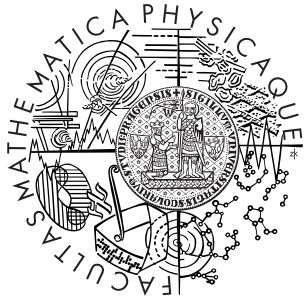
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Tomáš Jedlička

Generování trojrozměrného popisu objektu z multifokálních snímků

Katedra softwarového inženýrství

Vedoucí bakalářské práce: Mgr. Viliam Holub
Studijní program: Informatika, programování

2006

Na tomto místě bych chtěl poděkovat především svému školiteli, pod jehož vedením tato práce vznikla.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne 31. července 2007 Tomáš Jedlička

Obsah

Název práce: Generování trojrozměrného popisu objektu z multifokálních snímků
Autor: Tomáš Jedlička
Katedra (ústav): Katedra softwarového inženýrství
Vedoucí bakalářské práce: Mgr. Viliam Holub
e-mail vedoucího: holub@dsrg.mff.cuni.cz

Abstrakt: Předložená práce zkoumá metodu automatické tvorby trojrozměrného objektu ze série vhodně pořízených snímků.

Klíčová slova: multifokální, Delaunay, TIN

Title: Generation of 3D object description from multifocal pictures
Author: Tomas Jedlicka
Department: Department of Software Engeneering
Supervisor: Mgr. Viliam Holub
Supervisor's e-mail address: holub@dsrg.mff.cuni.cz

Abstract: The present work investigates method of automatical generation of 3D object from series of specialy prepared pictures.

Keywords: multifocal, Delaunay, TIN

Kapitola 1

Úvod

V současné době existuje několik metod pro rekonstrukci trojrozměrných objektů ze série snímků. Tato práce se zaměřuje na rekonstrukci trojrozměrného tělesa ze série multifokálních snímků. Jedná se o sérii snímků stejného objektu, vždy ze stejného místa, kdy se jednotlivé snímky od sebe liší pouze nastavením optické soustavy. Metody pracující s multifokálními snímky se dají rozdělit na dva typy.

Prvnímu typu metod stačí malý počet vstupních snímků k rekonstrukci tělesa. Metody jsou založeny na přesné znalosti optické soustavy, kterou byly snímky pořízeny. Jejich uplatnění se nachází především v robotickém vidění, kde není možnost pořizovat velký počet snímků, kvůli rychlosti jakou je třeba obrazy zpracovávat. Zároveň znalost optické soustavy je samozřejmostí, protože bývá navrhována spolu s robotem.

Druhý typ metod nevyžaduje pro rekonstrukci tělesa přesnou znalost optické soustavy, což usnadňuje celý proces rekonstrukce a vyžaduje minimální interakci uživatele. Nevýhodou je potřebná velká série vstupních snímků. Proto je tato metoda užívána pro rekonstrukci preprátů zobrazených mikroskopem. Cílem této práce je popsat tento typ rekonstrukce a provést její implementaci, která by měla odstranit některé nedostatky konkurenčních řešení a poskytnout lepší využití současného běžně dostupného hardware počítačů.

Kapitola 2

Teorie

Cílem implementace bylo nejen vytvořit trojrozměrný objekt, ale také poskytnout uživateli možnost náhledu objektu v reálném čase. Z tohoto důvodu bylo nutné rekonstrukci rozdělit do dvou částí. V první polovině dochází ke zpracování samotných multifokálních snímků. Výstupem je výšková mapa, normálová mapa a ostrá textura. K popsání tvaru tělesa bohatě postačuje pouze výšková mapa, bohužel takto uchovaná informace o tvaru je vzhledem k současným možnostem grafických karet nepraktická. Proto se v druhé fázi výpočtu ještě provádí konverze výškové mapy na síť trojúhelníků, které umí dnešní grafický hardware efektivně zpracovávat.

2.1 Konstrukce zaostřené textury

Prvním krokem k sestrojení trojrozměrného modelu je sestrojení jeho ostré textury. Ostrou texturu z několika multifokálních snímků lze získat vybráním jednotlivých ostrých částí z každého snímku a sloučením těchto částí do jednoho obrázku. Pokud jednotlivé snímky obsahují navzájem disjunktní ostré části a zároveň tyto ostré části pokrývají celý snímek, získáme ostrou texturu ve všech obrazových bodech. Typickým problémem je nedostatek vstupních snímků. Nastává tak situace, kdy sloučením ostrých částí nepokrývá celý obraz. Výsledkem je místo, které zůstane rozostřené a zároveň je problém stanovit jeho výšku. Sloučení multifokálních snímků do ostré textury se dá popsat následujícím algoritmem 2.1:

```
for i = 0 to width do
begin
    for j = 0 to height do
    begin
        max = 0
        for k = 0 to count do
        begin
            crit = kriterium(i, j, k, epsilon)
            if crit > max then max = k
        end
        pseudo(i,j) = max
    end
end
```

Obrázek 2.1: Algoritmus konstrukce ostré textury. V algoritmu *width* a *height* zastupují šířku a výšku multifokálních snímků v obrazových bodech, *count* určuje počet snímků a *epsilon* poloměr okolí obrazového bodu, které se užívá při vyhodnocování kriteria. Funkce *kriterium*, která je více rozvedena v 2.1.1, *pseudo* je zaostřovací pseudo obraz dle [1]

2.1.1 Kritéria ostrosti

Kritérium ostrosti je nejdůležitější funkce, která tvoří základ celé rekonstrukce. Tato funkce ohodnocuje každý obrazový bod (dále jen bod) reálnou hodnotou, která říká, jak hodně je daný bod zaostřený. Pro člověka není problém určit dané zaostřené body na každém multifokálním snímku, protože lidský mozek umí uvažovat nad celkem a nemusí zpracovávat každý bod. Pro počítač je nutné definovat nějaké pravidlo, které ostré body splňují a rozostřené nikoli. Samotná hodnota jednoho obrazového bodu nic nevypovídá o jeho zaostřenosti, protože hodnoty jednotlivých bodů jsou závislé na hodnotách okolních bodů [1, strana 250]. Zaostřenost bodu se tedy musí stanovit z hodnot bodů v jeho epsilonovém okolí.

$$v(x,y) = \frac{\max\limits_{c \in K_{xy}}(c) - \min\limits_{c \in K_{xy}}(c)}{C_{xy}} \tag{2.1}$$

$$v(x,y) = \frac{1}{C_{xy}} * \sum_{c \in K_{xy}} \left(\frac{c}{C_{xy}} - \sum_{c \in K_{xy}} \frac{c}{C_{xy}} \right)^2 \tag{2.2}$$

Ve výše uvedených vzorcích značí K_{xy} okolí bodu o souřadnici x a y, c značí hodnotu

obrazového bodu a $C_{xy} = \sum_{c \in K_{xy}} c$.

2.1.2 Detekce pozadí

V situacích, kdy se na multifokálních snímcích nachází preparát, který nepokrývá celý snímek, je nutné detekovat pozadí a body pozadí do dalších výpočtů neuvažovat. Při použití kritérií na body patřící do pozadí, dojde ke zvolení nejostřejšího bodu náhodně. Tento jev je dán tím, že pozadí se jeví na všech snímcích rozostřené a tedy kritéria dávají na všech snímcích velice podobný výsledek. V takovéto situaci autor [1] užívá k detekci pozadí statistické vyhodnocování, zda bod s danou barvou patří do pozadí nebo do preparátu. Během implementace této práce vznikla dvě výrazně jednodušší kritéria pro detekci pozadí, která jsou dostačující a proto autorova metoda nebyla implementována.

