Technická univerzita v Liberci

Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: B2612 Elektrotechnika a informatika
Studijní obor: 2612R011 Elektronické informační a řídicí systémy

Extrakce 3D grafického modelu z 2D dat

Extraction of 3D graphics model from 2D data

Bakalářská práce

Autor: **Petr Ječmen**

Vedoucí bakalářské práce: Ing. Petr Kretschmer

Konzultant: Ing. Jiří Hnídek

V Liberci dne 29. 5. 2009

Místo pro vložení zadání

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé DP a prohlašuji, že **s o u h l a s í m** s případným užitím mé diplomové práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít své diplomové práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

Datum

Podpis

Poděkování

Na tomto místě bych chtěl poděkovat vedoucímu mé bakalářské práce, Ing. Petru Kretschmerovi za seznámení s problematikou tvorby modelu z fotografií a také za rady a připomínky poskytnuté během tvorby vlastního programu a psaní průvodní zprávy.

Dále bych chtěl poděkovat své rodině, která mě podporovala jak po stránce finanční, tak po stránce morální a poskytla mi tak ideální prostředí pro tvorbu mé bakalářské práce.

Velké poděkování patří také mé přítelkyni, která mi poskytovala jiný úhel pohledu na všemožné problémy spojené s vytvářením programu a tím pádem umožnila jejich rychlejší řešení. Samozřejmě bych jí také chtěl poděkovat za její velikou trpělivost, která byla během tvorby práce mnohokrát zkoušena.

Abstrakt

Tématem této práce je vytvořit software schopný vytvořit s asistencí uživatele grafický 3D model předmětu zachyceného na fotografiích. Po uživateli je vyžadováno, aby na fotografiích vyznačil hraniční body modelu a následně je propojil mezi fotografiemi. První část práce je úvodem do problematiky modelování z fotografií, další část se zabývá implementací programu schopného rekonstrukce modelu předmětu z fotek v programovacím jazyku Delphi. Nakonec je provedeno porovnání užitých metod výpočtu z hlediska přesnosti modelu a také časové náročnosti.

Klíčová slova:

modelování, grafika, fotografie, extrakce

Abstract

Subject of this work is to create software capable of creating a 3D model of a photographed item with user assistance. User's part is to select model border points and link them between photos. First part of work is an introduction in photo modeling problematic, next part is focused on implementation of a program capable of model reconstruction from photos in Delphi programming language. Last part is about comparison of used computation techniques focused on model correctness and time needed for calculation.

Keywords:

modeling, graphics, photos, extraction

Obsah

Úvod

V poslední době dochází k velikému rozmachu modelování objektů. Hlavním důvodem jsou potřeby průmyslu, díky dobrému počítačovému 3D modelu se dá ušetřit mnoho peněz i času, protože lze provádět veliké množství testů a simulací přímo v počítači bez nutnosti vytvářet reálný předmět, takže se na spoustu vad může přijít již při návrhu.

Čím dál tím častěji ale CAD systémy využívají lidé pro své osobní potřeby (např. rozmístění nábytku v pokoji, předběžný návrh domu apod.). Zpočátku se takovéto věci řešily jako nákres půdorysu, případně několik pohledů na objekt z různých úhlů. Chyběl ale jakýkoliv náznak prostoru, to bylo až na samotném člověku aby si těch několik obrázků sestavil v hlavě dohromady. Existují sice programy schopné vytvoření 3D modelu např. domu či bytu, jedná se ale vždy o specializovaný software, který je navíc limitovaný předdefinovanou sadou objektů, které můžeme do prostoru umístit.

Vzniká tedy požadavek na software, který by byl schopný vytvořit 3D model nějakého objektu, aniž by musel uživatel strávit hodiny jeho měřením a následným kreslením v nějakém modelovacím programu. Výsledkem této bakalářské práce by měl být program schopný vytvořit 3D model objektu, aniž by uživatel musel mít jakékoliv zkušenosti s prací s CAD systémy, případně modelováním obecně.

1. Teoretický rozbor

Rekonstrukce modelu z fotografií je obecně velice složitý problém. Hlavním problémem je, že nevíme, co na fotografiích bude, kde to bude přesně umístěno a za jakých podmínek byly snímky pořizovány. Touto problematikou se zabývá obor jménem fotogrammetrie. Nezabývá se ale pouze rekonstrukcí modelu jednoho předmětu, jejím cílem je obecně fotografický záznam a zisk dat z něj.

V této práci nás ale zajímá pouze část, která se snaží rekonstruovat 3D model z fotografií. Proces rekonstrukce by se dal rozdělit na několik fází, kterým se budeme dále věnovat. Jedná se o zisk parametrů nutných pro výpočet, výpočet polohy kamer, výpočet polohy vyznačených bodů a případná finální manipulace s body.

1.1 Fotogrammetrie

Jak bylo zmíněno výše, fotogrammetrie se nezabývá pouze jednotlivými objekty, mezi obory zájmu patří například také mapy, architektura či balistika. Hlavním rozdílem od ostatních měřičských metod je fakt, že se nepracuje přímo s objektem, ale pouze s daty pořízenými z něho. Není tedy vyžadována fyzická přítomnost objektu během sběru a získávání informací. Stačí vyslat někoho, aby pořídil dostatečně velký počet fotografií a přinesl je zpět. Veškeré zpracování a výpočty pak mohou probíhat libovolně dlouho a prakticky kdekoliv.

V principu lze fotografie pořizovat dvěma způsoby. První je umístit kamery na předem daná místa a předmět umístit na pozici, kam jsou kamery namířeny. Poté stačí jedna či dvě fotografie z každého fotoaparátu a proces získání fotografických dat je hotov. Tato metoda má ale spoustu omezení. Hlavní je nutnost umístit fotoaparáty přesně na předem vypočtené či jinak určené místo, protože by jinak došlo k velkému zkreslení výsledného modelu. Naopak samotný výpočet modelu je již triviální, protože známe pozice i nastavení fotoaparátů. S řešeními rekonstrukce modelu postavenými na tomto principu se lze dnes občas ještě setkat, hlavně protože nabízejí bezkonkurenční přesnost rekonstrukce. Velkou nevýhodou je ale cena, protože je nutno požádat specializovanou firmu disponující fotící komorou, ve které je umístěno několik fotoaparátů, ze kterých se následně fotografie přenesou do specializovaného softwaru, který model dopočítá. Dalším omezením je maximální možná velikost objektu,

protože fotící komora má určitou velikost a bývá nerozebíratelná, aby nedocházelo k nepřesnostem vlivem špatného smontování komory.

Druhá metoda nemá omezenou polohu fotoaparátů. Proces focení je tedy vesměs jednoduchý, problém nastává při počítačovém zpracování a výpočtu modelu. Splněním některých podmínek lze ale dosáhnout zjednodušení výpočtu. Jmenoval bych například požadavek na malý pohyb kamery (fotoaparátu), což sice zvyšuje počet fotek, je ale možno rekonstruovat pohyb kamery dle podobností mezi fotografiemi. Do této kategorie lze zahrnout vlastně jakékoliv fotografování nějakého objektu ať už člověkem či například satelitem. Údaje o poloze kamer a jejich natočení, stejně jako umístění objektu ve scéně, je zjišťováno až ve vlastním algoritmu.

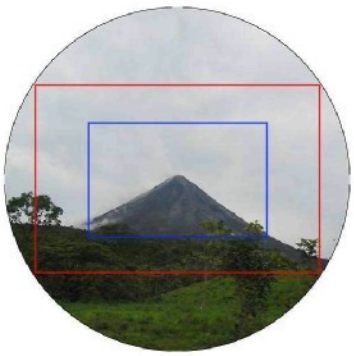
V této práci je využito jisté zjednodušení, a to že je vyžadována přítomnost alespoň jednoho obdélníku v modelu. Díky přítomnosti pravých úhlů lze vcelku přesně spočítat polohu fotoaparátů. Požadavek na obdélník se může zdát jako problém, v reálném světě se ale obdélník nachází téměř na každém objektu, není také problém obdélník domalovat či nějak uměle dotvořit před vlastním focením. Sice se bude muset vyřešit vynechání tohoto obdélníku ve finálním modelu, to je ale v porovnání se zjednodušením výpočtu zanedbatelná překážka.

1.2 Zisk parametrů

Po zachycení předmětu na dostatečný počet fotek je uživatel požádán, aby vyznačil na fotografiích body, které chce vidět ve výsledném modelu, propojil je a spojil do polygonů. Proces vyznačování a propojování bude podrobně probrán v kapitole „2. Implementace v jazyku Delphi" na příslušných stránkách. Tímto končí veškeré povinnosti uživatele a může spustit výpočet 3D modelu.

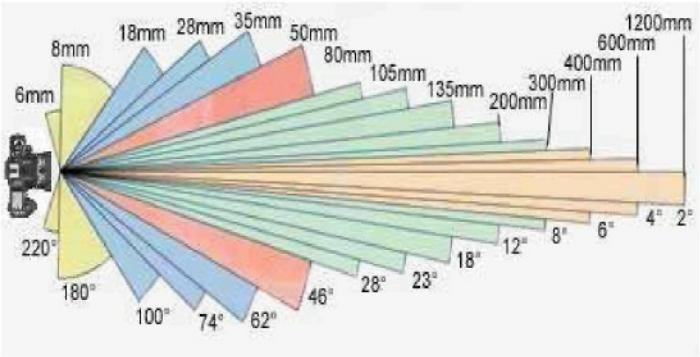
Než se ale bude moci přistoupit k vlastnímu výpočtu, je nutné získaná data sjednotit. Výsledný vzhled fotografie ovlivňuje mnoho parametrů. Vzhledem k asistenci uživatele nás nezajímají přírodní podmínky (např. osvětlení), protože si uživatel sám řekne, co považuje za důležitá data. Významná jsou tedy pro nás data o fotoaparátu a jeho vnitřním nastavení. Nejsou brány v potaz úplně všechny vlastnosti, protože většina z nich je pro tento výpočet nevýznamná, či přímo zanedbatelná.

Prvním parametrem je rozlišení obrázku. Tento parametr nám umožňuje převedení rozměrů zachycených na fotografii na jednotné rozměry. Nejedná se o nezbytně nutný parametr, hodí se ale, pokud byly fotografie pořízeny v různých rozlišeních (např. bylo potřeba doplnit další fotografie kvůli přesnosti apod.) a my chceme uživateli ušetřit práci se změnou velikosti fotografií. Druhým parametrem je ohnisková vzdálenost, s jakou byla fotografie pořízena. Zjednodušeně si lze tuto hodnotu představit jako vzdálenost promítací roviny od středu promítání. Velikost promítací roviny je dalším parametrem, který musíme získat. Původně se jednalo o obecně danou hodnotu (36x24 mm – š x v) z dob klasických fotoaparátů, kdy se jednalo o standardní velikost kinofilmového políčka. CCD snímač by měl být ideálně stejně veliký, aby co nejvíce napodobil chování klasického fotoaparátu, bohužel se ale výrobci snaží snižovat ceny fotoaparátů, takže občas sáhnou po zmenšení plochy CCD snímače. Velikost změny plochy snímače a tím pádem i promítací roviny se nazývá „Crop faktor". Jedná se o číslo v rozsahu <1, ∞>, které říká, kolikrát je snímač menší než kinofilmové políčko. Zmenšení snímače kromě snížení kvality znamená také změnu reálné ohniskové vzdálenosti vůči hodnotám napsaným na objektivu. Pro univerzálnost jsou všechny ohniskové vzdálenosti převáděny na ohniskovou vzdálenost vztaženou na kinofilm. Přepočet se děje pomocí rovnice, přesné znění naleznete v části „2. Implementace…".



Obrázek 1 - Crop faktor

Na obrázku č. 1 (získáno z [8]) je vidět vliv „Crop faktoru“ na velikost focené plochy. Červený obdélník ohraničuje oblast, kterou zachytí běžný fotoaparát se snímačem v plné velikosti, modrý obdélník označuje oblast, kterou zachytí fotoaparát, jehož Crop faktor je roven 1,3 čili jeho snímač je 1,3 krát menší než standardní snímač při stejné hodnotě ohniskové vzdálenosti. Jak je vidět, dochází k celkem značnému zmenšení plochy.



Obrázek 2 - Ohnisková vzdálenost

Na obrázku č. 2 (získáno z [8]) je ukázán vliv velikosti ohniskové vzdálenosti na velikost horizontálního úhlu objektivu. Můžeme si všimnout, že změna úhlu není lineárně závislá na změně ohniskové vzdálenosti.

1.3 Hledání polohy kamer

Výpočet polohy kamer je, dalo by se říci, nejdůležitější část výpočetního algoritmu. Pokud neznáme polohu kamery (je tím myšlena poloha a směr fotoaparátu), nemůžeme spočítat nic. Polohy kamer jsou také vždy relativní, protože bez použití nějaké lokalizační technologie není možné zjistit absolutní polohy kamer. Veškeré rozměry jsou tedy vztahovány k virtuálnímu počátku, který může být umístěn kdekoliv v prostoru cílového objektu.

