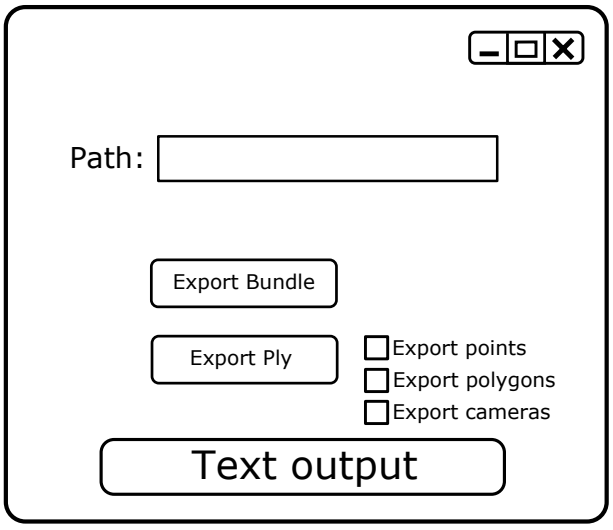
4.11.5 Export scény

Formulář pro exportování scény bude poměrně jednoduchý. Stejně jako u předchozích formulářů, i zde bude třeba, aby uživatel zadal cestu k adresářové struktuře s daty. Když je cesta zadána, uživateli bude stačit zvolit formát, do kterého má být proveden export. Při exportu do bundle formátu nejsou vyžadovány žádné další informace. Co se týče PLY formátu, zde si uživatel může zvolit, co vše chce exportovat. PLY formát nemá pevně stanoveno, co musí obsahovat. Uživatel zde může exportovat mračna bodů, otexturovaný model, kamery (pózy a parametry fotoaparátu odkud byly fotografie zachyceny), nebo jejich libovolnou kombinaci.

[21] roll - Rotace kolem vektoru směřujícího vpřed v lokálním souřadném systému kamery.



Obrázek 4.4: Návrh formuláře pro zpřesnění pózy.



Obrázek 4.5: Návrh formuláře pro exportování dat.

Kapitola 5

Implementace

5.1 Použité knihovny, API, frameworky

Pro zjednodušení a zrychlení implementace jsem využil řadu externích knihoven a API pro řešení určitých problémů. Zde je soupis jednotlivých knihoven a jejich využití.

5.1.1 .NET[24]

Framework .NET je využíván pro značnou část aplikace. Je používán především pro zjednodušení práce se sériovými porty v rámci práce a komunikace s řídící jednotkou MARS 2. Nejvíce jsou využívány knihovny WinForms, jež jsou součástí .NET frameworku. Ty jsou využity pro grafické uživatelské rozhraní v aplikaci.

5.1.2 OpenGL[6]

OpenGL a knihovny s ním spojené jsou využity ve fázi "zpřesnění pózy"pro vytvoření 3D scény, manipulaci s ní a především pro její vykreslení.

5.1.3 GLMathematics[1]

Knihovna GLM je využívána nejen v rámci částí spojených s OpenGL, ale všude kde se pracuje s body, maticemi a transformacemi ve 3D prostoru. Jediný případ, kdy se GLM nevyužívá, jsou maticové a vektorové operace, kde je použita knihovna Eigen.

5.1.4 Eigen[18]

Knihovna Eigen je využívána pro pokročilejší operace lineární algebry. Její využití je v algoritmu p3p pro reprezentaci matic (Které mají rozměry větší než 4×4 a nejde s nimi pracovat pomocí GLM.) a pro provádění operací, jako je výpočet vlastních čísel(Což je funkce, kterou GLM nepodporuje.).

5.1.5 ImageMagick[2]

Tato knihovna je v aplikaci využívána pro načítání souborů ve formátu .jpg. Toho se využívá ve fázi "zpřesňování pózy", kde je potřeba načíst fotografii a použít ji jako texturu pro 3D model.

5.1.6 libpng[10]

Tato knihovna je v aplikaci využita pro ukládání dat ve formátu .png. To je například použito při ukládání barevné informace získané skenerem o kalibračním objektu.

5.2 Ovladače zařízení

V této části jsou popsány ovladače vytvořené pro jednotlivá zařízení. Veškerá práce s externími zařízení, kterou bude aplikace vykonávat bude probíhat výhradně přes tyto tři třídy.

5.2.1 VividControl

Tato třída sama o sobě slouží jako wrapper[1] pro funkce SDK skeneru a pro odstínění zbytku aplikace od tohoto SDK. Cílem bylo zjednodušit práci se skenerem, aby každá akce, která má být provedena (připrav skener na skenování, naskenuj, získej data, etc.) byla prováděna pouze jednou funkcí.

Základní funkcí v této třídě je připojení ke skeneru. Dále jsou poskytovány funkce pro připravení skeneru na skenování objektu, skenování samotné a získání mračna bodů, barevné i černobílé informace a 3D modelu. Díky těmto funkcím je možné získat ze zařízení všechna data, která potřebujeme.

Další částí jsou funkce pro ukládání a načítání dat z/do souborů. Formáty těchto dat jsou popsány v části 4.4 a jde o ukládání mračen bodů s barevnou informací, 3D modelů a samotné barevné informace ve formátu PNG. Co se načítání dat ze souborů týče, každá ukládací funkce[2] má k sobě obdobnou funkci pro načítání dat. Dále jsou zde poskytnuty sloučené ukládací funkce které dohromady spojují načtení dat ze skeneru a jejich přímé uložení do souboru, což opět zjednoduší použití tohoto ovladače.

Nakonec jsou v této třídě funkce týkající se stavby kalibrační matice skeneru a výpočtu ohniskové vzdálenosti, jež byly popsány v části 3.7.7.

[1] Třída obalující jiné třídy a objekty za účelem změny rozhraní, které je navenek poskytováno. Zde je důvodem především zjednodušení přístupu a provádění operací.

[2] Kromě ukládání do PNG. Funkce pro načtení PNG souboru je poskytována .NET frameworkem.

5.2.2 CamControl

Třída CamControl je postavena na obdobných principech, jako VidivControl. Také funguje jako wrapper, aby odstínila aplikaci od EDSDK a poskytla jednoduché funkce pro použití základních funkcí fotoaparátu.

Základní je opět funkce pro připojení k fotoaparátu, která inicializuje EDSDK, nalezne správný fotoaparát a připojí se k němu. Další funkcí je zde "TakePictureSynch()", jež slouží k pořízení fotografie a k jejímu korektnímu stažení z paměti fotoaparátu na disk počítače. Dále jsou zde různé "handle"funkce (3.12.1), které volá samo EDSDK a také funkce pro extrakci informací, jako je ohnisková vzdálenost z EXIF metadat fotografie.

5.2.3 MARSControl

Tato třída je vytvořena odlišně než předchozí dva ovladače (Především proto, že MARS2 nemá žádné SDK.). Tato třída implementuje funkci pro připojení k sériovému portu s nastavením parametrů komunikace na daném portu a také sadu funkcí s jednoduchými i složenými příkazy, které byly popsány v částech 3.13 a 4.3.2.

5.3 Kolekce funkcí

V této části jsou popsány třídy sloužící jako kolekce statických metod. Byly vytvořeny především z toho důvodu, aby zabalili funkce s podobným účelem, nebo funkce, které jsou použity pro nějaký specifický účel.

5.3.1 CameraTool

Tato kolekce slouží k obalení p3p algoritmu (3.8), přičemž navenek jsou důležité pouze funkce "GetCameraIntrinsic", která vypočte ze zadaných parametrů kalibrační matici fotoaparátu, "LoadCamCalibrationFromFile", kterou se načítají kalibrační parametry fotoaparátu a "ComputeCameraExtrinsic", což je začátek p3p algoritmu. Tato funkce spustí nad zadanými parametry výpočet póz, které jsou jejím výsledkem.

Všechny ostatní funkce (Především ty, jež začínají na "p3p".) slouží jako části p3p algoritmu.

5.3.2 ExportTool

Tato kolekce slouží především k exportování už připravených dat do finální scény. K tomu jsou zde funkce "BuildBundle"a "BuildPly", které dostanou mračna bodů, definice kamer, jména textur a další data, ze kterých následně vytvoří finální scénu a uloží ji do souboru. Dále jsou zde pomocné funkce využité pro sestavování scén, jako funkce pro zapsání dat objektů do streamu ve formátu bundle nebo PLY, pro generování nejlepších korespondencí mezi pózami skeneru a fotoaparátu (3.7.11), nebo k výpočtům texturovacích souřadnic.

