Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

Diplomová práce

Využití metod umělé inteligence pro rekonstrukci povrchu

Plzeň, 2006 Ladislav Mrnka

Poděkování

Rád bych poděkoval Doc. Ing. Ivaně Kolingerové, Csc. za vedení této diplomové práce, Prof. Ing. Václavu Skalovi, Csc. za vedení v průběhu studia oboru počítačová grafika a Ing. Ivo Hanákovi za cenné rady při řešení problémů s vizualizací výsledků této práce. Dále bych rád poděkoval mé přítelkyni a přátelům za podporu a porozumění.

Abstract

The surface reconstruction is essential problem in many areas of computer graphics. During last four years implementation of CRUST reconstruction method with many improvements was developed at University of West Bohemia. This thesis continues in improvements of reconstructions produced by the given reconstructor. New improvements are based on artificial intelligence and evolution. Utilizations of several different methods are proposed and two stochastic methods with set of new heuristics are implemented. Results show an improvement in area of edges and corners reconstruction together with persistence of flat surface areas.

Obsah

Prohlášení

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

V Plzni dne 29.8.2006, Ladislav Mrnka, ……

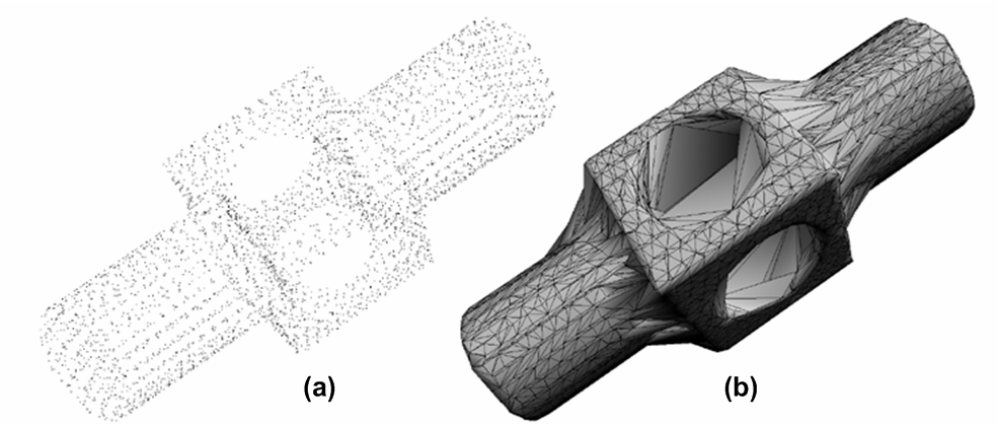
1. Úvod

1.1. Uvedení do problematiky

Výpočetní technika je v dnešní době využívána ve všech vědních oborech. Výhody využívání výpočetní techniky vzrůstají spolu s rozsahem zpracovávaných dat a složitostí řešených úloh. Není tedy divu, že řešení mnoha problémů je bez nasazení počítačů nemožné. Pro nasazení výpočetního výkonu nestačí pouhý algoritmus, který bude strojově řešit úlohu. Je potřeba také vhodně navrhnout interakci mezi uživatelem a výpočetním systémem a způsob, kterým budou uživateli zobrazeny výstupní data. V mnoha oblastech je textový nebo numerický výstup nedostatečný a je třeba zvolit pro uživatele srozumitelnější reprezentaci výstupních dat, např. grafický výstup.

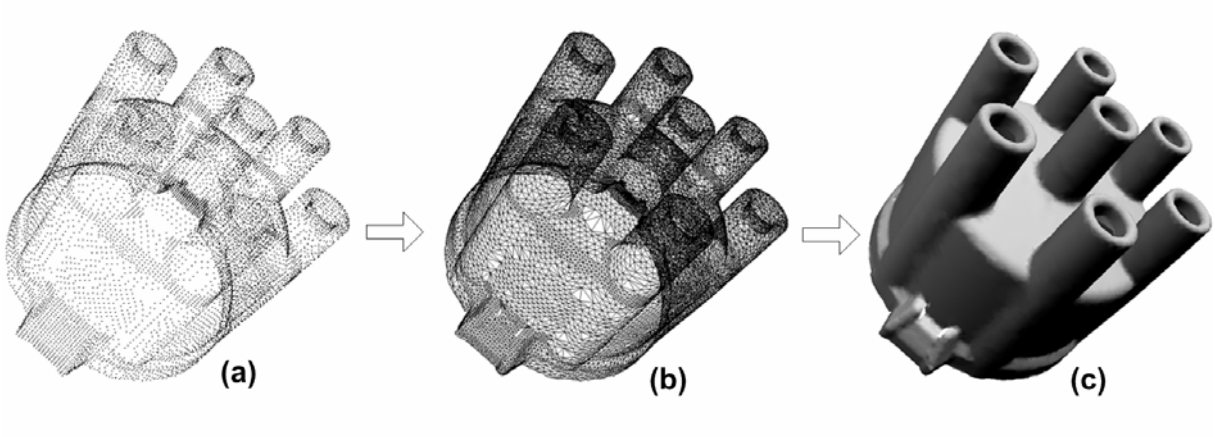
Zpracování a zobrazování grafických dat spadá do samostatné vědní disciplíny nazvané počítačová grafika. Mezi úlohy počítačové grafiky patří datová reprezentace objektů ze skutečného světa v počítači. Pořízení těchto dat je možné různými způsoby. Lze použít grafický software a ručně vytvořit počítačový model skutečného objektu. Tento způsob je ovšem časově náročný a v některých případech těžce proveditelný. Náročnost modelování roste se složitostí a detailností modelovaného objektu. Druhou možností je získat reprezentaci objektu ze speciálního vstupního zařízení, jakým je např. 3D scanner. Tento proces můžeme chápat jako digitalizaci objektu z reálného světa. Získanou digitální reprezentaci objektu můžeme nazývat obrazem objektu z reálného světa.

Snímací techniky jsou většinou založené na odměřování bodů (vzorků) v pravidelné mřížce. Získaná data jsou tedy jen prvotním krokem k získání požadovaného modelu. V dalším kroku je nutné z množiny sejmutých bodů rekonstruovat povrch, který bude představovat získaný model. Tento krok je velmi obtížný a značně závisí na kvalitě dat získaných snímací technikou. Nepovedená rekonstrukce je ukázána na obrázku 1-1. Nejčastější problémy, se kterými se metody pro rekonstrukci povrchu musí potýkat, jsou běžné neduhy všech digitalizovaných dat. Jedná se především o špatné vzorkování (podvzorkování, převzorkování) a šum. Tyto problémy jsou způsobeny technickým omezením snímacích zařízení a nedostatky snímacích technik.



Obr. 1-1 Nepovedená rekonstrukce povrchu. Ze vstupní množiny bodů (a), je nesprávně rekonstruován povrch (b).

Existuje více způsobů jak řešit rekonstrukci povrchu. Kromě metod pracujících nad množinou bodů existují i další metody, které předpokládají komplexnější vstupní data, např. množinu řezů objektem, soubor fotografií objektu snímaných pod různými úhly nebo objemovou reprezentaci (Magnetická rezonance a Počítačová tomografie). Dále lze metody dělit podle způsobu reprezentace získaného povrchu na metody pracující s parametrickými nebo implicitními plochami a metody pracující s trojúhelníkovými sítěmi. Proces rekonstrukce pracující s trojúhelníkovými sítěmi je znázorněn na obrázku 1-2. Obecně je problém rekonstrukce povrchu stále otevřený a nabízí mnoho prostoru pro další bádání. Stávající metody se většinou zaměřují na úzkou množinu vstupních objektů, které si musí být tvarově podobné nebo musí splňovat určité podmínky.



Obr. 1-2 Proces rekonstrukce povrchu z bodů (a), jehož výsledkem je trojúhelníková síť (b) a její následná vizualizace (c).

Na Západočeské univerzitě se pod vedením Doc. Ing. Ivany Kolingerové CSc. dlouhodobě vyvíjí nové metody pro práci s trojúhelníkovými sítěmi. Jedním z dlouhodobých projektů byla

i diplomová [1] a disertační [2] práce Ing. Michala Varnušky PhD., která se zaměřila na realizaci a v pozdější fázi i zdokonalení jednoprůchodové CRUST metody pro rekonstrukci povrchů z rozptýlených bodů. Byly dosaženy velmi zajímavé výsledky, které ukazovaly zlepšení oproti původní jednoprůchodové CRUST metodě Niny Amenty [5]. Stejně jako ostatní metody ani tato není dokonalá. Rekonstruované modely obsahují různé defekty, kterým se stávajícím postupem nedaří vyhnout.

Tato práce navazuje na výstup zmiňovaného rekonstruktoru a pokouší se vylepšit rekonstruované modely pomocí méně obvyklých postupů. Hlavním cílem práce je prozkoumat možnosti využití umělé inteligence pro vylepšení produkovaných rekonstrukcí a realizace některého z navrhnutých postupů. Zvýšená pozornost se zaměřuje na možnosti využití stochastických metod (Simulovaného žíhání) a evolučních metod (Genetických algoritmů), které se jeví jako velice zajímavý nástroj pro optimalizaci trojúhelníkových sítí.

1.2. Struktura diplomové práce

Následující text je rozdělen do několika logických bloků. V teoretické části bude podrobněji popsán problém rekonstrukce povrchů se zaměřením na metodu implementovanou na Západočeské univerzitě. Pro srovnání bude popsáno i několik dalších metod. Dále se teoretická část zaměří na popis vybraných metod umělé inteligence.

V praktické části bude proveden rozbor možností využití zmíněných metod umělé inteligence pro rekonstrukci povrchů. Jelikož se jedná o neprobádanou oblast, je tato část pojata z velké míry jako sbírka navrhovaných řešení. Dále bude v praktické části prodiskutována možnost propojení vznikajícího softwaru s existujícím softwarovým vybavením pro rekonstrukci trojúhelníkových sítí. Praktická část je zakončena popisem vybraných detailů implementované metody.

Pro plné porozumění dalšímu textu se od čtenáře předpokládá pokročilá znalost oblasti informatiky, především pak počítačové grafiky a výpočetní geometrie. Znalosti v oblasti umělé inteligence lze považovat za výhodu. Pro porozumění implementační části je nezbytná alespoň základní znalost objektově orientovaného programování a knihoven pro hardwarově akcelerovanou vizualizaci – OpenGL nebo Direct3D.

2. Metody rekonstrukce povrchu

Jak již bylo naznačeno v úvodu této práce, rekonstrukce povrchů je klíčovým nástrojem pro převod objektů z reálného světa do počítačové reprezentace. Takto získané modely se v současné době používají především v CAD (Computer Aided Design), reverzním inženýrství a medicíně. Pokud se budou metody dále rozvíjet a zdokonalovat, dá se do budoucna počítat s jejich větším využitím pro kompresi ukládaných modelů. Soubory s uloženými modely by poté nemusely obsahovat plnou geometrickou informaci, ale pouze souřadnice vrcholů a případně pomocné informace o tvaru objektu, např. definice ostrých hran, okrajů atd. Zbytek modelu by byl po jeho načtení ze souboru dopočítán metodou pro rekonstrukci povrchu.

Pro tuto práci je podstatná metoda rekonstrukce povrchů nazvaná CRUST [1], [4], [5], která patří do kategorie objemových metod pracujících s dělením prostoru Delaunayovou tetrahderonizací. V následujícím textu bude tato metoda podrobně popsána. Text se zaměří i na její modifikace zmiňované ve [2]. Popis metody CRUST a jejích vylepšení je převzat z [1] a [2]. Dále bude zmíněno několik dalších metod, které využívají naprosto odlišný způsob. Tyto metody jsou uvedeny hlavně pro srovnání výsledků.

2.1. CRUST

Předešlé práce [1], [2] se zaměřily na implementaci metody CRUST, jejíž původní verze byla vytvořena Ninou Amentou [4]. Výběr metody CRUST pro implementaci a následné vylepšování měl dva hlavní důvody. Prvním důvodem je samotný princip metody CRUST, která patří do kategorie metod provádějících rekonstrukci povrchu z trojúhelníků obsažených v Delaunayovy tetrahedronizaci množiny vstupních bodů P. Druhým důvodem byly teoretické důkazy, které alespoň částečně garantovaly funkčnost CRUST metody pro rekonstrukci povrchu.

Výzkum Delaunayovy tetrahedronizace má na Západočeské univerzitě dlouholetou tradici a byly zde dosaženy velmi dobré výsledky. V současnosti je k dispozici efektivní implementace řešená metodou inkrementálního vkládání s využitím datové struktury DAG (orientovaný

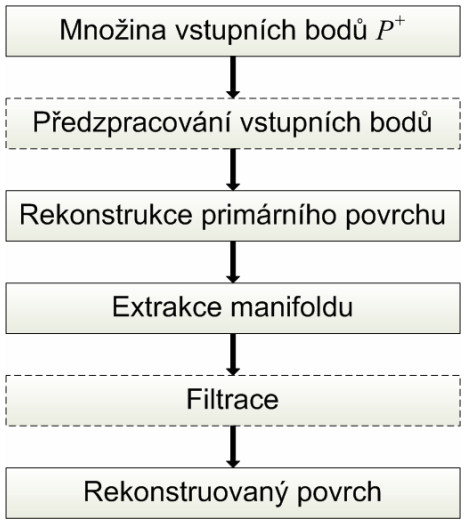
acyklický graf). Očekávaná algoritmická složitost této implementace je téměř lineární, ovšem rychlost si vybírá svou daň v podobě velkých paměťových nároků, které byly na 32 bitových počítačích silně limitující. Limitní velikost vstupní datové množiny pro tuto implementaci se pohybuje mezi 460 tisíci a 500 tisíci body. Proto byl v minulých letech vyvinut systém, který umožňuje distribuovaný výpočet Delaunayovy tetrahedronizace výše zmíněným postupem [13], čímž byly odstraněny všechny bariéry, které omezovaly počet bodů ve vstupní množině. Další práce se zřejmě zaměří na možnosti paralelizace samotného procesu rekonstrukce, který v současné době může pouze využívat paralelní řešení Delaunayovy tetrahedronizace, ale sám pak běží sériově.

Obr. 2-1 Ukázka špatně rekonstruovaných objektů metodou CRUST. Rekonstrukce neproběhla správně, protože objekty nesplňují vzorkovací kritérium.

Teoretické odvození, které garantuje rekonstrukci povrchu metodou CRUST, je možné jen pro vstupní datové množiny, které splňují vzorkovací kriterium. Pro dobrý vzorek musí platit, že jeho vzorkovací hustota je úměrná vzdálenosti ke střední ose. Tato úměrnost se definuje tzv. *r-vzorkem*. Vzorek P nazveme *r-vzorkem*, pokud je euklidovská vzdálenost každého bodu $p \in P$ k nejbližšímu bodu mimo vzorek nejvíce r-krát vzdálenost k nejbližšímu bodu na střední ose. Test vzdálenosti lze lehce provést, pokud je pro vstupní množinu bodů zkonstruován Voronoiův diagram. Podle původního textu Niny Amenty má být $r < 0{,}06$, ovšem ukázalo se, že tato hodnota může být několikanásobně vyšší. V případě, že není kritérium splněno, dochází k chybám při rekonstrukci, viz Obr. 2-1. Výhodou tohoto postupu je jeho nezávislost na distribuci bodů, na druhou stranu nevýhodou je špatná práce s povrchy, které obsahují ostré

rohy, hrany a okraje. Další nevýhodou metody CRUST je její velká citlivost na šum ve vstupních datech.

Obecná rekonstrukce povrchu využívající Delaunayovu tetrahedronizaci může být popsána diagramem z obrázku 2-2. Prvním nepovinným krokem je předzpracování množiny vstupních bodů P^+. Předzpracování je nejčastěji chápáno jako soubor vstupních filtrů, které se starají o odstranění šumu ve vstupních datech, odstranění tzv. *outliers*[1] a případně i decimaci vstupní množiny bodů na množinu, která stále dostatečně reprezentuje rekonstruovaný objekt, ale je co do počtu bodů mnohem menší. Výsledkem předzpracování je nová množina vstupních bodů P, nad kterou bude prováděna rekonstrukce povrchu. Poté začíná samotný proces rekonstrukce. Nejprve je z množiny vstupních bodů P rekonstruován primární povrch. Základním stavebním blokem rekonstrukce primárního povrchu je právě Delaunayova tetrahedronizace, protože hledaný primární povrch je podmnožina jejích trojúhelníků. Primární povrch je již vizuálně podobný původnímu snímanému objektu, ovšem z topologického hlediska může být stále nedokonalý. Dalším krokem rekonstrukce je tedy extrakce manifoldu z primárního povrchu. Po tomto kroku již je k dispozici topologicky správný povrch, ve kterém je okolí každého bodu na povrchu homeomorfní k otevřenému Euklidovskému kruhu a okolí každého bodu na okraji povrchu homeomorfní k otevřenému Euklidovskému půlkruhu. Po tomto kroku končí samotný proces rekonstrukce. Můžou následovat další kroky, které jsou v diagramu shrnuty jako filtrace. Mezi předpokládané filtrace patří dodatečné zdokonalení povrchu zaměřené například na správnou rekonstrukci hran, rohů a okrajů a na záplatování děr.



Obr. 2-2 Diagram rekonstrukce povrchu využívající Delaunayovu tetrahedronizaci

[1] Outliers si lze představit jako nízkofrekvenční šum v datech. Tedy body vstupní množiny, které do výsledného objektu na první pohled nepatří.

Metoda CRUST představuje postup pro rekonstrukci primárního povrchu a následnou extrakci manifoldu. Pokud není splněno vzorkovací kritérium bude primární povrch s největší pravděpodobností obsahovat různé defekty, jako jsou přebývající trojúhelníky nebo naopak chybějící trojúhelníky, které mohou v případě obzvlášť špatné hodnoty *r* způsobit i rozpadnutí povrchu na více částí. Tyto defekty je nutné řešit v dalších krocích rekonstrukce. Samotná metoda CRUST existuje ve dvou variantách, které se liší ve způsobu extrakce primárního povrchu. Starší variantou je dvouprůchodový CRUST, novější pak jednoprůchodový CRUST.

2.1.1. Dvouprůchodový CRUST

První verze metody CRUST [4] byla nazvána dvouprůchodovou, protože obsahuje stavbu dvou datových struktur – Voronoiova diagramu a Delaunayovy triangulace[1]. Základní princip dvouprůchodové CRUST metody se opírá o tzv. Voronoiovo filtrování [3]. Přímá aplikace Voronoiova filtrování je možná jen v případě rekonstrukce tvaru ve 2D. Pro rekonstrukci povrchu ve 3D je nutné postup modifikovat.