První a nejjednodušší metodou detekce pozadí je klíčování. Tato metoda dostává na vstupu hodnotu barvy pozadí K a povolenou odchylku δ, pro každý bod je pak proveden test, zda rozdíl mezi barvou pozadí a barvou bodu je menší než odchylka. Pokud ano, je bod detekován jako pozadí. Výhodou této triviální detekce je její velká rychlost, nevýhodou je špatné rozpoznávání bodů preparátu, které mají shodnou barvu s pozadím. Za těchto podmínek tato metoda občas detekuje body preparátu jako body pozadí.

$$b(X_{ij}) = |X_{ij} - K| < \delta \tag{2.3}$$

Druhou metodou je detekce pozadí s využitím maskovacího obrazu. Tato detekce nevyužívá žádný automatický algoritmus rozpoznávání bodů pozadí, jejím vstupem je maska obsahující bílé nebo černé body. Pro každý bod X_{ij} obrazu je testován jeho odpovídající bod M_{ij} v masce. Pokud je v masce na stejné souřadnici jakou má zpracovávaný bod černá barva, je daný bod detekován jako pozadí. Tato filtrace také zároveň dává uživateli možnost provést detekci pozadí v jiném programu.

$$b(X_{ij}) = (M_{ij} = 0) \tag{2.4}$$

2.2 Konstrukce výškové mapy

Série multifokálních snímků vzniká postupným vyfocením preparátu s různým nastavením optiky. Když se tato série snímků vhodně utřídí (provádí uživatel při tvorbě vstupních dat), dá se toho využít k stanovení výšky bodu. Předchozí kritéria určují, jak hodně je

daný bod zaostřen. Když se kritérium pro daný bod vyhodnotí na všech multifokálních snímcích, zjistí se, na kterém snímku je bod nejostřejší a díky utřídění vstupních obrazů získá informace i o jeho výšce. Z tohoto přístupu je jasně vidět, že čím více snímků je k dispozici, tím lepší bude výsledek. Z důvodu neznalosti optické soustavy musí uživatel poskytnout informaci o tom, jaká má být výsledná výška tělesa. V následujícím vzorci 2.5, je *depth* uživatelem zvolená výška, O_{ij} je hodnota zaostřovacího pseudo obrazu spočteného dle 2.1 a *count* je počet všech snímků.

$$h(X_{ij}) = \left(\frac{O_{ij}}{count}\right) * depth \tag{2.5}$$

Pokud se sestrojí těleso rovnou bez jakýchkoliv filtrací, bude mít výsledný model poblíž hran, kde se mění hodnota zaostřovacího pseudo obrazu, špičaté výstupky. To je způsobeno tím, že vstupní snímky nejsou úplně disjunktní a na hranách mezi dvěma snímky vrací kritéria podobný výsledek. Pro odstranění tohoto nedostatku stačí pseudo obraz filtrovat pomocí 2.6 a teprve poté zpracovat pomocí 2.5.

$$O_{ij} = round\left(\frac{\sum_{k \in K_{ij}} k}{(2 * \varepsilon + 1)^2}\right) \tag{2.6}$$

K_{ij} je ε okolí bodu na souřadnicích i a j v pseudo obrazu a je definováno následovně:

$$K_{ij} = \{O_{kl} | (i - \varepsilon \leq k \leq i + \varepsilon) \wedge (j - \varepsilon \leq l \leq j + \varepsilon)\}$$

Pro odstranění schodů v tělese stačí těleso filtrovat stejně jako pseudo mapu, ale pouze bez funkce *round*.

V tomto kroku je sestrojené těleso reprezentováno výškovou mapou. Tato mapa je matice výšek jednotlivých bodů, tedy se pro bod na souřadnici $[x, y]$ zjistí jeho výška tak, že se z výškové mapy vyzvedne hodnota h_{xy}. Obdobně se i výšková mapa sestrojí, pro každý bod se stanoví jeho výška a ta je do mapy zaznamenána na souřadnici $[x, y]$.

2.3 Konstrukce drátěného modelu

V druhé fázi zpracování vstupních dat je z výškové mapy vytvářena její trojúhelníková reprezentace. Trojúhelníková reprezentace, která se bude vytvářet, se nazývá Triangle Irre-

gular Network (dále jen *TIN*). Jak již název napovídá, je model reprezentován nepravidelnou sítí trojúhelníků. Základní algoritmus pro tvorbu *TIN* z výškové mapy je poměrně triviální, ale je náročný jak implementačně, tak z hlediska časové složitosti jednotlivých kroků algoritmu. Jednotlivé optimalizace jsou zmíněny v části implementace.

Před tvorbou *TIN* se provádí inicializace celého algoritmu. Je vytvořen trojúhelník, jehož rozměry jsou zvoleny tak, aby jakákoli souřadnice x a y výškové mapy ležela uvnitř trojúhelníku. Poté jsou vloženy čtyři rohové vrcholy výškové mapy do tělesa a provedem algoritmus tvorby *TIN*. Vložení rohových vrcholů není nutné, ale během implementace aplikace se ukázalo, že výsledky dosažené touto cestou jsou na pohled hezčí. Průběh algoritmu je uveden na 2.2. Po dokončení algoritmu tvorby *TIN* je ještě nutné těleso ořezat. Ořezání je provedeno odstraněním všech tří vrcholů trojúhelníku, který obklopoval celou výškovou mapu. Zároveň jsou odstraněny všechny hrany a trojúhelníky, které obsahují jeden ze tří odstraněných vrcholů.

```
vertex = 0
error = calculate_mesh_error(mesh)

while (error > delta) and (vertex < max_count) do
begin
    vertex = get_vertex_with_max_error(heightmap)
    insert_vertex_to_mesh(mesh, vertex)
    error = calculate_mesh_error(mesh)
    vertex = vertex + 1
end
```

Obrázek 2.2: Greedy Insertion algoritmus

2.3.1 Globální chyba tělesa

Jak je patrné z algoritmu 2.2, byla zvolena dvě kritéria, která algoritmus zastaví. První kritérium měří odchylku tělesa od výškové mapy. Pro jeho kalkulaci byl zvolen následující vzorec, kde e značí funkci počítající chybu vrcholu a V značí množinu vrcholů ve výškové mapě. Problémem tohoto kritéria je jeho kolísání, ne vždy se vložením vrcholu chyba celého tělesa sníží. Toto chování je podrobněji popsané v sekci 2.3.2.

$$me(mesh) = \frac{\sum_{v \in V} e(v)}{|V|} \tag{2.7}$$

Druhým kritériem je omezení počtu vkládaných vrcholů. Toto kritérium bylo použito ze dvou důvodů. Při vkládání vrcholů se sice celková chyba tělesa snižuje, ale od určitého počtu začleněných vrcholů do tělesa chyba začne klesat velmi pomalu, protože těleso už má téměř shodný tvar s výškovou mapou a zbývající nezačleněné vrcholy na něm tvoří pouze malé detaily. Důsledkem je typicky začlenění všech vrcholů do tělesa, což není nutné a hlavně vytvořit těleso s maximálními detaily jde daleko jednodušší a efektivnější metodou. Druhý důvod je úmyslné omezení počtu trojúhelníků, kvůli náhledu v reálném čase. Čím zastaralejší je grafický hardware, tím méně vrcholů dokáže zobrazit v reálném čase, tedy je vhodné mít možnost úmyslně omezit maximální počet vrcholů.

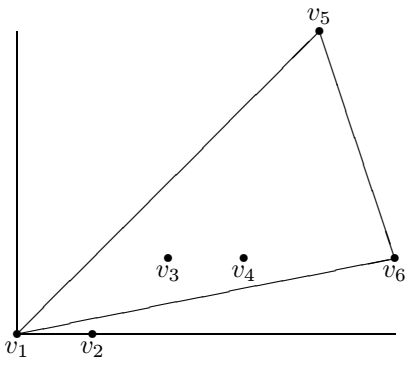
2.3.2 Chyba vrcholu

Definice chyby vrcholu má vliv na celý výsledek rekonstrukce. Pro tuto práci byla zvolena následující rovnice určující odchylku výšky spočtenou ve výškové mapě $h(x)$ a výšky vrcholu podle *TIN* $T(x)$.

$$e(x) = (h(x) - T(x))^2 \tag{2.8}$$

Změnou vzorce pro výpočet chyby lze změnit chování algoritmu. Podle 2.8 se snaží vždy začlenit vrchol, který je nejvíce vzdálen od výšky, ve které se má nacházet. V případě, že výšková mapa obsahuje šum, dá algoritmus priotu šumu místo tělesu. Proto je nutné, pokud preparát nezabírá celý snímek, použít metodu detekce pozadí.