V této práci se budeme zabývat zpracováním objektů, na kterých je přítomen alespoň jeden čtyřúhelník, který má všechny vnitřní úhly pravé – čtverec či obdélník. Nemusí se ale nutně jednat o část cílového předmětu, můžete se jednat například o podložku či uměle domalovaný obdélník, pouze se velikost všech jeho vnitřních úhlů musí blížit 90°. Dále v této práci budeme takovéto čtyřúhelníky nazývat „kalibrační obdélník" (dále jen KO), protože bude sloužit pro kalibraci poloh kamer. Ideálně by se měl jeden a ten samý KO nacházet na každé fotografii, protože polohy kamer jsou počítány vůči němu. Je ale možné vyznačit více těchto obdélníků a algoritmus se pokusí dopočítat všechny body za pomoci co nejméně KO.

Při použití více KO ale dochází ke snížení přesnosti modelu, protože program spočítá několik nezávislých částí modelu, které se poté musí spojit dohromady. Navíc je nutné, aby jednotlivé části měli alespoň 3 společné body, aby bylo možno jednoznačně určit jejich vzájemnou polohu. Není ale nutné, aby měli všechny části mezi sebou společné 3 body, protože po spojení dvou částí se z nich stává jeden celek. Celkový počet společných bodů lze uměle zvýšit vyznačením bodů např. v polovině spojnice dvou bodů. Není ale zaručeno, že to je dostatečné řešení vzhledem k lineární závislosti těchto třech bodů a tím pádem možné nejednoznačnosti řešení při hledání vzájemné polohy, protože se vlastně jedná stále o objekt s jedním stupněm volnosti, i když je tvořen třemi body.

Nyní již k samotnému algoritmu výpočtu polohy kamery. Víme, že na fotografiích jsou vždy vidět minimálně 4 body tvořící KO. Tyto body potřebujeme převést na prostorové vektory. Při fotografování je využíváno perspektivní projekce. Za snímačem fotoaparátu je virtuální střed promítání, ze kterého skrz snímač prochází shluk polopřímek. Každá polopřímka je charakterizována dvěma úhly – vertikálním

a horizontálním. Jejich maximální velikosti jsou dány konstrukcí fotoaparátu, běžně se charakterizují pomocí ohniskové vzdálenosti v mm, kterou jsme získali v prvním kroku výpočtu. Jsme tedy schopni pomocí jednoduché rovnice dopočítat maximální velikosti vertikálního a horizontálního úhlu. Tyto hodnoty ještě přepočítáme na šířku / délku promítací plochy pro ulehčení převodu bodů na vektory. Souřadnice bodů KO máme zatím uloženy jako vzdálenost od levého horního rohu obrázku. My je ale kvůli středovému promítání potřebujeme přepočítat na vzdálenosti od středu obrázku. Kromě tohoto také vzdálenosti vydělíme rozlišením obrázku, takže získáme hodnoty v rozsahu <-0.5, 0.5> pro obě souřadnice. Nyní již stačí tyto hodnoty vynásobit s šířkou / délkou a máme souřadnice X / Y námi hledaného vektoru. Hodnotu Z volíme 1, ta nehraje příliš velkou roli při dalším výpočtu, pouze uvádí vektory do prostoru. Musí se ale zachovat stejná hodnota Z pro všechny vektory, aby nedocházelo ke zkreslení. Máme tedy připraveny 4 prostorové vektory charakterizující polohu kamery vůči KO.

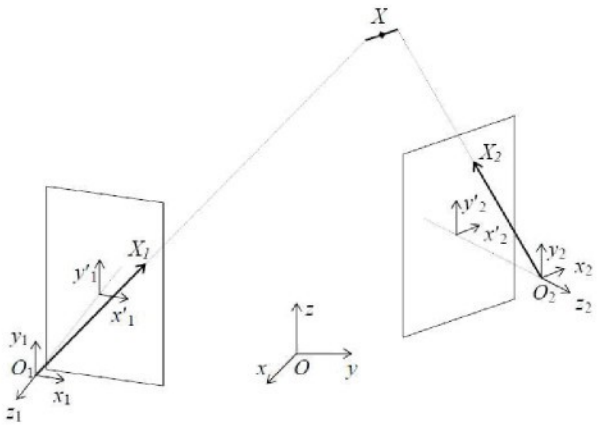
Dalším krokem je zjištění polohy a natočení promítací plochy vůči KO. Díky tomu, že víme, že KO je obdélník, můžeme toto hledání velice zjednodušit. V podstatě hledáme rovinu, na které vznikne obdélník po spočítání průsečíků vektorů KO s touto rovinou. Vlastní výpočet až tolik jednoduchý není, podrobně se mu budeme věnovat v kapitole „2. Implementace v jazyku Delphi." Po zjištění polohy této roviny si vezmeme souřadnice průsečíků a řekneme, že se jedná o přesný obdélník. Stanovíme si tedy počáteční bod obdélníku (nutno dodržet stejný pro všechny kamery počítané vůči jednomu KO) a tento bod určíme jako polohu kamery. Natočení kamery charakterizují úhly, ze kterých byl tvořen vektor hledané roviny a úhel, o který je natočen výsledný obdélník po promítnutí do roviny XY. Tyto úhly říkají, jak je kamera otočena okolo os x, y (úhly z vektoru roviny) a z (otočení obdélníku) vůči počátku.

1.4 Výpočet vlastních bodů

V tomto momentě známe polohy a orientace kamer vůči danému KO. Stačí již tedy pouze dopočítat body, které kamery vidí. Aby bylo možno spočítat polohu bodu v prostoru, musí být splněna podmínka, že je bod viděn minimálně dvěma kamerami. Samozřejmě čím vícekrát je bod zachycen na fotografiích, tím přesněji budeme moci spočítat jeho polohu.

Proces získání finální polohy bodu v prostoru je přirovnatelný k principu funkce našeho vidění prostoru. Oči představují dvě kamery s předem známou polohou a orientací. Po zaměření na konkrétní objekt získáme dva mírně odlišné obrazy. Takto získáme dvě informace o poloze objektu vůči našim očím. Díky známé poloze a orientaci očí nyní stačí vést dvě polopřímky vedoucí z našich očí a jdoucích zjištěnými směry. Tyto polopřímky se po nějaké době protnou a náš mozek je schopen tuto vzdálenost vnímat. Pro jednoduchost ale vnímáme pouze jednu vzdálenost, protože jednak jsou obě vzdálenosti velice podobné, navíc nás zajímá pouze relativní poloha objektu vůči naší osobě, nezajímá nás naprosto přesné umístění objektu.

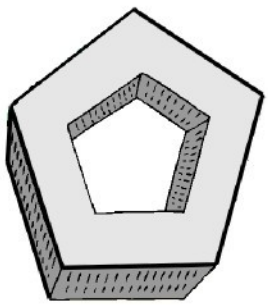
Výpočet realizovaný v počítači samozřejmě není tak přesný jako trénované lidské oko, navíc je informace o scéně deformována fotoaparátem (lineárnost čočky, kvalita snímače apod.), velkou roli zde samozřejmě hraje lidský faktor, protože body je nutno ručně vyznačit. Pro průběh samotného výpočtu to pouze znamená, že se nebude počítat průnik dvou přímek, spíše se bude hledat střední bod příčky mimoběžných přímek. Vzniká tak jistá nepřesnost polohy bodu, dá se ale korigovat zachycením bodu na co možná největším počtu fotografií.



Obrázek 3 - Rekonstrukce polohy bodu v prostoru

1.5 Finální manipulace s body

Pokud se na objektu nachází jen jeden KO, je výpočet hotov. Provede se pouze vystředění modelu, aby uživatel již nemusel s modelem dále hýbat, a může ho tedy rovnou zobrazit či uložit. Problém nastává, když je nutno použít více než jeden KO. Výpočet kamer a bodů proběhne stejně jako v případě jednoho KO, body se ale musí poté pospojovat. Důvodem je, že se polohy a orientace kamer počítají relativně vůči KO, nejsme tedy nějak jednoduše schopni zjistit vzájemnou polohu bodů, které byly spočítány při použití různých KO.



Obrázek 4 – Nákres focené budovy

Pro úspěšné spojení dvou částí modelu je proto potřeba splnit jednu podmínku, a to, že obě části musejí mít společné alespoň 3 body. Pro lepší představu společných bodů bych uvedl příklad. Chtěli bychom získat jednoduchý model budovy pentagonálního tvaru. Pro zjednodušení si budovu převedeme na obyčejný pětiúhelník (viz obrázek č. 1). Zanedbejme jakékoliv problémy s umístěním kamery do prostoru. Jako první je nutné zvolit KO. Střechu zvolit nemůžeme, protože se na ní nenachází žádný přirozený obdélník, jehož tvar by byl dobře lokalizovatelný ze všech úhlů pohledu. Volba tedy padá na stěny budovy. Ať se ale budeme snažit jakkoliv, nebudeme schopni zachytit společně námi zvolený KO společně s protilehlým rohem budovy (opět mějme na paměti, že jde o příklad, možná by se našel nějaký způsob, ale realizace takové fotografie by byla přinejmenším obtížná). Jsme tedy nuceni zvolit další KO, nejlépe jednu ze stěn ležících naproti prvnímu KO. Proto musíme zajistit alespoň 3 společné body. Nyní přichází na řadu střecha, protože tu jsme schopni zachytit společně s jakoukoliv stěnou. Splnili jsme tedy podmínku společných bodů, dokonce jsme poskytli algoritmu bodů 10.

Nutnost 3 bodů není svévolná. V prostoru má těleso šest stupňů volnosti – posuny podél všech os (x, y a z) a otočení okolo těchto os. Pro jednoznačnou lokalizaci tělesa v prostoru musíme zajistit, aby zadané parametry odstranily tyto stupně volnosti. Příkladem může být například krychle. Pokud bychom znali jeden společný bod, odstranili bychom první 3 stupně – znemožnili bychom posunutí počátku krychle. Krychle se ale může stále libovolně natáčet a přitom bude splněna podmínka pozice počátku. Potřebujeme tedy další body, abychom mohli jednoznačně určit polohu krychle v prostoru. Určíme si tedy další společný bod, nejlépe jeden z těch, které mají společnou hranu s prvním společným bodem. Tento stav by se dal představit jako krychle postavená na desku stolu tak, že první společný bod leží na desce stolu a druhý společný bod se nachází nad prvním společným bodem. Samozřejmě první společný bod musí setrvávat na jednom pevně daném místě. Krychle se tedy nemůže posunovat a ani se nemůže natáčet, stále se ale může otáčet kolem osy tvořené dvěma společnými body. Musíme tedy zvolit třetí společný bod (jeho poloha na krychli není příliš významná, pouze se nesmí krýt s jedním ze společných bodů či se nacházet na spojnici společných bodů), abychom této rotaci zamezili a určili tak polohu přesně. Ve finále tedy potřebujeme znát tři body, abychom mohli přesně rekonstruovat pozici krychle v prostoru. Běžnější je ale charakterizovat polohu objektu v prostoru pomocí třech úhlů a bodu pouze jednoho. Bod charakterizuje počátek, úhly pak natočení objektu podél všech třech os.

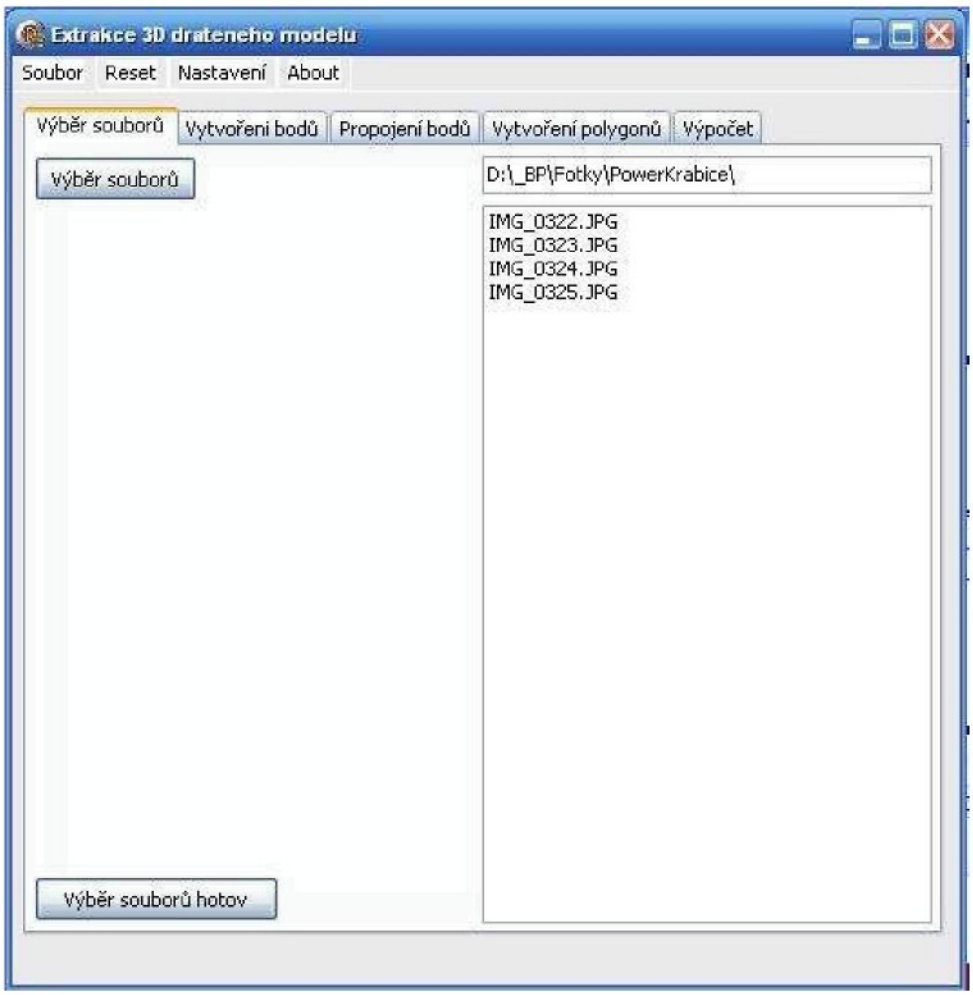
2. Implementace v jazyku Delphi

Volba programovacího jazyka nebyla v tomto případě příliš důležitá. Požadavky byly rozumná rychlost matematických výpočtů, což dnes splňuje snad každý programovací jazyk. Dále bylo zapotřebí vytvořit grafické prostředí programu (GUI), musel tedy být vybrán jazyk, pro který je dostupný vizuální editor prostředí, aby se nestrávilo příliš mnoho času ručním psaním GUI. Mělo by se také jednat o běžně používaný jazyk, aby bylo možné později program rozšiřovat a upravovat aniž by bylo nutné trávit hodiny, případně dny pouhým pochopením záměru autora kódu. Tyto požadavky dnes ale splňuje většina běžně používaných jazyků (např. Delphi, Java, C++ či C#). Volba padla na Delphi, hlavní důvodem byla největší zkušenost s tímto jazykem v porovnání s ostatními zmíněnými. Implementace v ostatních jazycích by měla být celkem bezproblémová, maximálně by se trochu změnili postupy používané pro práci s body v GUI. Samotný výpočetní algoritmus by mohl být zachován, nahradili by se nejspíše pouze názvy některých systémových funkcí a samozřejmě by se pozměnila syntaxe.