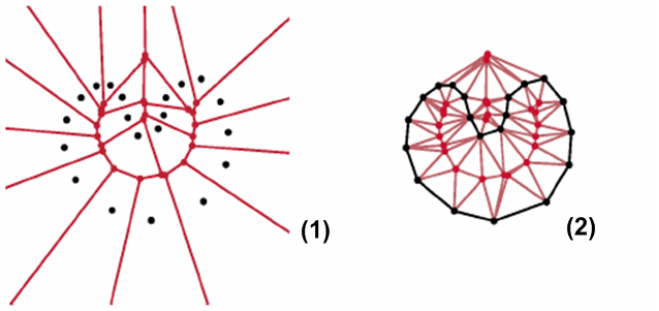
Voronoiovo filtrování ve 2D

Myšlenka rekonstrukce tvaru Voronoiovým filtrováním ve 2D je založena na vlastnostech *r-vzorku*. Platí, že pokud je vstupní množina *r-vzorek*, vrcholy Voronoiových buněk aproximují střední osu rekonstruovaného tvaru. Hodnota *r* může být pro tuto úlohu až 0,25. Z toho plyne, že spojnice (hrana) dvou bodů ze vstupní množiny *P* patří do rekonstruovaného tvaru právě tehdy, když kružnice opsaná této hraně neobsahuje žádný další bod z množiny *P* ani žádný bod z množiny vrcholů Voronoiových buněk *V*.

Rekonstrukce tvaru tedy nejprve nad množinou vstupních bodů *P* (množinou generátorů) vytvoří Voronoiův diagram, jehož vrcholy Voronoiových buněk tvoří množinu *V*. Voronoiův diagram lze také vytvořit jako duální strukturu k Delaunayově triangulaci zkonstruované na množině vstupních bodů *P* (tento postup je častější, a proto je v dalším textu předpokládán). Poté je zkonstruována druhá Delaunayova triangulace, jejíž vstup tvoří $P \cup V$. Výstupem rekonstrukce jsou všechny hrany z druhé triangulace, které spojují jen vrcholy z množiny *P*. Tyto hrany představují rekonstruovaný tvar (Obr. 2-3). Stejně jako v případě rekonstrukce

[1] Princip duality umožňuje jejich zaměnitelnost, takže se může jednat o dvě Delaunayovy tetrahedronizace nebo dva Voronoiovy diagramy.

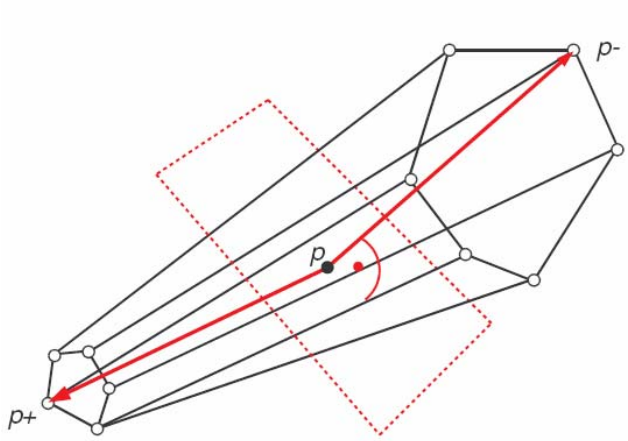
povrchu se zřejmě jedná jen o primární tvar, ze kterého bude dalším postupem určen finální tvar.



Obr. 2-3 Rekonstrukce tvaru dvouprůchodovou metodou CRUST. (1) Voronoiův diagram vstupní množiny bodů P. (2) Delaunayova triangulace množiny $P \cup V$ a zvýrazněný rekonstruovaný tvar, který tvoří jen hrany s vrcholy z množiny P. Obrázek je převzat ze [3].

Voronoiovo filtrování ve 3D

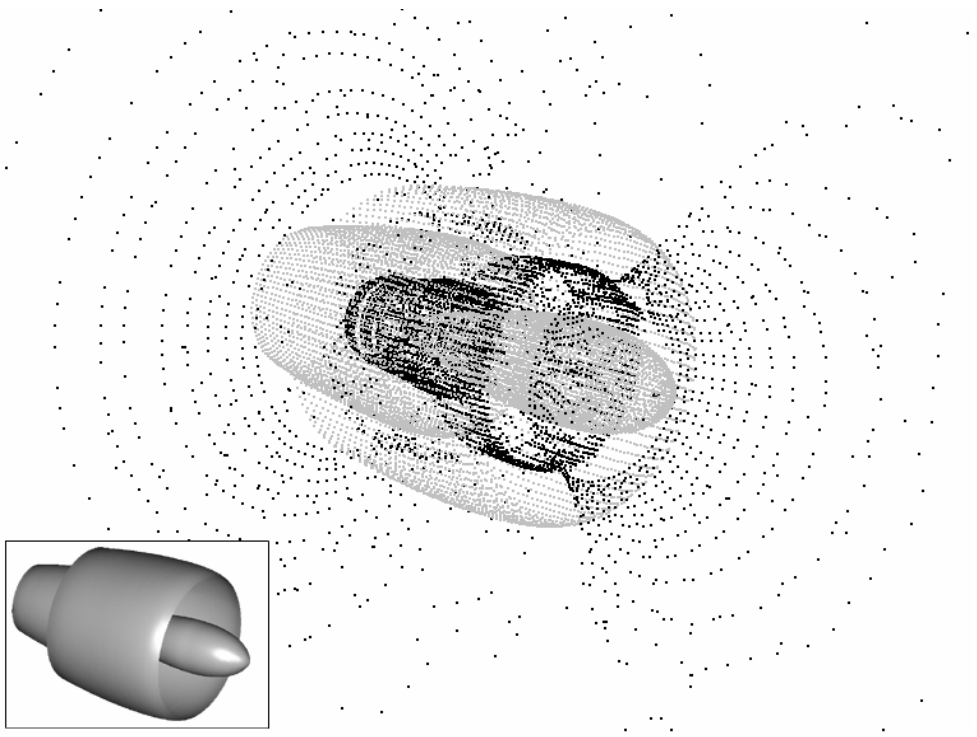
Využití Voronoiova filtrování pro rekonstrukci povrchu ve 3D vyžaduje úpravu postupu použitého ve 2D. Neplatí zde, že pro dobře vzorkovaný povrch (*r-vzorek*) aproximují všechny vrcholy z množiny V střední osu. Toto tvrzení lze ve 3D aplikovat pouze na podmnožinu vrcholů Voronoiova diagramu $V^* \subset V$. Oproti 2D variantě je tedy navíc nutné určit množinu V^* a teprve nad sjednocením vstupní množiny bodů P a množinou V^* vytvořit Delaunayovu tetrahedronizaci. Výsledný povrch je pak podmnožina trojúhelníků ze získané tetrahedronizace, které mají všechny vrcholy jen z množiny P.



Obr. 2-4 Voronoiova buňka generátoru p ve 3D. Vrcholy p+ a p– Voronoiovy buňky představují kladný a záporný pól generátoru p. Obrázek převzat ze [2].

Pro Voronoiovy buňky lze předpokládat, že budou velmi úzké, dlouhé a téměř ortogonální k hledanému povrchu. Budeme předpokládat, že jsou i ortogonální ke střední ose hledaného povrchu. Tyto předpoklady umožňují vybrat z každé Voronoiovy buňky až dva body nazývané póly, které patří do množiny V^*. Póly představují kladný a záporný extrém vůči

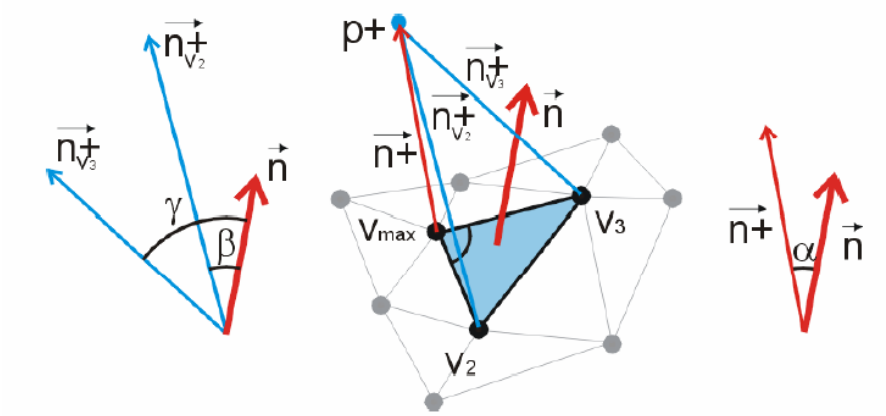
generátoru buňky $p \in P$. Kladný pól se běžně určuje jako nejvzdálenější bod Voronoiovy buňky od jejího generátoru p a je označován $p+$. Záporný pól označovaný $p–$ je určen jako nejvíce vzdálený vrchol Voronoiovy buňky od jejího generátoru p, který má navíc s kladným pólem zápornou hodnotu skalárního součinu. Vektory směřující z generátoru Voronoiovy buňky do kladného a záporného pólu se označují $n+$ a $n–$ (Obr. 2-4). Získané póly umožňují odvodit přibližný průběh hledaného povrchu bodem p. Vektor $n+$ pak slouží jako prvotní aproximace normály k povrchu v bodě p. V případě, že leží bod p na konvexní obálce hledaného povrchu, bude mít jeho Voronoiova buňka jen jeden pól $p–$, který vždy bude ležet uvnitř rekonstruovaného objektu. Určením pólů všech Voronoiových buněk (Obr. 2-5) je získána hledaná množina V^* a může být zkonstruována druhá Delaunayova tetrahedronizace, ze které jsou vybrány trojúhelníky, které mají všechny vrcholy z množiny P. Tyto trojúhelníky tvoří rekonstruovaný primární povrch.



Obr. 2-5 Ukázka vstupní množiny bodů (šedě) a množiny pólů (černě). Vlevo dole je ukázán rekonstruovaný povrch z množiny vstupních bodů.

Voronoiovo filtrování může mezi povrchové trojúhelníky zařadit i některé nevhodné trojúhelníky [1]. Již při extrakci primárního povrchu lze zvláštní skupinu těchto trojúhelníky identifikovat a odstranit je. Tuto skupinu tvoří velmi úzké trojúhelníky, které jsou kolmé na povrch. V předchozím kroku byly normály povrchových trojúhelníků aproximovány vektorem $n+$. Nyní je možné přidat volitelný test, který se pokusí tyto trojúhelníky odfiltrovat.

Kritérium pro filtraci bude odvozeno z odchylky normály trojúhelníků od normál $n+$ a $n-$. Nejprve se zjistí vrchol, u kterého má trojúhelník největší úhel. Pro tento vrchol je nalezen kladný pól $p+$ a určen normálový vektor $n+$. Pro zbývající vrcholy zkoumaného trojúhelníku jsou vytvořeny vektory jdoucí z těchto vrcholů k pólu hlavního vrcholu. Tyto vektory můžeme nazvat $n_{v2}+$ a $n_{v3}+$. Posledním důležitým krokem pro sestavení filtrovacího kritéria je určení vstupního parametru Θ. Trojúhelník je odfiltrován, pokud úhel α, který svírá jeho normála s vektorem $n+$ je větší než Θ nebo pokud úhly β a γ, které svírá jeho normála s vektory $n_{v2}+$ a $n_{v3}+$ jsou větší než $2{,}2\Theta$. Nastavení vstupního parametru Θ je kritické a může způsobit odfiltrování i správných trojúhelníků. Obecně by vstupní parametr Θ měl být závislý na r. Stávající rekonstruktor používá jako počáteční hodnotu 22,5°.



Obr. 2-6 Filtrace trojúhelníků podle normál. Vrchol v_{max} představuje vrchol, u kterého je nejvetší úhel zkoumaného trojúhelníku. K tomuto vrcholu je určen kladný pól $p+$ a normálový vektor $n+$. Od zbývajících vrcholů trojúhelníku jsou pak vedeny vektory ke kladnému pólu vrcholu v_{max}. Rozhodovací kritérium pro filtraci trojúhelníku je pak odvozeno od skutečné normály trojúhelníku a úhlů, které svírá se třemi získanými vektory a vstupního parametru Θ. Obrázek je převzat z [1].

2.1.2. Jednoprůchodový CRUST

Hlavní nevýhodou dvouprůchodové metody CRUST je vytváření druhé Delaunayovy tetrahedronizace, která má navíc přibližně třikrát tolik vstupních bodů než první Delaunayova tetrahedronizace. Pro vytvoření druhé tetrahedronizace je tedy potřeba mnohem více času a především mnohem více paměti. Vytváření druhé tetrahedronizace se přitom jeví nadbytečné, neboť povrchové trojúhelníky jsou obsaženy již v té první. Je tedy pouze nutné zvolit kritérium, podle kterého bude možné tyto trojúhelníky určit.

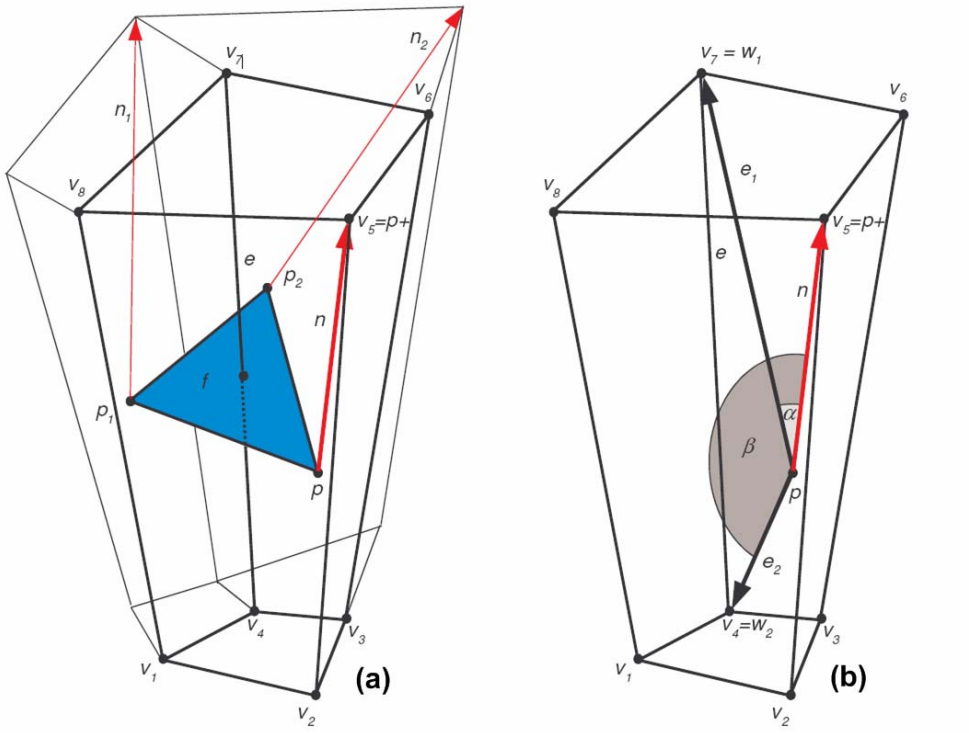
Nina Amenta se ve své další práci [5] vrátila k původním předpokladům, které musely splňovat Voronoiovy buňky, kterými prochází hledaný primární povrch u dvouprůchodové metody. Pokud hledaný povrch splňuje vzorkovací kriterium, jsou Voronoiovy buňky jejíchž stěny prochází tímto povrchem dlouhé, tenké a k hledanému povrchu téměř kolmé. Normálu k povrchu v daném bodě (generátoru Voronoiovy buňky) pak lze aproximovat stejně jako v dvouprůchodové metodě vektorem jdoucím z generátoru buňky do jejího kladného pólu. Z těchto poznatků a definice *r-vzorku* bylo sestaveno kriterium, které umožňuje vybírat povrchové trojúhelníky již z první tetrahedronizace. Kritérium je definováno následujícím teorémem (citace z [1], dále možno nalézt v [2], [5]):

> Nechť T je množina trojúhelníků povrchu S, přičemž T splňuje tyto tři podmínky:
>
> - T obsahuje pouze všechny trojúhelníky, jejíchž duální Voronoiovy hrany protínají S.
> - Každý trojúhelník v T je malý (poloměr kružnice opsané tomuto trojúhelníku je mnohem menší než vzdálenost vrcholu ke střední ose).
> - Všechny trojúhelníky v T jsou ploché (normály trojúhelníků mají pouze malé odchylky od normál povrchu ve vrcholech trojúhelníka).

Toto kriterium nám umožňuje provést Extrakci primárního povrchu jednoprůchodovou metodou následujícím postupem. Nejprve se zkonstruuje Delaunayova tetrahedronizace vstupní množiny bodů P a pro každý tetrahedron se spočítá střed kružnice opsané. Poté se pro každý bod vstupní množiny $p_i \in P$ určí všechny incidující tetrahedrony. Středy opsaných kružnic incidujících tetrahedronů k bodu p_i představují vrcholy Voronoiovy buňky generované bodem p_i. Ze získaných vrcholů Voronoiovy buňky lze stejným způsobem jako u dvouprůchodové metody určit kladný pól ($p+$) vrcholu p_i a tím získat aproximaci normály k hledanému povrchu v tomto bodě. Spojnice dvou Vornoiových vrcholů definovaných sousedícími tetrahedrony, které sdílí trojúhelník f, představuje hranu Voronoiovy buňky e, která je duálem k trojúhelníku f (Obr. 2-7a).

Nyní je třeba určit, zda trojúhelník f patří do hledané množiny T. Podle výše zmíněného kriteria pro rekonstrukci jednoprůchodovou metodou CRUST platí, že trojúhelník f je součástí hledaného povrchu S právě tehdy, když jeho duální hrana e protíná povrch S. Tento test lze provést velmi snadno, protože z předešlého kroku známe normály ve vrcholech trojúhelníka.

Zavedeme dva pomocné vektory e_1 a e_2, které budou směřovat z jednoho z vrcholů p trojúhelníku f do koncových bodů hrany e. Úhly, které svírají vektory e_1 a e_2 s normálou n v bodě p nazveme α a β. Pokud pro každý vrchol trojúhelníku protíná interval $<\alpha, \beta>$ interval $<\pi/2 - \Theta, \pi/2 + \Theta>$, pak je trojúhelník prvkem množiny T (Obr. 2-7b). Hodnota parametru Θ se výpočtu předává jako parametr a určuje rozmezí, ve kterém je ještě trojúhelník brán jako povrchový. V případě, že je Θ rovno nule, jsou přijaty jen trojúhelníky, které skutečně leží na povrchu. Získávání povrchových trojúhelníků bylo u dvouprůchodové metody nazván Voronoiovo filtrování [3] u jednoprůchodové metody může tento postup být nazván Delaunayovo filtrování [5].



Obr. 2-7 (a) Trojúhelník f a jeho vrcholy, které generují tři Voronoiovy buňky. Trojúhelníkem prochází jeho duální hrana e. Na obrázku jsou také vyznačeny normály ve vrcholech trojúhelníku, takže jsou vidět i kladné póly jednotlivých vrcholů. (b) Znázornění výpočtů úhlů α a β, které slouží k rozhodování o tom zda je či není trojúhelník povrchový. Úhlové kriterium musí být splněno pro všechny tři vrcholy trojúhelníku, aby byl přijat. Obrázek převzat z [2].

2.1.3. Extrakce manifoldu

Po získání primárního povrchu z jednoprůchodové nebo dvouprůchodové metody CRUST je třeba extrahovat manifold. Primární povrch může obsahovat různé defekty, které ho dělají topologicky nekvalitním. Mezi tyto defekty může patřit například nesprávný vějíř trojúhelníků

(*triangle fan*), překrývající se trojúhelníky, více než dva incidující trojúhelníky na jedné hraně atd. K odstranění těchto vad slouží právě tento krok. Tato část rekonstrukce povrchu může být považována za nejdůležitější a také nejhůře řešitelnou.