5.3.3 FileTool

Tato kolekce obsahuje řadu funkcí pro manipulaci se soubory, vytváření nových souborů a pro analýzu objektů, na které ukazuje zadaná cesta[3]. Dále jsou zde funkce pro vytváření jmen souborů (4.4), přípravu cest v adresářové struktuře (4.4.1) a především konstanty se jmény souborů, adresářů, přípon a všech dalších v aplikaci použitých řetězců, které se týkají souborů a adresářů.

5.3.4 ScannerTool

Tato třída je kolekcí statických metod, které se využívají především u skenovacího procesu a pro rekonstrukci pózy(Ale nehodily se svou povahou do kolekce CameraTool.). Jedná se o funkce pro načítání a ukládání póz z/do souborů, načítání kalibračních matic a ohniskových vzdáleností z "assembly scriptů"a především je zde funkce "ComputePoses", která zjednodušuje rekonstrukci a uložení póz na jediné volání funkce, kterému se předá cesta k uložišti a čtyři seznamy bodů, ze kterých má spočítat pózy jak pro fotoaparát, tak pro skener.

5.4 RigidModels

Tato třída slouží k inicializaci, uložení a k smazání geometrií, které budou vykreslovány v OpenGL scéně v rámci zpřesňování póz. Jde především o modely, které reprezentují přímku, texturovaný čtverec a mračno bodů. Mračno bodů je zde reprezentováno jako normální model pomocí sítí trojúhelníků, akorát je reprezentovaný úsporněji, neboť se sken vykresluje jen jako množina bodů[4].

5.5 CameraData

Tato třída slouží k reprezentaci dat jedné kamery. Především jde o pózu kamery a ohniskovou vzdálenost. Co se týče kalibrační matice, tu je nutné si udržovat externě. Tato třída umožňuje operace pro transformaci bodů ve světovém souřadném systému do lokálního souřadného systému[5] a také obráceným způsobem. Také poskytuje funkci pro vygenerování projekční matice kamery, pokud je jí zadána kalibrační matice kamery (3.6.2).

5.6 VertexCollection

Tato třída slouží k úspornějšímu uložení mračna bodů spolu s jeho barevnou informací[6] Tato třída slouží jako ekvivalent k reprezentaci mračna bodů pomocí CVIVertexData (3.11.4)

[3] Zda je na cestě soubor, adresář, zda je cesta validní, zda je objekt dostupný.

[4] Body jsou v modelu uloženy postupně po řádcích. V OpenGL jsou trojúhelníky vykreslovány tak, že jsou body brány po trojicích. První tři body tvoří jeden trojúhelník, druhé tři body další trojúhelník a tak dále. To by vedlo k tomu, že většina trojúhelníků by tvořila soustavu rovnoběžných přímek. Jelikož budou vykreslovány jen jako body, tak nás tato degenerace modelu netrápí a šetří nám paměť.

[5] Oba mají metr jako základní jednotku délky. Body získané skenerem je nejprve nutné vynásobit konstantou 10^{-3}, aby byly převedeny na metry z milimetrů.

[6] Mračno bodů i fotografie je reprezentováno jedním prokládaným polem desetinných čísel(float) o rozměrech $6 * 640 * 480$(6 hodnot na bod).

spolu s CVvdImage (3.11.5) akorát s tím rozdílem, že zde se využívá pouze jeden objekt a efektivnější uložení se snadnějším přístupem[7].

5.7 GUI

Tato skupina tříd reprezentuje grafické uživatelské rozhraní, se kterým uživatel pracuje. Dochází zde k vytváření jednotlivých grafických elementů, které jsou připojeny k formulářům, a k obsluhování událostí vyvolaných uživatelem na těchto elementech. Všechny třídy reprezentující formuláře dědí of tříd Form nebo NativeWindow z WinForms knihoven a jsou implementovány v .NET frameworku. To znamená, že i přes to, že se jedná o C++, tento kód je "managed"[8], díký čemuž se nemusíme starat o správu paměti pro použité managed objekty.

5.7.1 WindowMain

Tato třída reprezentuje hlavní menu aplikace. Po spuštění aplikace je vytvořen a zobrazen tento formulář, který během své inicializace vytvoří všechny další formuláře, na které se z něj dá dostat, tj. WindowScan, WindowPose a WindowExporter. Obsahuje pouze základní funkce pro otevření ostatních oken.

5.7.2 WindowScan

Tato třída reprezentuje formulář, ve kterém probíhá fáze "získání dat"(4.7 a 4.11.2). Akce umožněné uživateli z tohoto formuláře jsou ve funkcích reagujících na události vyvolané GUI elementy implementovány prakticky celé. Jedná se především o postavení adresářové struktury, tělo skenovací procedůry a také skenování kalibračního objektu.

5.7.3 WindowPose

Zde je implementován formulář pro rekonstrukci pózy. Obdobně jako u "WindowScan"i zde se jedná především o funkce reagující na události vyvolané uživatelem, které obsluhují jednotlivé GUI elementy a je zde využíván p3p algoritmus implementovaný v rámci třídy "CameraTool"pro samotnou rekonstrukci. Tato třída má také na starost vytvoření formuláře WindowOGL a přístup k němu.

[7] Data jsou zde uložena v jednom poli na rozdíl od objektům, které jsou použity v uvnitř CVIVertexData a CVvdImage.

[8] Managed C++ kód je překládán do Intermediate Language, který je just-in-time kompilován v runtime prostředí .NETu. Všechny knihovny, které .NET framework poskytuje jsou managed. Výhodou managed kódu je, že jeho paměť je spravována pomocí garbage collectoru .NET runtime prostředí. Nevýhodou je trochu jiná syntaxe, náročnější převádění neprimitivních datových typů mezi managed a unmanaged kódem a nutnost .NET runtime pro běh aplikace.

5.7.4 WindowOGL

V této třídě je implementován formulář pro zpřesňování póz skeneru a fotoaparátu. Důležitým prvkem tohoto okna je připojený formuláře CanvasOpenGL jako element. Zde je implementováno především přeposílání parametrů a událostí do CanvasOpenGL formuláře. Jedná se především o události manipulace pózami ve 3D scéně, jejich načítání a ukládání.

5.7.4.1 CanvasOpenGL

Tato třída je na rozdíl od ostatních oken implementována jako potomek třídy NativeWindow namísto třídy Form. Jedná se pouze o vykreslování OpenGL scény, takže tento formulář nemá sám o sobě žádné GUI elementy ani jiné ovládací prvky. Co se manipulace týče, tak je závislý na WindowOGL, aby mu předalo, co je třeba.

Je zde implementováno především vykreslování scény, ale také transformace vykreslovaných objektů, generování nových póz na základě toho, jak bylo s objekty manipulováno a správa objektů vytvořených pomocí třídy RigidModels.

5.7.5 WindowExporter

V této třídě je implementován formulář pro exportování scén. Tento formulář je poměrně jednoduchý a stejně jako předchozí formuláře, nejvíce práce je vykonáno ve funkcích volaných v reakci na událost vyvolanou GUI elementy. Z exportovacího procesu je zde naimplementována především část načtení a předzpracování dat a pro samotné exportování se využívají funkce třídy ExportTool.

Kapitola 6

Testování

6.1 Funkce poskytnuté uživateli

V rámci testování funkcionality aplikace bylo zjištěno, že by se uživatelům mohly hodit některé další funkce, které mohou zlepšit kvalitu práce nebo rozšíří využití aplikace. K tomu bylo potřeba příslušně upravit i grafické uživatelské rozhraní.

6.1.1 Získání dat

V této fázi byly uživateli umožněny další akce. První z nich je "Test table". Tato funkce uživateli umožňuje ověřit připojení aplikace k otočnému stolku a jeho správnou funkci. Toto umožňuje uživateli snadno zjistit, na jakém sériovém portu se MARS 2 nachází a zda funguje tak, jak má.

Druhou z akcí je "Test AF". V zásadě se jedná o otestování, zda skener dokáže zachytit skenovaný objekt v rámci přípravy na sken. Toto je rychlý způsob jak ověřit, zda projde v daném případě příprava na skenování bez toho, aby se ukládala nějaká data.