V následujícím textu bude použito tučného písma či kurzívy pro odkazy na zdrojový kód aplikace. *Kurzíva* bude použita pro názvy tříd, záznamů či datových typů. Pokud nebude uvedeno jinak, definice tříd se nacházejí v souboru „uData.pas." **Tučné** písmo bude odkazovat buď na prvky GUI, nebo na konkrétní proměnné použité během výpočtu.

2.1 Grafické rozhraní

Prvním krokem k úspěšnému vytvoření 3D grafického modelu objektu je určení bodů, které budou model tvořit. Tato práce toto nechává na uživateli, i když lze nalézt algoritmy schopné robustní extrakce klíčových bodů na obrázku. Uživatel by ale stejně musel alespoň označit pravoúhlé čtyřúhelníky, případně vybrat pouze ty body, které jsou součástí samotného modelu, ne např. okolí. Proces vyznačování má celkem 3 kroky a to vyznačení bodů na fotografiích, propojení bodů mezi fotografiemi a nakonec vyznačení polygonů. Jednotlivé kroky budou podrobně popsány níže, zmíněna bude také datová organizace projektu a některé významnější prvky GUI.



Obrázek 5 - Výběr souborů projektu

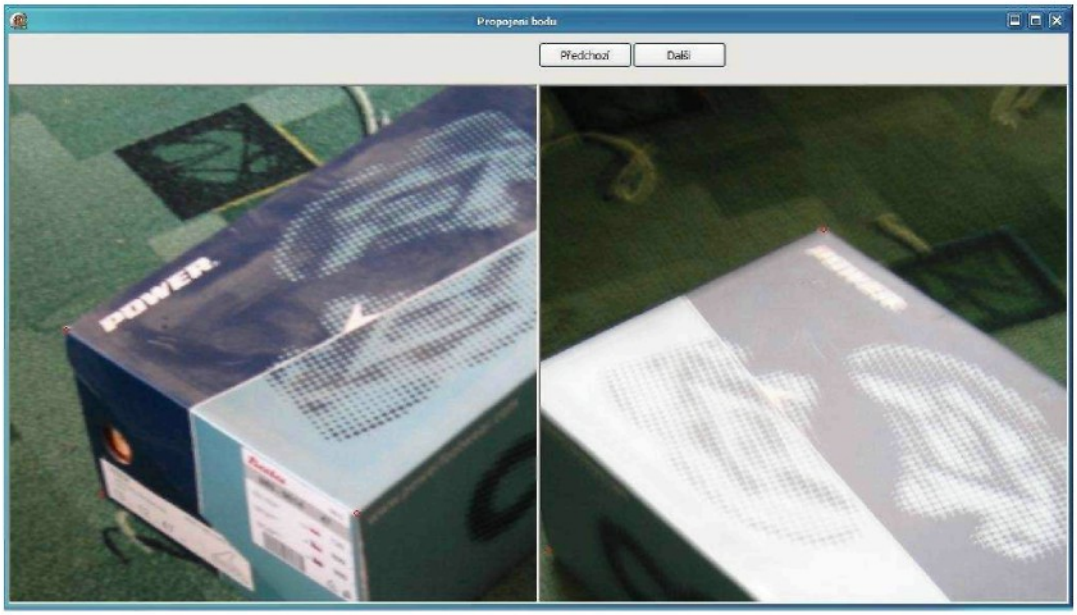
2.1.1 Vyznačení bodů

Obrázek 6 - Okno pro vyznačování bodů

Vzhledem k zaměření algoritmu na jednotlivé objekty, ne celé scény bylo vyznačování bodů přenecháno na uživateli. Odpadá tedy nutnost detekce bodů v obraze dle nějakého algoritmu, který by musel být dostatečně robustní, aby byl schopen ten samý bod označit i na jiných fotografiích bez ohledu na změnu polohy či natočení kamery, případně i po mírné změně osvětlení. Naopak vzniká problém s přesností. Fotografie jsou uživateli zobrazovány v plném rozlišení, ve kterém byly pořízeny, což případnou chybu lidského faktoru zmenší, neodstraní ji ale úplně. Další korekcí může být použití většího počtu fotek, protože se výsledné prostorové souřadnice průměrují, čímž se mohou menší nepřesnosti vzniklé během vyznačování odstranit.

Vlastní označování je realizováno jako označování polohy chtěným bodů obrázku, který si uživatel vybere. S označenými body je možno hýbat, případně je mazat. Jednotlivé ovládací prvky tohoto kroku a i dalších jsou popsány v programu v sekci ***About - Help*** dostupné přes hlavní menu programu.

2.1.2 Propojení bodů

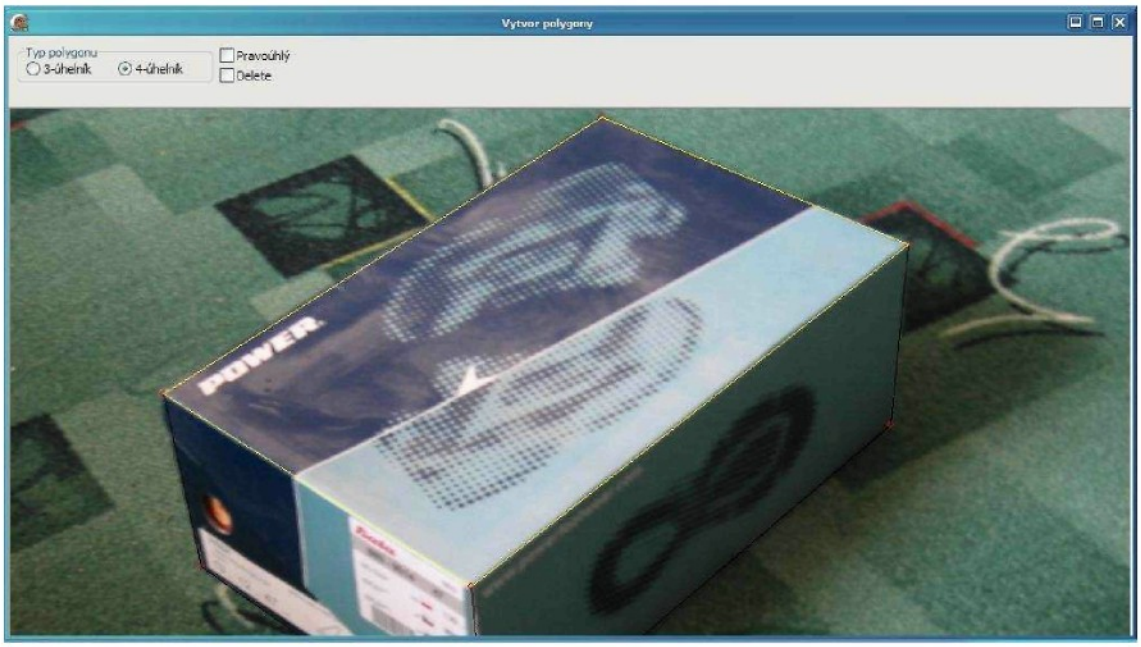


Obrázek 7 - Okno propojování bodů

Pouhé označení bodů však nestačí. Musíme ještě zjistit vzájemné korespondence bodů mezi fotografiemi (který bod je ten samý na druhé fotografii). Tato fáze může být pro uživatele trochu nepřehledná, je velice jednoduché některý bod zapomenout či zvolit nějaký jiný. Pro uložení korespondencí bodů je použit číselný index. Propojovací okno zobrazuje dvě fotografie (viz obr č. 4). Levá je dalo by se říci statická, ukazuje jednu fotografii, na které vybíráme zdrojový bod. Na pravé fotografii se pak označí korespondující bod, případně se pomocí tlačítka „**Další**“ přejde na další fotografii. Po vyznačení korespondence se automaticky přechází na další fotografii.

Ideálně by si uživatel měl vlevo vybrat jeden bod a vyznačit všechny jeho další výskyty na fotografiích zobrazovaných vpravo. Poté zvolit další bod a celý proces opakovat. Takto projde všechny body na jedné fotografii a může si vybrat další statickou fotografii. Zde již není ale nutné proces vyznačování opakovat pro všechny body, stačí to udělat pouze pro ty, kterým ještě nebylo přiřazeno ID. Aby si uživatel nemusel pamatovat, kam už klikal a kam ještě ne, zobrazuje se přiřazené ID jako nápověda po najetí kurzorem na bod.

2.1.3 Vyznačení polygonů



Obrázek 8 - Okno pro vyznačení polygonů

Toto je poslední krok výpočtu, který musí uživatel udělat. Z části se jedná pouze o formalitu – popis přesného tvaru modelu. Mimo to je to ale krok nezbytný pro úspěšný výpočet polohy kamer a tím pádem i polohy bodů. Uživatel má k dispozici vyznačení troj- či čtyř- úhelníku, přičemž u čtyřúhelníku je možnost ho označit s příznakem „**pravoúhlý**“. Hlavním tvarem jsou pro nás pravoúhlé čtyřúhelníky, protože se používají při výpočtu polohy kamer.

Jako v ostatních částech, i zde je možno polygony mazat. V nastavení je také možnost měnit barvy čar bodů, to je ale pouze formalita pro zlepšení viditelnosti prvků, aby uživatel na nějaký nezapomněl.

2.1.4 Parametry výpočtu

Před zahájením výpočtu je uživateli umožněno nastavit několik věcí. Může si vybrat metodu řešení některých částí algoritmu, což se ale týká pouze modelů, které používají více než jeden KO. Ovlivnit lze sjednocení velikostí jednotlivých částí a také způsob spojení těchto částí. Detailněji budou jednotlivé postupy popsány v sekci „2.2.4 Rotace bodů“. Pokud si uživatel přeje vynechat některé polygony z výsledného modelu, je zde možnost zapsat ID bodů tvořících daný polygon (y). Dále je tu možnost zobrazení statistiky modelu, jako je počet bodů dle ID, počet výskytu polygonů na fotografiích apod. S tímto souvisí také možnost korekce ID bodů, která má za úkol

odstranit neplatná ID, jež mohla vzniknout smazáním ID u nějakého bodu. Pokud nějaká odstraní, tak na ně samozřejmě také upozorní. Je zde i několik voleb týkajících se akcí po vypočtení modelu. Je možné zvolit rozlišení vnitřního prohlížeče, či zvolit formát souboru pro export výsledného modelu. Interní prohlížeč i formáty exportu budou zmíněny v sekci „2.3 Zobrazení modelu".

2.1.5 Datová struktura programu

Pro tuto část je vhodné mít otevřený zdrojový kód práce, konkrétně soubor „uData.pas." Tento soubor obsahuje definice téměř všech datových typů používaných při výpočtu. Výjimku tvoří pouze datové typy použité při ukládání / načítání projektu a při výpočtech řešených pomocí genetického algoritmu.

Velice důležitá třída je *TVector*. Jedná se o implementaci vektoru v prostoru s homogeními souřadnicemi. Kromě souřadnic $[x,y,z,w]^T$ má také identifikátor ID pro přiřazení do polygonu a booleovskou hodnotu done, používanou jako ukazatel úspěšného spočítání polohy bodu. Obsahuje celkem tři metody pro vytvoření vektoru. První možnost je přímé zadání souřadnic, dále zadáním dvou úhlů, což vytvoří jednotkový vektor dle jeho polárních souřadnic. Poslední volbou je vytvoření kopie jiného vektoru. Ostatní metody není třeba moc popisovat, jedná se o běžné operace s vektorem – posun, vynásobení / vydělení reálným číslem, vynásobení maticí, normalizace vektoru na jednotkovou délku, přičtení / odečtení vektoru a nakonec posun na dané souřadnice. Vzhledem k tomu, že se bude mnohokrát provádět stejná sekvence operací s větším množstvím vektorů, je výhodné využít matice, které mohou značně urychlit práci.