Extrakce manifoldu začíná nalezením tzv. startovacích trojúhelníků (ve skutečnosti stačí určit jen jeden startovací trojúhelník.) Startovací trojúhelníky jsou takové, o kterých můžeme s jistotou říci, že jsou součástí manifoldu. Hledání startovacích trojúhelníků je možné provádět pomocí tzv. deštníčků (*umbrella test*). Bod p z množiny vstupních bodů P má správný deštníček právě tehdy, když se průměty s ním incidujících trojúhelníků na rovinu definovanou tímto bodem a jeho normálou nepřekrývají. Tento test lze s jistotou provádět pouze na povrchu konvexní obálky, protože jen zde je možné správně určit směr normálového vektoru.

Normálu trojúhelníku (P_1,P_2,P_3), který leží na konvexní obálce lze určit z tetrahedronu, kterému trojúhelník tvoří jednu jeho stěnu. Jelikož se jedná o trojúhelník z konvexní obálky, bude takový tetrahedron existovat jen jeden. Normálový vektor startovacího trojúhelníku pak lze určit pomocí čtvrtého bodu tetrahedronu, který může být značen P_4. Rozhodnutí o orientaci trojúhelníku je provedeno skalárním součinem jeho normály a vektoru z jakéhokoliv vrcholu P_1,P_2,P_3 do vrcholu P_4. Pokud je tento skalární součin kladný, směřují oba vektory stejným směrem, tedy dovnitř objektu a je nutné normálu invertovat.

Počítání průmětů a jejich překrývání je matematicky náročné. Test deštníčků může být nahrazen podstatně jednodušším testem [1]. Pro vrchol p se nejprve určí incidující trojúhelníky a pro každý z těchto trojúhelníků se vyberou hrany, které obsahují bod p. Pokud má trojúhelník na každé takové hraně jen jednoho souseda, je test splněn.

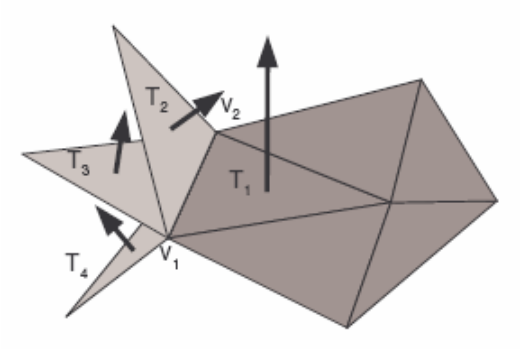
Dalším krokem extrakce je orientace všech trojúhelníků. Vstupem pro orientování je množina startovacích trojúhelníků, které jsou již správně orientovány. Další trojúhelníky jsou vyhodnocovány prohledáváním do hloubky. Postupně se vyhodnocují trojúhelníky, které sousedí s některým již vyhodnoceným a jejich normála je nastavena tak, aby směřovala stejným směrem.

Po dokončení orientace trojúhelníků může být provedena extrakce manifoldu. Extrakce probíhá postupným procházením jednotlivých trojúhelníků. Na každé hraně aktuálního

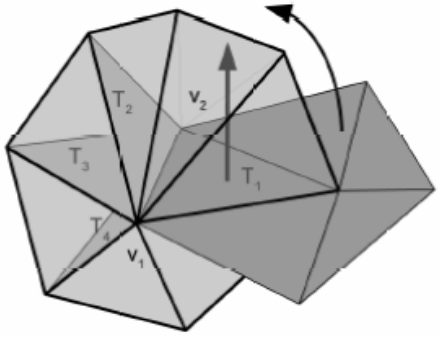
trojúhelníku se vyhodnotí jeho sousedi a ponechá se pouze ten, který je nejblíže předpokládaného povrchu. Toto kritérium je jedna z klíčových vlastností celého rekonstruktoru. Špatná volba kritéria způsobí hroší nebo dokonce nepoužitelné výsledky rekonstrukce. Základní strategie pro extrakci správných trojúhelníků jsou popsané v [2]. Jedná se o tři různé způsoby ohodnocování sousedů:

- Nejlepší trojúhelník svírá se stávajícím nejmenší úhel (Obr. 2-8)
- Nejlepší trojúhelník je v prvním nalezeném tetrahedronu (Obr. 2-9)
- Nejlepší trojúhelník obsahuje nejkratší hranu z dostupných trojúhelníků.

Ve volbě kritéria se liší i práce [1] a [2]. Práce [1] využívá první metodu výběru, práce dva využívá třetí metodu výběru. První metoda je numericky nestabilní, pokud je rekonstruovaný povrch téměř plochý. Primární povrch pak obsahuje tzv. *slivers* (ploché tetrahedrony), složené z trojúhelníků, pro které téměř není možné odlišit úhly. První dvě metody nejsou vhodné, kvůli jejich závislosti na svíraných úhlech mezi sousedními trojúhelníky.



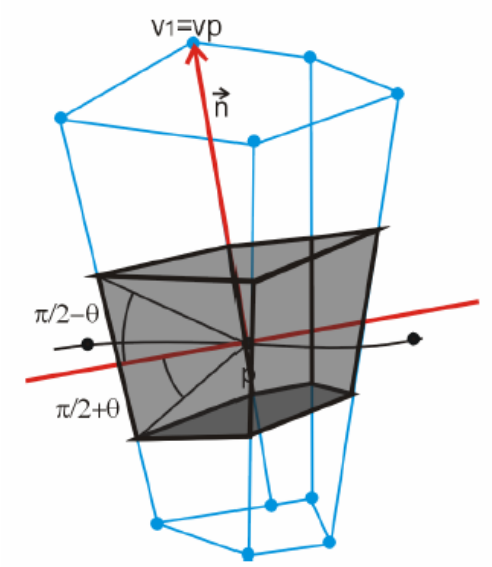
Obr. 2-8 V předešlých krocích extrakce manifoldu byl trojúhelník T_1 označen jako součást hledaného povrchu. Nyní je třeba se rozhodnout, který z jeho sousedů na hraně v_1v_2 bude vybrán jako další trojúhelník z hledaného povrchu. Úhlové kriterium vybere trojúhelník T_2, protože svírá s trojúhelníkem T_1 nejmenší úhel. Obrázek je převzat z [2].



Obr. 2-9 Situace je stejná jako na obrázku 2-8. Tetrahedronové kritérium nejprve určí startovní tetrahedron, který musí obsahovat trojúhelník T_1 a být ve směru jeho normály. Od startovního tetrahedronu jsou postupně procházeni sousední tetrahedrony, které obsahují hranu v_1v_2, dokud není nalezen první tetrahedron, který by obsahoval některého z možných sousedů T_1 – ten je pak přijat. Obrázek je převzat z [2].

2.1.4. Cocone

Dey a spol. publikovali velice podobný postup rekonstrukce primárního povrchu jako v jednoprůchodové CRUST metodě, který byl nazván COCONE [8]. Podobnost těchto metod není náhodná, protože Nina Amenta je spoluautorkou teoretických podkladů. Cocone je geometrický objekt, který je odvozen z tvaru příslušné Voronoiovy buňky (Obr. 2-10). Postup metody COCONE je rozdělen do tří kroků. V prvním kroku probíhá tzv. extrakce kandidátů. Kandidáti jsou trojúhelníky z Delaunayovy tetrahedronizace, které projdou přes filtrování využívající Cocone jednotlivých bodů vstupní množiny. Tento krok je obdobou Dalaunayova filtrování používaného v jednoprůchodové CRUST metodě.

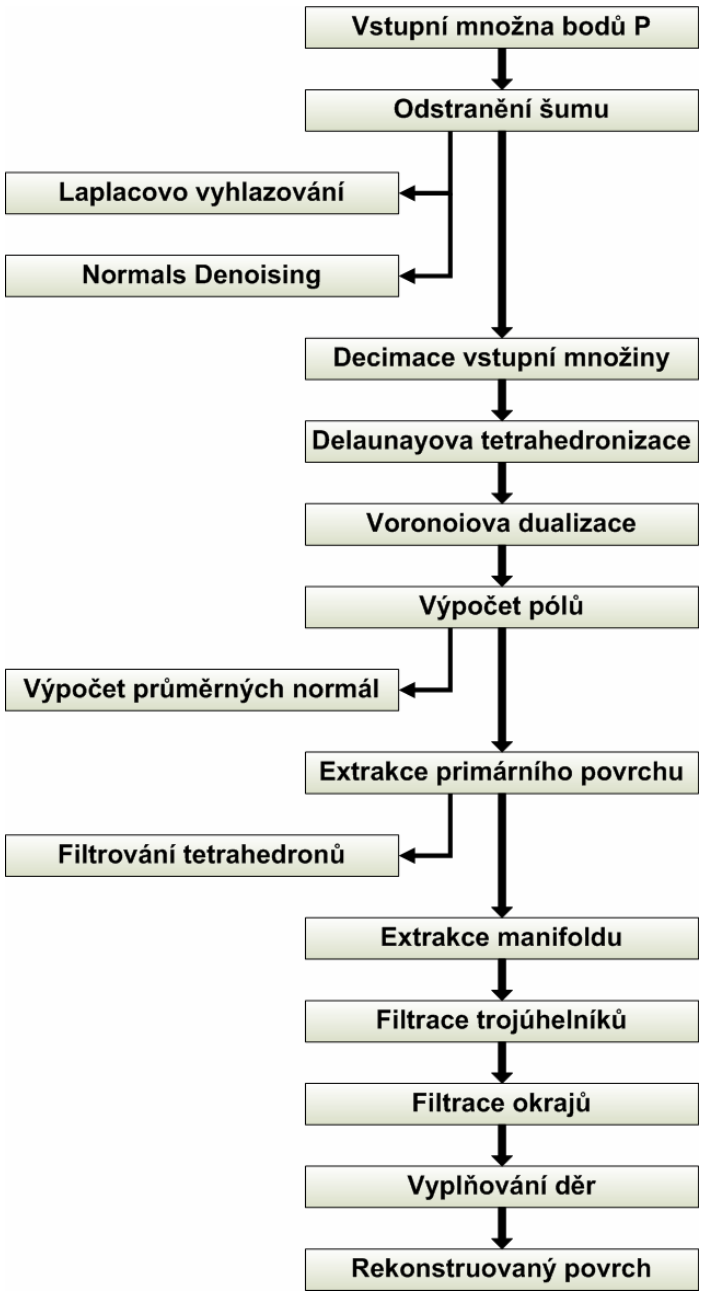


Obr. 2-10 Deyův Cocone (vystínovaná oblast) vytvořený na Voronoiově buňce bodu p. Vystínovaná oblast vznikla jako doplněk k průniku Voronoiovy buňky a dvou protilehlých kuželů umístěných vrcholem v bodě p. Úhel, který svírá normála se stěnou kužele u vrcholu p je dán vstupním parametrem Θ. Obrázek převzat z [1].

Po prvním kroku je získán primární povrch, který ovšem ještě netvoří manifold. Extrakci manifoldu představuje druhý krok nazvaný prořezávání (*pruning*) a třetí krok nazvaný procházení (*walk*). Během prořezávání jsou postupně odstraněny trojúhelníky, které vytvářejí se svým sousedem ostrou hranu, tj. svírají úhel větší než $3\pi/2$. V případě, že je hrana obsažena jen v jednom trojúhelníku, je také brána jako ostrá a trojúhelník je odstraněn. Základní verze metody COCONE tedy neumožňuje extrakci povrchů, které obsahují okraje. V posledním kroku rekonstrukce se prohledáváním do hloubky získá souvislý povrch.

2.1.5. Vylepšení jednoprůchodové metody CRUST

Aktuální implementace rekonstrukce metodou CRUST, která je dostupná na Západočeské univerzitě obsahuje několik vylepšení, které jsou popsány v [2]. Průběh rekonstrukce touto upravenou metodou je znázorněn na Obr. 2-11. Jednotlivá vylepšení budou blíže vysvětlena v této kapitole.



Obr. 2-11 Schéma aktuální implementace rekonstrukce povrchu jednoprůchodovou metodou CRUST, dostupné na Západočeské univerzitě.

Odstranění šumu

Každé snímací zařízení pro získání datové reprezentace objektu z reálného světa je zatíženo nějakou chybou a má nějaké technické omezení. Tyto vlastnosti vstupních zařízení způsobují,

že je ve vstupních datových množinách pro rekonstrukci šum. V popisu metody CRUST bylo zmíněno, že není vhodná pro rekonstrukci zašuměných datových množin, protože téměř jistě nesplňují vzorkovací kritérium. To lze považovat za velkou nevýhodu, protože metoda je tím pádem použitelná pouze na rekonstrukci „syntetických" modelů. Prvním krokem upravené metody je tedy potlačení tohoto šumu.

Laplacovo vyhlazování

K potlačení šumu je možné využít dva různé postupy. První postup je založen na Laplacově vyhlazování, které je ovšem nutné použít v jeho nejjednodušší podobě, protože zatím není znám vyhlazovaný povrch. Cílem je pouze zmenšit odchylky v pozici bodů vůči jejich nejbližším sousedům. Pro každý bod p vstupní množiny P se volá transformace, která vypočítá jeho tzv. centroid c. Centroid se použije jako nová poloha bodu p.

$$c = \frac{p + \sum_{i=1}^{k} p_i}{k+1}$$

Zvětšování hodnoty k přináší lepší vyhlazení modelu, ovšem za cenu ztráty detailů. Příliš malé k pro změnu nemusí stačit k odstranění šumu. Celý proces vyhlazování lze volat iterativně, přičemž jednou iterací je míněna transformace všech bodů ze vstupní množiny P. Výsledky pokusů v [2] ukazují, že není třeba více než jedna iterace (větší počet iterací vytváří shluky bodů) a počet sousedů je vhodné volit okolo dvaceti.

Normals Denoising

Druhý postup pro odstranění šumu byl nazván *Normals Denoising* [2]. Tento postup vyžaduje pro každý bod p vstupní množiny nalézt centroid c a tečnou rovinu v centroidu c pro k nejbližších sousedů původního bodů p. Centroid je určen stejně jako u Laplacova vyhlazovaní. Z polohy sousedů bodu p a polohy centroidu c je možno sestavit matici kovariance C, pro kterou jsou určeny vlastní čísla a vlastní vektory, které jsou dále použity pro vyhlazování dat. Vlastní vektory odpovídající dvěma největším vlastním číslům představují popis největší odchylky v datech. Tyto dva vektory definují tečnou rovinu, která je nejlepší aproximací použitých k bodů. Nyní je možné udělat průmět bodu p a každého z k sousedů bodu p na získanou rovinu. Opakováním tohoto postupu pro každý bod vstupní množiny získáme jednu iteraci vyhlazování. Na rozdíl od Laplacova vyhlazování tato metoda nezpůsobuje shlukování bodů, a tak lze použít i více iterací pro získání lepších výsledků. Nejlepší výsledky [2] dosahovala metoda pro k rovno dvacet.

$$C = \begin{bmatrix} \sum_{i=1}^{k}(p_{ix}-c_x)^2 & \sum_{i=1}^{k}(p_{ix}-c_x)(p_{iy}-c_y) & \sum_{i=1}^{k}(p_{ix}-c_x)(p_{iz}-c_z) \\ \sum_{i=1}^{k}(p_{ix}-c_x)(p_{iy}-c_y) & \sum_{i=1}^{k}(p_{iy}-c_y)^2 & \sum_{i=1}^{k}(p_{iy}-c_y)(p_{iz}-c_z) \\ \sum_{i=1}^{k}(p_{ix}-c_x)(p_{iz}-c_z) & \sum_{i=1}^{k}(p_{iy}-c_y)(p_{iz}-c_z) & \sum_{i=1}^{k}(p_{iz}-c_z)^2 \end{bmatrix}$$

Tato metoda obsahuje i některé na první pohled skryté výhody. Pro každý bod vstupní množiny je možné ukládat vektor, o který byl posunut v průběhu odstraňování šumu. Po dokončení rekonstrukce povrchu je možné aplikovat inverzní posunutí, které vrátí vrcholy do původní polohy. Získáme tím rekonstrukci povrchu na původní nevyhlazené datové množině. Díky tomu je v některých případech možné správně rekonstruovat i objekty s hranami a rohy.

Decimace vstupní množiny

Vzhledem k omezením, které klade na vstupní datovou množinu použitá Delaunayova tetrahedronizace, je nutné některé datové množiny zmenšit. Nejjednodušším řešením je iterativní decimace vstupní datové množiny. V každé iteraci jsou spojeny dva nejbližší body ze vstupní množiny. Pozice nového bodu odpovídá průměru z pozic předcházejících dvou bodů. Tento postup se opakuje dokud není dosaženo cílového počtu bodů.

Výpočet průměrných normál

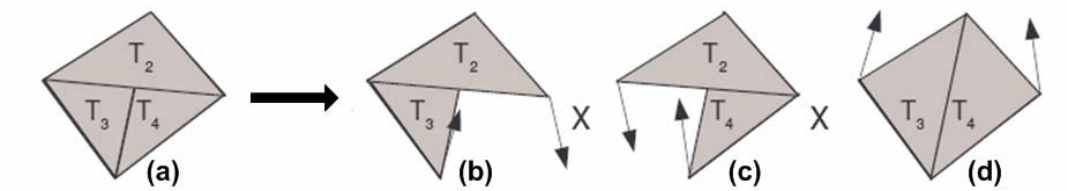
Běžná CRUST metoda rekonstrukce povrchů předpokládá, že Voronoiova buňka procházející povrchem je tenká, dlouhá a téměř kolmá k tomuto povrchu. Normála se pak dá aproximovat jako vektor jdoucí z generátoru buňky do nejvzdálenějšího Voronoiova bodu této buňky (kladného pólu). Podmínkou pro tyto předpoklady je dostatečné dobré vzorkování, tedy vstupní množina musí být *r-vzorek*. To ovšem na reálných datech nebývá splněno. Voronoiova buňka pak nesplňuje předpoklady a normála v jejím generátoru je aproximována špatně.

Tento nedostatek lze odstranit výpočtem průměrné normály. Původní aproximace normály se bere pouze jako normála dočasné plochy, která dělí buňku na dvě poloviny. Skutečná aproximace normály se určí jako součet vektorů jdoucích z generátoru Voronoivy buňky do všech Voronoiových vrcholů, které leží ve stejné polorovině jako kladný pól. Výsledný vektor

není po dobu rekonstrukce normalizován, protože by tím všechny vektory, ze kterých byl složen, získaly stejnou váhu. To ovšem není žádoucí, protože vektory do vzdálenějších Voronoiových vrcholů představují kvalitnější aproximaci než vektory do bližších Voronoiových vrcholů.

Filtrování tetrahedronů

Při extrakci manifoldu můžou být odstraněny některé trojúhelníky, které pak způsobí malé díry ve výsledném povrchu. Tento jev může vzniknout pokud primární povrch obsahuje tři trojúhelníky z jednoho plochého tetrahedronu. Jeden z nich pak překrývá zbylé dva, které ovšem vyplňují větší prostor než on sám (Obr. 2-12a). Pokud je při extrakci vybrán tento trojúhelník, jsou zbylé dva trojúhelníky odstraněny a může vzniknout díra.