Následující akce "Test visibility"se trochu váže na předchozí akci. Jde o to, že kdykoliv během celého skenovacího procesu může dojít k tomu, že se skenovaný objekt otočí to takové polohy, že skener nedokáže během přípravy na skenování nalézt žádný odraz laserového paprsku a pro dané otočení nedojde k získání skenu (mračna bodů i 3D modelu). Tato akce tedy provede simulaci skenovacího procesu bez samotného získávání dat a ve výsledku uživateli vrátí informaci o tom, zda by se povedlo získat všechny skeny, nebo které skeny se získat nepodařilo. Tímto způsobem uživatel dokáže rychle získat informaci o tom, zda má cenu snažit se o skenování s daným nastavením a nebo zda je třeba objekt lépe umístit.

Došlo také k úpravě akce skenování kalibračního objektu a skenování uživatelem zvoleného objektu. Nyní, pokud není připojen fotoaparát, aplikace se uživatele zeptá, zda chce pokračovat bez fotoaparátu (dříve by aplikace dále nepokračovala). Pokud uživatel bude chtít, budou nasbírány pouze mračna bodů a 3D modely. I tato data mohou být užitečná pro další zpracování a uživatel nebude vázán pouze na konkrétní fotoaparát, který nemusí být vždy dostupný.

6.1.2 Rekonstrukce pózy

V této fázi došlo pouze k menším úpravám. K ukládání nových 3D souřadnic bodů jsou využívána tlačítka pro přechod na předchozí, nebo následující bod, která nejprve uloží novou pozici bodu a až poté provedou posun. Je to především pro předcházení chybám, které by mohly nastat, když uživatel zapomene stisknout tlačítko pro aplikaci změn a rovnou přejde na další bod.

Další změna, která byla provedena v rámci rekonstrukce pózy je možnost zvětšení a zmenšení fotografií. Jde především o to, že při zadávání bodů je potřeba klikat co nejpřesněji a velikost jednoho pixelu je značně malá. Proto je vhodné uživateli umožnit fotografii vhodně naškálovat. U fotografie získané fotoaparátem docházelo k problému, že byla příliš veliká a uživatel nedokázal vidět celý kalibrační objekt v okně fotografie. U fotografie skeneru zase docházelo k nepřesnostem, neboť rozlišení 640×480 je velmi malé a nepřesnost několika pixelů je znát na zrekonstruované póze.

Proto byly přidány 2 elementy typu "slider"do formuláře rekonstrukce, kterými uživatel může jednotlivé fotografie přiblížit nebo oddálit. Element "slider"byl zvolen místo běžnějšího přístupu, tj. zvětšování/zmenšování pomocí otáčení kolečka myši, proto, aby byla usnadněna práce uživatelům, kteří nemají k dispozici myš s kolečkem. Takto mohou aplikaci snadno využívat jak uživatelé PC, tak uživatelé s notebookem.

6.1.3 Zpřesnění pózy

V rámci této fáze došlo pouze k menším úpravám. Bylo zde přidáno několik nastavení pro vykreslování krychle v počátku pro snadnější zarovnávání skenů/fotografií. Nyní zde uživatel může zadat velikost hrany a informaci o tom, zda má být krychle invertovaná v nějakém směru(Zda má být krychle zrcadlena podle roviny procházející počátkem, kolmé na zadanou osu.).

Další modifikací je přidání možnosti nastavit průhlednost fotografie. To umožní uživateli zobrazit fotografii přesně tak, jak bude chtít. Bude možné, aby byla úplně průhledná a uživateli nebránila v modifikaci pózy skeneru, nebo aby byla méně průhledná a byly dobře vidět obrysy kalibračního objektu.

Poslední modifikací je umožnění uživateli měnit zorný úhel kamery. To je především pro zvětšení objektů ve scéně a zjednodušení zarovnávání fotografie.

6.1.4 Expot scény

V rámci exportování scény nebyly uživateli přidány žádné nové akce k použití, ale při exportování 3D modelů do PLY formátu bude docházet nejen k vytvoření celé výsledné scény, ale budou ukládány i jednotlivé 3D modely tak, jak mají být ve scéně. Je to především kvůli tomu, aby uživatel mohl pracovat s jednotlivými 3D modely v dalších programech, protože s jednou celou 3D scénou už moc úprav provést nemůže. Takto uživatel bude moci zpracovat

jednotlivé 3D modely samostatně a jejich výslednou scénu získá jejich složením například v programu MeshLab. Zároveň bude stále uložena celá složená scéna.

Toto uživateli umožní zpracování jednotlivých modelů samostatně, jejich přesnější napozicování a především, pokud se 3D modely příliš překrývají, některé vynechat, nebo odebrat některé trojúhelníky a části modelů.

6.2 Vliv světla

V této části bude popsáno testování vlivu osvětlení na výsledky získané skenerem.

6.2.1 Parametry testování

Dle informací z dokumentace a instrukčního manuálu[19] ke skeneru, pro práci skeneru je vhodné zvolit tmavší prostředí. Skener sám o sobě modifikuje v rámci přípravy na skenování sílu laserového paprsku, který bude použit, tak, aby v daných podmínkách pro daný předmět zachytil co největší množství dat s co největší přesností.

V rámci testování tedy půjde o zjištění, jak veliký vliv na výsledky má různá kvalita osvětlení skenovaného objektu a v jakém osvětlení byly získány nejlepší výsledky. Jelikož dle instrukcí je skener stavěn pouze pro práci ve budovách a ne v terénu, lze předpokládat, že pro nás bude nejvýznamnější umělé osvětlení a nebo sluneční světlo přicházející oknem. V rámci testování tedy půjde především o tři scénáře. První scénář je objekt v osvětlené místnosti se slunečním světlem přicházejícím z okna. Takto bude objekt osvětlen jak nejvíce je to možné a zároveň bude osvětlen více z jedné strany. Druhý scénář je takový, že objekt bude osvětlen pouze zářivkami na stropě místnosti. Toto je nejpravděpodobnější scénář, který bude nastávat. Ve třetím scénáři je objekt v místnosti se zataženými žaluziemi a zhasnutým světlem tak, aby byl v co největší možné tmě.

Pro testování budou provedeny testy se třemi různými objekty. Prvním objektem je měřící přístroj značky Bosch, který je vyrobený z tmavého lesklého plastového materiálu a má hodně drobných detailů na svém povrchu. Druhým objektem je plastový hranol s matným povrchem. Třetím objektem je plastový obal od žvýkaček vyrobený z velmi lesklého plastu s hladkým povrchem bez detailů.

V rámci testů provedených na jednom objektu bude jediný parametr, který se bude měnit, osvětlení. Scéna sama o sobě, jako například pozice skeneru, pozice objektů nebo předměty ve scéně se měnit nebudou. Tím se zaručí co největší podobnost mezi jednotlivými testy na jednom objektu a vyloučí se vlivy, které s osvětlením nesouvisí.

Pro každý objekt pro každý scénář bude získáno 35-50 skenů daného objektu.

6.2.2 Způsob zpracování dat

Naším cílem je zjistit vliv osvětlení na kvalitu skenů. První otázkou je, kolik bodů dokáže při jednotlivých scénářích skener zachytit. Druhou otázkou jak kvalitní body jsme získali.

Na první otázku dokážeme odpovědět snadno. Zajímat nás bude především množství bodů získaných skenerem pro každý scénář. Lze předpokládat, že množství bodů v rámci jednoho scénáře pro jeden objekt bude stejné, nebo velmi podobné.

Pro zodpovězení druhé otázky bude nejprve nutné definovat, co myslíme kvalitou bodů. Jde nám především o to, aby body co nejlépe odpovídaly skutečným bodům. Tyto skutečné body sice změřit s naprostou přesností nedokážeme. Lze předpokládat, že naměřené body se vychylují od skutečné pozice do všech stran a s normálním rozdělením(Tj. nejvíce bude blízko skutečného bodu a s větší vzdáleností bude četnost bodů rapidně klesat.). Lze předpokládat, že pokud všechna měření jednoho bodu budou mít za výsledek stejné hodnoty, tak je tento bod změřen přesně (nebo došlo ke stejné systematické chybě u všech měření.). Čím více se budou body od sebe lišit, tím nekvalitněji máme body naměřené.