Datový typ *TMatrix* je pouze pole 4x4 obsahující koeficienty matice. Díky použití homogeních sořadnic je možné vytvořit matice nejen pro rotaci a změnu velikosti, ale také pro posun vektoru. Generování matic je pouze otázkou uložení čísel na správné pozice, případně užití funkcí cos a sin, takže nemá cenu tyto metody dále rozebírat. Je potřeba ale zmínit zavedení vektoru a matic z pohledu násobení. Výše již bylo naznačeno, že vektor je zaváděn jako sloupcový, čili se maticí vektor násobí „zleva" (**A** * x = v). Pokud bychom chtěli provést více operací pomocí násobení matic najednou, je možné nejdříve požadované matice mezi sebou vynásobit a vektor násobit

až výslednou maticí, je ale nutné dodržet správné pořadí matic (první se provede operace uložená v matici nejblíže vektoru).

Veškerá data projektu jsou uložena v proměnné ***data***, která je typu *TData*, což je datový záznam zastřešující data potřebná pro jednotlivé části výpočtu. ***Settings*** obsahuje grafická nastavení projektu společně s jeho názvem a cestou. Uchovává také údaj ***maxID*** o počtu přiřazených ID, ukazatel ***status*** jak daleko je uživatel s projektem a pole ***skippedPolygons*** držící indexy polygonů, které budou při zobrazení / exportu modelu vynechány. Další část dat je položka ***imgData***. Do té se ukládají údaje o jednotlivých fotografiích použitých v projektu. Kromě jména obsahuje informace získané z EXIF hlavičky souboru, konkrétně se jedná se šířku a výšku fotografie a ohniskovou vzdálenost, se kterou byla fotografie pořízena. Kromě těchto údajů se zde ukládají informace o bodech, které uživatel vyznačil, případně propojil s ostatními.

Část ***polygon***, jak již název napovídá, uchovává data o polygonech, která uživatel vyznačil. Kromě indexů bodů tvořících polygon se ukládá také údaj o pravoúhlosti. Každý polygon má ještě dvě booleovské položky ***done*** a ***used***, které jsou použity při výpočtu a budou zmíněny v kapitole „2.2.1 Výběr kalibračního obdélníku". Poslední část dat tvoří pole ***final***, které po dokončení výpočtu obsahuje souřadnice bodů modelu. Vzhledem k tomu, že výpočet nemusí úspěšně proběhnout, obsahují body hodnotu ***done***, která říká, jestli byl bod úspěšně spočítán či nikoliv. Toto pole je po dokončení výpočtu bráno jako zdroj dat pro export nebo pro zobrazení v interním prohlížeči.

Vzhledem k tomu, že proces vyznačování bodů a jejich propojování může být časově náročný, je možno projekt ukládat. K dispozici jsou dva formáty uložení, textový a binární. Textový je výhodný, je možno ručně upravovat data projektu, takže pokud uživatel ví o nějaké chybě a neví si s ní rady v GUI, může projekt uložit a chybu napravit v uloženém souboru. Někomu ale nemusí z nějakých důvodů textový formát vyhovovat, takže je k dispozici i binární zápis dat projektu. Pořadí zápisu dat do jednotlivých formátů je vidět ve zdrojovém kódu, v souboru „uFiles.pas," není ale moc podstatné, takže zde nebude dále popisováno.

2.2 Výpočet modelu

Nyní uživatel vyznačil všechny body, propojil je mezi sebou, vyznačil všechny požadované polygony a nastavil parametry výpočtu. Po spuštění výpočtu proběhne získání parametrů fotek z EXIF hlavičky souboru. Je použit externí program jHead [7], který parametry z hlavičky uloží do textového souboru. Počet parametrů v EXIF hlavičce se u každého výrobce liší, obecně tam ale nalezneme specifikace fotoaparátu a hlavně vnitřní nastavení fotoaparátu v době pořizování fotografie. My si tedy ze souboru načteme pouze ty parametry, které nás zajímají – rozlišení fotografie, ohnisková vzdálenost a velikost CCD čipu. Následně proběhne inicializace některých proměnných potřebných pro výpočet, jsou také smazány veškeré výsledky předchozího výpočtu. Po inicializaci může začít výpočetní smyčka. Ta je opakována, dokud nejsou dopočítány všechny body nebo již nelze žádné nové body spočítat.

2.2.1 Výběr kalibračního obdélníku

Tento krok není nutné provádět vždy, je potřeba pouze při použití více KO. Implementace výběru není příliš složitá, nepředpokládá se příliš velký počet obdélníků. Kritérium je jediné a to počet nově spočítatelných bodů. Pro každý KO se projdou obrázky, na kterých je vidět, a zaznamenávají se ID bodů, které ještě nejsou spočítány. Po projití všech fotek se spočte počet bodů, které jsou vidět alespoň 2x s daným KO. Toto číslo již říká, kolik nových bodů se dá teoreticky spočítat. Stačí tedy najít nejvyšší číslo a tak vybrat nejvhodnějšího kandidáta. Z výběru jsou samozřejmě vynechány KO, pro které byly body již úspěšně dopočítány. Může se ale stát, že se nějaký KO použije, jen se nepodaří dopočítat nové body, např. kvůli nedostatečnému počtu společných bodů nutných pro spojení částí. Tento KO je označen jako použitý a není dále ve výpočtu používán (obdobně jako KO, pro který se body dopočítaly). Může se ale stát, že po dopočítání dalších bodů se již najde dostatečný počet společných bodů, a proto by bylo možno použít zmiňovaný KO. Takovému KO se nastavuje příznak *Used* (ne *Done*), který je po dopočítání nových bodů resetován.

2.2.2 Výpočet polohy a orientace kamer

Po výběru správného KO můžeme postoupit k výpočtu polohy kamer. Příprava bodů pro výpočet byla zmíněna v teoretickém rozboru, přejděme tedy rovnou k vlastnímu algoritmu. Plocha snímače fotoaparátu se dá charakterizovat pomocí roviny, na které jsou umístěny čtyři body označující kraje snímače. Mezi těmito body se pak nachází obraz zachycený na našem snímku. My jsme z množiny bodů tvořících snímek vybrali čtyři body, které označují rohy našeho KO. Polohy těchto bodů byly z prostoru převedeny na polohu na ploše dle jejich umístění vůči směrovému vektoru fotoaparátu (vycházejícího ze středu snímače a jdoucím skrz objektiv) a také dle vzdálenosti od fotoaparátu. Známe rovnici této deformace, převedli jsme tedy tyto body na polopřímky jdoucí přesně daným směrem. Stále ale nevíme polohu ani natočení kamery.

V tuto chvíli využijeme zjednodušení zmíněného na počátku této práce, předpokládáme, že KO je pravoúhlý. Musí tedy existovat rovina, na které se po vypočtení průsečíků vektorů tvořících KO s touto rovinou, vytvoří pravoúhlý obdélník. Natočení této roviny a poloha průsečíků pak bude charakterizovat polohu a natočení kamery vůči KO. Vzhledem k tomu, že nelze ani přibližně určit hledanou rovinu, musí se použít algoritmus hledající řešení pomocí náhody. Jediným předpokladem ulehčujícím výpočet je, že se kamera nachází vždy nad daným KO, ne pod ním, což je v drtivé většině případů lehce splnitelný předpoklad (vnitřní strany ploch tělesa nebývají vidět). Hledanou rovinu budeme charakterizovat jejím normálovým vektorem, protože ho lze lehce generovat a plně postačuje pro výpočet průsečíku vektoru s rovinou. Normálový vektor bude generován pomocí dvou úhlů, charakterizujících ho ve sférických souřadnicích (délka vektoru je jedna). Výhodou při našem výpočtu je fakt, že vektory KO vycházejí z počátku, což zjednoduší výpočet průsečíku.

Výpočet průsečíku by se dal rozdělit na dvě části – v první se zjistí, jestli vůbec existuje, v druhé se případně průsečík dopočítá. Jediný případ, kdy neexistuje průsečík vektoru s rovinou v trojrozměrném prostoru, je, když je vektor s rovinou rovnoběžný. To se dá lehce ověřit pouhým skalárním součinem vektoru KO s normálovým vektorem roviny. Pokud vyjde nula, znamená to, že vektory jsou rovnoběžné a tím pádem je rovina s vektorem rovnoběžná, takže nemá smysl pokračovat ve výpočtu průsečíku ani

v ověřování kvality této roviny. V případě nenulového výsledku můžeme postoupit k výpočtu průsečíku. Hodnota získaná skalárním součinem je převrácenou hodnotou charakterizující vzdálenost roviny od počátku ve směru daném vektorem KO. Stačí tedy vydělit vektor KO daným číslem a máme souřadnice průsečíku vektoru KO s vygenerovanou rovinou. Tento postup zopakujeme pro všechny čtyři vektory a můžeme přistoupit k ověření kvality roviny.

Ukazatel kvality roviny je odchylka úhlů vektorů, tvořících čtyřúhelník vzniklý z průsečíků, od devadesáti stupňů. Vezmeme tedy jednotlivé body a odečtením od sebe ve správném pořadí (zadaném uživatelem při vytváření polygonu) získáme vektory charakterizující tvar čtyřúhelníku. Úhel mezi dvěma vektory se spočítá arccosinem jejich skalárního součinu. Hledáme odchylku od devadesáti stupňů, odečteme tedy od této hodnoty číslo rovno polovině π (úhly jsou interně počítány v radiánech) a vybereme největší hodnotu ze čtyř získaných. Tu poté porovnáme s hodnotou kvality prozatím nejlepšího řešení, a pokud bude menší, uložíme ji společně s úhly normálového vektoru a souřadnicemi vzniklého obdélníku pro další použití. Tento proces generování vektoru normály roviny a výpočtu kvality roviny je mnohokrát opakován (přesný počet a další popis viz část „3. Testování"), aby se nalezlo pokud možno co nejlepší řešení.

Po dosažení požadovaného počtu opakování či po dosažení dostatečné přesnosti už pouze zbývá převést získané polohy a úhly na vektor polohy kamery a na vektory natočení kamery. Natočení kolem os X a Y charakterizují úhly použité pro vygenerování normálového vektoru roviny, natočení podél osy Z pak natočení vzniklého obdélníku v ploše XY. Struktura matic charakterizující rotace podél os je vidět ve zdrojovém kódu, jedná se o matice rotace běžně používané např. v počítačové grafice. Matice posunutí charakterizující polohu kamery je pak vytvořena pomocí prvního bodu obdélníku.

2.2.3 Výpočet polohy bodů

Když známe polohy kamer, jsme již velice blízko k úspěšnému dopočítání polohy bodů v prostoru. Budeme ale potřebovat nejdříve převést body označené na fotografiích na polopřímky v prostoru a následně vybrat polopřímky se stejnými ID, abychom spočítali jejich průnik a tím také zjistili koncovou polohu bodu v prostoru. Polopřímky v prostoru budeme charakterizovat pomocí bodu v jejím počátku a směrovým vektorem. Počáteční bod polopřímek bude vždy tvořit souřadnice kamery pro příslušnou fotografii. Směrový vektor bude nutné dopočítat, použijeme k tomu vektory charakterizující orientaci kamery a polohu bodu na fotografii. Bod na fotografii převedeme pomocí postupu uvedeném v teoretickém rozboru na vektor orientovaný vůči středu se složkou Z rovnou jedné. Vezmeme jednotlivé složky tohoto vektoru a vynásobíme jimi odpovídající vektory charakterizující směr kamery (X složka vektoru krát X vektor kamery). Takto spočteme všechny polopřímky pro jedno ID a můžeme přistoupit k výpočtu jejich průsečíku.

Polopřímky budeme brát obecně jako mimoběžky, budeme tedy muset počítat střední bod jejich příčky. Příčka mimoběžek je úsečka kolmá na obě mimoběžky, jejíž vrcholy leží na obou křivkách. My těchto středů spočteme několik, protože budeme počítat průniky všech polopřímek mezi sebou, abychom zvýšili přesnost výpočtu. Výsledné body poté zprůměrujeme a získáme tak konečné řešení polohy bodu v prostoru. Přesný algoritmus řešení výpočtu průniku mimoběžek naleznete v metodě *Intersection_Skew*, zde bude zmíněn výpočet pouze jednoho z krajních bodů. Jako první vektorově vynásobíme směrové vektory obou polopřímek, čímž získáme normálový vektor roviny, ve které se nachází hledaný bod. Tento vektor dále vektorově vynásobíme s jedním ze směrových vektorů a získáme normálový vektor roviny kolmé na rovinu možných řešení. Nakonec spočteme průnik druhého směrového vektoru s právě dopočtenou rovinou a máme první krajní bod příčky. Toto opakujeme i pro druhý bod, pouze se prohodí použité vektory. Výpočet středu příčky je pak už jen otázkou přičtení poloviční vektoru tvořeného krajními body příčky k jednomu ze dvou bodů. Musíme ale dát pozor na orientaci tohoto poloviční vektoru.