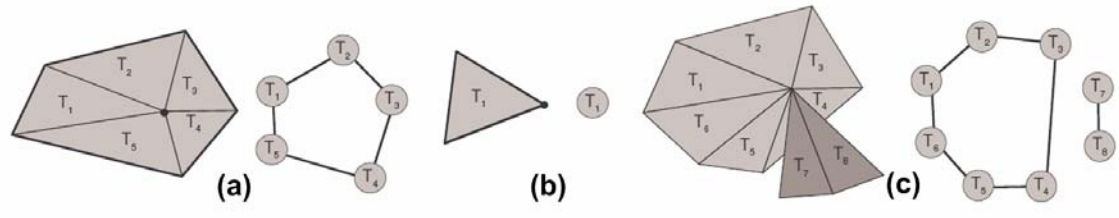
Obr. 2-12 (a) Tři trojúhelníky z jednoho plochého tetrahedronu. Pokud bude při extrakci manifoldu přijat trojúhelník T_2 budou v dalším průběhu trojúhelníky T_3 a T_4 odstraněny, což může způsobit vznik díry v trojúhelníkové síti. (b),(c),(d) Filtrace špatného trojúhelníku probíhá na základě normál. Ponechány jsou ty dva trojúhelníky, jejichž normály svírají nejmenší úhel. Obrázek převzat z [2].

Aby k tomuto jevu nedocházelo, je nutné před začátkem extrakce manifoldu spustit filtraci tetrahedronů, která projde všechny tetrahedrony a zjistí pro které z nich byly vybrány tři trojúhelníky. Jeden z těchto trojúhelníků je poté odstraněn. Výběr trojúhelníku k odstranění je prováděn na základě úhlu mezi normálami. Ponechány jsou ty dva trojúhelníky, jejichž normály svírají nejmenší úhel (Obr. 2-12b-d).

Filtrování trojúhelníků

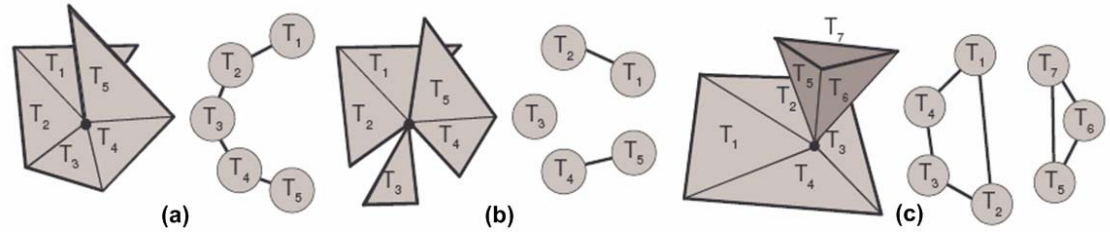
Po extrakci manifoldu dosud zmiňovaným způsobem může model stále obsahovat konfigurace trojúhelníků, které by manifold obsahovat neměl. Extrakce se zabývá jen řešením problémů, které mohou vzniknout na hranách trojúhelníkové sítě. Manifold ovšem klade požadavky i na konfigurace, které mohou obklopovat jednotlivé body trojúhelníkové sítě. Je tedy nutné zavést další krok, který bude provádět analýzu vějíře trojúhelníků (*triangle fan*) kolem každého bodu extrahované trojúhelníkové sítě. Každý vějíř trojúhelníků lze popsat neorientovaným grafem. Uzly grafu představují trojúhelníky okolo zkoumaného vrcholu a

hrany grafu představují sousednost mezi trojúhelníky. Pouze dva typy konfigurací jsou automaticky považovány za správné. První automaticky přijatý typ lze znázornit uzavřeným cyklem v grafu (Obr. 2-13a). Druhým automaticky akceptovatelným typem je jeden uzel. Tato varianta je možná, jen pokud je vrchol trojúhelníků rohem nějakého okraje (Obr. 2-13b).



Obr. 2-13 Trojúhelníky incidující se zkoumaným bodem lze popsat neorientovaným grafem. (a) a (b) Představuje správné konfigurace. (c) Představuje špatnou konfiguraci, protože zde vzniká bodový spoj. Obrázek je převzat z [2].

Odstraňování nadbytečných trojúhelníků je jednoduché, pokud v grafu nalezneme jeden cyklus. Všechny trojúhelníky mimo tento cyklus jsou odstraněny. Větší pozornost je nutné věnovat konfiguracím, které obsahují částečný vějíř (*sub fan*). Tyto konfigurace je nutné promítnout na tečnou rovinu ve zkoumaném bodě. Na průmětu je vypočtena suma všech úhlů, které mezi sebou svírají sousedící trojúhelníky u zkoumaného vrcholu. Pokud je suma menší než 2π je konfigurace přijata. V opačném případě (Obr. 2-14a). je nutné odstranit některé okrajové trojúhelníky z vějíře tak, aby se suma úhlů zmenšila pod 2π. Nejhorší konfigurace obsahují více oddělených částí vějíře (Obr. 2-14b). nebo několik cyklů (Obr. 2-14c).. První situaci lze řešit ponecháním té části, jejíž průmět na tečnou rovinu ve zkoumaném bodě má největší sumu úhlů přilehlých ke zkoumanému bodu. Druhá situace je považována za kritickou chybu, protože o jejím řešení nelze automaticky rozhodnout.



Obr. 2-14 (a) Špatná konfigurace, která je odhalena při projekci na tečnou rovinu zkoumaného bodu. Pro její opravu je nutné odstranit trojúhelník T_1 nebo T_5. (b) Problémová konfigurace kterou je možné řešit jen ponecháním jedné části grafu a smazání ostatních. Ponechává se ta část, jejíž průmět na tečnou rovinu zkoumaného bodu má u bodu největší sumu úhlů. Zde se bude jednat o trojúhelníky T_1 a T_2. (c) Kritická konfigurace, kterou není možné automaticky rozhodnout. Obrázek převzat z [2].

Filtrace okrajů

Častým vstupem do rekonstrukce jsou modely, které obsahují okraje. Běžná implementace metody CRUST si s takovými modely neumí správně poradit a považuje okraj za podvzorkovanou oblast. Tato oblast je poté vyplněna trojúhelníky, které v povrchu zůstanou i po extrakci manifoldu. Rekonstrukci okrajů je tedy nutné provést dodatečně. Stávající implementace za tímto účelem definuje speciální ohodnocovaní funkci, která na základě délky hran a svírajících úhlů rozhodne, které trojúhelníky budou odstraněny.

Filtrace okrajů je prováděna na všech bodech vstupní množiny. Pro každý bod jsou určeny všechny hrany, které s ním incidují a z těchto hran je vybrána jedna referenční. Trojúhelník incidující k právě zkoumanému bodu je odstraněn, pokud jsou délky obou jeho hran, které obsahují zkoumaný bod, delší než délka referenční hrany vynásobené konstantou m. Konstanta m je určena podle následujícího vzorce:

$$m = c_{be} + c_{se}\left|n_p.n_t\right|$$

Vzorec se skládá z konstant c_{be} a c_{se}, normálového vektoru zpracovávaného bodu n_p a normálového vektoru testovaného trojúhelníku n_t. Konstanta c_{be} představuje maximální povolenou délku hrany, pokud je úhel mezi normálami 90°. Druhá konstanta upravuje váhu úhlu v celkovém výpočtu. Nejlepší hodnoty byly pro testované modely odvozeny empiricky [2], $c_{se} = 5.0$ a $c_{be} = 2.0$.

Problém může představovat určení referenční hrany. Pokud je tato hrana určena špatně, nemusí docházet k filtraci nebo je naopak odfiltrováno příliš mnoho trojúhelníků. V práci [2] bylo provedeno několik experimentálních pokusů s výběrem referenční hrany a jako nejlepší se ukázala metoda, která vybírá medián ze všech hran incidujících se zkoumaným vrcholem a všech hran incidujících se sousedy zkoumaného vrcholu.

Vyplňování děr

V tomto kroku se dá říci, že byly provedeny všechny úpravy, které mají za cíl vytvořit povrch, který se co nejvíce podobá původnímu objektu reprezentovanému vstupní datovou množinou. Poslední krok, který je možné provést, je vyplnění děr, které vznikly v průběhu rekonstrukce nebo v průběhu dodatečných úprav. Většina děr, které rekonstruovaný povrch obsahuje, má podobu několika chybějících trojúhelníků. Proto nemá význam používat některou

z existujících robustních a složitých metod určených pro vyplňování složitých děr či spojování částí různých objektů.

Před vyplňováním je nutné nalézt všechny díry. Při hledání děr se postupně zkoumají všechny trojúhelníky a dále se zpracovávají jen ty, které na některé hraně nemají souseda. Na této hraně se vybere startovní vrchol a podle incidujících hran se postupuje dále po obvodu díry, dokud není celá díra ohraničena. Problém může nastat, pokud některý z vrcholů ležící na okraji díry je zároveň součástí i nějaké další díry. V tomto případě je nutné rozhodnout, kterou hranou se vydat dále, protože s vrcholem inciduje více než jedna hrana obsažená jen v jedenom trojúhelníku. Tato situace se řeší vytvořením dělicí roviny, která je dána poslední zpracovanou hranou a normálou aktuálního vrcholu. Rovina dělí model do dvou poloprostorů. Hrany mezi kterými je vybíráno, mohou být posuzovány vzhledem k této rovině. Nejprve je snaha najít hranu v poloprostoru, kde se nalézá díra. Z tohoto poloprostoru se vybírá hrana, která s předešlou zpracovanou hranou svírá nejmenší úhel. Pokud zde žádná taková hrana není, vybere se z druhého poloprostoru hrana, která s předešlou zpracovávanou hranou svírá největší úhel.

Poté co jsou nalezeny všechny díry, je nutné zvážit, co je ještě považováno za díru a co je již bráno jako okraj. Aktuální implementace bere jako díru vše, co má po obvodu méně než padesát hran. Poté může začít vyplňování nalezených děr. Pro vyplňování je použita jednoduchá rovinná metoda pro triangulaci polygonu „odřezáváním uší“. Přesný popis této metody lze nalézt v [16]. Jelikož se jedná o rovinou metodu, je nejprve nutné díry promítnout do roviny. Tato operace se nemusí zdařit pro každou díru. Některé mohou být příliš tvarově komplikované, takže se jejich průmět v rovině bude různě překrývat. Tyto díry nejsou ve stávajícím programu nijak řešeny.

2.1.6. Jiné úpravy metody CRUST

Souběžně s vývojem úprav popsaných v [2] se autoři původních metod také snažili o jejich úpravy. Nina Amenta předvedla značně upravenou metodu nazvanou PowerCRUST [6], která je schopná pracovat i s tenkými modely a s modely, které obsahují okraje. Později bylo uvedeno i vylepšení metody PowerCRUST [7], které je schopné pracovat se zašuměnými daty. Další vývoj zaznamenala metoda COCONE. Thamal Dey předvedl variantu nazvanou SuperCocone [9], která je schopná pracovat nad extrémně velkými datovými množinami. Data jsou nejprve rozdělena pomocí stromové struktury OctTree a každý list tohoto stromu je

zpracováván samostatně. Aby bylo možné jednotlivé části pospojovat, pracuje se v každém listu i s body jeho sousedů, které jsou „dostatečné“ blízko ke zpracovávanému listu. Další úpravou metody COCONE je TightCocone [10], který umí rekonstruovat i tenké povrchy. Zatím poslední varianta této metody byla nazvána RobustCocone [11] a je určená pro rekonstrukci ze zašuměných dat.

2.2. Další metody pro rekonstrukci povrchu

Jedním z hlavních vstupů této práce byl i dokument [1] (diplomová práce Ing. Michala Varnušky PhD.), který je přímo zaměřen na metody rekonstrukce povrchu. Teoretická část tohoto dokumentu obsahuje rozsáhlý přehled a popis různých metod pro rekonstrukce povrchů. Proto zde tyto metody nebudou dále rozebírány. Zmíněna bude pouze Hoppeho metoda, která patří mezi první algoritmy pro rekonstrukci povrchu z bodů. Dále budou zmíněny dvě zajímavé metody, které ve zmiňovaném dokumentu nejsou uvedeny.

2.2.1. Metoda „Hoppe et al.“

Jednou z prvních a poměrně známých metod pro rekonstrukci povrchu z bodů byl algoritmus, který uvedl Hoppe ve své disertační práci [12]. Původní Hoppeho metoda využívá dělení prostoru pravidelnou třírozměrnou mřížkou, ve které je vyhodnocována pozice každého voxelu vůči hledanému povrchu. Povrch je poté rekonstruován metodou Contour Tracing, která patří mezi jednu z metod pro extrakci povrchu z volumetrických dat. Některé implementace této metody využívají pro rekonstrukci povrchu metodu Marching Cubes, ovšem autor v původním textu [12] tuto variantu zavrhuje, protože Contour Tracing údajně produkuje lepší výsledky.

Obtížnou částí tohoto postupu je právě vyhodnocení pozice jednotlivých voxelů, protože je nutné vytvořit jakýsi prvotní popis hledaného povrchu, podle kterého by byly vrcholy voxelů ohodnocovány znaménkovou funkcí. Tento prvotní popis se vytváří odhadem tečné roviny k hledanému povrchu v každém bodu vstupní množiny. Tento algoritmus produkuje pouze aproximaci povrchu reprezentovaného vstupní množinou bodů, což je jeho hlavní nevýhoda. Další nevýhodou je zahlazování ostrých hran a rohů.

2.2.2. Metoda využívající MPU implicitní plochy

Zajímavě vypadá metoda, kterou představil H. Xie a kol. [14]. Tato metoda volí zcela jiný přístup a nepoužívá Delaunayovu tetrahedronizaci. Nad množinou vstupních bodu je nejprve vytvořeno dělení prostoru pomocí OctTree. Hloubka OctTree je nastavována podle úrovně detailu a požadovaného odstranění šumu. Poté jsou vrcholy tvořící OctTree shlukovány metodou aktivní kontury do tzv. mono-orientovaných skupin. Mono-orientované skupiny se vyznačují tím, že všechny jejich prvky jsou buď vně a nebo uvnitř rekonstruovaného povrchu. V tuto chvíli se ovšem nedá říci, která skupina představuje vně a která je uvnitř. V dalším kroku je každým listem OctTee, který je mezi více mono-orientovanými skupinami proložena implicitní kvadrika. Používají se tzv. MPU implicitní plochy (kombinace radiálních bázových funkcí a algebraických plátů). Následuje rozhodování o orientaci jednotlivých mono-orientovaných skupin, z čehož je poté odvozena orientace jednotlivých implicitních ploch a posléze i normály. Pokud se ukáže, že implicitní plocha nebyla proložená správně, může se celý proces iterativně opakovat. Posledním krokem metody je prolnutí jednotlivých implicitních ploch a rekonstrukce povrchu z volumetrických dat – používá se metoda Marching Cubes. Detailní znázornění postupu metody v 2D případě je ukázáno na obrázku 2-15.

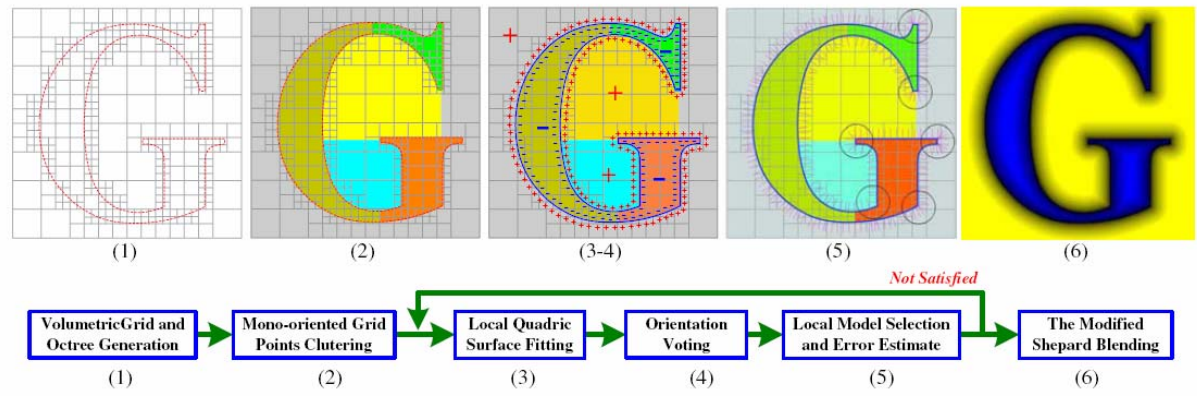
Výsledky prezentované autory vypadají velmi zajímavě stejně jako tvrzení o vlastnostech zmiňovaného algoritmu. Algoritmus je údajně schopný rekonstruovat data ze zašuměných a poškozených datových množin, má robustní postup pro volbu orientace a odstraňování *outliers*. Dále je schopný rekonstruovat detaily a ostré hrany. Využití OctTree v sobě nese i vestavěnou podporu pro proměnnou úroveň detailu (*level of detail*). Stejně jako v předchozí metodě je výsledný povrch pouze aproximací vstupní množiny bodů. To je ovšem u práce ze zašuměnými daty vcelku očekávané.

2.2.3. Metoda “pokrývání kuličkami“

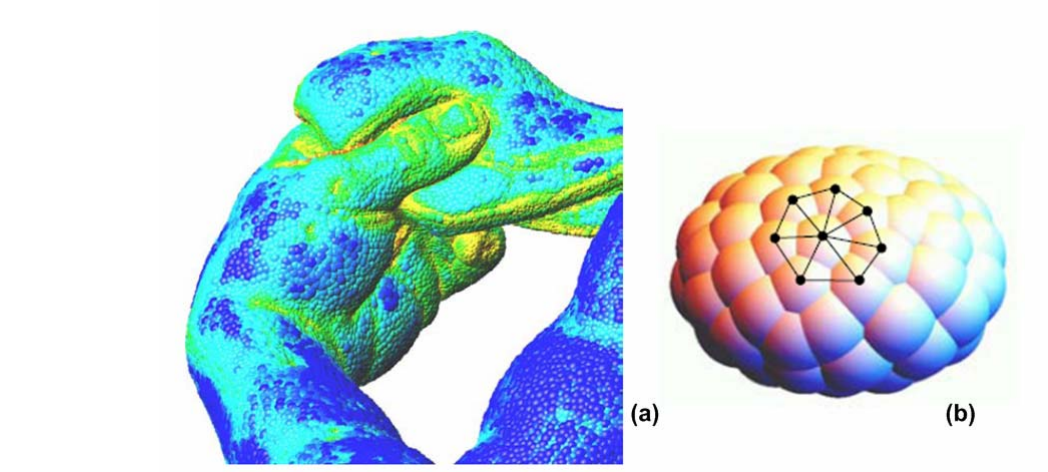
Další zajímavý přístup k rekonstrukci povrchu představil Y. Ohtake a kol. [15]. Opět se jedná o metodu, která nepoužívá Delaunayovu tetrahedronizaci. Proces rekonstrukce je rozdělen do tří kroků. První krok představuje předzpracování, ve kterém je každý vrchol vstupní množiny ohodnocen vahou, jejíž výpočet se odvíjí i od hustoty vzorkování v dané oblasti a od případného šumu v datech. Ke každému vrcholu vstupní množiny je také určena jeho normála.