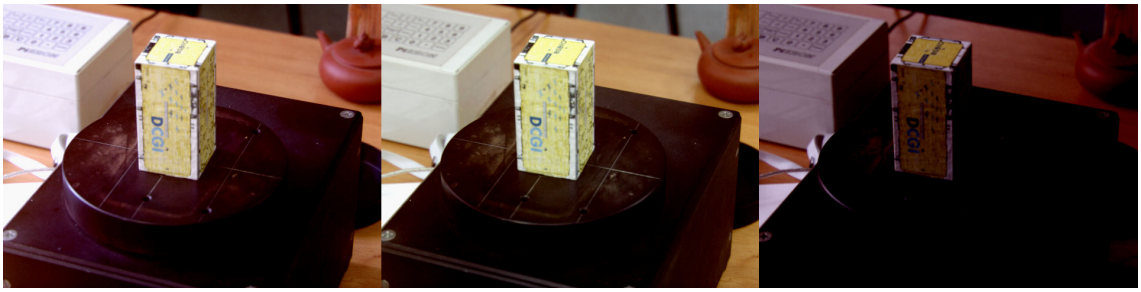
Každý skenm se skládá z matice 640×480 trojrozměrných bodů. Každá sada obsahuje 35 až 50 skenů. Pro každou sadu dat je vypočtena matice průměrných bodů tak, že průměrný bod na pozici $[i, j]$ je vypočten jako aritmetický průměr(těžiště) bodů na pozici $[i, j]$ všech skenů v dané sadě. Tím získáme nejlepšího reprezentanta výsledné pozice.

Následně spočítáme pro každý bod euklidovskou vzdálenost(odchylku) od jeho průměrného bodu a toto provedeme pro všechny body ve všech 35 až 50 maticích. Následně spočteme průměr ze všech těchto odchylek od průměrných bodů a také maximální hodnotu odchylky. Tím získáme představu o tom, o jakou vzdálenost se přibližně každý získaný bod lišil od pozice, na které se měl vyskytovat a také informaci o tom, jaká byla největší chyba pro všechny body dané sady skenů.

6.2.3 Naměřené výsledky

Obrázek 6.1: Fotografie pořízené skenerem z prvního měření. Postupně zleva doprava jsou to "sluncem osvětlená scéna", "umělé světlo", "šero".

Obrázek 6.2: Fotografie pořízené skenerem z druhého měření. Postupně zleva doprava jsou to "sluncem osvětlená scéna", "umělé světlo", "šero".

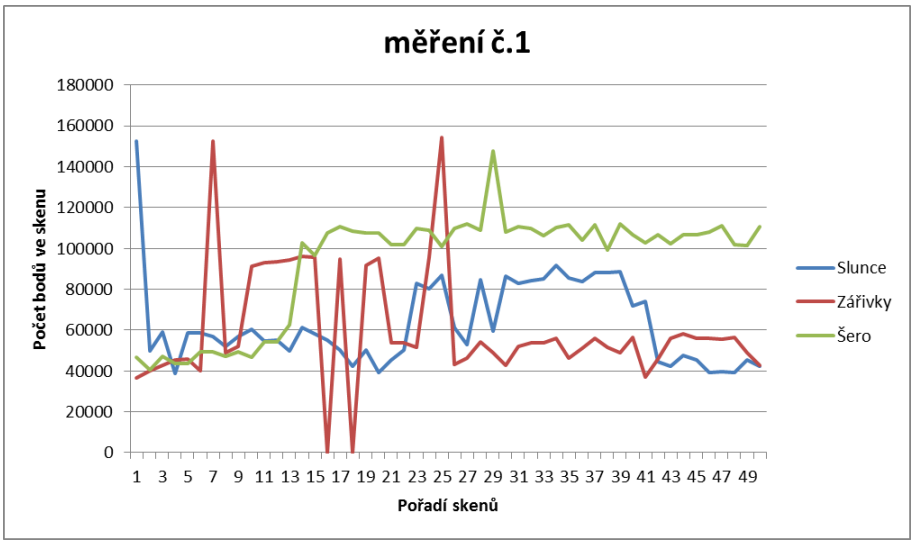


Obrázek 6.3: Fotografie pořízené skenerem z třetího měření. Postupně zleva doprava jsou to "sluncem osvětlená scéna", "umělé světlo", "šero".

Z prvních tří grafů (6.4, 6.5, 6.6) je dobře vidět, jak se v rámci jednotlivých měření měnily počty bodů získané skenerem. Je zde velmi dobře vidět, že co se množství bodů týče, jsou zde ohromné rozdíly v počtech bodů mezi jednotlivými skeny stejné scény za stejných podmínek. Často se zde mezi dvěma skeny, které by měly mít stejné výsledky, vyskytují větší rozdíly, než mezi skeny za jiných světelných podmínek. U prvního měření se v pozdější fázi skenování dařilo získávat lepší výsledky u skenování v šeru než u ostatních dvou, ale u dat z druhého a třetího měření nelze říci nic o tom, že by data pro jeden ze scénářů byla lepší než pro ostatní scénáře.

měření		č.1			č.2			č.3	
osvětlení	Slunce	Zářivky	Šero	Slunce	Zářivky	Šero	Slunce	Zářivky	Šero
dErrAvg	0.158	0.153	0.083	0.073	0.06	0.067	0.154	0.08	0.330
dErrMax	6.165	4.672	5.453	3.225	3.04	2.005	5.088	1.870	2.212

Tabulka 6.1: Tabulka obsahující výsledky měření kvality bodů. dErrAvg označuje průměrnou hodnotu odchylky každého bodu od průměrných bodů v milimetrech. dErrMax označuje nejvyšší hodnotu odchylky naměřených bodů od průměrných bodů.



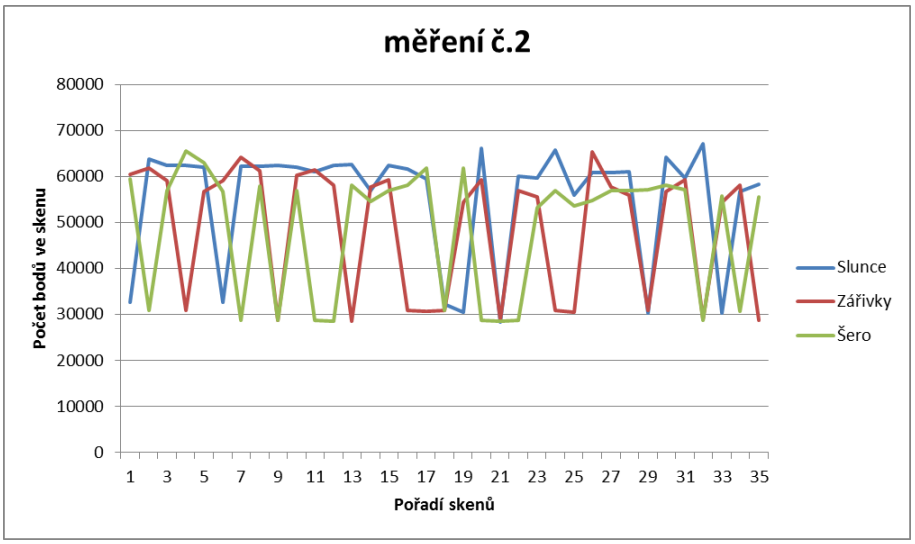
Obrázek 6.4: Počty bodů ve skenech z prvního měření. Osa x označuje pořadí skenů a osa y označuje počet bodů v daném měření. Výsledky měření za slunečního světla jsou označena modře, pouze s umělým světlem červeně a v šeru modře.

Nejpravděpodobnější vysvětlení těchto rozdílů mezi skeny stejné scény za stejných podmínek, je příprava skeneru na skenování. Před každým skenem probíhala příprava skeneru na skenování, při které se nastavuje intenzita laseru, která bude použita při samotném skenu. Právě různá intenzita laserového paprsku byla pozorována mezi jednotlivými skeny, což je důvod, proč se přikláním k tomuto vysvětlení. Proč v této fázi dochází k takto velikým inkonzistencím není známo. Jelikož rozdíly jsou tak obrovské, nelze říci, že by kvalita osvětlení hrála nějak významnou roli ve skenovacím procesu, co se počtu získaných bodů týče.

Co se kvality bodů týče, ani zde nelze říci, že by některé z osvětlení mělo nějaký významný vliv na průměrné nebo maximální odchylky bodů. Jak je vidět z výsledků v tabulce 6.1, ani v jednom z osvětlení se nejeví, že by docházelo k významnému zlepšení přesnosti oproti ostatním typům osvětlení. U prvního měření sice došlo u skenování v šeru k velmi výraznému zlepšení průměrné odchylky, ale u měření číslo dva došlo při skenování v šeru k horší průměrné odchylce a u třetího měření šero dopadlo nejhůře. Ani co se týče maximální odchylky nelze říci, že by některé z osvětlení mělo lepší výsledky než ostatní.