2.2.4 Rotace bodů

Poslední krok algoritmu nemusí nutně proběhnout. První podmínkou spuštění této části je použití více KO při rekonstrukci modelu. Dále se musí nalézt dostatečné množství společných bodů mezi již spočítanými a nově dopočítanými body. Počet potřebných bodů vychází z nutnosti určení jednoznačné polohy množiny bodů v prostoru. Jak bylo zmíněno v teoretickém rozboru, tyto body jsou potřeba 3, aby se pokryli všechny body volnosti (3 posuny a 3 rotace). Důležitým předpokladem jsou ale také shodné vzdálenosti mezi všemi body jednotlivých modelů (hotové části a nově dopočítané části). Algoritmus výpočtu polohy kamer a bodů ale toto bohužel nezaručuje zcela přesně, čili je nutno ještě vyrovnat vzdálenosti mezi body než se bude moci přistoupit k samotnému umístění bodů do správné pozice.

Metody pro nastavení správných vzdáleností v modelu mají na začátku názvu slovo *Scale*... První je metoda průměrování. Porovnáme délky vektorů a ukládáme si koeficienty poměru délek již hotových a nových vektorů. Všechny je sečteme a následně vydělíme jejich počtem. Získáme tak koeficient, který říká kolikrát je vektor v nové množině bodů průměrně delší než jeho dříve spočítaná předloha (vektory předlohy jsou tvořeny ze společných bodů, proto jsme schopni určit poměr délek). Teď stačí pouze vydělit vektory získaným koeficientem. Společné vektory pak můžeme vydělit přímo jejich vlastním koeficientem, abychom zvýšili přesnost. Přesnost nastavení stejně jako rychlost jednotlivých variant nastavení velikostí bude probrána v části „3. Testování." Druhý způsob změny velikosti také nastavuje známé velikosti rovnou. Polohy dalších bodů jsou ale počítány vektorově. Před nastavením velikostí jsou uloženy vektory jdoucí ze společných bodů do bodů neznámých. Takto získáme celkem dobrý popis vnitřní geometrie předmětu. Po nastavení známých velikostí následuje rekonstrukce polohy dalších bodů. To provedeme jako hledání průsečíků polopřímek jdoucích z již správně umístěných společných bodů předem spočítaným směrem. K tomu použijeme metodu z výpočtu polohy bodů.

Poslední možnost nastavení správných velikostí má oproti předešlým metodám širší teoretický základ, protože se jedná o metodu obecně používanou v úlohách optimalizace. Metoda zde bude rozebrána podrobněji, protože se stejný princip používá u rotace bodů do správné polohy. Řeč je o genetických algoritmech (dále jen GA). Teorie GA je založena na teorii Charlese Darwina o vývoji druhu. Hlavními prvky

převzaté GA, je přežití silnějších jedinců, člen nové generace je potomkem dvou jedinců z generace předchozí a vlastnosti prvku jsou popsány řetězcem čísel či znaků s pevně daným významem (lze přirovnat ke chromozomům u organismů). V ideálním případě tak při vzniku nové generace dojde ke smíšení dvou kvalitních genetických informací a vznikne o něco silnější jedinec. Samozřejmě toto se nestane vždy, může naopak vzniknout velice slabý jedinec, ten ale do další generace nezasáhne, protože bude příliš slabý.

Proces vzniku nové generace by se dal rozdělit do třech kroků – reprodukce (někdy označována jako selekce), křížení a mutace. Pro potřeby reprodukce je nutné jasně definovat metodu, kterou určíme kvalitu řetězce. V našem případě se metoda jmenuje *CalculateQualityScale*. Jejím úkolem je provést ohodnocení kvality jednotlivých řetězců. Jako ukazatel kvality byla zvolena odchylka společných bodů - první z již hotových bodů, druhý z nově spočtených. To, jestli větší číslo znamená vyšší kvalitu, opět záleží na naší volbě, pro jednoduchost byla vybrána nula jako ideální kvalita a nekonečno jako nejhorší případ.

Reprodukce má za úkol odstranit slabé jedince, aby se umožnil vznik nových, silnějších jedinců. V zásadě se používají dva přístupu pro zvolení jedinců postupujících dále. První je turnajový způsob, který je v této práci také využíván (metoda má název *Selection*). Ze staré generace jsou náhodně vybírány vždy dva prvky a do nové generace postoupí ten kvalitnější. Druhý způsob je o něco zajímavější, je ale v porovnání s prvním způsobem výpočetně mnohem náročnější. Říká se mu systém rulety, protože každému prvku se přiřadí určitá pravděpodobnost, odvíjející se od jeho kvality, že bude zvolen, následně se náhodně vybere číslo v rozsahu nula až součet všech pravděpodobností a tím se určí postupující prvek.

Během křížení se generace rozdělí do párů, které si mezi sebou vymění genetické informace (určité části řetězců) a tím dají vzniknout dvěma novým jedincům. To, jakou část řetězce si mají mezi sebou rodiče vyměnit, je opět možno vybrat několika způsoby. Mezi nejjednodušší možnosti patří například prohození lichých či sudých prvků, či prohození první nebo druhé poloviny řetězce. My jsme zvolili prohození počátečních částí řetězce, délka je však vždy náhodně volena v rozsahu délky řetězce. Aby se zvýšila rozmanitost řetězců a zamezilo se postupné stagnaci řešení okolo jednoho bodu, určuje pravděpodobnost, s jakou bude křížení úspěšné. Pokud křížení

neuspěje, doje k pouhému nakopírování rodičů do nové generace. Implementace křížení je v metodě *Cross*. Posledním krokem je mutace. Ta by měla zvyšovat rozmanitost generace. Vychází z představy, že čas od času se narodí jedinec, který má něco navíc či je v něčem jiný. Pokud mu tato deformace poskytne výhodu, projde sítem reprodukce a obohatí generaci o nový prvek, pokud mu naopak uškodí, dostane špatné hodnocení a neprojde do další generace. Pravděpodobnost mutace je ale velice nízká (přibližně pět procent) v porovnání s pravděpodobností křížení (až šedesát procent), což zajišťuje občasné obohacení generace, zároveň předchází jejímu znehodnocení vlivem přílišného počtu nevhodných mutací.

Za zmínku stojí také implementace vlastního řetězce, kterým je jedinec charakterizován. Byl zvolen řetězec nul a jedniček, protože umožňuje velice jednoduchou práci během všech částí algoritmu. Změnu velikosti provádíme ve třech dimenzích, potřebujeme tedy do řetězce uložit tři čísla. Řešením je převést tato čísla do binárního kódu a seřadit je za sebou v pevně daném sledu. Výhodou genetických algoritmů je, že vůbec nepotřebují vědět, co za problém řeší, protože pro ně je důležitá pouze informace o kvalitě, nikoliv o konkrétním obsahu a významu řetězce. Čistě binární reprezentace čísel ale není také úplně nejvhodnější, protože změna o jeden dílek rozsahu vpravo či vlevo může v binárním kódu znamenat v nejhorším případě změnu všech bitů. Je tedy běžné pro potřeby GA používat Grayův kód, který ulehčí hledání nejpřesnějšího řešení.

Po dokončení nastavení správných rozměrů modelu můžeme přistoupit k vlastnímu spojení dvou částí dohromady. Jak bylo zmíněno v teoretické části, obecný objekt ve třech rozměrech má 6 stupňů volnosti. Potřebujeme tedy zjistit 6 čísel, abychom mohli části spojit. První tři čísla budou říkat, jak jsou části vůči sobě posunuty v osách x, y a z. Zvolíme tedy jeden společný bod, který budeme brát jako počátek modelu a vypočteme vektor spojující výchozí bod z hotové části a výchozí bod v nové části. Takto jsme zajistili, že se části překrývají minimálně v jednom bodě. Potřebujeme ale ještě zjistit vzájemné natočení obou částí v prostoru.

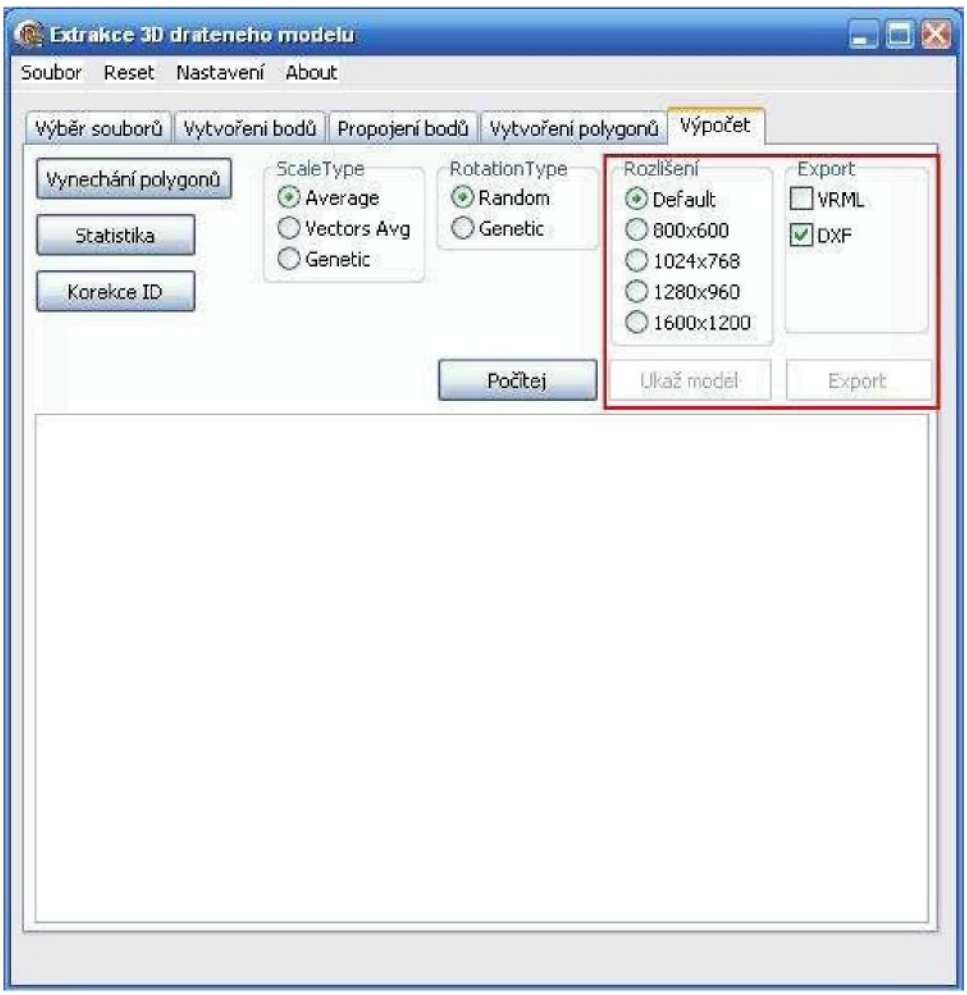
Problémem je, že naše dvě množiny bodů nejsou tvarově identické, není tedy možné přímo zjistit jejich natočení pouze řešením soustavy rovnic. Bylo tedy zvoleno řešení této úlohy pomocí náhody. Implementovány jsou dvě metody hledání úhlů – jedna je náhodné generování úhlů rotace okolo třech os, druhá je hledání úhlů pomocí

genetického algoritmu. Hledání úhlů pomocí náhody není úplně vhodné řešení, obzvláště pokud je vyžadována vysoká přesnost. Pokud bychom například chtěli přesnost půl stupně pro každou osu, dostaneme přes tři sta miliónů možných řešení, která by se musela prozkoumat. Zde nám pomůže předpoklad, že funkce reprezentující kvalitu narotování obou částí k sobě má maximum v jednom bodě (před chvíli bylo sice řečeno, že řešení může být více, ale všechna řešení jsou při zachování konečné přesnosti ekvivalentní a nám nevadí, že jsme našli jiné řešení, než jsme předpokládali). Zvolíme si tedy mezní hodnotu přesnosti narotování bodů, pod kterou když klesneme, bude to znamenat, že jsme našli oblast, ve které se nachází maximum funkce reprezentující kvalitu narotování a omezíme změnu úhlů na menší hodnotu, než bylo původních tři sta šedesát stupňů (díky jejich náhodnému generování). Takto se lze dopracovat ke kvalitnímu řešení v rozumně dlouhé době, bližší informace o době trvání hledání řešení a jeho kvalitě lze nalézt v části „3. Testování."

Hledání úhlů pomocí genetického algoritmu využívá stejné třídy pro výpočet jako úloha nastavení velikostí v modelu. Výpočet tedy probíhá stejně, pouze jsou změněny některé parametry výpočtu. Je zvolen vyšší počet generačních cyklů, aby se zvýšila teoretická úspěšnost algoritmu, bitová reprezentace čísla byla prodloužena o pět bitů, aby se dosáhlo lepšího pokrytí rozsahu hledaných čísel (nula až tři sta šedesát stupňů). Použita je také jiná metoda výpočtu kvality řetězce, opět ale menší číslo znamená vyšší kvalitu řetězce. Kvalita narotování je reprezentována jako podíl délek dvou vektorů. První vektor vede ze společného počátku do některého ze společných bodů v již hotové části modelu. Druhý vektor tvoří spojnici tohoto společného bodu se společným bodem v nově spočtené části modelu, která byla předtím narotována dle vygenerovaných úhlů. Čím je jejich podíl větší, tím je větší odchylka obou částí modelu a tím pádem je i samotné řešení méně kvalitní. Toto ověření se provádí pro všechny společné vektory, aby náhodou nedošlo k situaci, kdy jeden společný bod z obou částí modelu splývá, zatímco ostatní body jsou umístěny špatně. Stejného ověřování kvality je použito i u algoritmu využívajícího náhodu pro zjištění úhlů.