Po tomto kroku přichází konstrukce primárního povrchu. Nejprve je pro každý vrchol vstupní množiny vytvořena obalující koule, jejíž poloměr závisí na vstupních parametrech rekonstruktoru a váze daného bodu (Obr. 2-16a), Trojúhelníková síť je z původních bodů vystavěna podle průniků obalových koulí (Obr. 2-16b). Poslední krok představuje extrakci manifoldu a vyplňování případných děr.

Metoda je podle autorů schopná rekonstruovat objekty ze zašuměných dat, dále umí rekonstruovat detaily a ostré hrany.



Obr. 2-15 Postup metody z kapitoly 2.2.2. pro 2D případ. Obrázek převzat z [14].



Obr. 2-16 Obrázky k metodě z kapitoly 2.2.3. (a) Pro každý vstupní bod je vygenerována obalující koule, jejíž poloměr je definován vahou daného bodu a vstupními parametry. Barvy naznačují rozdílné poloměry. (b) Ukázka jakým způsobem je ze systému koulí konstruována trojúhelníková síť. Obrázky převzaty z [15].

3. Metody umělé inteligence

Umělá inteligence (UI) je odvětvím informatiky a kybernetiky, které se zabývá inteligentním chováním, učením a přizpůsobováním počítačových programů. Vývoj v oblasti umělé inteligence se zaměřuje na vytvoření strojů / programů, které jsou schopny automaticky řešit úlohy, jejichž řešení vyžaduje schopnosti, které by v případě řešení člověkem byly považovány za projev jeho inteligence. Mezi tyto úlohy patří např. řízení, plánování, rozpoznávání a pro tuto práci důležité optimalizování. V této kapitole se blíže zaměříme na teoretický popis některých technik, které by mohly být zajímavé pro problém rekonstrukce a optimalizace trojúhelníkových sítí. Konkrétní možnosti využití těchto technik bude popsáno v následující kapitole.

Hlavním zdrojem informací použitých pro sepsání této kapitoly byly první tři díly série knih *Umělá inteligence* [19], [20], [21] nakladatelství Academia a internetové stránky předmětu *Umělá inteligence a rozpoznávání* [web1] vyučovaném na Západočeské univerzitě. Pro kapitolu věnovanou Genetickým algoritmům byla hlavním zdrojem kniha *An Introduction to Genetic Algorithms* [18] nakladatelství MIT Press.

3.1. Terminologie

Strojové řešení optimalizační úlohy je možné jen v případě, že je vytvořen její formální popis. Jako formální popis úlohy může sloužit stavový model S používaný v oblasti Řešení úloh (*Problem Solving*) [19]. Každá hodnota optimalizovaného objektu je ve stavovém modelu charakterizována jako stav s, který má kvalitu $|s|$. Množina všech možných stavů $L = \{s_i\}$ se nazývá stavový prostor. Přechod mezi stavy ze stavového prostoru je možný jen aplikací elementárního operátoru ϕ_i, definovaného v množině operátorů $\Phi = \{\phi_i\}$. Stavový model S je definován stavovým prostorem a množinou operátorů.

$$S = (L, \Phi)$$

Stavový model S lze také popsat jako orientovaný graf, jehož uzly představují jednotlivé stavy stavového prostoru L a hrany představují přechody mezi stavy způsobené aplikací operátorů z množiny Φ.

Řešení úlohy popsané stavovým modelem představuje posloupnost aplikovaných operátorů z množiny Φ, které způsobí přechod z počátečního stavu s_0 do některého koncového stavu g_i, definovaného v množině koncových stavů $G = \{g_i\}$ (řešení úlohy je tedy ekvivalentní hledání cesty v orientovaném grafu). Množina koncových stavů G je podmnožinou stavového prostoru L. Koncové stavy nemusí být pevně definovány, mohou být definovány pouze omezující podmínky – klasický případ pro optimalizační úlohu, kde je cílem získat stav jehož kvalita představuje globální optimum z kvalit všech stavů ve stavovém prostoru L.

$$g_i \in G \ \wedge \ G \subset L \ \wedge \ |g_i| = \max\{|s_i|\}$$

Získání globálního optima je hlavním problémem optimalizačních úloh. Běžné metody řešení mají tendenci skončit při nalezení lokálního optima. Dalším problémem může být obtížné nalezení globálního optima kvůli rozsáhlosti stavového prostoru. Často se proto spokojíme s co možná nejlepší aproximací globálního optima.

3.2. Řešení úloh

Řešení úloh (*Problem Solving*) [19], [web1] je technika založená na systematickém prohledávání stavového prostoru L. K prohledávání se váže další termín, tzv. expanze stavu. Expanze aktuálního stavu znamená odvození jeho následovníků, které je možné získat aplikací operátorů z množiny Φ na aktuální stav. Prohledávání lze pak chápat jako určité pořadí, ve kterém jsou stavy expandovány, dokud není nalezen stav, který splňuje cílové podmínky. Konkrétní řešení úlohy je možné znázornit stromovým grafem, který ukazuje, v jakém pořadí byly uzly expandovány a kde byl nalezen cílový stav. Efektivitu, neboli rychlost konvergence dané metody, lze chápat jako podíl mezi potřebnými expanzemi stavů pro získání cílového stavu a celkovým počtem expanzí stavů. Pro porovnávání celkové efektivity různých metod je nutné brát v úvahu i algoritmickou složitost vyhodnocování kvality jednoho stavu.

Prohledávání stavového prostoru může být prováděno neinformovaným (slepým) nebo informovaným (cíleným) způsobem. Neinformované metody nejsou z pohledu umělé inteligence příliš zajímavé, protože při prohledávání stavového prostoru nevyužívají kvalitu stavů. Mezi tyto metody patří algoritmy prohledávání do hloubky (*Depth-First Search*) [19], [web1] a prohledávání do šířky (*Breadth-First Search*) [19], [web1]. Informované metody obecně expandují nejdříve uzly s vyšší kvalitou. Lze je rozdělit do dvou základních skupin. První skupina využívá při prohledávání stavového prostoru samotnou kvalitu stavu, druhá

skupina algoritmů vyžaduje dodatečnou znalost řešené úlohy, která má urychlit prohledávání. Ve většině případů představují dodatečnou znalost empiricky odvozené poznatky týkající se řešeného problému. Takto získaná znalost se nazývá heuristika nebo též heuristická funkce. Skutečná kvalita stavu f_i je pak vyjadřována jako součet základní kvality stavu $|s_i|$ a hodnoty heuristické funkce h_i pro daný stav.

$$f_i = |s_i| + h_i$$

Rychlost konvergence je závislá na vhodnosti použité heuristické funkce pro konkrétní problém. V některých případech může být heuristická funkce jediným způsobem, jak stav ohodnotit. Nevhodně zvolená heuristika pak může způsobit, že prohledávání stavového prostoru L nepovede k řešení úlohy. Do této kategorie úloh spadá i optimalizace trojúhelníkových sítí.

Příkladem informovaného algoritmu využívajícího heuristickou funkci je algoritmus A* [19], [web1]. Tento algoritmus se běžně používá pro hledání cesty mezi dvěma místy na diskrétní mapě. Stavový prostor L představuje celá mapa, množina operátorů Φ jsou všechny možné elementární pohyby po mapě. Kvalita stavu $|s_i|$ při hledání cesty se určuje jako náročnost (cena) přesunu z počátečního stavu do aktuálního stavu. Heuristická funkce h_i použitá pro A* algoritmus představuje vzdálenost aktuálního stavu od cílového stavu vzdušnou čarou. Součet těchto dvou hodnot představuje skutečnou kvalitu stavu. Pro další výpočty se vždy bere v úvahu jen nejnižší skutečná kvalita f_i, se kterou byl stav dosažen. Tento algoritmus vždy s velkou rychlostí konvergence najde nejlepší možnou existující cestu z počátečního do cílového stavu.

3.2.1. Gradientní algoritmus

Gradientní neboli horolezecký algoritmus (*hill-climbing algorithm*) [19], [web1] patří mezi nejjednodušší informované metody prohledávání stavového prostoru L. Princip hledání globálního optima je v případě tohoto algoritmu velmi přímočarý. Algoritmus začíná expanzí aktuálního (počátečního stavu). Z možných následovníků je vybrán vždy ten, který má nejvyšší kvalitu a ten je přijat jako nový aktuální stav. Ostatní stavy včetně předešlého aktuálního stavu jsou zapomenuty. Z toho plyne, že horolezecký algoritmus nevytváří strom řešení. Není tedy možné zpětně získat posloupnost operátorů z množiny Φ, která způsobila

přechod z počátečního do cílového stavu. Toto přímočaré řešení je jednoduché na pochopení i na implementaci, ovšem nese sebou mnoho problémů.

Nejhorším problémem tohoto algoritmu je uvíznutí (zacyklení) v lokálním optimu, které je způsobeno samotným výběrem dalšího aktuálního stavu. Ve chvíli kdy je dosaženo nějaké optimum, začne algoritmus cyklovat mezi tímto optimem a jeho nejkvalitnějším následovníkem. Proto je nutné při spouštění výpočtu předat algoritmu maximální možný počet stavů, které mohou být při řešení úlohy expandovány, aby bylo možné ukončit jinak nekonečný cyklus. Cyklování se dá potlačit úpravou algoritmu, při které se nejkvalitnější následovník použije jako nový aktuální stav jen tehdy, pokud je kvalitnější než současný aktuální stav. V opačném případě algoritmus končí a současný aktuální stav je vrácen jako nalezené optimum. Tato úprava ovšem v tuto chvíli nebude brána v úvahu, protože na původní verzi se lépe vysvětluje jeho následující úprava.

3.2.2. Algoritmus prohledávání se zakázaným seznamem

Hlavní nevýhodou gradientního algoritmu je vysoká pravděpodobnost uvíznutí v lokálním maximu. Tento problém se snaží řešit úprava tohoto algoritmu nazývaná Prohledávání se zakázaným seznamem (*Tabu search algorithm*) [web1]. Původní algoritmus je doplněn o speciální paměť typu FIFO (fronta) pevné velikosti *k*, nazývanou zakázaný seznam. Tato paměť udržuje inverzní operátory k operátorům, které byly použity pro výběr následovníka v předchozích *k* krocích algoritmu. Výběr následujícího aktuálního stavu je prováděn stejným způsobem jako v případě gradientního algoritmu s tím rozdílem, že mezi možné následovníky nejsou zahrnuty stavy, které jsou dosaženy přes operátory obsažené v zakázaném seznamu. Tato úprava má zabránit zacyklení algoritmu. Po vybrání následovníka je inverzní operátor k použitému operátoru vložen do zakázaného seznamu. V případě, že je délka zakázaného seznamu větší něž *k*, je z něj odstraněn operátor, který je v něm obsažen nejdéle.

Princip úlohy tedy spočívá v možnosti opustit dosažené optimum a vydat se ke stavům, které mají horší kvalitu s nadějí, že bude nalezeno lepší optimum. Pro správné použití tohoto algoritmu je nezbytné vhodně určit velikost *k* zakázaného seznamu. Příliš malé *k* nemusí stačit k opuštění lokálního optima. Naopak příliš velké *k* může způsobit přeskočení globálního optima. Pro správné určení *k* je tedy nutné provést důkladnou analýzu úlohy a poté testovat, se kterými hodnotami dává řešení nejlepší výsledky. Ztrátě nejlepšího nalezeného optima může

zabránit úprava algoritmu, ve které je navíc přidána pomocná proměnná uchovávající nejlepší nalezené optimum.

3.3. Stochastické metody

Velmi účinným nástrojem pro optimalizační úlohy jsou stochastické metody, které při řešení úlohy využívají jistou míru náhody. Síla těchto metod se projevuje především ve chvíli, kdy i úplná expanze jednoho stavu je tak náročná, že úlohy na ní postavené jsou časově téměř neřešitelné. Stochastické metody využívají pro výběr následujícího stavu náhodný jev. Do rozhodování zda bude náhodně vygenerovaný stav přijat jako nový aktuální stav, lze dále přidat další náhodný jev (např. pro pravděpodobnostní metody).

3.3.1. Stochastický gradientní algoritmus

Nejjednodušší stochastickou metodou je upravená verze gradientního algoritmu. Originální verze předpokládá úplnou expanzi aktuálního stavu a následný výběr nejkvalitnějšího následovníka. Oproti tomu stochastická verze neprovádí expanzi stavu, ale vygeneruje jen několik následovníku vytvořených náhodně vybranými operátory z množiny Φ, které lze na aktuální stav aplikovat. Z těchto následovníků je opět ponechán ten nejkvalitnější. Stochastickou verzi lze dokonce zredukovat na generování jednoho náhodného následovníka, jehož kvalita je porovnávána proti kvalitě aktuálního stavu a jako aktuální stav je ponechán kvalitnější z těchto dvou stavů. Pro tuto verzi ovšem není možné použít ukončení prohledávání, pokud v daném kroku nebyl nalezen kvalitnější následovník.

Další možnost úpravy původního gradientního algoritmu předpokládá jeho spouštění původní varianty v cyklu. Každý cyklus je spuštěn s náhodně vygenerovaným počátečním stavem a mezi cykly se uchovává nejlepší dosažené oprimum. Tato verze je ovšem značně pomalá, protože iterací bývá většinou více (v závislosti na typu úlohy).

3.3.2. Simulované žíhání

Simulované žíhání (SA) [22] je velmi často používaná metoda pro řešení globální optimalizace nad velkým stavovým prostorem. Název i princip této metody je odvozen od techniky, která se běžně používá v metalurgii a vychází z principů termodynamiky. Pokud chceme ze surového materiálu získat kvalitní kov, musíme ho nejprve zahřát na vysoké teploty, aby přešel do tekutého stavu. Poté je nutné materiál vhodně ochlazovat, aby se

zlepšila struktura jeho krystalické mřížky. Pokud je ochlazování příliš rychlé, nestačí se krystaly v krystalické mřížce správně vyrovnat a materiál nebude mít požadované vlastnosti – vnitřní energie materiálu bude příliš vysoká a materiál tak bude např. křehký. Proto se používá tzv. žíhání, materiál je nejprve zahřát na vysokou teplotu a poté začne být řízeně ochlazován. Pokud se v materiálu začnou vyskytovat defekty, je opět trochu zahřán a řízené ochlazování pokračuje. Řízené ochlazování dává čas nalézt krystalům správnou konfiguraci v krystalické mřížce a minimalizovat tak vnitřní energii materiálu, dodatečné zahřívání umožňuje vyskočit z lokálně dosaženého optima.

Simulované žíhání je analogií k tomuto procesu. Algoritmus začíná v počátečním stavu, který může být náhodně vygenerován. Dále je nastaveno několik vstupních parametrů algoritmu. Prvním je počáteční teplota, která udává zahřátí žíhané úlohy. Dále je zadána koncová teplota, tedy teplota při jejímž dosažení řešení úlohy končí a koeficient, který udává, jak se bude teplota v jednotlivých krocích žíhání snižovat. Posledním parametrem je počet optimalizačních kroků, které se mají provést pro každou úroveň teploty. Průběh algoritmu lze popsat pseudokódem 3-1., který minimalizuje hodnotu stavu.

```
teplota := pocatecniTeplota;

WHILE (teplota > minTeplota) DO
BEGIN
  FOR (i = 0 TO maxOptimalizaci) DO
  BEGIN
    novyStav := NahodneGenerujNovyStav(aktualniStav);

    IF (Hodnota(aktualniStav) > Hodnota(novyStav))
    BEGIN
      aktualniStav = novyStav;
    END IF
    ELSE
      /* 0 <= nahodneCislo <= 1 */
      nahodneCislo := GenerujNahodneCislo();
      rozdilHodnot := Hodnota(novyStav) - Hodnota(aktualniStav);

      IF (nahodneCislo <= e^-rozdilHodnot/teplota)
      BEGIN
        aktualniStav := novyStav;
      END IF
    END ELSE
  END FOR

  teplota := teplota * koeficient;
END WHILE
```

Pseudokód 3-1. Simulované žíhání, hledající globální minimum mezi hodnotami stavů.

Algoritmus tedy pro každou úroveň teploty provede pevně daný počet pokusů o optimalizaci aktuálního stavu, poté je teplota vynásobena koeficientem, jehož hodnota musí být z intervalu (0;1). Vnější cyklus pokračuje dokud není aktuální teplota nižší než minimální teplota předaná jako vstupní parametr algoritmu.

V každém pokusu o optimalizaci je náhodně vygenerován nový stav *s´*. Náhodné generování probíhá výběrem náhodného operátoru z množin všech operátorů *Φ*, který je možné aplikovat na aktuální stav *s*. Pro nový stav *s´* se zjistí jeho hodnota |*s´*| a porovná se s hodnotou aktuálního stavu |*s*|. Jelikož výše popsaná metoda předpokládá hledání globálního minima (obdoba žíhání z metalurgie, kde se hledá minimální vnitřní energie materiálu), je za lepší hodnotu považována ta menší. Pokud má nový stav lepší hodnotu, je automaticky přijat a cyklus pokračuje.

Do této chvíle se algoritmus choval podobně jako stochastický gradientní algoritmus. Rozdílnost se projevuje až u zpracování nového stavu, který nemá lepší hodnotu. Pro tento stav je vygenerováno náhodné číslo z intervalu <0;1> a toto číslo je porovnáno proti pravděpodobnostní funkci. Pravděpodobnostní funkce udává s jakou pravděpodobností bude přijat nový stav, pokud je jeho hodnota vyšší než hodnota aktuálního stavu. Její hodnota musí být při minimalizační úloze pro záporné hodnoty stavů z intervalu <0;1> a pro kladné hodnoty stavů musí být větší nebo rovná jedné. Hodnota funkce se vždy určuje z hodnot nového a aktuálního stavu a z aktuální teploty. Pro simulované žíhání je nezbytné, aby s klesající teplotou byly i hodnoty pravděpodobnostní funkce menší, tj. aby s klesající teplotou byly horší stavy přijímány mnohem méně. Nejpoužívanější pravděpodobnostní funkcí pro simulované žíhání je tzv. Metropolitova funkce, která je použita i v pseudokódu 3-1.

$$pst = e^{-\frac{|s'|-|s|}{teplota}}$$

3.4. Genetické algoritmy

Jedním z nejmocnějších ovšem také nejsložitějších nástrojů pro řešení optimalizační úlohy jsou genetické algoritmy [21]. Genetické algoritmy patří do skupiny tzv. evolučních výpočetních technik, které se snaží simulovat vývojové procesy známé z přírody. Evoluční výpočetní techniky využívají stejně jako stochastické metody náhodné jevy, jejich princip je ovšem mnohem složitější a propracovanější. Kromě genetických algoritmů patří mezi

evoluční techniky také genetické programování, evoluční strategie a evoluční programování. Tato práce se zaměřuje pouze na genetické algoritmy, popis dalších evolučních metod lze nalézt ve [21]. Jelikož jsou genetické algoritmy analogií ke skutečné genetice, na které je založena celá biologie, budou nejprve vysvětleny biologické termíny (převzato z [18] a [web4]) využívané i pro počítačovou analogii.