6.2.4 Závěr o vlivu světla

S množstvím naměřených dat se nepodařilo prokázat, že by některé z osvětlení skenovaného objektu a scény vedlo k výrazně lepším nebo horším výsledkům co se počtu získaných bodů a jejich kvality týče. Zároveň se ale podařilo zjistit, že i v podmínkách, které se neměnily docházelo k velikému výkyvu množství bodů získaných skenerem.

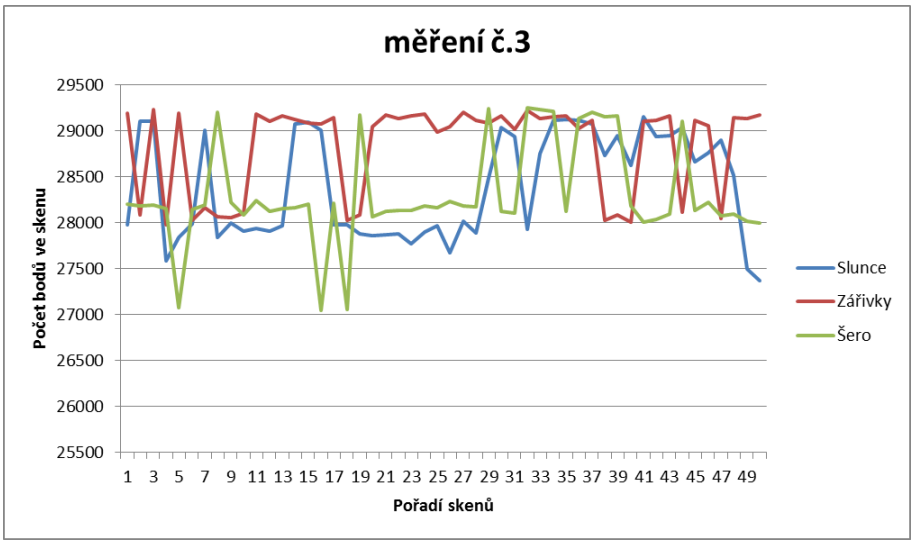


Obrázek 6.5: Počty bodů ve skenech z druhého měření. Osa x označuje pořadí skenů a osa y označuje počet bodů v daném měření. Výsledky měření za slunečního světla jsou označena modře, pouze s umělým světlem červeně a v šeru modře.

V budoucnu by tedy bylo vhodné věnovat se tomuto úkazu blíže a zjistit, proč k tomu dochází, jak tomu zabránit, jak zajistit nastavení, které získá co největší možný počet bodů a jak se kvalita těchto bodů bude lišit od kvality bodů získaných při nastavení zvoleném skenerem, jako je tomu teď.

Na závěr k osvětlení je ještě potřeba zmínit jednu věc. Pokud nám jde pouze o skeny, světlo nám nedělá až tak veliké problémy, ale pokud chceme získat i barevnou informaci ať fotoaparátem, nebo skenerem samotným, bude lepší volit scénu s lepším osvětlením. To je možné vidět na fotografiích 6.1, 6.2 a 6.3. Na fotografiích získaných v šeru je objekt sotva vidět a jako textura tyto fotografie nejsou v hodné. Co se týče umělého osvětlení zářivkami, tyto fotografie jsou poměrně dobré, barva objektu není příliš zkreslená a celá scéna je dobře viditelná. Co se týče slunečního světla spolu s umělým, je zde vidět výrazné barevné zkreslení a jelikož jsou objekty osvětleny zleva zezadu, dochází vlivem světlejších objektů v pozadí k automatickému ztmavení celé fotografie. Proto v tomto případě bude nejlepší zvolit lépe osvětlené scény, aby byla barevná informace získána v co nejlepší kvalitě. Nejvhodnější by bylo, aby byl skenovaný objekt osvětlen co nejrovnoměrněji ze všech stran.

Co se týče použití blesku pří získávání fotografie fotoaparátem, zde bude docházet k odleskům a k artefaktům s tím spojených(Přivrácené plochy budou lépe osvětleny než plochy s normálou kolmou na směr fotografování.). To je věc, kterou u textur nechceme a proto bude vhodné fotit bez blesku v místnosti s rovnoměrným osvětlením ze všech stran, které nebude příliš ostré.



Obrázek 6.6: Počty bodů ve skenech z třetího měření. Osa x označuje pořadí skenů a osa y označuje počet bodů v daném měření. Výsledky měření za slunečního světla jsou označena modře, pouze s umělým světlem červeně a v šeru modře.

Místnost, ve které se 3D skener nachází a používá, je vybavena stropními zářivkami produkujícími bílé světlo a díky bílým stěnám dochází k dobrému osvětlení objektů ze stran bez vznikání příliš velikého množství odlesků. To je dobře vidět na fotografiích 6.2 a 6.3, kde prostřední fotografie je právě za použití tohoto osvětlení a mají nejlepší kvalitu oproti ostatním dvěma. Použití blesku, jak bylo zmíněno výše, není při fotografování vhodné. Co se týče fotografování objektu 6.1, zde i přes stejné světelné podmínky jako u ostatních došlo k výraznému ztmavení fotografií. K získání barevné informace ale bude použit fotoaparát a u něj je možné v nastavení zabránit tomu, aby docházelo k adaptaci na světlo.

Kapitola 7

Závěr

Výsledkem této práce je aplikace, s jejíž pomocí uživatel může použít 3D skener VIVID 9i, fotoaparát Canon EOS 100D a otočný stolek řízený skrz řídící jednotku MARS 2 k naskenování uživatelem zvoleného objektu ze všech stran. Pro jednoduchost práce, aplikace poskytuje grafické uživatelské rozhraní s co nejmenším množstvím elementů tak, aby uživatel mohl pracovat a nebyl příliš zahlcen velikým množstvím možností. Uživateli je umožněno získat mračna bodů, 3D modely a fotografie objektu pro libovolné množství pozic otočného stolku.

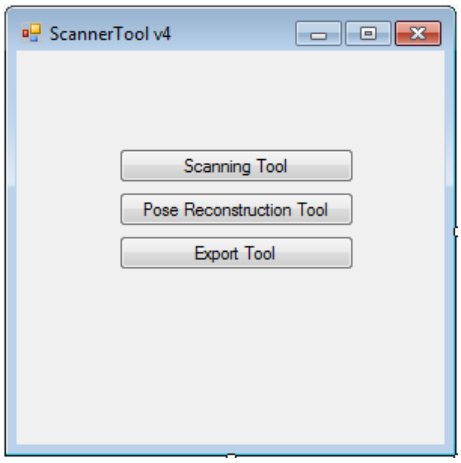
Pomocí nástroje pro rekonstrukci pózy je mu umožněno jednoduchým způsobem zjistit, jak byly skener i fotoaparát umístěny v prostoru vůči otočnému stolku, zvolením několika bodů ve fotografiích a nastavením jejich skutečné pozice. V případě, že uživatel nebyl příliš přesný při rekonstrukci pózy, nástrojem pro zpřesnění pózy je mu umožněno pózy fotoaparátu i skeneru ještě více zpřesnit do požadované kvality.

Nakonec uživatel může výslednou scénu exportovat do formátu bundle, nebo do formátu PLY. Pro PLY formát si může zvolit, zda chce exportovat body, 3D modely, kamery nebo libovolnou kombinaci předchozích možností. Při exportování 3D modelů je kromě vytvoření jedné úplné scény vytvořena jedna scéna pro každý 3D model, aby je uživatel mohl dále zpracovávat i samostatně, pokud bude chtít.

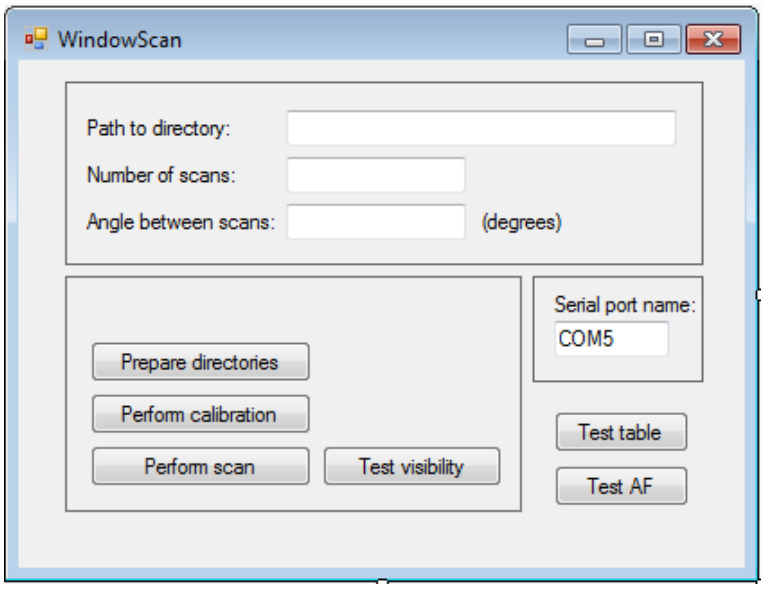
Co se týče vlivu světla na kvalitu skenování, nepodařilo se prokázat, že by intenzita okolního osvětlení hrála nějakou roli ve výsledcích skenovacího procesu. Jednotlivé skeny za stejných podmínek se od sebe lišily více, než jak se výsledky lišily při různých intenzitách osvětlení. Kvalita osvětlení má ovšem výrazný vliv na získanou barevnou informaci o scéně a proto je lepší mít skenovaný objekt rovnoměrně dobře osvětlen ze všech stran, aby výsledná textura měla co nejméně ostrých barevných přechodů a měla co nejvyšší kvalitu.