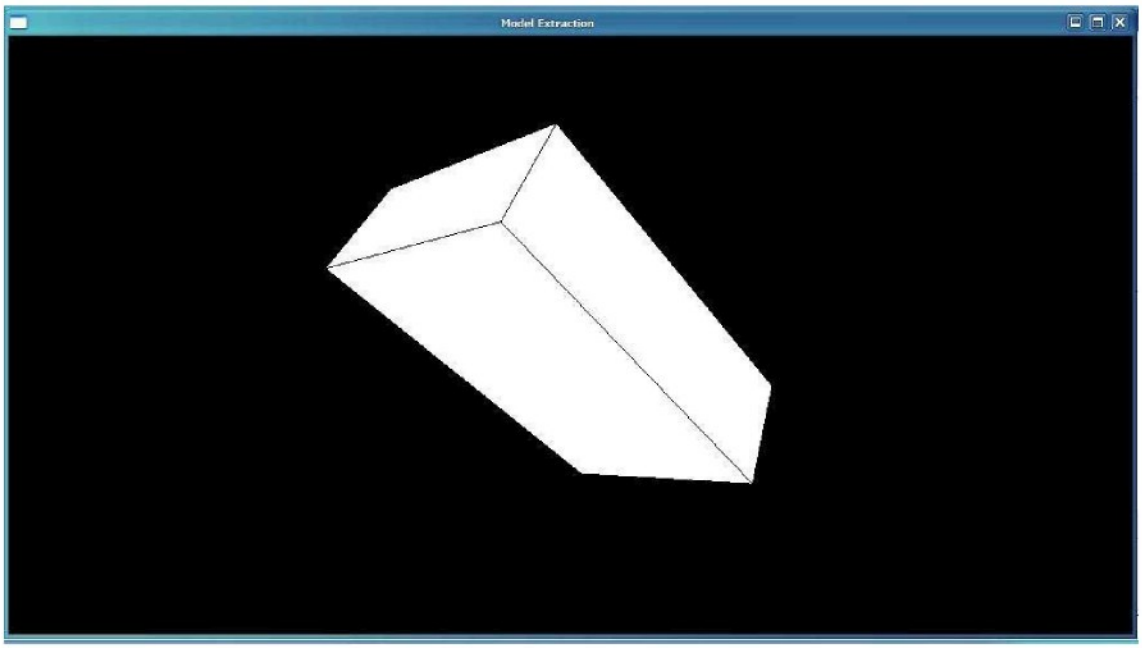
2.3 Zobrazení modelu

Za předpokladu, že během výpočtu nedošlo k nějaké chybě, máme nyní k dispozici množinu bodů v prostoru plus informace o přímkách mezi nimi. Jediným problémem nyní je, že uživatel data nevidí a nemá k nim ani přístup. Momentálně jedinou možností, jak zjistit souřadnice bodů, je uložit projekt v textové podobě a přečíst si jejich hodnoty. To ale není příliš pohodlné, navíc bez vložení těchto souřadnic do nějakého modelovacího programu je téměř nemožné si vypočtený tvar představit. K tomuto účelu bylo do programu přidáno několik možností, jak vypočtený model zobrazit. Pro rychlé ověření správnosti výpočtu byl vytvořen prohlížeč využívající grafickou knihovnu OpenGL. Pokud chce uživatel data používat v nějakém jiném programu, což je více než pravděpodobné, je k dispozici export buď do zdrojového souboru jazyka VRML či do CAD formátu DXF.



Obrázek 9 - Možnosti exportu modelu

2.3.1 OpenGL prohlížeč



Obrázek 10 - Okno interního prohlížeče modelu

OpenGL je standard definující API pro tvorbu počítačové grafiky, aniž by programátor musel vědět konkrétní typ grafického hardwaru, který bude použit pro vykreslení scény, či platformu, na které bude program běžet. API nabízí sadu cca 250 příkazů pro vykreslování 2D/3D objektů pomocí grafických primitiv (bod, přímka, trojúhelník či polygon). Umožňuje definovat barvy, implementuje i možnost interpolace barev. Dává k dispozici základní geometrické transformace v prostoru (posuny a rotace), umožňuje zobrazit scénu pomocí perspektivní či ortogonální projekce. Obsahuje také základní osvětlovací model, který má sice svá omezení, na druhou stranu zapnutí osvětlení nemá příliš veliké nároky na výpočetní výkon. Kromě míchání barev také umožňuje pokrytí ploch texturami, pro kreslení složitějších tvarů nabízí evaluátory bezpérových křivek či ploch. API nabízí spoustu dalších funkcí, navíc existuje nemálo nadstavbových knihoven zachovávajících multiplatformnost a nabízejících funkce pro ulehčení často používaných postupů.

Jako první věc před samotným kreslením našeho modelu je nutné vytvořit okno, ve kterém se bude vykreslování odehrávat a vhodně nastavit kontext zařízení. Při tvorbě okna nenarazíme na větší komplikace, musíme pouze nastavit, jak má okno vypadat, jeho velikost apod. Za zmínku stojí nastavení formátu pixelu pomocí „Pixel Format

Deskriptoru“ (dále jen PFD). Jedná se o strukturu, která říká, jak bude vypadat pixel na cílové kreslící ploše. Pro zájemce je možno najít kompletní popis této struktury na MSDN (Microsoft Developer Network) nebo ve velké většině tutoriálů pro OpenGL bez použití speciálních komponent, které toto nastavení dělají automaticky. PFD je celkem velká struktura umožňující spoustu nastavení týkající se zobrazení scény. První je položka *dwFlags*, pomocí které řekneme, že budeme kreslit do okna pomocí OpenGL a zapneme Double Buffering, což by mělo zamezit vzniku artefaktů na obrazovce či vykreslení nekompletní scény. Poté si vybereme RGBA formát barev (3 barvy a průhlednost) a barevnou hloubku.

Dalším krokem po nastavení PFD je zvolení vhodné projekce, v našem případě to je perspektivní. OpenGL obsahuje matici charakterizující vlastnosti projekce (např. poměr stran, viditelnost nebo úhel pohledu) a dává k dispozici funkce pro vložení vhodných čísel dle zvoleného typu projekce a jejích parametrů (v našem případě to je funkce *gluPerspective*()). Poslední akcí před kreslením je propojení renderovacího kontextu okna a kontextu grafické karty, aby se v okně zobrazovalo to, co karta spočte.

Po provedení těchto úkonů se program dostane do opakující se smyčky, která bude mít za úkol překreslovat scénu, případně reagovat na vstup z klávesnice či na pohyb myši. Překreslení scény by se dalo rozdělit na celkem čtyři části. V první se vymaže vše, co se nachází na obrazovce. OpenGL neumožňuje překreslování pouze části scény, které bylo populární před mnoha lety, protože bylo možno snížit výpočetní a paměťovou náročnost aplikace na úkor složitosti zobrazovacích rutin. Následuje nastavení pozice a natočení kamery pozorovatele. Pro charakterizaci polohy a natočení kamery a objektů ve scéně používá OpenGL „ModelView“ matici. Dle aktuálních hodnot v ní uložených se objekty umísťují do scény a také se dle ní nastavuje poloha a natočení kamery. K dispozici je několik funkcí aplikujících základní geometrické transformace na aktuální obsah matice – posun, změna velikosti a rotace okolo zadaného vektoru. Pokud tedy chceme nakreslit objekt na danou pozici, libovolně natočený či zmenšený/zvětšený, můžeme použít dostupné transformace a vykreslit objekt aniž bychom museli někde mimo vykreslovací smyčku přepočítávat polohu jednotlivých bodů modelu.

Důležité je také nezapomenout, že OpenGL je navrženo jako stavový automat, čili jakmile něco nějak nastavíme, zůstane to tak nastaveno do konce programu

či do doby než to změníme. Maticové transformace tedy nejsou absolutní, nýbrž offsetové. Samozřejmě se to netýká pouze manipulací s maticemi, dalším příkladem mohou být barvy. Jakmile provedeme instrukci nastavení barvy, všechny body budou mít danou barvu, dokud neprovedeme změnu barvy na jinou. Abychom tedy nakreslili body na správné místo, je nutné nahrát do ModelView matice matici jednotkovou, protože jinak bychom začali kreslit neznámo kde. Uživatel může s modelem otáčet, takže po resetu je nutné natočit kameru podle toho, jak uživatel s modelem otáčel.

Dostáváme se ke třetí části – kreslení vlastního modelu. To bylo rozděleno na dvě části – kreslení ploch a kresleni hran. Tento postup byl zvolen, aby byli patrny přesné tvary modelu, aniž bychom museli použít jakýkoliv osvětlovací model. Jakékoliv zadávání primitiv je nutno obalit příkazy *glBegin(typ primitiva) a glEnd()*, čímž řekneme grafickému adaptéru, že budou následovat data bodů. V našem případě budeme používat trojúhelníky a čtyřúhelníky, čili za názvy primitiv dosadíme *GL_TRIANGLES* nebo *GL_QUADS*. Mezi těmito dvěma příkazy se nachází sekvence příkazů glVertex3f(souřadnice_bodu). Mezi glBegin a glEnd je možno napsat libovolné množství souřadnic bodů, OpenGL vytvoří odpovídající počet primitiv dle celkového počtu zadaných bodů (pokud bylo zadáno šest bodů pro kreslení trojúhelníku, nakreslí se celkem dva celé trojúhelníky). Pokud by byl zadán počet bodů, který by nebyl beze zbytku dělitelný počtem bodů nutných pro nakreslení primitiva, nakreslila by se pouze ta, pro která jsou k dispozici všechny body, nevyužité body by byly zahozeny. Takto se postupně nakreslí všechny plochy objektu, následně se zvýrazní hrany nakreslením toho samého v módu drátěného modelu. Uživatel má nyní před sebou na obrazovce již kompletní obraz modelu pozorovaného z nějakého bodu v prostoru.

Musíme ale ještě provést kontrolu vstupních zařízení, aby si uživatel mohl model prohlédnout ze všech stran. Klávesnice i myš se obsluhují pomocí fronty zpráv aplikace. Zprávy jsou způsob komunikace mezi jednotlivými částmi operačního systému. Každá zpráva obsahuje identifikátor a parametr. Dle identifikátoru jsou zprávy zachytávány správnými aplikacemi, které vykonají předem nastavenou činnost v závislosti na hodnotě parametru. Toto je pouze hrubé nastínění systému zpráv operačního systému Windows, další informace je možno nalézt na internetu. My si tedy

vybereme správné identifikátory zpráv, zajímat nás bude stlačení klávesnice či tlačítka myši a jejich puštění, dále otočení kolečkem a budeme také zachytávat zprávu o ukončení aplikace, abychom mohli uvolnit kontext grafického adaptéru. Akce prováděné po zachycení zprávy není nutné příliš popisovat, jedná se vždy o načtení hodnoty parametru a změny správné proměnné.

2.3.2 VRML

„Virtual Reality Modeling Language" je grafický formát určený primárně pro popis trojrozměrných scén. Kromě statické definice rozložení objektů umožňuje specifikovat animace či reakce na různé události. Jedná se tedy o kompletní platformu pro virtuální realitu. Samotné zdrojové soubory scény jsou textové, není tedy složité je editovat (postačí obyčejný poznámkový blok), spojení více modelů do jedné scény je pak záležitostí pouhého kopírování. Celá VRML scéna je popisována pomocí objektů, které jsou uspořádány ve stromové struktuře. Není tedy složité realizovat všemožné pohyby objektů ve scéně, ať už se jedná o např. celého člověka či pouze jeho ruku. Specifikace VRML nabízí veliké množství příkazů umožňující detailní popis celé scény, nám opět bude stačit malá část, abychom byli schopni popsat vypočtený model.

Popis jednotlivých částí souboru bude vztažen ke struktuře, kterou generuje náš program, je proto vhodné mít otevřený WRL soubor, který byl vygenerován z vypočteného modelu pro lepší přehled. Položka *WorldInfo* obsahuje název scény a také několik řádek informací. Grafickou podobu modelu zvolíme podobnou jako u OpenGL prohlížeče, což znamená, že plochy objektu budou bílé a hrany budou černě zvýrazněny pro lepší orientaci a abychom nemuseli řešit vhodné nastavení osvětlení. Uzel *Shape* ohraničuje definici jednoho předmětu ve scéně. Může obsahovat definice vlastností materiálu, posunutí či natočení a samozřejmě obsahuje body tvořící daný předmět společně s informacemi o jejich propojení. V jazyku VRML se tvary definují pomocí seznamu souřadnic vrcholů následovaného plochami, které jsou specifikovány pomocí indexu svých vrcholů.

Pro naši potřebu nám budou stačit uzly *IndexedLineSet* a *IndexedFaceSet*, které říkají, že se budou kreslit nevyplněné, respektive vyplněné, polygony. Ty mohou mít libovolný počet vrcholů, jednotlivé polygony se od sebe

oddělují pomocí indexu „-1." Měli bychom ještě nastavit správné barvy materiálů, protože není jisté, jaké barvy VRML prohlížeč zvolí. Přidáme tedy k definici každé *Shape* uzel *Apperance*, ve kterém nastavíme barvu materiálu na bílou pro kreslení vyplněných ploch a na černou pro kreslení obrysů. Tímto jsme úspěšně uložili vypočtený model do zdrojového souboru jazyka VRML.