3.4.1. Biologická terminologie

Základem všech živých organismů jsou buňky. Všechny buňky obsažené v jednom organismu se skládají ze stejné sady jednoho nebo více chromozomů – řetězce DNA, které tvoří „předlohu“ pro organismus. Chromozomy lze rozdělit na elementární části, které se nazývají geny. Konkrétní pozice genu v chromozomu se nazývá lokus. Každý gen tvoří funkční blok DNA, kterým je zakódován příslušný protein. Např. si lze představit, že lidský organismus obsahuje gen, kterým je zakódovaná barva očí každého jedince. Každá hodnota, která může být v daném genu zakódována se nazývá alela.

Složitější organizmy mají v každé buňce více různých chromozomů. Konkrétní popis organismu představuje jeho genotyp. Genotyp je sestavení řetězce DNA, tedy všechny obsažené geny, jejich lokus a alely. Změna lokusu některého z genů již představuje jiný genotyp.

Chromozomy mohou být v organismu uspořádány do párů (buňky organizmu se pak nazývají diploidní, tj. mají dvojnásobek chromozomů) nebo se mohou vyskytovat samostatně (buňky se pak nazývají haploidní). Organismy, u kterých probíhá reprodukce rozmnožováním mají zpravidla chromozomy v párech, např. buňky lidského organismu mají 23 párů chromozomů, tedy 46 chromozomů celkem. Reprodukce organismů jejichž chromozomy tvoří páry, se provádí tzv. gamety. Gameta je speciální haploidní pohlavní buňka, která má jen polovinu chromozomů obsažených v původním diploidním organismu. Tyto chromozomy vznikají křížením (cross-over) původních chromozomů v diploidní buňce rodiče. Při tomto procesu dochází k předání genetického materiálu rodiče – jedná se o hlavní zdroj genetické rozmanitosti. Celý proces reprodukce je zakončen pospojováním (rekombinací) chromozomů z gamet obou rodičů zpět do párů, tak aby v každý pár obsahoval chromozomy z obou rodičů. Vzniká tak diploidní buňka. Reprodukce u organismů, jejichž chromozomy netvoří pár probíhá stejně jako cross-over u gametů. Kromě reprodukce (křížení) může během evolučního

procesu docházet i k mutaci, kdy jsou elementární části DNA náhodně (nebo v případě genetického inženýrství cíleně) změněny.

Celý evoluční proces (vývoj organismů) je řízen kvalitou jednotlivých organismů. Kvalita organismu představuje pravděpodobnost přežití organismu a jeho schopnost se dále rozmnožovat. Organismy jsou v evolučním procesu zvýhodňovány či znevýhodňovány právě podle jejich schopnosti přežít a rozmnožovat se, což vede k přežití a adaptaci jednotlivých druhů. Tento jev se nazývá selekce.

3.4.2. Realizace genetických algoritmů

Předchozí sekce popsala, jakým způsobem funguje evoluční proces v přírodě. Nyní je třeba tento proces převést do strojové reprezentace. Základní charakteristika genetických algoritmů tedy předpokládá, že v danou chvíli je k dispozici větší množství dílčích řešení úlohy (stavů ze stavového prostoru řešené úlohy) reprezentovaných jako chromozomy. Kompletní množina aktuálně dostupných chromozomů je nazývána generace. Nad aktuální generací jsou prováděny genetické operátory, které způsobí přechod k nové generaci, která posléze nahradí generaci předchozí a proces se opakuje. Výpočet končí, pokud jsou splněny cílové podmínky úlohy nebo je vygenerován limitní počet generací. Při vhodně zvolené ohodnocovací funkci a správně odvozených genetických operátorech, se v generacích objevují stále kvalitnější jedinci – dochází k evoluci. Po dokončení výpočtu je jako řešení úlohy považován chromozom, který má z poslední generace nejvyšší kvalitu. Základní genetické operátory používané při řešení úlohy jsou křížení, mutace a selekce. První pokusy s genetickými algoritmy obsahovaly i operátor inverze části chromozomu, ovšem ukázalo se, že většině řešených úloh tento operátor způsobuje spíše potíže. Proto tento operátor nevystupuje sám, ale může být aplikován jako zvláštní typ mutace.

3.4.3. Nejjednodušší verze genetických algoritmů

Nejjednodušší verze aplikace genetických algoritmů se snaží kopírovat přírodní proces evoluce haploidních organismů. Chromozom je reprezentován jako bitový řetězec. Možné alely pro každý gen jsou tedy jen 0 nebo 1. Všechny chromozomy musí mít stejnou délku. Pro chromozom je dále určena ohodnocovací funkce, která určuje jeho schopnost přežít a dále se rozmnožovat (jeho kvalitu). Tato funkce je zcela závislá na řešené úloze. Nad bitovými řetězci se velice lehce sestavují genetické operátory.

Operátor selekce lze realizovat např. pomocí „rulety“. Ruletu si lze představit jako koláčový graf, který představuje rozmezí od 0 do 2π (plný úhel). Každému chromozomu aktuální generace přísluší dílek tohoto grafu, který je přímo úměrný jeho kvalitě. Chromozomy s vyšší kvalitou tak zabírají větší část ruletového kola (jejich výseče mají větší úhel). Pro operátory mutace a křížení se chromozomy vybírají tímto ruletovým kolem. Pokud je třeba vybrat chromozom, vygeneruje se náhodné číslo z intervalu <0;2π> (roztočí se ruleta) a zjistí se, ke kterému chromozomu daná hodnota v koláčovém grafu patří. Tento chromozom je vybrán. Chromozomy s větší kvalitou mají tedy větší šanci, že budou vybrány.

Aplikace operátoru křížení v nejjednodušší variantě genetických algoritmů je analogií ke křížení haploidu v přírodě. Selekcí jsou vybrány dva chromozomy a náhodně je určen lokus, ve kterém budou chromozomy roztrženy a jejich díly vzájemně zaměněny tak, aby vznikly opět dva stejně dlouhé chromozomy. Operátor mutace je implementován jako změna alely na náhodně zvoleném lokusu.

Operátory křížení a mutace nejsou volány samovolně. Vstupním parametrem genetických algoritmů musí být procentuálně udaný počet křížení a mutací, které se mají provést při vytváření nové generace. Počet křížení se obvykle pohybuje nad šedesáti procenty, počet mutací obvykle nepřekročí pět procent. Zbylé pozice v nové generaci jsou obsazeny chromozomy z předchozí generace. Tyto chromozomy mohou být zvoleny náhodně operátorem selekce nebo mohou být automaticky zvoleny chromozomy s nejvyšší kvalitou, případně je možné varianty kombinovat. Zde je nutné zvolit cestu „menšího zla“. První varianta může způsobit nenávratnou ztrátu nejkvalitnějších chromozomů, druhá může způsobit uvíznutí v lokálním optimu.

3.4.4. Rozšířené verze genetických algoritmů

Existují dva přístupy k řešení úloh genetickými algoritmy. První přístup předpokládá, že se úloha formalizuje tak, aby bylo pro její řešení možné využít nejjednodušší formu genetických algoritmů. Samotné použití genetických algoritmů je pak velmi jednoduché a hodně se podobá přírodní analogii. Bohužel převést složité úlohy na reprezentace v binárních řetězcích je většinou velmi těžké, ne-li nemožné. Druhý způsob řešení naopak předpokládá, že formulace úlohy zůstane stejná a přizpůsobovat se budou genetické operátory. Toto je mnohem častější řešení, nicméně u složitých úloh je také velmi obtížné.

Nejjednodušší genetický algoritmus tvoří základní stavební blok, pro všechny další úpravy, protože princip metody zůstává stejný. Ve většině případů se stále zůstává u reprezentace v podobě haploidu. Diploidy jsou využívány velice zřídka. Příklad pro použití diploidu lze nalézt v [18], kde se tento způsob používá pro optimalizaci počtu potřebných porovnání při řazení 16ti čísel. Jednotlivé chromozomy pak představují posloupnost indexů od 1 do 16. Při propojení těchto chromozomů vzniká diploid, kde alely genů na stejném lokusu představují jedno porovnání při operaci řazení. Posloupnost těchto dvojic tedy reprezentuje celý proces řazení. Selekce a ohodnocování kvality je prováděno vždy na celém páru chromozomů. V odborných materiálech lze nalézt i zmínky o komplikovanějších konfiguracích chromozomů, jedná se např. o triploid či komplikovanější propojení obecně nazývaná polypoid. Tyto konfigurace lze výjimečně nalézt i v přírodě, ale jedná se zpravidla o genetickou vadu.

Úpravy mohou být provedeny i na operátoru mutace. Nejjednodušší algoritmus obsahuje pouze jednoduchou mutaci, která způsobí změnu alely genu na náhodně zvoleném lokusu. Výše byla také zmíněna verze mutace, která provádí inverzi části chromozomu. Mezi další možnosti patří např. odstranění genu, přidání genu nebo vložení kopie části chromozomu.

V některých úlohách řešených genetickými algoritmy lze v pozdějších generacích pozorovat jakési vyrovnávání kvalit jednotlivých chromozomů. Tento jev nemusí být vždy žádoucí, např. pro optimalizační úlohy to znamená uvíznutí v aktuálním lokálním optimu. Tento jev se nazývá koevoluce a jeho odstranění je poměrně složité. V publikaci [18] je tento problém řešen vytvořením jakési ekvivalence virů, která napadá optimalizovanou úlohu a znehodnocuje tak dosažená řešení. Tyto viry představují druhou množinu řešenou genetickými algoritmy. Jednotlivé prvky množiny tvoří malé podřetězce chromozomů ze skutečně řešené úlohy. Ohodnocovací funkce řešené úlohy pak upravuje kvalitu chromozomu podle množství virů, které obsahuje. Úloha má tak snahu se dále vyvíjet stejně jako viry, které chtějí napadat i nově vznikající chromozomy. Jedná se o analogii závodu ve zbrojení.

Hlavní nevýhodou použití genetických algoritmů je doba jejich výpočtu a případně i paměťová náročnost. Pro složitější úlohy, od kterých očekáváme kvalitní výsledky se jedná minimálně o hodiny. Pokud je k uložení jednoho řešení úlohy potřeba velké množství paměti, nastává další problém, protože pro použití genetických algoritmů je třeba mít k dispozici

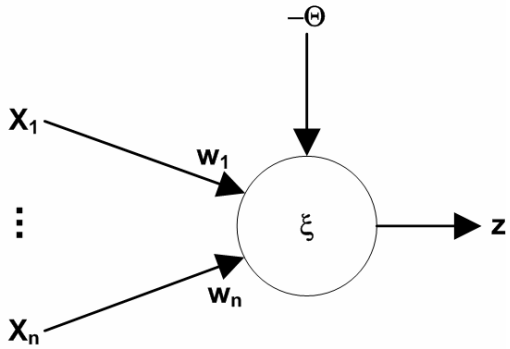
celou generaci, tedy stovky až tisíce řešení. Tyto nedostatky do značné míry řeší vysoká míra paralelizace výpočtu.

3.4.5. Optimalizace genetickými algoritmy

Řešení optimalizační úlohy genetickými algoritmy představuje evoluci počáteční generace stavů ze stavového prostoru úlohy, jejíž cílem je získat jedince (stav), který má nejlepší kvalitu. Získání počáteční generace stavů může být provedeno dvojím způsobem. Lze použít náhodně vygenerované stavy ze stavového prostoru nebo lze použít výsledky z nějaké jiné metody. Druhá možnost vytvoření počáteční generace stavů zvyšuje šanci na úspěch a snižuje předpokládanou dobu optimalizace, protože výsledky pravděpodobně již budou splňovat některé požadavky na kvalitu stavu. Většinou je ovšem nutné tyto možnosti zkombinovat, protože generace by měla mít řádově stovky až tisíce jedinců, což bývá příliš mnoho pro vytvoření v jiné aplikaci. Určení počáteční generace lze považovat za inicializaci optimalizačního procesu. Po dokončení inicializace je možné spustit optimalizaci.

3.5. Neuronové sítě

Další metoda umělé inteligence, která představuje obdobu procesů z biologie jsou umělé neuronové sítě [19] (dále jen neuronové sítě). Neuronová síť je síť mnoha jednoduchých procesorů (umělých neuronů) uspořádaných do konkrétní topologie. Hlavní vlastností neuronové sítě je reakce na vnější a vnitřní podněty a její přizpůsobování (adaptace) – tato vlastnost se dá někdy také chápat jako učení.



Obr. 3-1. Schéma perceptronu.

3.5.1. Perceptron

Nejjednodušší model neuronové sítě obsahuje jen jeden neuron. Tento model se nazývá perceptron (Obr. 3-1). V tomto neuronu se zbíhá *n* spojů, které se nazývají analogicky k biologii axony. Axony obecně představují vstup z jiných neuronů nebo podněty z okolního

prostředí. Axony přichází do neuronu vstupní stimuly x_i (příznaky). Podnět v dané chvíli lze tedy charakterizovat jako vektor příznaků ze všech axonů $x = [x_1, ..., x_n]$. S každým axonem je navíc spojena příslušná váha w_i, která určuje důležitost příznaku, který tímto axonem přichází. Celkový podnět neuronu (excitaci) ξ lze vyjádřit váženým součtem příznaků násobených příslušnou vahou, od kterého je odečtena prahová hodnota daného neuronu Θ. Prahová hodnota představuje minimální hodnotu podnětu, která je nutná pro aktivaci neuronu. Často se prahová hodnota modeluje jako další vstup neuronu s konstantním příznakem $x_0 = \Theta$ a váhou $w_0 = -1$.

$$\xi = \sum_{i=1}^{n} w_i x_i - \Theta = \sum_{i=0}^{n} w_i x_i = wx$$

Celkový součin w.x umožňuje neuronu klasifikovat vstup do dvou tříd. Vstup způsobující odezvu a vstup nepůsobující odezvu. Pokud je součin w.x > 0, reaguje neuron odezvou z, která je dána přenosovou nelineární funkcí S.

$$z = S(wx)$$

Funkce S má nejčastěji podobu tzv. sigmoidy s danou strmostí λ.

$$S(\xi) = \frac{1}{1 + e^{-\lambda\xi}}$$

3.5.2. Dělení neuronových sítí

Neuronové sítě lze dělit podle dvou základních kritérií. První kriterium popisuje topologické vlastnosti neuronové sítě. Může se jednat o sítě rekurentní, kde konfigurace neuronů tvoří cyklický orientovaný graf. V rekurentních sítích vrací neurony všechny své odezvy jako stimuly zpět do sítě. Druhým mnohem důležitějším typem topologie jsou sítě vrstvené. Vrstvené sítě mají neurony rozdělené do vrstev. Výstup každého neuronu ve vrstvě n je předáván jako stimul každému neuronu ve vrstvě $n+1$. Jiné propojení mezi neurony vrstvené sítě neexistuje. Dolní vrstva celé sítě představuje vstup, vrchní vrstva pak výstup.

Druhým kriteriem pro dělení neuronových sítí je způsob jejich práce. Lze je dělit na neuronové sítě s učitelem a samoorganizující neuronové sítě. Sítě s učitelem vznikají vkládáním tzv. trénovací množiny dat, která dostatečně dobře pokrývá úlohy, které budou neuronovou sítí řešeny. Trénovací množina je soubor dvojic podnět a očekávaná odezva. Vytváření těchto sítí vyžaduje nastavování vah pro vstupy jednotlivých neuronů, případně nastavování počtu vrstev a počtu neuronů v neuronové síti tak, aby výsledná síť byla schopna

co nejlépe zpracovat trénovací množinu – tedy, aby výsledek co nejlépe odpovídal očekávanému výstupu. Nastavení sítě získané jejím trénováním lze považovat za uložení znalostí o dané úloze. Toto nastavení neuronové sítě představuje optimalizační problém, který se často řeší stochastickými nebo evolučními metodami, např. simulovaným žíháním nebo genetickými algoritmy. Princip vzniku těchto sítí tedy spočívá v tom, že je znám vnější činitel (učitel), který zná správnou odezvu na testovací podněty. Síť se tedy upravuje na základě vnějších stimulů. To je hlavní rozdíl od samoorganizujích neuronových sítí, které se naopak učí (adaptují) jen na základě vnitřních stimulů vyhodnocovaných z historie sítě.

3.5.3. Kohonenova samoorganizace

Jednou z nejdůležitějších neuronových sítí pro oblast počítačové grafiky, vizualizace dat a rozpoznávání, je Kohonenova samoorganizující mapa, která se často využívá pro reprezentaci dat ve vyšších dimenzích. Jedná se o jednovrstvou neuronovou síť, která je nejčastěji topologicky uspořádána jako dvojrozměrná nebo trojrozměrná mřížka. Prvním krokem k použití Kohonenovy samoorganizující mapy je inicializace váhových vektorů všech jejích neuronů. Váhové vektory mohou být inicializovány malými náhodnými hodnotami, nebo mohou být inicializovány podle znalosti řešeného problému. Váhové vektory musí mít stejnou dimenzi jako vektory podmětů. Po dokončení inicializace může začít trénování neuronové sítě.

Trénování neuronové sítě probíhá vkládáním jednotlivých podmětů. Pro každý podmět se vyhodnotí jeho Euklidovská vzdálenost k váhovému vektoru každého neuronu. Neuron s nejmenší Euklidovskou vzdáleností k vektoru podmětu je považován za reprezentanta podmětu. Váhové vektory reprezentanta podmětu a jeho sousedů jsou upraveny směrem k vektoru podmětu. Míra této úpravy klesá se vzdáleností upravovaného neuronu od reprezentanta podmětu a s časem. Pro starší neuronové sítě tedy platí, že se adaptují mnohem méně než mladé neuronové sítě (analogie z biologie). Výpočet nového váhového vektoru může být popsán následující rovnicí.

$$w_i(t+1) = w_i(t) + \Theta(i,j,t) \cdot \alpha(t)(X - w_i(t))$$

Cílem je tedy vypočítat nový váhový vektor *i*-tého neuronu v čase $t + 1$. Nový váhový vektor je dán váhovým vektorem v čase t upraveným o daný posun. Posun je dán rozdílem vektoru podmětu X a starého váhového vektoru w_i, vynásobeným koeficientem α, který udává míru přizpůsobení závislou na čase t. Koeficient α nazývaný též plasticita se s přibývajícím časem

(stářím neuronové sítě) zmenšuje. Tento součin je dále násoben funkcí Θ, která upravuje posun v závislosti na vzdálenosti neuronu i od reprezentanta podmětu j. Tato funkce má nejčastěji gaussovský průběh a její rozsah se zmenšuje s časem t, což vede k tzv. kompetitivnímu učení.

Čas, neboli stáří neuronové sítě vyjadřuje číslo iteraci trénování pro vstupní množinou trénovacích podmětů. Jedna iteraca představuje zpracování všech podmětů z této množiny. Natrénovanou Kohonenovu samoorganizující mapu lze např. velmi dobře použít pro klasifikaci (rozpoznávání) podmětů do k tříd, kde k je počet neuronů v neuronové síti.