Vzhledem k tomu, že osvětlení v místnosti skeneru je dostatečné, není třeba dělat žádné úpravy.

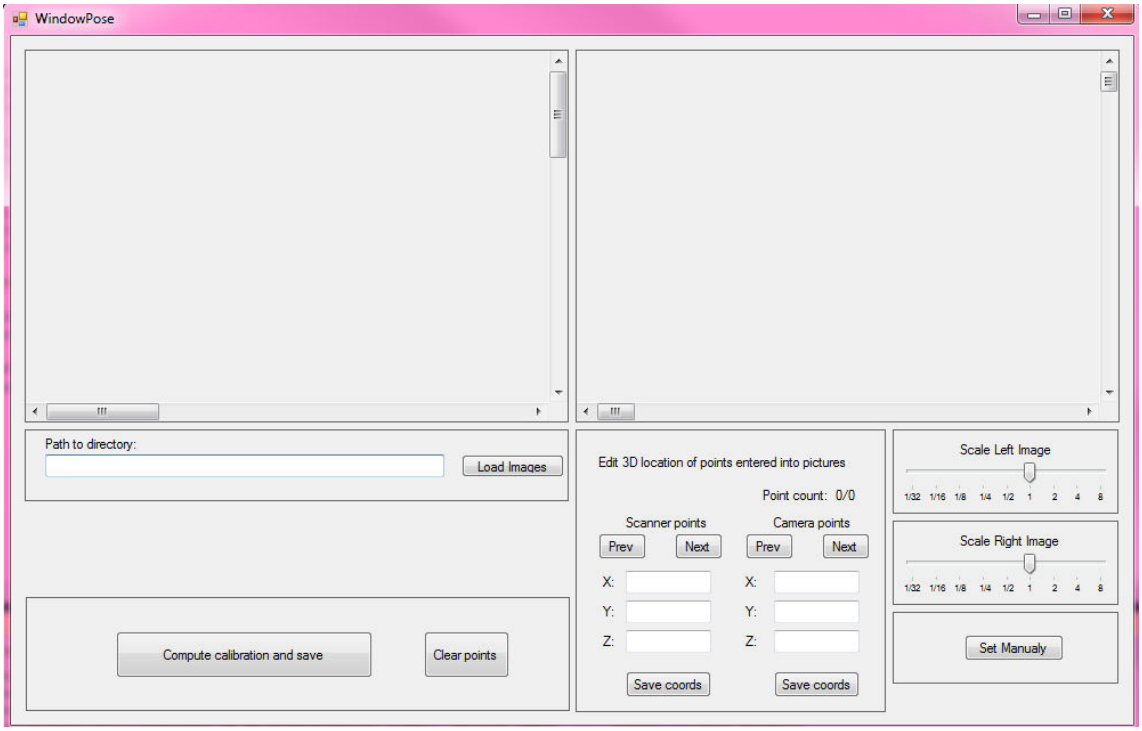
7.0.1 Výstupní scény



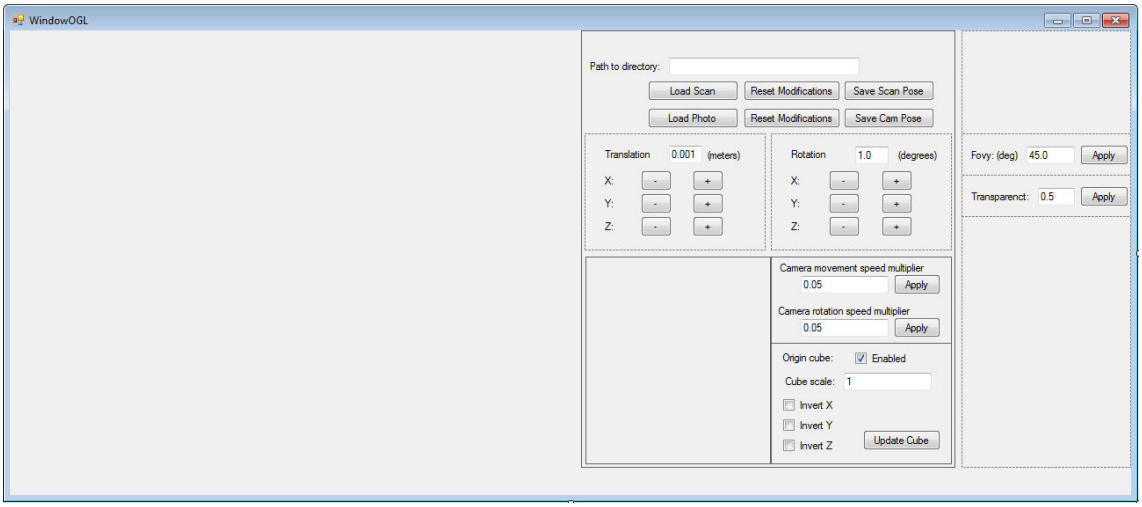
Obrázek 7.1: Hlavní menu - finální verze GUI



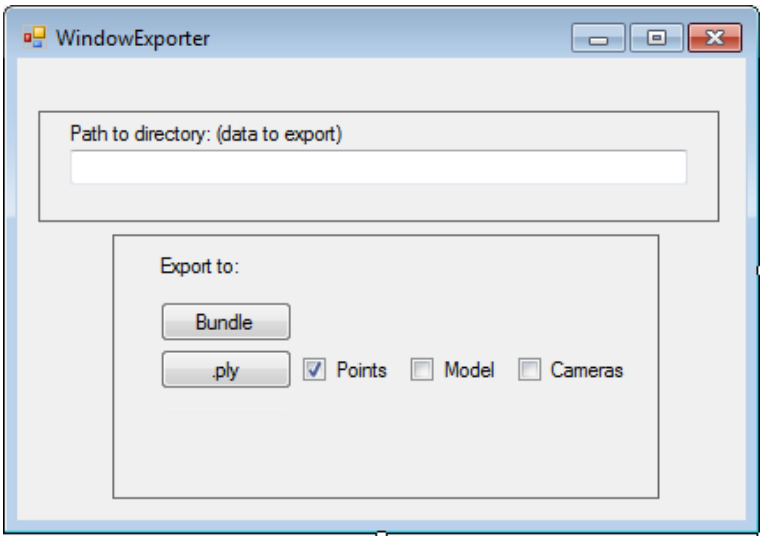
Obrázek 7.2: Formulář získání dat - finální verze GUI



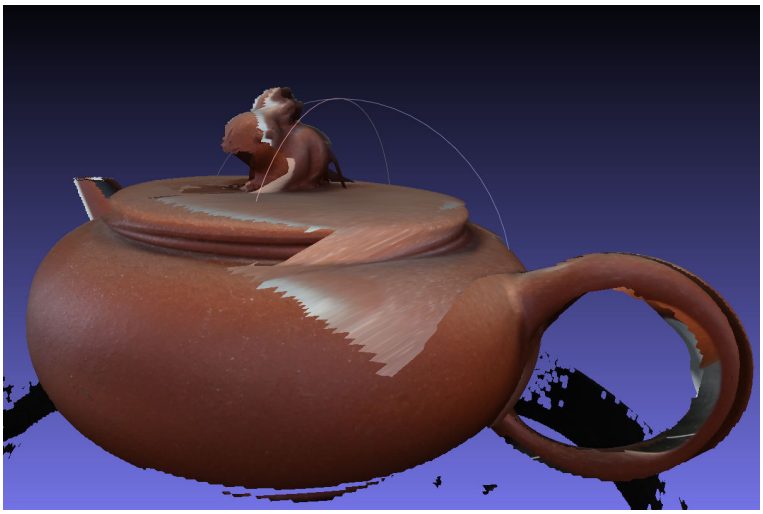
Obrázek 7.3: Formulář rekonstrukce pózy - finální verze GUI