2.3.3 DXF

Firma Autodesk vyvinula tento formát pro zjednodušení výměny dat mezi různými CADovými programy. Původně se jednalo o čistě binární soubor, další verze ale přidali i ASCII verzi. Tu také v našem programu využijeme. Textová verze byla zvolena, protože umožňuje velice lehce zasahovat do struktury i po vytvoření souboru aniž by se musela použít specializovaná aplikace. Zároveň umožňuje jistý náhled, co se ve scéně nachází i bez otevření v nějakém prohlížeči modelů.

DXF soubor je rozdělen do několika sekcí, my opět všechny nevyužijeme, už jen proto, že DXF je formát určený primárně pro výkresy a my prvky jako například kóty, nepotřebujeme. Význačným prvkem formátu DXF je použití dvou řádků pro zápis jedné hodnoty. Na prvním (typicky lichém řádku souboru) se nachází číselná hodnota označující datový typ hodnoty následující na dalším řádku. Kromě datového typu ale číslo také přesně vymezuje význam následující hodnoty. Jako příklad si vezměme třeba hned první číslo, čili nulu. Ta jako jedna z mála znamená více věcí – buď to je oddělovač sekcí, začátek grafické entity či značí hodnotu uloženou v tabulce. Význam čísel použitých v programu naleznete v následující tabulce.

Číslo	Význam
0	Oddělovač sekcí, začátek grafické entity či hodnota uložená v tabulce
2	Název následující sekce
8	Jméno hladiny nebo typ entity
10-18	x-ová souřadnice (1. – 9.)
20-28	y-ová souřadnice (1. – 9.)
30-38	z-ová souřadnice (1. – 9.)

Tabulka 1 - Identifikátory DXF

Sekce využijeme dvě (vlastně tři, EoF čili konec souboru se uvádí také jako sekce), z čehož jedna bude uvedena pouze, protože tam být musí. V hlavičce by se měli nacházet proměnné a informace o programu, ve kterém byl soubor generován, to ale pro naše potřeby není podstatné. Důležitější je pro nás sekce „*Entitites*", která bude obsahovat souřadnice bodů, které budou vhodně seřazeny a spojeny do skupin, aby tvořili vypočtený model. Pro nás nevhodnější entitou bude „*3DFACE*," která vytvoří troj- či čtyř- úhelníkovou plochu. Po identifikátoru následují souřadnice bodů ve třech osách za použití správných číselných identifikátorů (viz. Tabulka č. 1). Proces opakujeme pro všechny polygony a můžeme ukončit sekci Entitites. Poslední dalo by se říci sekcí je konec souboru, který musí být explicitně označen pomocí nuly a textového řetězce „EOF." Za tímto řetězcem mohou následovat libovolná data, protože běžný parser pro DXF soubory je bude ignorovat. Takto je možno společně s univerzálními daty o modelu přenášet i další informace, které jsou vhodné pouze pro jednu konkrétní aplikaci. Ta si je může uložit a i později načíst aniž by znemožnila načítání modelu ostatním aplikacím podporujícím standard DXF.

3 Testování

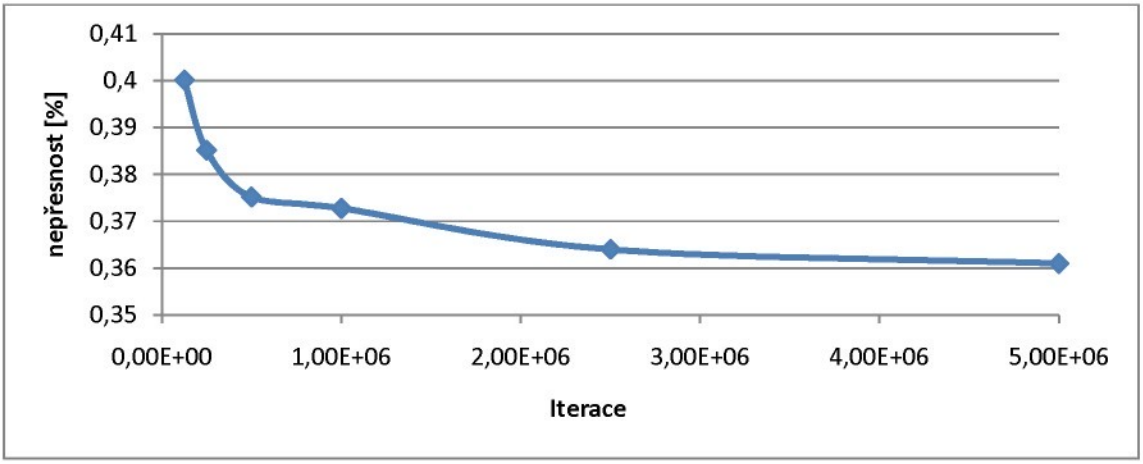
Pro řešení některých částí programů nebylo možné vytvořit algoritmus, který by byl schopný zjistit konečné řešení nějakým přímým postupem. Největší překážkou jsou nepřesnosti ve vypočítaných částech modelu, nejvíce to je zřejmé u spojení dvou či více částí modelu dohromady. Vzhledem k tomu, že známe minimálně tři společné body, měly bychom být schopni za pomoci analytické geometrie charakterizovat vzájemnou polohu všech částí modelu v prostoru. Polohy kamer nejsou ale úplně přesné (lze ale dosáhnout celkem slušné přesnosti), vzájemné rozmístění společných bodů v prostoru není pro každou část úplně stejné. Museli tedy být implementovány metody schopné nalezení maximálně přesného řešení jiným než analytickým způsobem.

Způsoby byly implementovány celkem dva – generování výsledku náhodně nebo za pomoci genetických algoritmů. Náhodné hledání řešení není zcela elegantní, natož efektivní, lze ale provést pár optimalizací, které zajistí rychlejší konvergenci k přesnému řešení. Navíc dnešní počítače nabízejí velký výpočetní výkon, takže projít několik milionů možných řešení trvá pouze pár minut. První úlohou, která je řešena náhodou, je hledání polohy kamer. Princip hledání řešení byl popsán v kapitole „2.2.2 Výpočet polohy a orientace kamer“, zde se budeme zabývat pouze vlivem nastavení konstant na výslednou přesnost nalezeného řešení. U hledání kamer se nepoužívají žádné optimalizace, jde nám tedy čistě o nalezení vhodného poměru počtu iterací algoritmu vůči přesnosti řešení. Pro testování byl zvolen jednoduchý model krabice, jež byl opakovaně počítán, a zaznamenávány byly přesnosti polohy kamer. Pro každý počet iterací byl výpočet opakován 5x, aby se odstranil vliv náhody na přesnost výsledku. Výsledky jsou vidět v následující tabulce, kromě přesnosti byl měřen i čas nutný pro dokončení výpočtu (testovací sestava obsahovala procesor Intel Atom 1,6 GHz a 2 GB operační paměti.).

Iterace	1,25E+05	2,50E+05	5,00E+05	1,00E+06	2,50E+06	5,00E+06
Čas [s]	3,5	7	13	27,5	64	129
1. výpočet	0,405	0,398	0,375	0,376	0,369	0,362
2. výpočet	0,4	0,375	0,371	0,369	0,365	0,36
3. výpočet	0,396	0,376	0,378	0,376	0,362	0,358
4. výpočet	0,405	0,395	0,386	0,369	0,363	0,363
5. výpočet	0,395	0,382	0,366	0,374	0,361	0,362
Nepřesnost [%]	0,4002	0,3852	0,3752	0,3728	0,364	0,361

Tabulka 2 - Přesnost řešení polohy kamer

Čas nutný k nalezení řešení roste víceméně lineárně s počtem iterací, přesnost výsledku již ale ne. Pro lepší představu je lepší se podívat na obrázek č. 11, na kterém je vidět grafická závislost přesnosti řešení na počtu iterací. Je vidět, že pro získání celkem přesného řešení potřebujeme alespoň 5e5 iterací, další zvyšování iterací ale nemá příliš velký vliv na přesnost řešení. Byla zvolena hodnota 1e6, při které nalezení řešení netrvá příliš dlouho, navíc nabízí dobrou přesnost a také stálost řešení.



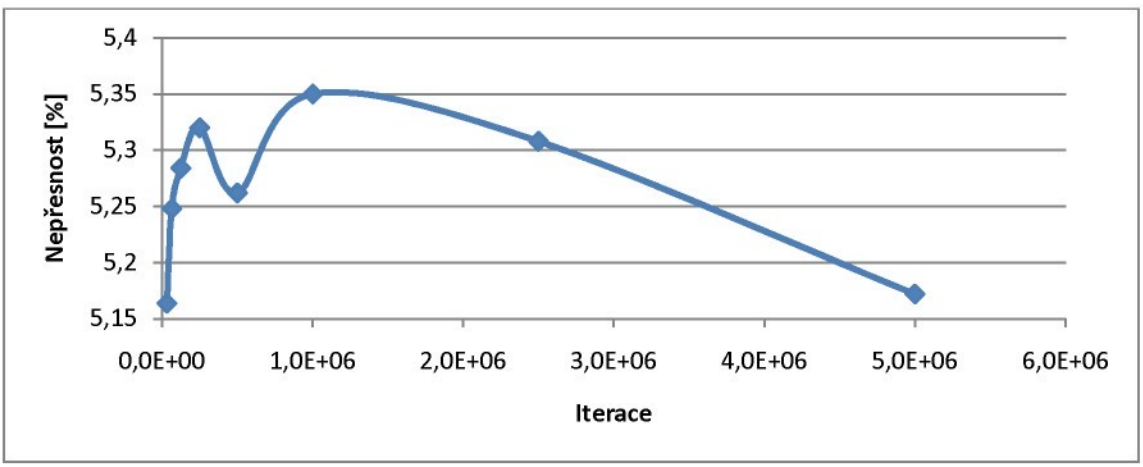
Obrázek 11 - Přesnost polohy kamer

Další část, kterou je nutno řešit numericky, je spojování částí modelu dohromady. Před vlastním narotováním je provedeno srovnání délek hran v modelu, přesto to nestačí k možnosti hledání analytického řešení. Jako první možnost hledání řešení byla zvolena opět náhoda. Nyní ale byla přidána mírná optimalizace algoritmu, která zrychlí a zpřesní hledání řešení. Úhly nejsou při každé iteraci generovány „nanovo," nýbrž se generuje přírůstek k prozatím nejlepšímu řešení. Tato změna samotná nemá příliš velký vliv na kvalitu řešení, pouze mění trajektorii, po které se dostáváme k finálnímu řešení. Vlastní optimalizace spočívá v omezení rozsahu generovaných přírůstků vzhledem ke kvalitě posledního řešení. Tato optimalizace vychází z představy, že v okolí přesného řešení dochází k rychlému poklesu chyby řešení, čili jakmile získáme celkem přesné řešení, stačí již jen pár menších kroků, abychom dosáhli přesného řešení. Opět je nutné nalézt vhodný počet iterací, který by zajistil přesné a numericky stálé řešení v přijatelném čase. V následující tabulce jsou vidět naměřená data, která byla pořízena podobně jako předchozí data na modelu krabice, upraven byl pouze soubor projektu tak, aby se musely spojovat dvě části modelu.

Iterace	3,3E+04	6,5E+04	1,3E+05	2,5E+05	5,0E+05	1,0E+06	2,5E+06	5,0E+06
Čas	0,5	0,5	1	2	4	7	17,5	35
1. výpočet	5,42	5,51	5,18	5,31	5,47	5,09	5,18	4,92
2. výpočet	4,86	5,14	5,21	5,38	5,35	5,37	4,98	5,22
3. výpočet	4,94	5,4	4,91	5,55	5,17	5,68	5,61	5,16
4. výpočet	5,12	4,87	5,59	5,14	4,96	5,48	5,29	5,37
5. výpočet	5,48	5,32	5,53	5,22	5,36	5,13	5,48	5,19
Nepřesnost [%]	5,164	5,248	5,284	5,32	5,262	5,35	5,308	5,172

3 - Přenost řešení narotování částí modelu

Přesnost rotace je bohužel závislá na kvalitě předchozího řešení, nelze tedy jednoznačně říci, které nastavení je nejlepší. Pro lepší představu je vidět na obrázku č. 12 opět graf charakterizující závislost přesnosti nalezeného řešení na počtu iterací. Pro jistotu jsou ve výpočtu použity dvě hodnoty pro počet iterací, aby bylo jisté, že dojde k nalezení co možná nejpřesnějšího řešení. Byly voleny z různých stran charakteristiky, aby se co nejvíce eliminovala možná nepřesnost vlivem náhody.



Obrázek 12 - Přesnost narotování při různém počtu iterací

Druhou možností, jak hledat řešení spojení dvou částí modelu, je pomocí genetických algoritmů. Princip funkce a detailnější popis byl uveden v kapitole „2.2.4 Rotace bodů“, zde probereme pouze význam jednotlivých parametrů na průběh a výsledek výpočtu. Začněme od velikosti vlastního prvku generace. Tím si zvolíme maximální dosažitelnou přesnost výsledku, zároveň ovlivňujeme také délku výpočtu. Pro naše potřeby byla zvolena reprezentace jednoho úhlu pomocí dvanáctibitového čísla, čímž dosáhneme maximálně nepřesnost rovnou čtyřem setinám stupně, což je plně

dostačující vzhledem k přesnosti výpočtu předchozích částí algoritmu. Počtem prvků v jedné generaci lze regulovat rozmanitost jednotlivých prvků (ať už při počátečním generováním či po mutaci prvku). My jsme zvolili velikost 255, čímž zajistíme vcelku dobrou rozmanitost prvků vhodnou pro křížení, zároveň však minimalizujeme šanci výskytu stejných prvků v počátečních iteracích algoritmu.