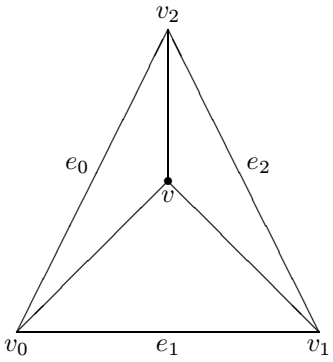
Další problém tohoto způsobu stanovení chyby vrcholu, je nemožnost zaručit, že po vložení vrcholu do tělesa se vždy globální chyba tělesa sníží. Celý problém je vyobrazen na obrázku 2.3, kde je zakreslena *TIN* z profilu před vložením vrcholu a po jeho vložení spolu s velikostí jednotlivých chyb vrcholů.



Obrázek 2.3: Zvýšení chyby tělesa po vložení vrcholu v_5.

2.3.3 Vložení vrcholu do TIN

Vložení vrcholu do *TIN* je přímočarý postup. Nejprve se zjistí, do kterého trojúhelníku v *TIN* se má vrchol umístit. Poté se trojúhelník nahradí třemi novými, které pokrývají původní trojúhelník a jako jeden z vrcholů mají nově vložený vrchol (na obrázku 2.4). Dále se provádí rekurzivně aktualizace hran e_0, e_1 a e_2 podle Delaunayho triangulace [2]. Ačkoli se jedná o triviální algoritmus jeho implementace je více rozvedena v kapitole 3.



Obrázek 2.4: Vložení vrcholu v do *TIN*

2.3.4 Konstrukce normál

Závěrečným krokem při konstrukci tělesa je výpočet normál. Bez vypočtených normál nelze těleso nasvítit světelným zdrojem při náhledu v reálném čase. Normálové vektory se užívají pro tzv. *per vertex* osvětlení tělesa. Pro každý vrchol tělesa je spočten jeho normálový vektor podle 2.9. $T(x)$ značí množinu trojúhelníků, které mají x jako jeden z jejich vrcholů, x značí vrchol, pro který počítáme normálový vektor a $nv(t)$ značí normálový vektor trojúhelníku t.

$$n(x) = normalize\left(\sum_{t \in T(x)} nv(t)\right) \tag{2.9}$$

2.4 Konstrukce normálové textury

Největším nedostatkem *TIN* je, že pokud má *TIN* co nejvíce odpovídat výškové mapě, je nutné do ní vložit co nejvíce vrcholů a ideálně dokonce všechny. To je ale v rozporu s požadavkem na náhled v reálném čase, kde je naopak potřebné vytvořit těleso s co nejméně trojúhelníky. S nástupem moderních grafických karet přišly i nové technologie, jednou z

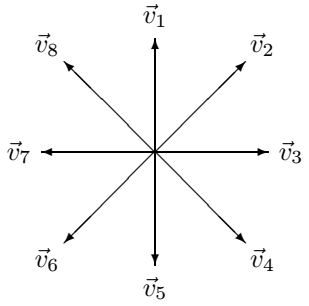
nich je i *DOT3* mapování. Základním principem je nanesení speciální textury na objekt. Tato textura obsahuje v barevné složce zakodovány normály povrchu, při vykreslování tělesa pak tyto normály ovlivňují dopad světla. Výsledkem je detailní zvlněný povrch i když je těleso rovné. Jediným nedostatkem této metody je pohled zblízka rovnoběžně se stěnou tělesa, kdy uživatel uvidí, že těleso je rovné i když z dálky vypadá jinak.

Výpočet textury je obdobný jako v sekci 2.3.4, jen nemusí uvažovat okolní trojúhelníky. Pro každý vrchol sestrojí vektory k jeho sousedním vrcholům a vypočte normálový vektor. Značení vektorů vedoucích k sousedním vrcholům je uvedeno na obrázku 2.5. Při výpočtu $\vec{n}_8$ se dosazuje do vzorce 2.11 $\vec{v}_1$ místo $\vec{v}_9$.

$$norm(X_{ij}) = normalize\left(\sum_{i=1}^{8} \vec{n}_i\right) \tag{2.10}$$

$$\vec{n}_i = \vec{v}_i \times \vec{v}_{i+1} \tag{2.11}$$

Takto spočtené normálové vektory jsou v tzv. *Object Space* neboli v prostoru objektových souřadnic tělesa. V současné době většina normálových map uchovává normálové vektory v tzv. *Tangent Space*. Výhodou tohoto přístupu je možnost opakovaně využívat jednu texturu pro více částí tělesa. V tomto případě by generování normálové mapy v tangent space jen zbytečně zpomalilo výpočet a nepřineslo žádné výhody, protože textura je jen jedna a pokrývá celé těleso.



Obrázek 2.5: Vektory v_i vedoucí k okolním vrcholům

Kapitola 3

Implementace

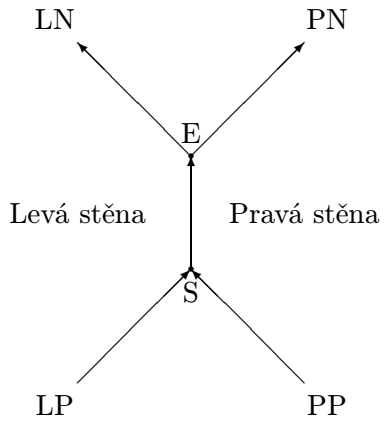
Tato kapitola se zabývá implementací algoritmů z kapitoly 2. Nejsou zde zmíněny všechny implementační detaily, protože implementace vzorců je dosti přímočará a lze ji snadno vyčíst ze zdrojového kódu. Naopak cílem této kapitoly je zmínit veškeré speciality a důležité implementační detaily algoritmů.

3.1 Konstrukce drátěného modelu

3.1.1 Reprezentace tělesa

Během konstrukce drátěného modelu je nutné zvolit vhodnou reprezentaci tělesa. Poměrně často bývají trojrozměrná tělesa reprezentována jako seznam vrcholů a seznam trojic indexů do seznamu vrcholů, které reprezentují trojúhelníky. Tato reprezentace je vhodná pro prezentaci tělesa, nikoli však pro manipulaci s tělesem. Po rozhodování byla zvolena struktura *Winged Edge*[3] místo *Quad Edge* [3], protože je o něco jednodušší a naprosto postačuje pro účely aplikace.

Při této reprezentaci těleso také obsahuje seznam vrcholů a indexů, které tvoří trojúhelníky. Navíc obsahuje seznam hran, který tvoří hlavní popis tělesa. Struktura windged edge je vyobrazena na obrázku 3.1. Struktura se dá implementovat dynamicky (každá položka je ukazatel do paměti na příslušný vrchol, hranu nebo trojúhelník) nebo polem (každá položka je indexem do pole položek příslušného typu). Při implementaci polem je bohužel nutná realokace, tedy dochází k zpomalení výpočtu. Výhodou pole je možnost sériového zpracování, které se hodí více při výpočtu normál a exportu tělesa do výstupního formátu. Při kompletně dynamické reprezentaci je výhodou, že není nutno pole realokovat, ale neumožňuje sekvenční přístup k vrcholům a trojúhelníkům.



Obrázek 3.1: Windged edge reprezentace, S a E označují počáteční a koncový vrchol. *LN* a *LP* označují levou následující a levou předchozí hranu, to samé platí i pro *PN* a *PP*. *Levá stěna* a *Pravá stěna* označují trojúhelníky nalevo nebo napravo od hrany.

Během implementace se ukázalo, že je poměrně nepřehledné provádět aktualizace tělesa, protože zde většinou probíhá více operací s různými indexy najednou. Také je zde potřeba ošetřovat mnoho speciálních situací (každá je ještě ovlivněna orientací hrany). Existuje však zlepšení, které šetří čas hlavně vývojáři. Jak již bylo zmíněno v kapitole 2, je celá výšková mapa zabalena do jednoho velkého trojúhelníku. Pokud se vrcholy tohoto trojúhelníku uvedou v pořadí proti směru hodinových ručiček, získá se určitá pravidelnost. Díky pravidelnosti vrcholů získají pravidelnosti i hrany tělesa a při vkládání dalších vrcholů se již snadno zajistí, aby tato pravidelnost byla dodržena. Vývojáři tím odpadá řešení mnoha specifických situací a omezí se počet chyb při implementaci.