3.6. Využití znalostí

Všechny dosud zmíněné umělo inteligentní metody nějakým způsobem využívaly znalosti o řešené úloze. V řešení úloh, stochastických i genetických algoritmech byly znalosti reprezentovány ve formě ohodnocovaní funkce daného stavu. U neuronových sítí znalosti vznikaly průběžně úpravou neuronové sítě na základě trénovací množiny podmětů. Hlavní nevýhodou obou těchto postupů je obsažení znalostí přímo v metodě pro řešení úlohy. Takovou reprezentaci nazýváme implicitní [19]. Změna implicitních znalostí v případě počítačového programu obvykle představuje změnu zdrojových kódů programu a jeho nový překlad.

Tato kapitola se blíže zaměří na explicitní reprezentaci znalostí, která odděluje znalosti od výkonného programu. Znalosti jsou uchovávány v externím uložišti, obecně nazývaném databáze. Extrakce znalostí o dané úloze je tak velmi jednoduchá, navíc je možné znalosti průběžně měnit, mazat nebo přidávat. Pro správnou aplikaci tohoto principu je nutné vytvořit jednoznačný popis znalosti, který bude umožňovat vyhledávání v databázi podle charakteristických rysů. Pokročilé metody využívající explicitní reprezentaci znalostí dokáží z aktuální databáze odvozovat další znalosti, které původně do databáze vloženy nebyly. Stejně jako v případě řešení úloh, je nutné vytvořit formální popis systému, který bude reprezentovat řešenou úlohu. Existuje více možných popisů [19] úloh pracujících s explicitními znalostmi. Zde budou zmíněny jen produkční systémy.

3.6.1. Produkční systémy

Produkční systémy [19], [web1] se v mnohém podobají stavovému modelu popsanému v kapitole 3.1. V případě běžné problematiky řešení úloh jsou dokonce ekvivalentní. Produkční systém je charakterizován třemi částmi. První částí je soubor produkčních pravidel tvořený množinou dvojic [Situace; Akce], které představují podmínku, za které je možné vykonat příslušnou akci. Druhou částí je pracovní paměť neboli báze dat, ve které se uchovávají vstupní data a aktuální stavu úlohy. Poslední částí je tzv. inferenční stroj neboli interpret pravidel, který představuje výkonnou jednotku produkčního systému. Inferenční stroj provádí analýzu aktuálního stavu a vybírá pravidlo, které na aktuální stav bude použito, čímž je vytvořen nový aktuální stav a proces se může opakovat. Proces se zastaví, pokud je aplikován limitní počet pravidel, pokud aktuální stav splňuje cílové podmínky nebo pokud na aktuální stav již není možné aplikovat žádné další pravidlo.

Produkční systémy mohou pracovat dvěma způsoby. První způsob se nazývá přímé řetězení nebo též odvozování řízené daty. Tento způsob začíná počátečním stavem a snaží se dojít k cílovému stavu. Opačný postup práce, tedy od cílového stavu k počátečnímu, se nazývá zpětné řetězení nebo též odvozování řízené cílem.

Během analýzy pravidel je porovnáván aktuální stav v paměti se souborem pravidel a je vytvořena množina kandidátů (pravidla, které mohou být na aktuální stav aplikována). Některá pravidla z množiny kandidátů mohou obsahovat proměnné. Z těchto pravidel je třeba vytvořit instance odpovídající aktuálnímu stavu. Produkční systém předpokládá sériový běh inferenčního stroje. Je tedy možné aplikovat jen jedno pravidlo z množiny kandidátů – tzv. řešení konfliktu, které může být řešeno např. vahou (prioritou) daného pravidla, neopakováním pravidla použitého v předešlém kroku nebo preferencí novějších dat. Pro správnou funkčnost a rozšiřitelnost produkčního systému musí produkční pravidla splňovat některá omezení:

- Interakce mezi pravidly je možná jen přes pracovní paměť
- Vyhodnocováním pravidla ani vytvářením instance pravidla nesmí být měněna pracovní paměť
- Pravidla musí představovat elementární operace

Dodržení těchto omezení umožňuje vytvářet složité systémy, které jsou lehce zpravovatelné. Na druhou stranu dodržení všech těchto podmínek při vytváření pravidel spolu s modularitou produkčního systému je velmi nelehký úkol.

4. Možnosti integrace se stávajícím SW

Jedním z bodů zadání této práce je i rozbor možností integrace se stávajícím softwarovým vybavením dostupným na Západočeské univerzitě. Tímto vybavením je myšlen rekonstruktor, který vytvořil Ing. Michal Varnuška PhD. v rámci své disertační práce [2]. Celkem byly zváženy tři rozdílné možnosti, které by umožnily propojení mezi rekonstruktorem a aplikací, nazvanou Optimizer, která vznikla v této diplomové práci. V této kapitole se nejprve zaměříme na samotný rekonstruktor a poté budou probrány jednotlivé možnosti propojení obou aplikací s jejich výhodami a nevýhodami.

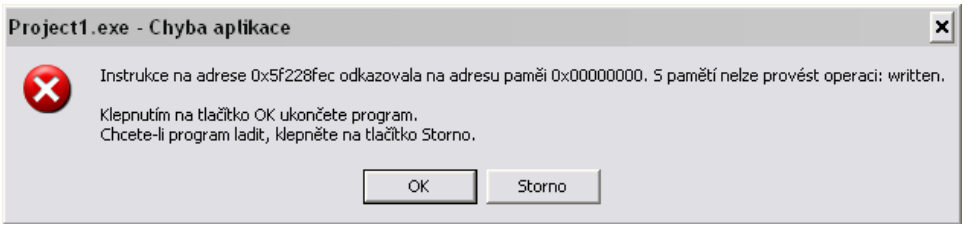
4.1. Rekonstruktor

Než budou popsány možnosti propojení aplikací, je vhodné se pozastavit nad samotným programem a zdrojovým kódem rekonstruktoru. Tato část je věnována rozboru několika poskytnutých verzí rekonstruktoru. Dá se předpokládat, že se nejedná o finální verze, protože některé výsledky prezentované v teoretické práci [2] se dostupnými verzemi vůbec nedařilo dosáhnout (např. rekonstrukce datové množiny Nascar s filtrováním okrajů). Pokud se jedná o výsledné verze, bylo by nutné tuto pasáž začít poněkud tvrdým tvrzením, že tento program byl vyvíjen téměř čtyři roky a nikdy nebyl dokončen. Minimálně o verzích, které byly poskytnuty jako vstup této diplomové práce, to platí. Není totiž problém program během pár desítek vteřin přivést do chybového stavu, ve kterém se sám ukončí. Velmi obvyklé chyby představuje přístup do paměti, která programu nepatří, případně problémy s OpenGL, které dokonce mohou způsobit „vypnutí“ grafické karty, která znovu naběhne až po tvrdém restartu počítače. První poskytnutá verze rekonstruktoru, která byla údajně finální dokonce nefungovala vůbec, protože v ní každý pokus o otevření souboru končil jeho decimací na čtyři body, tj. jeden tetrahedron a následnou rekonstrukci povrchu tohoto tetrahedronu.

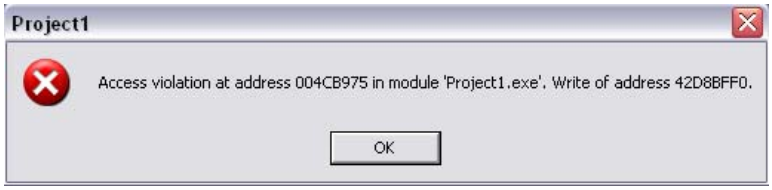
4.1.1. Chyby v aplikaci

Chyby lze celkově rozdělit do čtyř skupin. V první skupině jsou přístupy do špatné paměti (Obr. 4-1), které vznikají pokud se uživatel pokusí při jednom spuštění programu otevřít více než jeden soubor (tedy provést rekonstrukci nad více než jednou datovou množinou), což je vcelku očekávaný požadavek. Pokud má uživatel v plánu provést více rekonstrukcí, musí před každou znovu program vypnout a zapnout. Nepříjemnost tohoto postupu vzrůstá se

skutečností, že si program po vypnutí neukládá žádná nastavení (např. pro vizualizaci), takže je po každém zapnutí nutné vše opět nastavit. Druhá skupina chyb se týká nastavování průběhu rekonstrukce z grafického rozhraní. Jelikož grafické rozhraní nebylo upravováno a používáno téměř tři roky, dávno jeho možnosti neodpovídají funkcionalitě v aplikaci a co více, změna nabízených hodnot a spuštění rekonstrukce většinou končí chybou (Obr. 4-2). Třetí skupinou je chybná práce s OpenGL. Samo OpenGL občas vyhodí chybu, která hlásí přístup do špatné paměti. Extrémním případem je již zmiňované „vypnutí" grafické karty, které se projevovalo na první testovací sestavě (viz. příloha 9.4), např. u datové množiny Demi nebo Hypersheet. Poslední skupinou jsou chyby, při kterých se aplikace sama vypne, aniž by uživateli sdělila proč. Na první testované sestavě se toto chování objevovalo např. při pokusu o rekonstrukci datové množiny CTHead60.



Obr. 4-1 Chyba, která vznikla při pokusu o otevření třetí datové množiny. Stejná chyby vzniká i po ukončení aplikace po otevření druhé datové množiny.



Obr. 4-2 Chyba, která vznikla při pokusu o spuštění rekonstrukce dvouprůchodovou metodou CRUST s filtrováním úhlů. Standardní nastavení pracuje s jednoprůchodovou metodou CRUST.

4.1.2. Ovládání

V předešlém odstavci byly načaty možnosti ovládání rekonstruktoru a jeho uživatelské rozhraní. Dá se říci, že v této oblasti program naprosto šokuje. Grafické rozhraní programu využívá vlastní ovládací prvky a je řešeno přinejmenším velmi neprakticky. Základ programu tvoří malý vertikální panel s několika tlačítky a konzolový výstup. Panel je z nějakého neznámého důvodu zespoda špatně oříznut, takže je z posledního tlačítka vidět jen půlka. Jednotlivá tlačítka způsobují otevírání dalších okének, které se objevují různě po pracovní

ploše počítače. Ze subjektivního pohledu grafické rozhraní nebudí celistvý dojem, chybí zde něco hlavního co by sdružovalo jednotlivá okénka do souvisejícího bloku. Takto je uživateli dána najevo jejich příbuznost jen díky odlišnému vzhledu od standardních ovládacích prvků operačního systému Windows. Na druhou stranu je třeba zmínit, že podobný systém grafického rozhraní volí např. vývojářské prostředí Delphi nebo grafický editor Gimp a aplikace Optimizer částečně využívá tento přístup pro některá výběrová okna či ladící okna.

Vzhled ovládání ovšem představuje minimální problém proti jeho funkcionalitě. Jak již bylo zmíněno výše, ovládat rekonstrukci z grafického rozhraní aplikace je téměř nemožné, protože změny nastavení způsobují chyby. Jak tedy rekonstrukci ovládat? Průběh rekonstrukce a nastavení jejích jednotlivých částí je řízen na úrovni podmíněného překladu. To znamená, že změna nastavení je možná jen v případě, že jsou k dispozici i zdrojové soubory aplikace. Pokud tedy uživatel chce používat program rekonstruktor a měnit nastavení pro rekonstrukci musí k tomu mít nainstalován minimálně překladač Delphi a umět ho používat. Před každou změnou nastavení musí uživatel opět vypnout aplikaci a upravit soubor s konstantami pro podmíněný překlad tak, aby odpovídal jeho požadavkům na nastavení rekonstrukce. Tento soubor není nikde zdokumentován a komentáře nepokrývají celý jeho obsah, takže je občas nutné provádět změnu nastavení metodou pokus – omyl. Naštěstí mají konstanty většinou srozumitelný název, takže se v případě, že má uživatel velmi dobře nastudovanou teoretickou práci [2] dá jejich smysl odvodit (ovládání filtrování okrajů ale nalezeno nebylo). Po úpravě těchto konstant může aplikaci přeložit a znova spustit. Toto řešení „uživatelského rozhraní“ by se tedy dalo pokládat za odstrašující případ a jasně ukazuje, že tento program není určen pro další používání, ale pouze jako test teoretických poznatků z práce [2].

Dalším problémem uživatelského rozhraní je ovládání vizualizované scény. Problém nastává pokud se uživatel pokusí používat kolečko myši pro přibližování a oddalování objektu. Chování této funkcionality je velmi podivné a zřejmě zde dochází k nějakému přetečení nebo špatnému zaokrouhlování. Po chvilce pohybu jedním směrem sebou objekt začne škubat – chvilku jde jedním směrem a po chvilce opět zpátky, přestože kolečkem na myši se točí pořád stejně. V tu chvíli je problém i s používáním šipek na klávesnici a nepomáhá ani volba *Reset scene*, která se zřejmě nezabývá anulováním transformací. Nesprávné chování lze pozorovat i pokud je nejprve objekt pootočen a teprve poté oddalován nebo přibližován. Vypadá to, že se při této situaci neupraví vektor pro pohyb v ose *z* o vzniklé pootočení, takže je objekt oddalován a zároveň se pohybuje někam šikmo do strany.

4.1.3. Zdrojové soubory

Na závěr rozboru rekonstruktoru se zaměříme i na jeho zdrojové soubory, protože ty v případě integrace s jinou aplikací hrají největší roli. Tato část je brána hodně subjektivně, a tak s ní pravděpodobně nebude každý souhlasit. Dá se předpokládat, že s touto částí budou souhlasit spíše programátoři využívající objektové programovací jazyky syntakticky odvozené od jazyka ANSI C, tedy např. jazyk C++, C# či Java. Naopak nesouhlas se dá očekávat od uživatelů Delphi.

Rekonstruktor byl vytvořen objektovým přístupem ve vývojovém prostředí Delphi. Je tedy škoda, že je objektový přístup hyzděn některými jazykovými konstrukcemi, které v Delphi zůstaly jako pozůstatek starého procedurálního Pascalu. Mezi tyto konstrukce patři vnořené procedury a funkce a konstrukce `WITH`, kterou je ovšem možné najít i u moderního jazyka Visual Basic .NET. Zmíněné konstrukce spolu s nedostatkem komentářů silně ztěžují porozumění kódu a jeho *reverse engineering*. Použití těchto konstrukcí je přitom naprosto zbytečné, protože stejná část kódu se dá nahradit ekvivalentním způsobem, který je mnohem srozumitelnější a v případě vnořených procedur a funkcí nekazí objektový dojem.

Použití konstrukce `WITH` pro přístup k položkám přímo dostupného záznamu nebo proměnným a metodám k přímo dostupné instance třídy sice usnadňuje programování, protože identifikátory jsou kratší, ovšem kód pak nemusí být na první pohled srozumitelný. Pokud tedy má být kód v budoucnu využíván někým jiným, je rozhodně lepší trocha nepohodlnosti při vytváření programu, která přinese jeho vyšší srozumitelnost. V případě, že je konstrukce `WITH` využívána k přístupu k položkám vnořeného (agregovaného) záznamu nebo proměnným či metodám vnořené (agregované) instance třídy, pak je možné provést její jednoduché nahrazení lokální proměnnou typu pointer na příslušný datový typ, který bude odkazovat na řídící část konstrukce `WITH`. Celý kód by se tak stal podstatně přehlednější a pro programátora, který kód nevytvářel i mnohem čitelnější, neboť s použitím konstrukce `WITH` není na první pohled jasné, k čemu se daná proměnná nebo metoda váže. Dohledávání lze ve vývojovém prostředí Delphi řešit stisknutím Ctrl + kliknutí levým tlačítkem na zkoumaný identifikátor, ale to z nějakého důvodu nefunguje vždy. Zmatek ve zkoumaném kódu navíc vzrůstá s počtem vnořených konstrukcí `WITH`.

Vnořené funkce a procedury jsou v objektovém programování zbytečností. Jejich využití je již řešeno samotným principem zapouzdření a přístupovými právy k metodám. Vnořené funkce a procedury jsou tak téměř ekvivalentní k soukromým (privátním) metodám objektu. Jediný rozdíl představuje možnost, přímého přístupu vnořených procedur a funkcí k lokálním proměnným nadřazené metody. Opět se nedá hovořit o čistém řešení, neboť to zvyšuje nepřehlednost kódu. Řešením je tedy udělat z vnořených procedur a funkcí soukromé metody příslušné třídy a vše jim předávat formou parametrů. Pokud dojde k tomu, že by metoda potřebovala pro svojí práci příliš mnoho parametrů, je to první náznak toho, že problém byl špatně dekomponován a metoda se pravděpodobně snaží dělat příliš mnoho věcí najednou.

Poslední poznámkou ke zdrojovým kódům jsou jmenné konvence souborů. Není skutečně dobrý nápad nechat soubory pojmenované stylem Project1, Form1, Form2, Unit7 atd. Tyto názvy nevypovídají nic o obsahu těchto souborů a jsou prakticky nepoužitelné.

4.1.4. Zhodnocení programu

Předchozí tři kapitoly mohou být vyloženy jako tvrdá kritika zmiňovaného programu, ovšem není tomu tak. Před výsledným hodnocením programu je třeba se zamyslet nad tím, za jakým účelem byl program vyvíjen. Nejedná se o program, který by měl být používán další osobou nebo masově využíván jako software pro rekonstrukci povrchu. Ani sám autor to nepokládá za dobrý nápad. Program je určen pouze jako test myšlenek a poznatků popsaných v teoretické práci, žádné další využití v této podobě nemá. Tuto úlohu program splňuje velmi dobře a není mu tedy možné nic vytknout. Pokládat tento program za softwarové vybavení pro rekonstrukci povrchu je ale velký omyl.

K větším možnostem využití stávajícího kódu by bylo třeba velké množství úsilí pro opravu všech zmiňovaných chyb a předělání uživatelského rozhraní. Dále by bylo velmi vhodné kód rozdělit do několika vzájemně oddělených modulů s pevně definovaným rozhraním tak, aby bylo možné použít pouze některé moduly nebo některé vyměnit za jinou implementaci. Minimální dělení si lze představit jako rozdělení na tři knihovny, první by obsahovala Delaunayovu tetrahedronizaci, druhá samotný rekonstruktor a třetí uživatelské rozhraní a vizualizaci. Vzhledem k použitému programovacímu jazyku a objektovosti programu by moduly musely být pravděpodobně řešeny jako COMy.

Navrhované řešení je zřejmě tak složité, že by se skoro vyplatilo vzít teoretickou část a stávající řešení jako referenční a vytvořit novou implementaci s propracovaným návrhem, takže by se skutečně dalo hovořit o softwarovém vybavení pro rekonstrukci povrchu. Kvůli rychlosti a znuvupoužitelnosti by pro vývoj nové verze rekonstruktoru byl asi nejvhodnější programovací jazyk C++. Jádro rekonstruktoru by mělo být vytvořeno podle ANSI/ISO normy C++ a nad tímto jádrem by bylo možné vytvořit knihovny pro operační systém Windows (pravděpodobně COM) i Linux. Toto řešení by pravděpodobně již zahrnovalo i úpravy pro paralelizaci rekonstrukce. Dále by bylo nutné vytvořit nový systém pro vizualizaci (nebo použít nějaký dostupný), který by v první řadě měl mít možnost autodetekce dostupného grafického hardwaru a přizpůsobení vykreslovaného obsahu tomuto hardwaru. Tato vizualizační knihovna by mohla být řešena i v řízeném kódu, tedy např. v jazyce C#.NET. Tímto řešením by se Centrum počítačové grafiky a vizualizace dat na Západočeské univerzitě mohlo hrdě prezentovat a dále ho rozvíjet. Obě řešení dalece přesahují možnosti této práce, rozsahem se jedná o samostatné zadání diplomových prací zaměřené na kombinaci počítačové grafiky a softwarového inženýrství a v případě úpravy pro paralelizaci i zaměřením na distribuované systémy.