Další volbou je počet iterací algoritmu. Ideální hodnota byla zjištěna experimentálně, hlídala se průměrná kvalita řešení a čas nutný k jeho nalezení. Byla vybrána hodnota pět tisíc, protože zaručuje dobrou kvalitu a zároveň stabilitu řešení v rozumném čase. Nyní zbývá pouze nastavit pravděpodobnosti křížení a mutace. Tyto hodnoty zabraňují stagnaci vývoje populace, obzvláště pak mutace dokáže zajistit nalezení vhodného řešení díky náhodnému měnění řetězců. Pravděpodobnost křížení byla stanovena na šedesát procent, takto zajistíme zachování části generace, což je vhodné hlavně ke konci výpočtu, kdy je většina řetězců natolik kvalitních, že je lze považovat za finální řešení a nelze jejich kombinací najít lepší řešení. Šance, že některý prvek řetězce zmutuje, byla naopak zvolena pouze pět procent, protože při větší pravděpodobnosti by mohlo docházet ke znehodnocování celé generace vlivem příliš velkého počtu nevhodných mutací. Při těchto nastavením trvá GA nalézt řešení zhruba 18 sekund s tím, že kvalita řešení je srovnatelná s řešením nalezeném „náhodou," kvalita řešení GA ale nekolísá tak, jako se děje u náhodného hledání.

Závěr

Aplikace, vytvořená v rámci této práce, je schopná rekonstruovat libovolný předmět za předpokladu, že se na něm či v jeho nejbližším okolí nachází pravoúhlý čtyřúhelník. K dispozici je uživatelsky jednoduché rozhraní pro vytváření dat projektu a následný výpočet. Pro okamžitou verifikaci modelu je k dispozici interní prohlížeč, pro další použití dat je možno je exportovat do běžných formátů používaných pro prostorové modely. Pro jednodušší orientaci v programu je možno náhlednou do sekce help, kde je popsáno celé rozhraní programu a i jak správně postupovat při tvorbě projektu. Samozřejmě je možno kdykoliv projekt uložit pro pozdější použití.

Přesnost rekonstrukce sice nedosahuje přesnosti vhodné pro průmyslové použití, přesto při dodržení několika zásad lze získat velice přesný model. Vlastní algoritmus, bez ohledu na použité metody výpočtu, je schopen počítat velice přesně, problémem je nalezení vhodné implementace pro řešení numerických úloh během algoritmu. Nejjednodušším řešením tohoto problému je implementace vhodných optimalizačních metod, které jsou schopné nalézt velice přesné řešení pro jakoukoliv úlohu. Problémem může být ale rychlost výpočtu, protože pro získání přesnějšího výpočtu musí být použita složitější metoda, která je obecně výpočetně náročnější. Ovšem i toto lze obejít, či nechat na uživateli, zda si raději počká a dostane přesnější řešení, nebo chce mít výsledek rychle, ovšem za cenu méně přesného modelu.

Ideální by bylo současnou implementaci výpočetních metod pro výpočet polohy kamer, bodů a vzájemné polohy množin bodů nahradit rozhraními, která by definovala pouze vstupní a výstupní formát dat a jednotlivé metody by se dodávali ve formě pluginů. Ty by kromě vlastního algoritmu obsahovali i záznam o délce výpočtu, případně o kvalitě výsledného řešení. Uživatel by si tedy mohl vybrat metodu dle požadované přesnosti, případně by bylo možno spočítat řešení pomocí více metod a vybrat to nejvhodnější. Zlepšení by jistě zasloužila i výstupní část aplikace. Kromě rozšíření možností exportu (více formátů, lepší správa polygonů) by se mohl rozšířit i OpenGL prohlížeč, protože nyní je schopen pouze vykreslovat základní tvar. Minimálně by bylo vhodné zobrazovat help k ovládání pohledu, případně doplnit možnosti zobrazení modelu (barva, nasvícení, pouze drátěný model apod.).

Seznam použité literatury

[1] Eller Frank: *Delphi 6 – příručka programátora*, Grada, 2002, ISBN 80-247-0303-3

[2] Shreiner Dave, Woo Mason, Neider Jackie, Davis Tom: *OpenGL – průvodce programátora*, Computer Press, 2006, ISBN 80-251-1275-6

[3] Žára Jiří, Beneš Bedřich, Felkel Petr: *Moderní počítačová grafika*, Computer Press, 1998, ISBN 80-7226-049-9

[4] Wikipedia - The Free Encyclopedia, http://wikipedia.org/, různé doplňující informace

[5] http://www.web3d.org/x3d/specifications/vrml/, standart definující VRML

[6] http://www.autodesk.com, standart definující DXF

[7] Matthias Wandel – jHead v2.87, http://www.sentex.net/~mwandel/jhead/

[8] http://www.fotoaparat.cz, doplňující obrázky


POWER

- Výběr provedete kliknutím na tlačítko „Výběr souborů.“ Program umí pracovat se soubory typu BMP a JPG. Všechny soubory ale musí být stejného typu. Pokud vyberete soubor jiného typu, než jste dosud používali, dojde ke smazání všech dosavadních souborů a začínáte vlastně odznova.
- Soubory je možno přidávat i odebírat, musí se ale dodržet dvě věci. Zaprvé výše zmíněný stejný dat. typ všech souborů, zadruhé pořadí souborů, které již byli uloženy v projektu. To vlastně znamená, že nemůžete mazat soubory, které byly přidány před uložením a znovuotevřením projektu. Pokud z nějakého důvodu nechcete některý obrázek používat a je již uložen v projektu, prostě na něm smažte všechny body a při výpočtu nebude používán.
- Přidání dalších souborů se provede kliknutím na tlačítko „Vyber soubory.“
- Pokud chcete obrázek smazat, označte ho v seznamu a stiskněte klávesu „d.“

PŘIŘAZENÍ ID

- Nejdříve vpravo vyberte jeden bod, který budete označovat na dalších obrázcích.
- Následně na pravém obrázku označte bod, který je v reálu ten samý, jako bod označený na levém obrázku. Pokud se na obrázku takový bod nenachází, přejděte na další obrázek pomocí tlačítka „Další“
- Takto projděte všechny obrázky a označte, kde se nachází daný bod. Poté na levém obrázku označte další bod a proces opakujte.
- Při vybrání bodu na levém obrázku je vytvořeno nové ID, dbejte prosím tedy na postupné propojování bodů. Mohlo dojít na situaci, kdy je více ID než počet reálných uzlových bodů a výpočet je tedy znehodnocen, případně může počítat úplné nesmysly
- Po vyznačení všech bodů stiskněte tlačítko „Ověřit ID.“ Ukáže se vám seznam obrázku, u kterých budou vypsány body, které nemají přiřazeny žádné ID, čili nebyly propojeny s nějakým jiným bodem.
- Je doporučeno propojit všechny body, když to není nezbytně nutné. Zvýší se ale přesnost a i úspěšnost výpočtu.

VYMAZÁNÍ ID

- Na obrázku označte daný bod (je jedno jestli na levém nebo pravém) a stiskněte klávesu „d“
- Nezapomeňte ale že ID se přiřazuje zleva doprava, takže než dáte bodu nové ID, najděte si nějakou jinou fotku, kde je daný bod s již správným ID, tu označte aby byla v levém obrázku a do pravého načtěte fotku bodem, u kterého jste ID mazali

ÚHLY

- Pro výpočet je nutné zadat některé parametry fotoaparátu. Pokud si nejste příliš jisti, koukněte do manuálu fotoaparátu, kde ohniskovou vzdálenost téměř určitě najdete. Nezapomeňte ale, že se ohnisková vzdálenost mění s velikostí aktuálního zoomu. Pokud váš fotoaparát má např. ohniskovou vzdálenost 20-160mm a až 8x zoom, spočtete aktuální vzdálenost jako aktuální zoom * ((160-20)/8).
- Crop faktor udává poměr velikosti CCD čipu vůči typickému rozměru (24*36mm). Tento parametr není běžně uváděn, obecně ale klesá s rostoucí cenou fotoaparátu. Typická hodnota je 1.5, případně můžete zkusit experimentovat s touto hodnotou pro dosažení lepších přesnosti modelu.
- Nejjednodušší způsob získání těchto údajů je ale zobrazeni EXIF informací o fotce. Najdete tam údaj „focal length," případně přímo ohnisková vzdálenost. Toto číslo ale nemusí nutně to, co potřebujeme. Koukněte, jestli o kousek níže není něco jako „35mm" ekvivalent. Pokud ano, tak použijte toto číslo a crop faktor nastavte na 1. Pokud tam není, zkuste hledat položku „CCD height" nebo „CCD width." Pokud naleznete jedno z těchto čísel, vydělte jím rozměr „plného" CCD čipu (24x36mm) a získané číslo použijte jako crop faktor, přičemž jako ohniskovou vzdálenost použijete parametr „focal length"

STATISTIKA + KOREKCE

- Je možné, že během propojování bodu (tedy přiřazování ID) došlo k vytvoření nadbytečným ID, kterým není přiřazen žádný bod, v horším případě jim je přiřazen jeden či více bodů, které mají mít jiné ID
- Tlačítko „Statistika" vám vypíše, kde se jaké body nacházejí plus zákl. statistiku polygonů
- Tlačítko „Korekce ID" najde a smaže ID, kterým není přiřazen žádný bod
- Pokud tedy zjistíte, že ID nejsou správně přiřazena, napravte to v záložce „Propojení bodů" a následně spusťte korekci ID

SCALE TYPE

- Touto volbou ovlivňujete způsob výpočtu při použití 2 a více kalibračních obrazců. Obecně je vhodnější první volba, pro některé případy ale zvolením „Vectors" může dojít ke zpřesnění modelu

VÝPOČET

- Po nastavení parametrů výpočtu spusťte výpočet tlačítkem „Počítej"
- Pokud se model úspěšně dopočítá, máte možnost uložit vypočtená data do některého z grafických formátů (DXF / VRML) a následně zobrazit.

UKAŽ MODEL

- Otevře vypočtený model v interním prohlížeči (vyžaduje OpenGL), je možno si zvolit rozlišení okna
- **LMB** - pootočí modelem směrem ke kurzoru
- **RMB** - tažením můžete modelem v okně hýbat
- **MMB** – kolečkem můžete model přibližovat / oddalovat
- **F1** – přepnutí okno / fullscreen

LMB

- funkce závisí na tom, v jaké fázi tvoření modelu jste. Bližší info viz help jednotlivých sekcí
- v integrovaném prohlížeči modelu se přidržením dá s modelem pohybovat do stran

RMB

- při držení pravého tlačítka myši můžete pohybovat obrázkem
- v integrovaném prohlížeči slouží k natáčení modelu (model se točí směrem k pozici kurzoru)

MMB

- v integrovaném prohlížeči slouží k přiblížení/ oddálení modelu (kolečko vpřed/ vzad)

„d“

- po označení konkrétního objektu, který chcete smazat, dojde k jeho smazání. Bližší info viz naleznete v helpu jednotlivých sekcí

HLAVNÍ MENU

- Soubor-Nový – vytvoří nový projekt
- Soubor -Načíst – načte uložený projekt
- Soubor -Uložit – uloží projekt
- Reset – vymaže vybraná data a všechna data, která závisí na vymazaných datech
- Nastavení – grafické nastavení programu
- About – info o autorovi, tento help

- V seznamu si vyberte obrázek, na němž chcete vyznačovat body. Poté pomocí LMB vyznačte všechny body, které chcete, aby byly použity při výpočtu a tím pádem také zahrnuty ve výsledném 3D modelu.
- Body je možné posouvat, stačí bod přetáhnout na požadované místo.
- Mazání bodu se provádí označením daného bodu a stisknutím klávesy „d."

- Nejdříve vyberte v horní části obrazovky typ polygonu. Pokud budete vyznačovat 4úhelník, který má všechny vnitřní úhly pravé, zaškrtněte prosím volbu „Pravoúhlý." Následně označte body tvořící polygon, po označení všech bodů polygonu dojde k jeho automatickému vytvoření. Prosím vyznačte všechny polygony, obzvláště ty pravoúhle, protože to zvýší pravděpodobnost úspěchu výpočtu modelu.
- Pokud budete chtít vymazat nějaký polygon, zaškrtněte v horní části obrazovky „delete," označte body tvořící polygon a zmáčkněte klávesu „d."

- Pokud chcete z nějakého důvodu vynechat ve výsledném modelu některé polygonu, můžete tak učinit kliknutím na tlačítko „Vynechání polygonů“ na záložce „výpočet“
- Nejdříve si zjistěte indexy bodů tvořících daný polygon (otevřete si např. záložku vytváření polygonů, najděte si obrázek, kde je polygon vidět, poté najeďte kurzorem nad jednotlivé body a ukáže se vám index bodu)
- Ty napište do připraveného pole, indexy oddělujte čárkami, pokud chcete odebrat více polygonů, oddělte je středníkem
- Příklad možného zápisu – 1,2,3,4;2,5,43
  odebere z finálního modelu 2 polygony – čtverec z bodů 1,2,3,4 a trojúhelník 2,5,43


POWER