Samozřejmě lze vrcholy trojúhelníků uspořádat i po směru hodinových ručiček, ale tento přístup se nehodí z důvodu užívání Delaunayho triangulace, která vyžaduje znát vrcholy v pořadí proti směru hodinových ručiček. Tohoto pořadí by se dalo docílit tříděním vrcholů, ale to je zbytečné pokud takovéto podmínky jde zajistit přímo.

3.1.2 Volba vrcholu a její optimalizace

Prvním problémem je implementace funkce `get_vertex_with_max_error` v algoritmu 2.2 v sekci 2.3. Podmínky pro volbu jsou jednoduché, zvolí se takový vrchol, který v *TIN* ještě není a zároveň má nejvyšší chybu (určenou podle 2.8 na straně 12) ze všech nezačleněných vrcholů. Tento krok je značně přímočarý, ale hlavní problém tkví v jeho časové složitosti. Vyhledávat vrchol s nejvyšší chybou trvá $O(n)$, kde n je počet prohledávaných vrcholů, zároveň se musí zajistit, že daný vrchol není v *TIN* již začleněn. Za předpokladu,

že v *TIN* je celkem m vrcholů, tak test, zda-li je vrchol v *TIN* potrvá $O(m)$. Spojená podmínka, že vrchol musí mít největší chybu a zároveň nesmí být v *TIN*, má tedy při naivním řešení velice velkou časovou složitost. Ideální datovou strukturou pro tyto podmínky je halda nezačleněných vrcholů do *TIN*. Vždy se odebírá vrchol v konstantním čase s reorganizací haldy tedy v čase $O(\log_2 n)$ a zároveň není vůbe nutné testovat zda vrchol v *TIN* je nebo není, protože každý, který byl do *TIN* přidán, byl z haldy odstraněn.

Podstatným detailem je ale i doba, která je potřebná pro aktualizaci chyby vrcholu. Při implementaci vyhledávání vrcholů haldou ztrácíme možnost aktualizovat chyby vrcholů, vyhledávání v haldě totiž není možné provádět efektivně. Z tohoto důvodu jsou samotné vrcholy a jejich pořebné informace uloženy ve výškové mapě jako struktura. Nad touto strukturou je vytvořena halda ukazatelů do ní. Tato halda je utříděna tak, aby vrchol s nejvyšší chybou byl na vrcholu. Změnit hodnotu chyby vrcholu lze tedy v čase $O(c)$. Poté je ještě nutné aktualizovat haldu (jen tehdy pokud vrchol není v *TIN*). Z tohoto důvodu má vrchol uloženou informaci o jeho pozici v haldě,toto řešení zvýší paměťovou složitost, ale aktualizace chyby vrcholu se provádí v čase $O(\log_2 n)$.

Čím více je vrcholů v *TIN*, tím menší je halda a celá tato operace nalezení vrcholu s největší chybou a aktualizace ostatních trvá stále méně času.

3.1.3 Vkládání vrcholu do *TIN*

Vkládání vrcholu je nejnáročnějším krokem konstrukce modelu. Je třeba vložit nový vrchol, provést aktualizaci *TIN* a aktualizovat chyby jednotlivých vrcholů. Samotná aktualizace sítě je závislá na zvoleném algoritmu triangulace. Pro tuto práci byla zvolena Delaunayho triangulace. Dnes existuje několik algoritmů, které provádí efektivně Delaunayho triangulaci, bohužel jsou konstruovány tak, že pro danou množinu vrcholů sestrojí *TIN*. Tento přístup se příliš nehodí, protože začleňuje vrcholy postupně podle jejich spočtené chyby a tedy není vhodné po každém vložení nového vrcholu znovu sestrojit celou *TIN*, proto byla zvolena daleko efektivnější metoda a to aktualizovat *TIN* průběžně pomocí inkrementální Delaunayho triangulace.

Aby bylo možno provést jen nezbytné aktualizace, je nutno nejprve určit, ve kterém trojúhelníku se nachází vkládaný vrchol. Jedním z řešení je prohledávat nějakým způsobem trojúhelníky v *TIN* již začleněné, ale tento přístup není zdaleka nejefektivnější. Pokud je třeba dosáhnout efektivnějšího lokalizování trojúhelníku, je třeba získat rychlost na úkor potřebné paměti. Pro každý vrchol je uložena ještě informace, ve kterém trojúhelníku se nachází, tímto je možno najít rodičovský trojúhelník v čase $O(c)$.

Samotná aktualizace je pak již jednoduchá. Zvolí se vrchol, který se má vložit a zjistí ve kterém trojúhelníku se nachází. Tento trojúhelník se odstraní z *TIN* a nahradí třemi novými podle 2.4. Poté se projdou všechny tři hrany e_0, e_1 a e_2 a provede se aktualizace,

tak aby byly zachovány podmínky nutné pro Delaunayho triangulaci. Tento postup je rekurzivní a je popsán v [2].

3.2 Uživatelské rozhraní

Pro implementaci uživatelského rozhraní bylo rozhodováno zda užívat rozhraní *GTK*[4] nebo rozhraní *Qt*[5]. Obě dvě rozhraní jsou téměř srovnatelná. Nakonec bylo zvoleno rozhraní *Qt* a to z následujících důvodů :

- Poskytuje objektově orientované rozhraní
- OpenGL podpora je zde přímo a snadno použitelná
- Obsahuje nejen grafické prvky, ale i spoustu pomocných objektů.
- Obsahuje stejně jako *GTK* nástroj pro vytváření uživatelskch prostředí

3.2.1 OpenGL

Implementace byla zaměřena na maximální výkon aplikace, aby byla manipulace s preparátem co nejrychlejší. Zároveň musí být kompatibilní se starším hardwarem. Důsledkem je, že si uživatel se starším hardwarem musí nainstalovat některé knihovny, které se sice nepoužijí, ale aplikace bez nich nepůjde spustit. Celkem byly implementovány tři různé metody kreslení preparátu.

- První metoda podávající nejnižší kvalitu zobrazení, ale nejvyšší kompatibilitu s grafickým hardwarem. Pro kreslení preparátu se používá kreslení elementů z dat uložených v operační paměti počítače. Nasvícení modelu je prováděno pomocí *per vertex* metody, pro kterou se užívají normály spočtené dle 2.3.4.
- Druhá metoda je identická s první, ale je zaměřena na karty s podporou Vertex Buffer Objects. Veškerá data modelu jsou načítána do paměti spravované ovladačem grafické karty, tedy buď do video ram nebo do sdílené paměti. Tato akcelerace ovlivní hlavně modely s velkým počtem trojúhelníků. Nároky jsou na hardware o něco vyšší, hlavně na dostatek video paměti na kartě a její rychlost. V ideálním případě karta nahraje do video paměti celý model a pokud se nejedná o nejlevnější kartu, bude pravděpodobně vybavená DDR3 nebo DDR4 pamětmi, které jsou výrazně výkonnější než paměť počítače.

- Třetí metoda je určena pro současné nové karty. Organizace paměti je shodná s druhou metodou. Samotné stínování, transformování a rasterizování je řízeno přímo napsaným kódem pro grafické karty. Při tomto režimu akcelerace se kompletně o model stará grafická karta. Výhodou je možnost manipulace s texturami a je tedy implementován DOT3 mapping, který užívá normálovou texturu generovanou podle 2.4. Těleso má nejvyšší kvalitu vzhledu.

Nejvýkonnější metoda byla implementována za použití jayka *Cg*[6] firmy *NVIDIA*. I když OpenGL poskytuje vlastní shader jazyk, bylo zvoleno externí řešení. Použití externího řešení si vynucuje mít nainstalovány runtime knihovny, které jsou k dispozici téměř pro každý operační systém. Výhodou je, že při implementaci se nemusí řešit veškeré detaily týkající se kompatibility s různým hardwarem. Nejprve se vytvoří CGcontext, poté získá profily pro vertex a fragment shader. Pak se předá zdrojový kód kompilátoru, který zvolí nejvhodnější instrukční sadu podle možností hardwaru na kterém je aplikace spuštěna. Implementace byla prováděna na počítači s grafickou kartou *ATI Radeon X1950Pro* a přestože je karta osazena čipem od konkurenční firmy, je řešení pomocí *Cg* jazyka plně kompatibilní.