Obrázek 7.4: Formulář zpřesnění pózy - finální verze GUI

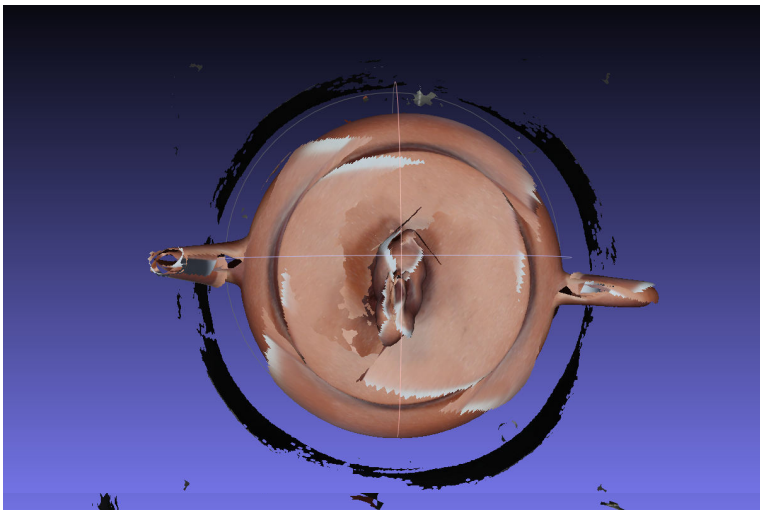


Obrázek 7.5: Formulář exportu scény - finální verze GUI



Obrázek 7.6: Finální scéna - konvička 1. Následky nedokonalého zarovnání dílčích modelů v důsledku nepřesnosti v póze je možné vidět na uchu konvičky. Dále je možné vidět artefakty vzniklé při texturování okrajových trojúhelníků v důsledku nepřesné pózy fotoaparátu (v pozadí byl šedý koberec).

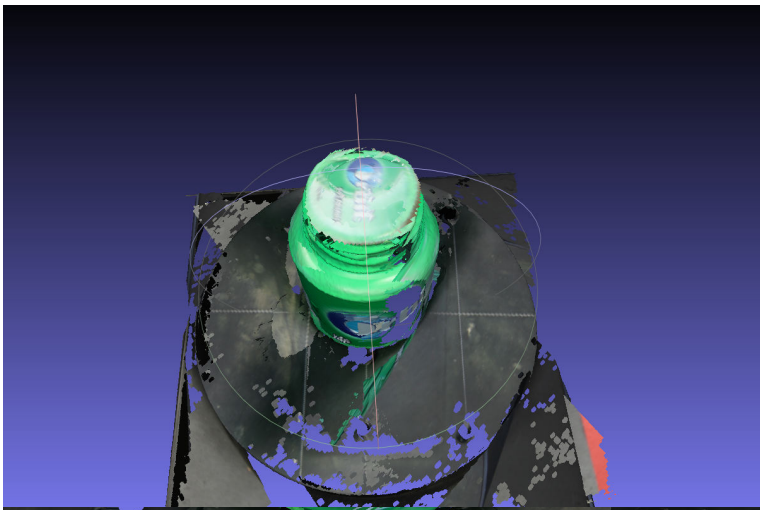
Obrázek 7.7: Finální scéna - konvička 2.



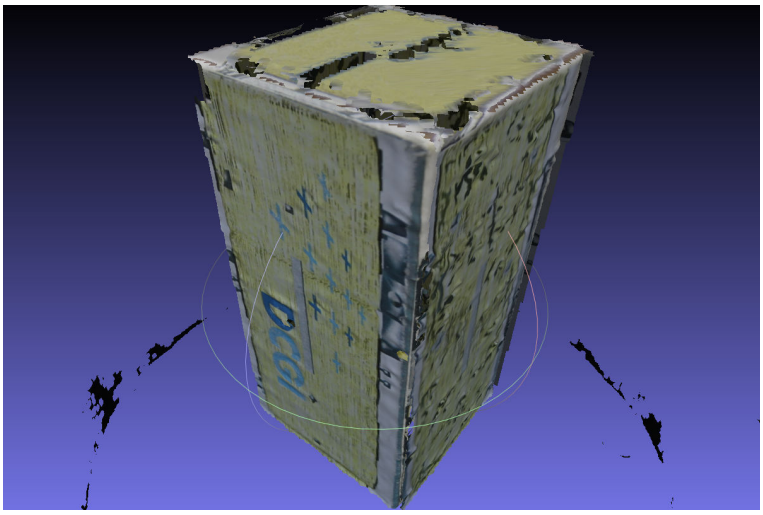
Obrázek 7.8: Finální scéna - konvička 3. Zde je vidět, jak nepřesnost pózy skeneru ovlivní výsledné složení modelů. Proto jsou 3D modely exportovýny i samostatně, aby uživatel mohl pozice modelů dodatečně poupravit.

Obrázek 7.9: Finální scéna - balení žvýkaček 1. Zde je vidět, že skener má občas problémy se skenováním velmi lesklých objektů, proto na přední části modelu chybí trojúhelníky.

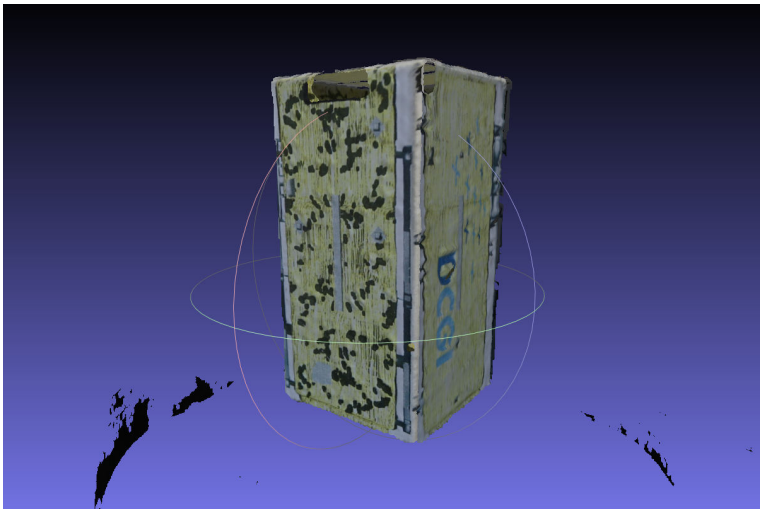
Obrázek 7.10: Finální scéna - balení žvýkaček 2. V tomto záběru je patrné, že skener není stavěn pro skenování průhledných a průsvitných objektů, v tomto případě jsou části modelu zdeformovány tím, že laserové paprsky prošly obalem a odrážely se od žvýkaček uvnitř.



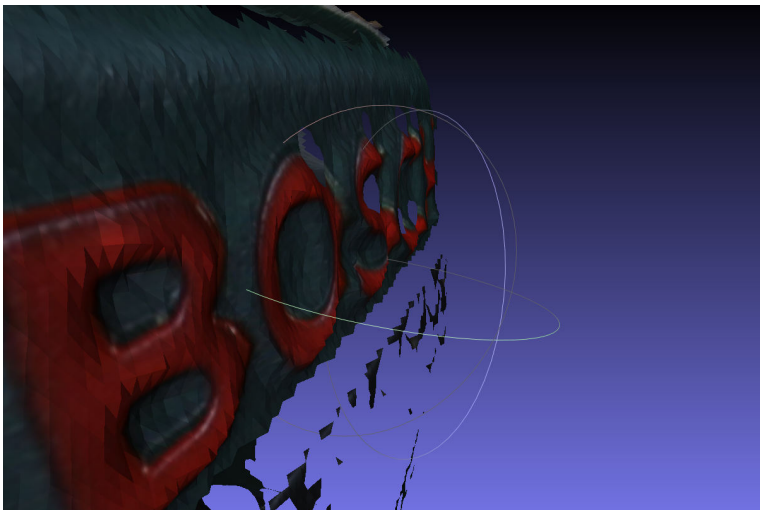
Obrázek 7.11: Finální scéna - balení žvýkaček 3. Tato scéna, oproti scéně s konvičkou má pózu skeneru i fotoaparátu mnohem přesnější a proto na sebe dílčí modely velmi dobře navazují.