3.3 Paralelní zpracování

Cílem návrhu bylo napsat aplikaci, která by fungovala na běžně dostupném stolním počítači. Nejvýkonnější procesory dnes dostupné veřejnosti disponují celkem čtyřmi jádry, byla by proto škoda je nevyužít. Z tohoto důvodu je celá první polovina výpočtů textur, kriterií implementována pomocí vláken. Stejně tak je paralelizován i výpočet normál tělesa. Protože většinu času běhu aplikace tvoří konstrukce trojrozměrného tělesa, není zrychlení až tolik patrné. Zpracování vstupů na více procesorech se projeví až při větším počtu vstupních souborů nebo při nastavení velkého epsilon.

3.4 Výstupní formáty

Tato sekce již neobsahuje žádné specifické algoritmy, nicméně je dobré zmínit, proč aplikace umožňuje výstup v konkrétních formátech a jaká jsou omezení. I když aplikace, která je výsledkem této práce, umožňuje prohlížení rekonstruovaných trojrozměrných modelů, není konstruována pro náročnější práci s modely. Proto je nezbytné dát uživateli možnost přenést modely do jiné aplikace, ve které bude možno provádět další operace, které může požadovat. Nejčastějším případem může být např. prezentace modelu v publikaci, proto je nutné vygenerovat detailní model a nechat jej vykreslit některým komerčním nebo nekomerčním nástrojem, aby získal obraz vysoké kvality.

Při volbě výstupního formátu bylo přihlédnuto k současným aplikacím pro práci s trojrozměrnými modely :

1. *3D Studio MAX*[11] patří mezi nejrozšířenější komerční grafické nástroje a jeho 3DS soubory podporuje mnoho dalších produktů.
2. *Autodesk Maya*[12] je konkurecí předchozí aplikace. Původní název býval Alias Wavefront Maya.
3. *Povray*[13] je open source projekt a tedy ideální pro raytracing v linuxu.
4. *Blender*[14] je open source projekt s možností editace modelů, jako jediný je konkurencí komerčním aplikacím.

Výstupní formáty aplikace byly zvoleny podle komerčních aplikací. První velice dobře podporovaný výstupní formát je pro *Autodesk Maya*, který generuje optimální výstupní soubor podle [9]. Nevýhodou tohoto výstupního formátu je jeho velikost, protože je zapisován jako textový soubor. Mnohem vhodnější volbou je tedy binární formát, kterým byl zvolen *3D Studio MAX* 3DS formát. Bohužel jedná se již o starší specifikaci a formát umí uložit jen 65536 vrcholů nebo 65536 stěn tělesa. Proto při exportu dochází rozdělení ukládaného tělesa na menší, což vede k opakování někerých vrcholů a tedy výstupní soubor je validní, ale jeho velikost je díky opakování vrcholů větší než by bylo nutné. Výstup se provádí pomocí knihovny, která je zdarma pro operační systémy Linux [10].

Kapitola 4

Uživatelská dokumentace

Tato kapitola je zaměřena na vše, co by měl uživatel aplikace znát, aby ji byl schopen plnohodnotně užívat a uměl nastavit veškeré parametry ve vstupním souboru. Samotná aplikace je rozdělena do dvou částí, na konzolovou aplikaci a grafické uživatelské prostředí (dále jen GUI). Dokumentace se zabývá jednotlivými částmi zvlášť a speciální oddíl je věnován formátu vstupního souboru.

4.1 Instalace

Aplikace se skládá pouze ze dvou spustitelných souborů, které se nainstalují do adresáře `/usr/bin`. Pro snadnou instalaci jsou na CD přiloženy instalační balíčky pro některé operační systémy. Pokud má uživatel možnost instalace binárních balíčků ve svém operačním systému, měl by ji využít. V ostatních případech je možnost instalace ze zdrojových kódů pomocí :

```
make
make install
```

Pro instalaci projektu je nutné nainstalovat i veškeré knihovny, které aplikace využívají. Všechny knihovny by měly být dostupné pomocí nástrojů operačního systému. Pro případ nemožnosti instalování závislostí automaticky, jsou jejich balíčky přiloženy na CD, odkud si je může uživatel sám nainstalovat.

4.2 Vstupní soubor

Tato sekce je určena pro uživatele, kteří plánují užívat pouze textové rozhraní aplikace. Uživatelé kteří budou užívat GUI mohou tuto sekci přeskočit, protože o validní vstupní soubor se postará GUI za ně. Pokročilejší uživatelé, kteří znají jazyk Extensible Markup Language[7] (dále jen *XML*) mohou přeskočit úvod této sekce a věnovat se části, která popisuje strukturu vstupního dokumentu.

4.2.1 Stručný úvod do jazyka *XML*

Pro popis informací ve vstupním souboru byl zvolen jazyk *XML* a to z několika důvodů. V dnešní době je poměrně snadné generovat dokumenty v jazyce *XML* a také existuje spousta knihoven, které umožňují jejich parsování. Další výhodou je možnost *XML* dokument vytvářet v libovolném textovém editoru, jako je například *vi*, tedy uživatel má velkou volnost, jak vstupní data zapsat do jazyka *XML*, případně může užít přiloženou GUI aplikaci.

Každý dokument v jazyce *XML* musí začínat informací o verzi jazyka *XML* a o znakové sadě, ve které byl dokument vytvořen. Vypuštěním nebo špatným vyplnění položky kodóvání vstupního souboru může program špatně intepretovat české znaky a také názvy vstupních souborů.

Dokument v jazyce *XML* se skládá z tzv. *elementů*. Každý element se skládá z dvojice značek a z obsahu uvedeného mezi těmito značkami. Značka může mít libovolný název zapsaný malými písmeny, tato aplikace rozeznává pouze konkrétní značky a konkrétní atributy, uvedené v tabulce 4.1 na stránce 24. Jiné značky jsou aplikací ignorovány a nebudou nijak zpracovány. Otevírací značka může obsahovat *atributy*, název každého atributu musí být zapsán malými písmeny a hodnota atributu musí být uzavřená do úvozovek.

Otevírací značka se zapisuje ve tvaru `<znacka atribut="hodnota">`, při tomto zápisu nezáleží na pořadí atributů v otevírací značce. Ke každé otevírací značce musí dokument obsahovat příslušnou značku uzavírací, která se zapisuje ve tvaru `</znacka>`. Pokud je třeba v dokumentu uvést prázdný element, je možné použít zkráceného zápisu `<znacka/>`, při tomto zápisu lze uvést pouze atributy.

Dalším důležitým pravidlem je dodržet správnou strukturu dokumentu. Hlavním pravidlem, které je nutné respektovat, je správné vnořování elementů. Elementy lze do sebe vnořovat, nesmí se však překrývat. Validní je tedy zápis `<zn1><zn2></zn2></zn1>` nikoli však `<zn1><zn2></zn1></zn2>`. Správná struktura vstupního souboru je uvedena na obrázku 4.1 na stránce 23. Pro správné strukturování dokumentu je k aplikaci přiložen [1][8]

[1] XML Schema

```
<?xml version="1.0" encoding="utf-8" ?>
<?xml-stylesheet type="text/xml" href="project.xsl" ?>
<project name="Ukázkový projekt" width"640" height="480">
  <description>Detailní popis k ukázkovému projektu</description>
  <infiles>
    <file>soubor1.png</file>
    <file>soubor2.png</file>
  </infiles>
  <filters>
    <average radius="20" />
    <background method="colorkey" data="120 200 15" />
  </filters>
  <outfiles dest="./" depth="100" detail="0.5" const="4.0" epsilon="4" method="1">
    <output type="texture" format="png">texture.png</output>
    <output type="heightmap" format="bmp">heightmap.bmp</output>
    <output type="model" format="3ds">model.3ds</output>
    <output type="mask" format="jpg">mask.jpg</output>
  </outfiles>
</project>
```

Obrázek 4.1: Validní vstupní soubor

soubor, který slouží validátorům pro kontrolu správné struktury vstupního souboru.

4.2.2 Sruktura vstupního souboru

Větší znalosti z jazyka *XML* uživatel nepotřebuje, nyní bude probrán význam jednotlivých elementů pro aplikaci. Hlavním elementem je *project*, všechny nastavení aplikace musí být do něj vnořeny. Obsahuje jeden povinný atribut *name*, do kterého se vyplní krátký název reprezentující vstupní data. Další povinné atributy jsou *height* a *width*, které určují rozlišení výstupních obrázků. Element *description* slouží k detailnímu popisu projektu, nemusí však být v dokumentu uveden vůbec. Vstupní soubory jsou zadány pomocí elementu *infiles*. Uvnitř elementu se pak opakuje element *file*, který obsahuje název vstupního souboru. Je uveden tolikrát, kolik je vstupních souborů. Soubory se budou zpracovávat v pořadí v jakém jsou uvedeny ve vstupním souboru od shora dolů. Obdobou vstupu je i výstupní konfigurace a to element *outfiles*. Povinné atributy jsou *detail*, *depth*, *epsilon*, které nasavují úroveň detailů výstupního trojrozměrného modelu. Vzhledem k tomu, že aplikace umožňuje celkem čtyři druhy výstupů, je třeba specifikovat, o které má uživatel zájem a v jakém formátu chce jejich výstup. To se provádí uvedením elementu *output*, který obsahuje název výstupního souboru a atributy *type*, který nabývá hodnot uvedených v tabulce 4.1 na stránce 24, určujících, který výstup se má do souboru zapsat. Atribut *format* udává v jakém formátu se dají data uložit.

element	atribut	hodnota	význam
project	name	text	Krátký název projektu
	height	číslo	Výška výstupu v obrazových bodech
	width	číslo	Šířka výstupu v obrazových bodech
description			
infiles	src	text	Cesta k vstupním souborům
file			
outfiles	dest	text	Cesta k výstupním souborům
	detail	číslo	Úroveň detailů výstupního modelu
	epsilon	číslo	Poloměr okolí bodu při výpočtu kritérií
	depth	číslo	Výška objektu v obrazových bodech
	method	číslo	Volba metody výpočtu kritérií (1 nebo 2)
output	type	heightmap	Výšková mapa
		mask	Maska objektu
		texture	Ostrý snímek
		model	Trojrozměrný model
	format	png	Obrázek ve formátu *PNG*
		bmp	Obrázek ve formátu *BMP*
		jpg	Obrázek ve formátu *JPG*
		3ds	3D model ve formátu *3DS*
		obj	3D model ve formátu *Wavefront*

Tabulka 4.1: Atributy a jejich povolené hodnoty

4.3 Konzolová aplikace

Celý projekt byl rozdělen na dvě části. První částí je konzolová aplikace, která má za úkol převádět vstupní data na výstupní. Výhodou je možnost provozovat samotný výpočet na jiném výkonnějším počítači, který nemusí být vybaven grafickým rozhraním. Ovládání konzolové aplikace je velice jednoduché, samotná aplikace přijímá pouze argumenty uvedené v následující tabulce. Jediný možný způsob jak ovlinit průběh samotného výpočtu, je úpravou vstupního souboru. Ve skutečnosti většina uživatelů konzolovou aplikaci užívat nebude, ti, kteří se rozhodnou pro její užívaní, musí dodržet pravidla uvedená v sekci 4.2 na straně 22.

-f	Povinný argument, určuje název vstupního souboru.
-t	Vygeneruje pouze textury
-m	Vygeneruje i těleso, automaticky implikuje -t

Tabulka 4.2: Argumenty konzolové aplikace

4.4 Grafické uživatelské rozhraní

Uživatelské prostředí bylo navrženo tak, aby zakrylo tvorbu vstupního souboru. Uživatel nemusí znát nic o struktuře vstupního souboru, validní vstupní soubor bude vygenerován aplikací. Zároveň grafické rozhraní umožňuje prohlížet rekonstruované modely, ale pouze pokud byl jako výstupní soubor zvolen formát *obj*. Po spuštění aplikace je rovnou vytvořen prázdný dokument a uživatel má možnost ihned vkládat vstupní soubory a provádět jejich zpracování. Pro jednodušší začátek je v aplikaci zabudován dialog, který se vyvolá zvolením `Soubor->Nový`. Tento dialog provede uživatele všemi částmi tvorby projektu. Po konfiguraci parametrů projektu spouští uživatel samotnou rekonstrukci zvolením `Projekt->Spočti`. Před provedním výpočtu musí být soubor s projektem uložen na pevném disku, k tomu slouží položky `Ulož` a `Ulož jako`. Pro uložení aktuálního obrázku modelu je k dispozici menu `Soubor->Ulož obrázek`.

Pořadí vstupních souborů je stanoveno shora dolů, což znamená že nejvyšší vstupní soubor bude tvořit nejnižší výšku v rekonstrukci. Pro manipulaci se vstupními soubory aplikace poskytuje celkem tři způsoby a je na uživateli, který mu nejvíce vyhovuje. První byl již zmíněn v minulém odstavci, druhým je použití rychlého menu pomocí stisku pravého tlačítka myši v okně *Vstupní soubory* a zvolením příslušné akce z menu, které se objeví. Touto akcí je buď přidat soubor, odebrat soubor, posunout aktuálně označený soubor nahoru, resp. dolů. Při dvojkliknutí na obrázek v jakémkoli okně dojde k zobrazení obrázku v jeho skutečné velikosti. Posledním způsobem je volba `Projekt->Nastavení`, kde může uživatel modifikovat veškeré nastavení projektu a tedy i vstupní soubory.

Okno s názvem *Výstupní soubory* zobrazuje vypočtené soubory nutné k rekonstrukci. Uprostřed aplikace je zobrazen trojrozměrný model preparátu, kterým může uživatel otáčet za současného stisku levého tlačítka myši a pohybu zvoleným směrem otáčení. Poslední okno s názvem *Výstup* slouží k informativnímu výstupu o průběhu výpočtu.

Kapitola 5

Zhodnocení práce

Kromě této práce existují i další implementace podobné rekonstrukce. První skupina, která tuto techniku používá, je robotika a počítačové vidění. Pro tuto oblast tato práce není konkurencí, protože robotika je dosti specifická oblast jak hardwarově tak algoritmicky.

Druhou skupinou je aplikace v geologii a dalších biologických vědách. Zde se používaný software dá dělit na komerční a nekomerční. Komerční software bývá dodáván s konkrétním mikroskopem a má tedy lepší možnost pořizování potřebných snímků. Může například automaticky ovládat krokovací motorky mikroskopu a tak pořídit přesné snímkování a podstatně kvalitnější vstupy, než uživatel ručním ovládáním. Dále může provádět přesnější rekonstrukce díky přesné znalosti konstrukce optiky mikroskopu.

Poslední sekce softwaru je volně šiřitelná a do této kategorie zapadá i tato implementace. Hlavním přínosem je implementace pro operační systém *Linux*, protože ostatní implementace jsou zaměřené na operační systém *Windows*. Zároveň je zde snaha o implementaci pro současný běžně dostupný desktopový systém, který se dá dnes již za rozumnou cenu pořídit. GUI část je zaměřena na akcelerované grafické karty, protože i většina dnešních on-board karet již 3D grafiku umí akcelerovat. Výsledkem je možnost plynule prohlížet i velice detailní těleso.

Tento přístup je jiný v použití TIN, výsledkem je, že se trojúhelníky tvoří tam, kde jsou zapotřebí a na velkých rovných plochách se s nimi šetří. Některé implementace tvoří síť rychleji, ale ne zdaleka optimálně, tedy pro stejnou úroveň detailů modelu spotřebují daleko více trojúhelníků.

I když práce pokrývá celou rekonstrukci, jsou zde části které se dají dále rozvíjet.