Obrázek 7.12: Finální scéna - krabička 1. U této scény je vidět velmi dobré zarovnání dílčích modelů i textur díky přesnosti póz.

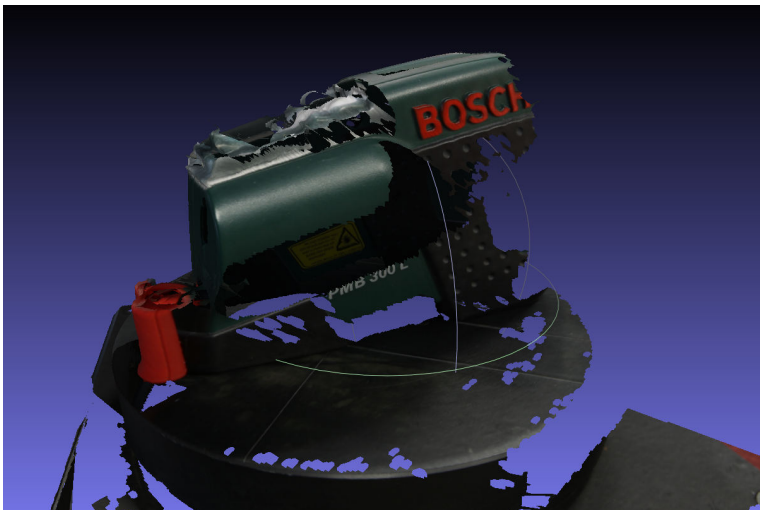


Obrázek 7.13: Finální scéna - krabička 2.



Obrázek 7.14: Finální scéna - měřící přístroj Bosch 1.

Obrázek 7.15: Finální scéna - měřící přístroj Bosch 2.



Obrázek 7.16: Finální scéna - měřící přístroj Bosch 3.

Literatura

[1] OpenGL Mathematics. Dostupné z: <http://glm.g-truc.net/0.9.8/index.html>. [vid. 16.5.2017].

[2] ImageMagick webpage. Dostupné z: <http://www.imagemagick.org/script/index.php>. [vid. 16.5.2017].

[3] MeshLab webpage. Dostupné z: <http://www.meshlab.net/>. [vid. 16.5.2017].

[4] StackOverflow - EDSDK Message Loop help 1, . Dostupné z: <http://stackoverflow.com/questions/13669312/c-sharp-canon-sdk-no-callback-after-cameracommand-takepicture>. [vid. 16.5.2017].

[5] StackOverflow - EDSDK MEssage Loop help 2, . Dostupné z: <http://stackoverflow.com/questions/18360402/no-callback-from-event-handler-canon-sdk-2-12>. [vid. 16.5.2017].

[6] OpenGL API Documentation. Dostupné z: <https://www.opengl.org/documentation/>. [vid. 16.5.2017].

[7] PNG (Portable Network Graphics) Specification, Version 1.1. Dostupné z: <http://www.libpng.org/pub/png/spec/1.1/PNG-Contents.html>. [vid. 16.5.2017].

[8] Fotografie skeneru VIVID 9i celá, . Dostupné z: <http://www.hiphealth.ca/media/3DDig1.jpg>. [vid. 16.5.2017].

[9] Fotografie skeneru VIVID 9i detail, . Dostupné z: <https://www.researchgate.net/profile/Mercedes_Farjas/publication/239590802/figure/fig5/AS:298580423266310@1448198528488/Figure-10-Minolta-Vivid-9i-laser-scanner.png>. [vid. 16.5.2017].

[10] libpng webpage. Dostupné z: <http://www.libpng.org/pub/png/libpng.html>. [vid. 16.5.2017].

[11] AUTODESK. FBX webpage. Dostupné z: <https://www.autodesk.com/products/fbx/overview>. [vid. 16.5.2017].

[12] BOULOS, S. Object Files. Dostupné z: <http://www.cs.utah.edu/~boulos/cs3505/obj_spec.pdf>. [vid. 16.5.2017].

[13] BOURKE, P. PLY - Polygon File Format. Dostupné z: <http://paulbourke.net/dataformats/ply/>. [vid. 16.5.2017].

[14] CANON. EDSDK webpage, . Dostupné z: <https://www.didp.canon-europa.com/>. [vid. 16.5.2017].

[15] CANON. EOS 100D webpage, . Dostupné z: <http://www.canon-europe.com/for_home/product_finder/cameras/digital_slr/eos_100d/>. [vid. 16.5.2017].

[16] CIPA. EXIF specification. Dostupné z: <http://www.cipa.jp/std/documents/e/DC-008-Translation-2016-E.pdf>. [vid. 16.5.2017].

[17] HUGGEL, A. EXIF reference table. Dostupné z: <http://www.exiv2.org/metadata.html>. [vid. 16.5.2017].

[18] JACOB, G. G. e. a. B. Eigen. Dostupné z: <https://eigen.tuxfamily.org/dox/>. [vid. 16.5.2017].

[19] KONICA MINOLTA. VIVID 9i instructions, . Dostupné z: <https://www.konicaminolta.com/instruments/download/instruction_manual/3d/pdf/vivid-9i_vi-9i_instruction_eng.pdf>. [vid. 16.5.2017].

[20] KONICA MINOLTA. VIVID 9i Brochure, . Dostupné z: <http://www.dirdim.com/pdfs/DDI_Konica_Minolta_Vivid_9i.pdf>. [vid. 16.5.2017].

[21] MICROSOFT. Using MEssages and Message Queues, . Dostupné z: <https://msdn.microsoft.com/en-us/library/windows/desktop/ms644928(v=vs.85).aspx>. [vid. 16.5.2017].

[22] MICROSOFT. MFC - MicroSoft Foundation Class, . Dostupné z: <https://msdn.microsoft.com/en-gb/library/d06h2x6e.aspx>. [vid. 16.5.2017].

[23] MICROSOFT. Windows Forms documentation, . Dostupné z: <https://msdn.microsoft.com/en-us/library/dd30h2yb.aspx>. [vid. 16.5.2017].

[24] MICROSOFT. .NET webpage, . Dostupné z: <https://www.microsoft.com/net>. [vid. 16.5.2017].

[25] PíšA, P. Uživatelský manuál k jednotce MARS 2. Dostupné z: <http://cmp.felk.cvut.cz/~pisa/mars/mars_man_cz.html>. [vid. 16.5.2017].

[26] PAJDLA, T. Elements of Geometry for Computer Vision. *FEE CTU,[online].[cit. 2013-05-19], March*. 2013. Dostupné z: <http://cmp.felk.cvut.cz/~pajdla/gvg/GVG-2016-Lecture.pdf>.

[27] PIKRON. MARS 2 webpage. Dostupné z: <http://www.pikron.com/pages/products/motion_control/mars_2.html>. [vid. 16.5.2017].

[28] ŽÁRA, J., BENEŠ, B., SOCHOR, J., FELKEL, P. *Moderní počítačová grafika*. Praha : Computer Press, 2nd edition, 2005. ISBN 80-251-0454-0.

[29] RATOC SYSTEMS, INC. U2SCX convertor. Dostupné z: <http://www.ratocsystems.com/english/products/U2SCX.html>. [vid. 16.5.2017].

[30] SNAVELY, N. *Bundler v0. 4 User's Manual* [online]. 2009. [vid. 16.5.2017]. Dostupné z: <http://www.cs.cornell.edu/~snavely/bundler/bundler-v0.4-manual.html>.

[31] SNAVELY, N. Bundler webpage. Dostupné z: <http://www.cs.cornell.edu/~snavely/bundler/>. [vid. 16.5.2017].

[32] TON. FBX specification, 2013. Dostupné z: <https://code.blender.org/2013/08/fbx-binary-file-format-specification/>. [vid. 16.5.2017].

[33] WU, C. VisualSFM: A visual structure from motion system. 2011, 9. Dostupné z: <http://ccwu.me/vsfm/>.

Příloha A

Obsah přiloženého DVD

DVD přiložené k práci obsahuje především text této práce jak ve zdrojovém formátu (LaTeX), tak i ve formátu PDF. Kromě toho také obsahuje projekt pro Visual Studio 2015 s aplikací "Scanner Tool v4", jež je výsledkem této práce,dokumentaci a zkompilovanou verzi aplikace. Podrobnější informace jsou k nalezení v souboru README.txt na přiloženém DVD.

Obsah:

\text	Adresář s textem práce ve zdrojovém formátu a v pdf.
\project	Adresář obsahující projekt, dokumentaci, testovací data a další.
\README.txt	Soubor obsahuje detailní popis obsahu DVD.