- Lepší detekce pozadí, která vyžaduje minimální zásah od uživatele

- Paralelní zpracování tvorby *TIN*

Literatura

[1] Dalibor M., PaeDr., Ph.D (2002): Matematické principy grafických systémů, Littera Brno

[2] Miler G.L., Pav S.E., Walkington N.J. (2002) : An Incremental Delaunay Meshing Algorhitm

[3] Kettner L. : Designing a Data Structure for Polyhedral Surfaces

[4] Gtk:
http://www.gtk.org

[5] Qt:
http://www.trolltech.com

[6] NVIDIA Cg:
http://developer.nvidia.com/page/cg_main.html

[7] XML:
http://www.w3.org/XML/

[8] XSL:
http://www.w3.org/XML/Schema

[9] Wavefront OBJ File Format:
http://www.fileformat.info/format/wavefrontobj/

[10] 3DS File Format Library:
http://sourceforge.net/projects/lib3ds/

[11] Autodesk 3D Studio MAX :
http://www.autodesk.com

[12] Autodesk Maya :
http://www.autodesk.com

[13] Povray :
http://www.povray.org

[14] Blender :
http://www.blender.org

Příloha A

Struktura přiloženého CD

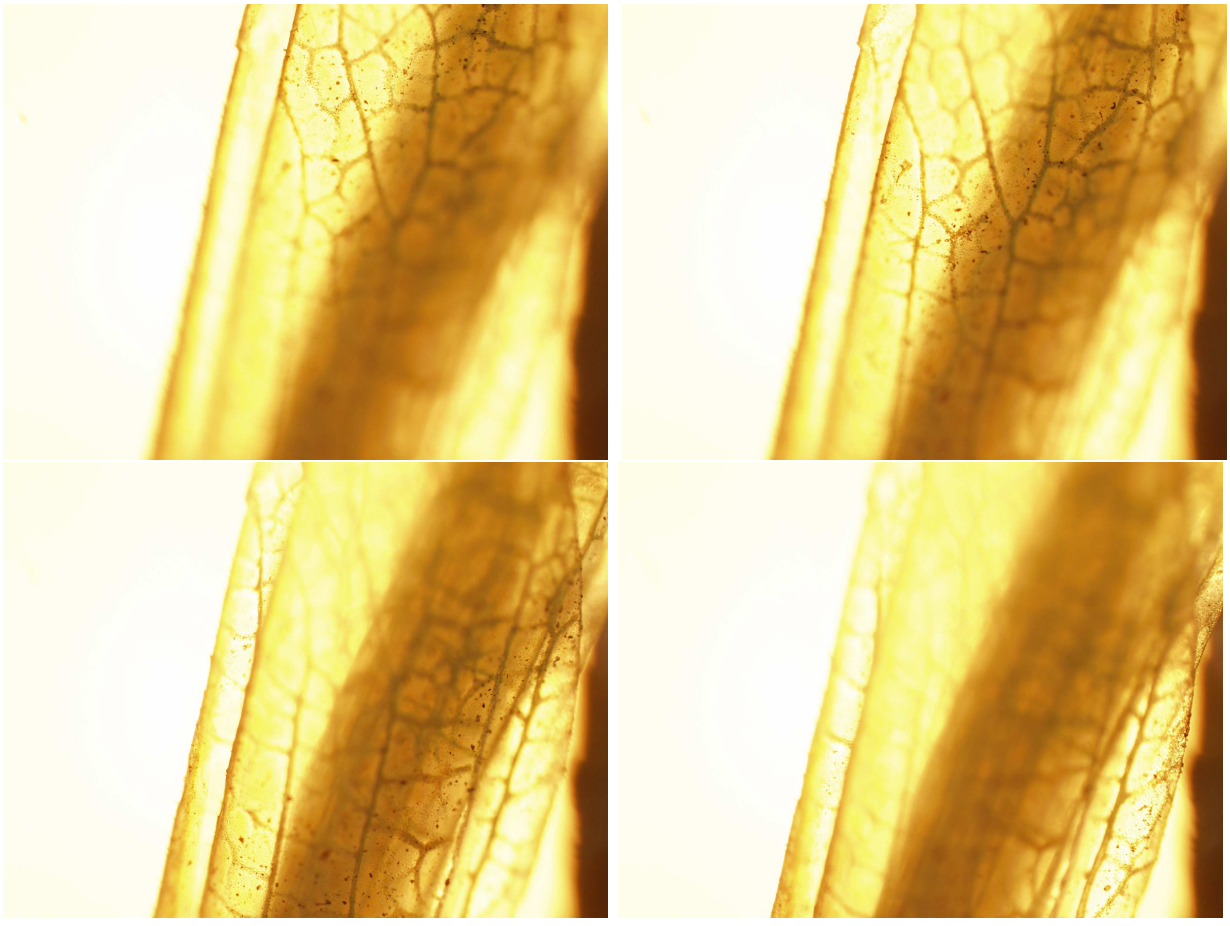
Na CD jsou přiloženy všechny soubory pro případný další vývoj a užívání aplikací. Zároveň je v kořenovém adresáři informační soubor s konkrétními návody na instalaci.

/sources	Zdrojové kódy aplikací
/install	Instalační soubory aplikací
/text/dokumentace	Vývojová dokumenace
/text/prace	Elektronická verze této práce
/examples	Ukázkové vstupy a rekonstrukce
index.html	Obsah CD a instalační návody

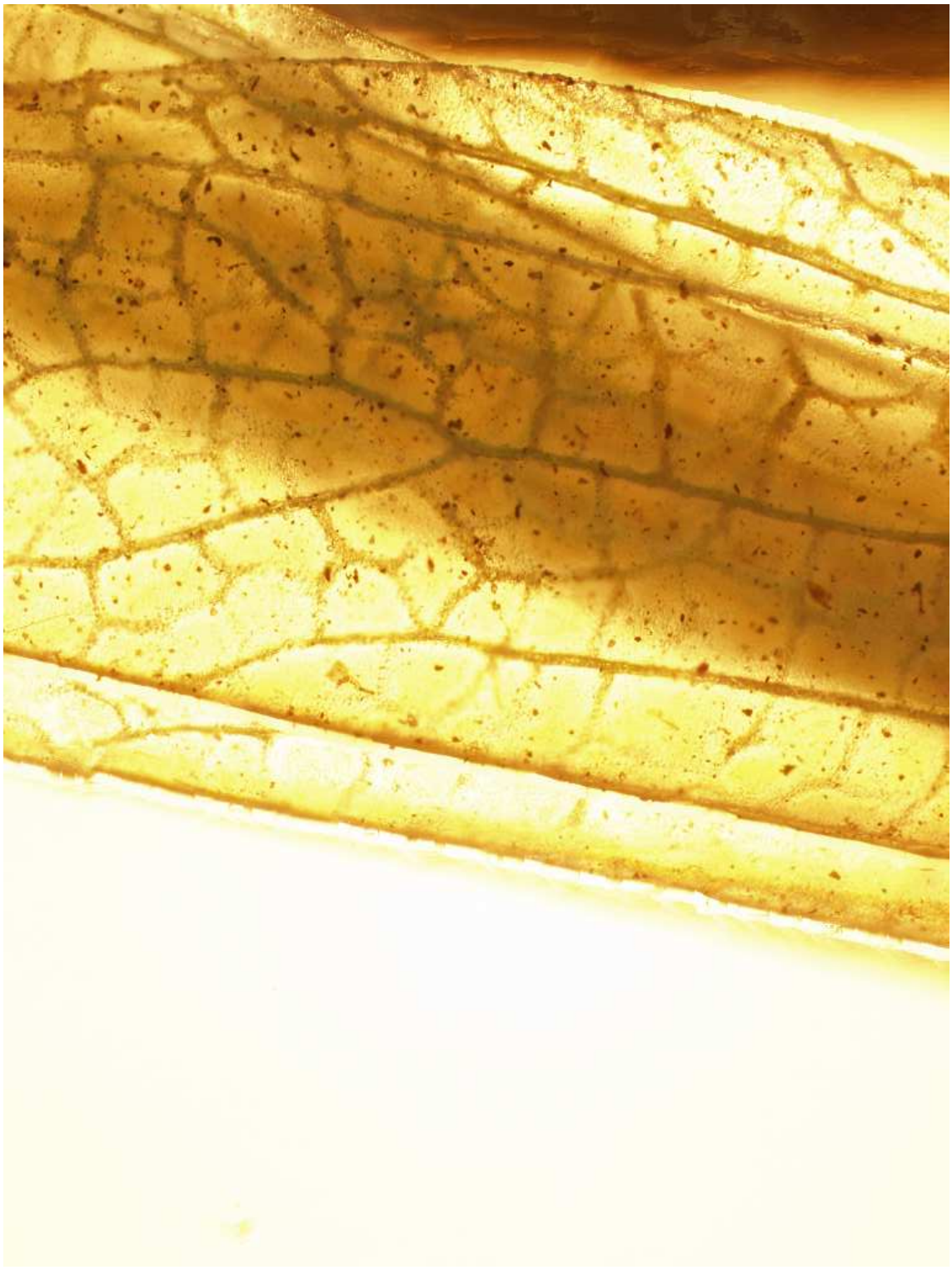
Tabulka A.1: Struktura přiloženého CD

Příloha B

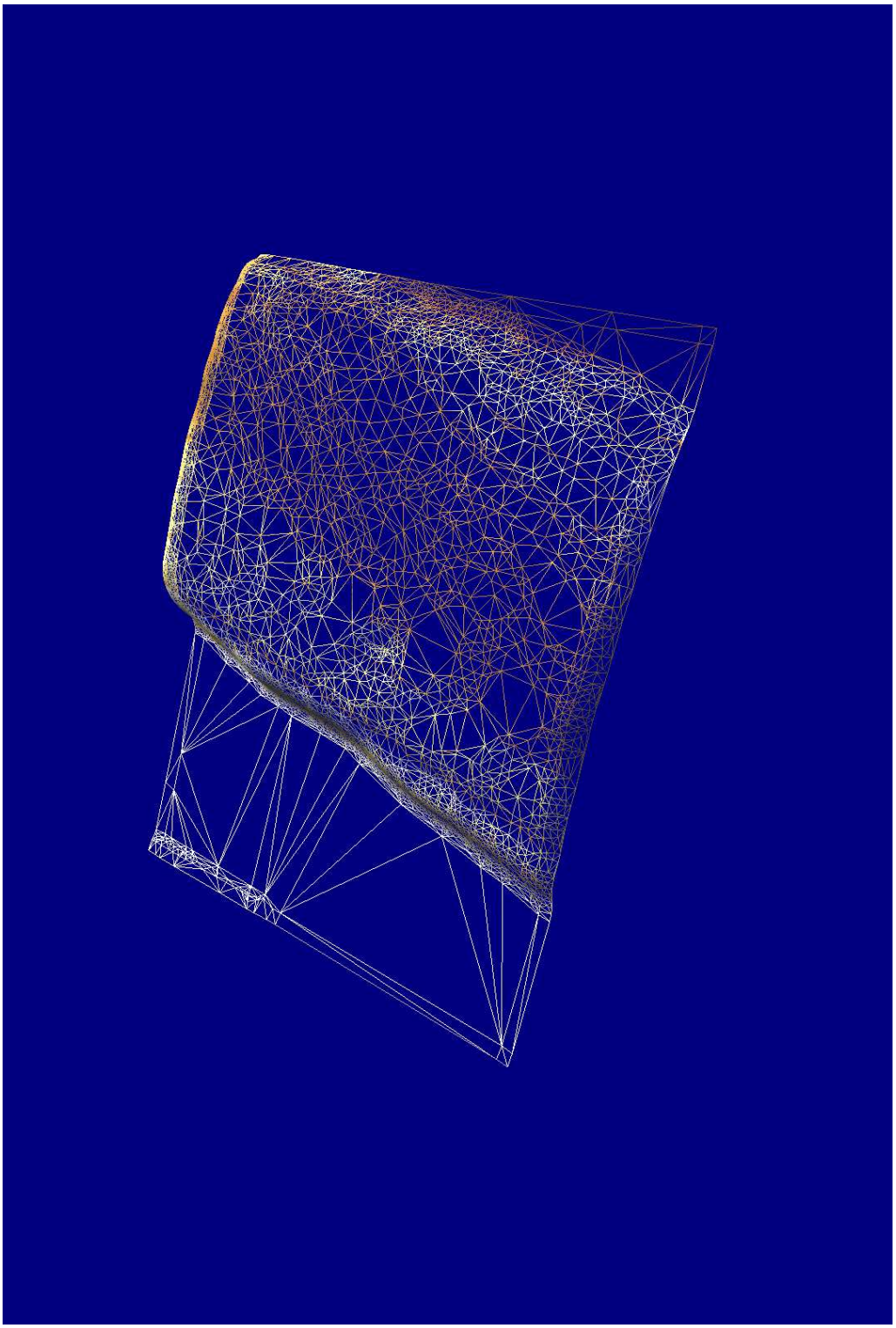
Ukázky rekonstruovaných modelů



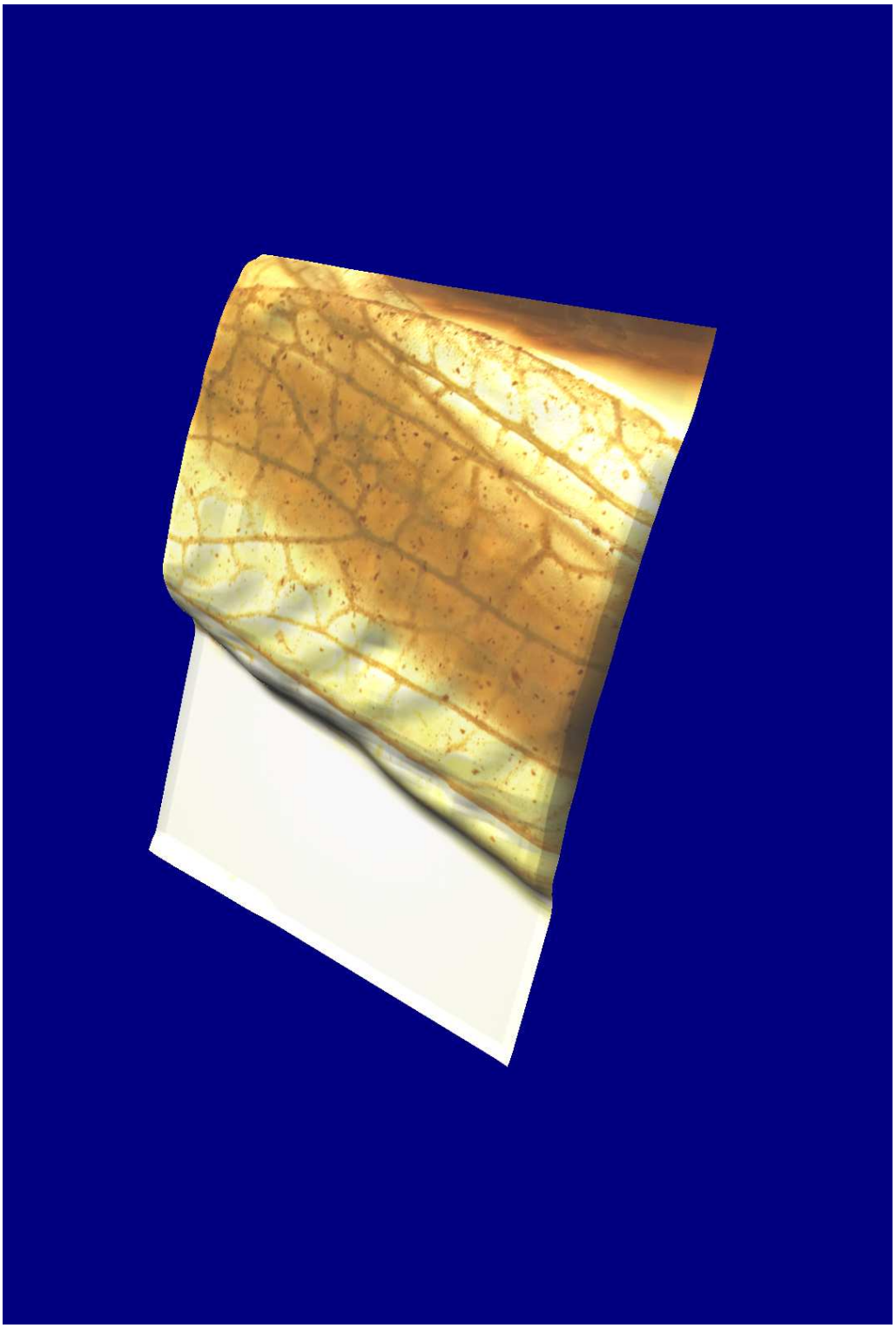
Obrázek B.1: Výběr několika vstupních snímků.



Obrázek B.2: Složený ostrý snímek.



Obrázek B.3: Rekonstruovaný trojrozměrný model jako síť trojúhelníků.



Obrázek B.4: Rekonstruovaný trojrozměrný model.