4.2. Optimizer jako modul pro Rekonstruktor

První, a z pohledu vedoucí této diplomové práce, nejžádanější možností propojení obou aplikací bylo vytvoření nové aplikace v podobě modulu nebo knihovny, jejíž metody by se volaly přímo z rekonstruktoru. Tato varianta je svým způsobem nejlepší, neboť umožňuje rozšířit možnosti stávajícího programu a zlepšit tak přímo výstup z rekonstruktoru. Nespornou výhodou tohoto řešení je připojení nového kódu kamkoliv do průběhu rekonstrukce. Bylo by tedy možné provádět pokusy s předzpracováním bodů vstupujících do Delaunayovy tetrahedronizace, předzpracováním tetrahedronů vstupujících do metody CRUST či předzpracováním trojúhelníků vstupujících do procesu extrakce manifoldu. Dále by mohl být finální povrch optimalizován. Při této optimalizaci by byly k dispozici informace o předchozích krocích, tedy všech trojúhelníky před extrakcí manifoldu nebo rovnou všechny trojúhelníky z Delaunayovy tetrahedronizace. Možností tohoto řešení je skutečně mnoho. Bohužel tyto možnosti narážejí na nedostatky aktuální implementace rekonstruktoru, které byly zmíněny v předchozí kapitole. Sám autor rekonstruktoru nepovažoval jeho rozšiřování za vhodný postup.

Ze strany vedoucí diplomové práce byla snaha přesvědčit autora rekonstruktoru o drobné úpravě, která by spočívala v rozdělení programu do dvou částí. V první části by se vykonalo veškeré předzpracování a Delaunayova tetrahedronizace. Výsledek by byl uložen do souboru. Druhá část by jako vstup měla dva soubory, jedním by byla původní Delaunayova tetrahedronizace a druhým váhy pro trojúhelníky z této tetrahedronizace. Extrakce primárního povrchu by pak tyto váhy využívala pro rozhodování o povrchových trojúhelnících. Některé by mohly být automaticky přijaty nebo naopak zamítnuty u jiných by váha mohla přispět k rozhodování. Cílem této práce by poté bylo určit tyto váhy. Tento nápad ovšem neuspěl, protože autor rekonstruktoru opustil vysokoškolskou půdu a na tyto úpravy již neměl čas. Podobnou myšlenky by bylo možné aplikovat po rekonstrukci primárního povrchu.

4.3. Rekonstruktor jako modul pro Optimizer

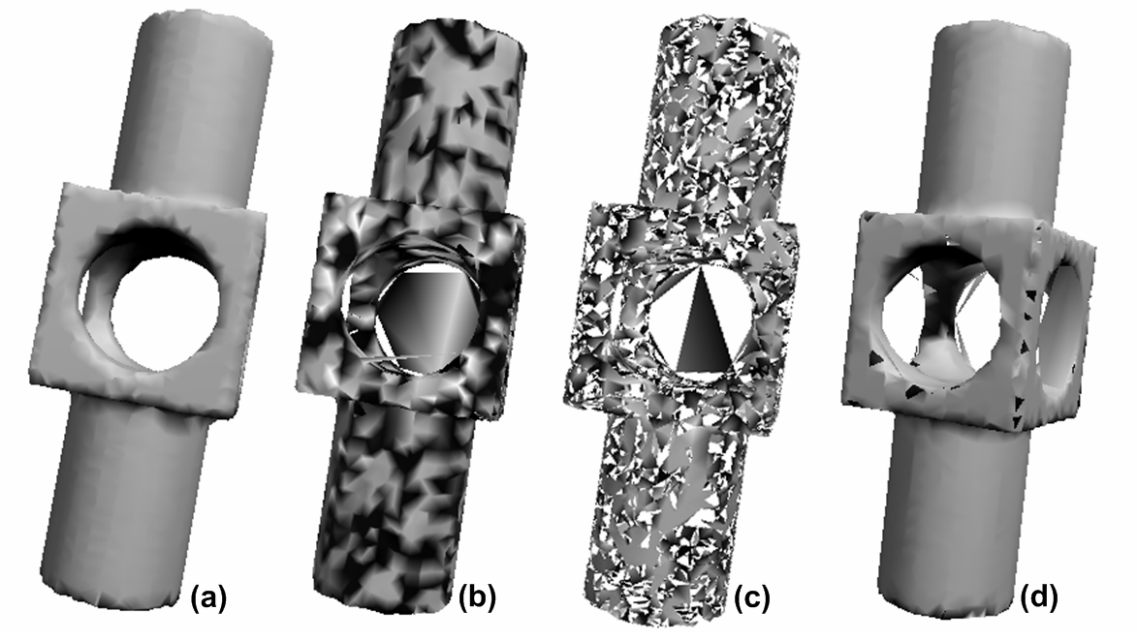
Druhou možností je opačný postup, který spočívá v integraci rekonstruktoru do nové aplikace, která by volala jeho metody. Tento postup má podobné výhody jako předešlý návrh řešení, ale je implementačně mnohem náročnější. Novou výhodou tohoto postupu je odbourání problematické vizualizace stávajícího řešení. Implementace tohoto řešení by vyžadovala úpravu rekonstruktoru zmiňovanou v kapitole 4.1. Rekonstruktor by musel být k dispozici v podobě znovupoužitelného modulu, jehož metody by byly postupně volány v nové aplikaci. Pokud by byl požadavek na úpravu dat po jednotlivých fázích rekonstrukce, musely by data existovat v nové aplikaci a do rekonstruktoru by byly pouze předávány v podobě parametrů volaných metod. To by pravděpodobně vyžadovalo velkou změnu stávajícího kódu. V případě, že by tato změna nebyla provedena, bylo by možné upravovat pouze vstupní data, ze kterých je povrch rekonstruován a výstupní data – tedy rekonstruovaný povrch. Pro tento typ úlohy ovšem není třeba vynakládat takové úsilí na úpravu rekonstruktoru. Místo toho lze zvolit přístup navržený v následujícím oddílu.

4.4. Rekonstruktor produkuje data pro Optimizer

Nejjednodušším řešením komunikace mezi stávajícím rekonstruktorem a novou aplikací je komunikace prostřednictvím datových souborů. Toto řešení nepřináší zdaleka tolik možností jako předešlé dva návrhy, ovšem není nijak závislé na programových nedostatcích existujícího programu. Existující program tedy není nutné nijak upravovat, čímž je ušetřeno mnoho času, který může být věnován samotnému řešení problému zlepšování rekonstruovaných

trojúhelníkových sítí. To byl hlavní důvod, proč byl tento způsob propojení aplikací zvolen pro tuto diplomovou práci.

Možnosti úprav a zlepšování výsledku rekonstrukce jsou zredukovány na úpravu (optimalizaci) výsledného povrchu, který je reprezentován trojúhelníkovou sítí. Rekonstruktor tyto trojúhelníkové sítě (Obr. 4-3a) ukládá do .tri souborů, které poté mohou být zpracovány v programu Optimizer a znovu uloženy. Pro optimalizaci rekonstruovaného povrchu nejsou k dispozici žádné dodatečné informace, proto byla snaha programu předat trojúhelníkové sítě, které ještě neprošly extrakcí manifoldu. Rekonstruktor byl za tímto účelem upraven tak, aby před každou extrakcí manifoldu uložil primární povrch do pomocného .tri souboru[1]. Získané sítě (Obr. 4-3b) ovšem nepřinesly dostatek nových informací, aby se s nimi vyplatilo pracovat při optimalizaci. Tyto trojúhelníkové sítě měly naopak některé nežádoucí vlastnosti. Nejnepříjemnější vlastností trojúhelníkových sítí před extrakcí manifoldu je často rozdílná orientace trojúhelníků, nastavení orientace se provádí až v průběhu extrakce (Obr. 4-3c). To se týká i trojúhelníků, které po extrakci manifoldu v trojúhelníkové síti zůstanou. Dalším problémem je velké množství děr v trojúhelníkové síti (Obr. 4-3d).



Obr. 4-3 (a) Rekonstruovaný povrch získaný běžnou cestou. Tato síť má 8212 trojúhelníků. (b) Trojúhelníková síť před extrakcí manifoldu. Tato síť má 8210 trojúhelníků. (c) Stejná síť jako v b, ale jsou zobrazeny jen trojúhelníky, které mají orientaci po směru hodinových ručiček. (d) Stejná síť jako v b po ruční orientaci špatných trojúhelníků. Černé skvrny na popředí modelu jsou díry v trojúhelníkové síti. Velká černá skvrna uvnitř výřezu je způsobena špatnou orientací těchto trojúhelníků, způsobenou propojením vnitřního povrchu s vnějším.

[1] Tuto verzi rekonstruktoru lze nalézt na přiloženém DVD v adresáři `Modified Reconstructor`. Při každé extrakci se do tohoto adresáře uloží do souboru beforeManExt.tri trojúhelníkovou síť před extrakcí manifoldu.

Existuje i čtvrtá možnost propojení obou aplikací. Tuto možnost by bylo možné nazvat „Optimizer produkuje data pro Rekonstruktor“. Tato metoda by měla za úkol zbavit data šumu a odstranit *outliers*. Vzhledem ke stávajícím možnostem práce se vstupními daty (jejich filtrování, vyhlazování a decimování) se ovšem nejedná o nijak přínosnou možnost. Datové množiny, které projdou dostupnými úpravami jsou zbaveny všech očividných chyb a zůstává vysokofrekvenční šum, který je obsažen přímo v datech. Ten je již ve stávající implementaci rekonstruktoru částečně řešen metodou *Normals Denoising*. Problémy, se kterými si neporadí stávající způsoby jsou tedy těžko řešitelné i metodami umělé inteligence.

5. Využití UI pro rekonstrukci povrchů

Poté, co byla probrána teorie týkající se vybraných umělo inteligentních metod a princip rekonstrukce povrchu dostupným rekonstruktorem, mohou být popsány možnosti využití některých z těchto metod pro řešení rekonstrukce povrchů nebo jejich následnou optimalizaci. Tato kapitola obsahuje soubor myšlenek a nápadů, které se zrodily v průběhu vytváření práce. Ze zmiňovaných nápadů byl vybrán jeden, jehož implementační náročnost se jevila odpovídající rozsahu práce. Ostatní nápady tak zůstávají pouze na papíře a jejich případná funkčnost nebo nefunkčnost nemůže být doložena.

surface reconstruction	triangle mesh	neural network
surface optimization	simulated annealing	artificial intelligence
edge swap	genetic algorithms	stochastic
probabilistic	point clouds	scattered points
manifold	heuristic	weights

Tabulka 5-1. Seznam termínů, ze kterých byly skládány dotazy na odborné články.

Problém využití umělé inteligence pro rekonstrukci povrchů zatím není ve vědecké sféře příliš řešen, a proto je velmi obtížné k němu nalézt jakékoliv informace. Při hledání informací byly použity webové stránky ACM, IEEE, CiteSeer.IST a WSCG. Seznam termínů, ze kterých byly skládány dotazy lze nalézt v tabulce 5-1. Výsledky hledání byly ovšem velmi neuspokojivé. Tento fakt je zmiňován i v dokumentu [23], který se zabývá shrnutím některých metod, které využívají umělou inteligenci pro práci s trojúhelníkovými sítěmi. Většina současných metod využívá umělou inteligenci pro úpravy trojúhelníkových sítí v rovině, což je podstatně lehčí problém než operace s trojúhelníkovými sítěmi v prostoru. Je zde popsáno několik metod zaměřených na optimalizaci geometrických vlastností trojúhelníkových sítí s využitím simulovaného žíhání a genetických algoritmů. Při rekonstrukci povrchu jsou ovšem v první řadě důležité topologické vlastnosti, protože kvalita sítě z pohledu tvaru a počtu trojúhelníků nic nevypovídá o výsledném tvaru objektu. Důležitým faktem, který je v průběhu dokumentu zmiňován několikrát, jsou velikosti datových množin, na kterých autoři testují své metody založené na genetických algoritmech. Jedná se řádově o desítky bodů v rovině, což upozorňuje na problémy, které by mohly vzniknout pokud při využití genetických algoritmů v této prácí, protože nejmenší datové množiny rekonstruovaných objektů mají řádově tisíce vrcholů či trojúhelníků.

V této kapitole budeme pojmy rekonstrukce a triangulace pokládat za termíny, které vyjadřují naprosto rozdílné vlastnosti trojúhelníkové sítě nad množinou vstupních bodů. Triangulace bude reprezentovat obecnou trojúhelníkovou síť zkonstruovanou nad množinou vstupních bodů. Rekonstrukce bude reprezentovat manifold zkonstruovaný nad množinou vstupních bodů, který se navíc má podobat co nejvíce původnímu objektu, ze kterého byla množina vstupních bodů snímána.

5.1. Neuronové sítě

Jako první budou popsány možnosti využití neuronových sítí, protože se jedná o jedinou oblast, ke které se podařilo nalézt dokument, který by svým obsahem přímo odpovídal názvu této diplomové práce. Y. Yo ve své práci [24] představil metodu, která pro rekonstrukci povrchu využívá Kohonenovu samoorganizující mapu s topologií 2D mřížky, ve které je každá buňka rozdělena dalším axonem na dva trojúhelníky. Vstupním parametrem této metody je počet neuronů sítě. Každý neuron představuje jeden budoucí vrchol rekonstruované trojúhelníkové sítě. Souřadnice budoucího vrcholu jsou dány vahou příslušného neuronu. Souřadnice bodů vstupní množiny jsou brány jako množina vstupních podmětů pro trénování neuronové sítě.

Počáteční hodnoty váhových vektorů jednotlivých neuronů jsou nastaveny tak, aby neurony tvořily obalovou kouli množiny vstupních bodů, ze které je rekonstruován povrch. V každé iteraci trénování neuronové sítě vstupními podměty se váhy bodů přesunou blíže k hledanému povrchu – povrch je postupně obalován. Problém vzniká v případě, že objekt obsahuje konkávní oblasti, neuronová síť se snaží tyto oblasti vyplnit trojúhelníky, které tam nepatří. Tuto situaci je nutné řešit identifikováním těchto trojúhelníků a následnou operací prohození hrany mezi dvěma trojúhelníky → prohození axonu v neuronové síti, případně dodatečnou úpravou váhového vektoru problémového neuronu.

Výstupem této metody je natrénovaná neuronová síť, ve které váhy jednotlivých neuronů představují souřadnice bodů v prostoru a axony mezi neurony definují trojúhelníkovou síť nad těmito body. Výsledná síť představuje rekonstrukci, která je aproximací původního povrchu reprezentovaného vstupní množinou bodů. Počet neuronů použité neuronové sítě může charakterizovat úroveň detailu (*level of detail*) výsledné rekonstrukce.

Zajímavou část metody představuje její urychlení. Trénování neuronové sítě je obecně časově velmi náročné. Při rekonstrukci povrchu můžeme pracovat s neuronovými sítěmi, které mají řádově tisíce a více neuronů. Tato metoda přichází s velmi zajímavým poznatkem. Na začátku trénování jsou rekonstruovány hrubé obrysy modelu a s přibývajícím stářím neuronové sítě, se metoda dostává blíže k povrchu a začíná rekonstruovat detaily. Proto metoda začíná s malým počtem neuronů a v průběhu rozpoznávání jsou do neuronové sítě vkládány další neurony, které umožní rekonstruovat i detaily povrchu. S plnou neuronovou sítí se tak pracuje jen zhruba v deseti procentech všech iterací nad trénovací množinou podmětů.

Yo uvedl i druhou variantu metody, která pracuje se čtyřúhelníkovými sítěmi. Tato varianta se používá pro rekonstrukci 2,5D povrchů, tedy např. krajiny. Tato metoda nebyla vybrána pro implementaci, protože nesplňuje zadání. Jedná se o vytvoření zcela nového rekonstruktoru, nikoliv o úpravu stávajícího nebo optimalizaci výstupu stávajícího rekonstruktoru.

5.2. Genetické algoritmy

Optimalizace trojúhelníkové sítě genetickými algoritmy vypadá na první pohled velice lákavě. Před samotným zamyšlením nad konkrétní podobou genetických algoritmů je třeba uvažovat nad tím, jak bude generován vstup, tedy počáteční generace. V kapitole 3.4.5 byl uveden postup generování počáteční generace pro obecnou optimalizační úlohu. Obě tyto možnosti představují pro optimalizaci trojúhelníkových sítí značný problém. Dalším problémem je reprezentace jednotlivých chromozomů v populaci s čímž souvisí i způsob jakým budou pracovat genetické operátory.

5.2.1. Vytvoření počáteční generace

Náhodné generování počáteční generace pravděpodobně nepovede k řešení úlohy. Toto tvrzení je založeno na několika domněnkách. V první řadě se dá předpokládat, že optimalizace z náhodných triangulací bude trvat příliš dlouho – příliš zde může znamenat řádově dni, týdny či déle. Tato domněnka vznikla z údajů obsažených ve [23], kde jsou popisovány některé aplikace genetických algoritmů na rovinné triangulace, doba výpočtu těchto úloh pro řádově desítky bodů se podle autorů pohyboval v hodinách, tyto údaje je samozřejmě nutné brát s velkou rezervou, protože se většinou jedná o testy prováděné před mnoha lety. Nicméně v této úloze řešíme problémy jejichž vstupní množina čítá řádově tisíce bodů v prostoru. Další

domněnka, která předpovídá neúspěch optimalizace z náhodných triangulací je konstrukce ohodnocovací funkce. Vytvoření obecné ohodnocovací funkce, která by byla schopná kvalitativně ohodnotit libovolnou triangulaci na n bodech v prostoru, je samo problém, který se v této práci nepodařilo vyřešit. Dá se tedy předpokládat, že výsledek této optimalizace pravděpodobně ani nebude manifold, tj. nebude to rekonstrukce. Další možností by pravděpodobně bylo generování náhodných rekonstrukcí nad množinou vstupních bodů, ale tento proces náhodného generování různých manifoldů by sám musel být řešen nějakou velmi silnou umělo inteligentní metodou.

Vytvoření počáteční populace pomocí dostupného rekonstruktoru je zjevně také nemožné, protože se nepodaří vygenerovat dostatečný počet rekonstrukcí. Testy pro jednoduché datové množiny, které reprezentovaly průběh funkce dvou proměnných ukázaly, že je možné vygenerovat asi dvacet různých rekonstrukcí, z nichž některé se lišily jen minimálně. Zvětšení počtu získaných rekonstrukcí je možné použitím i dalších rekonstruktorů, např. Inria [web2] a Cocone [web3]. Všechny použité rekonstruktory musí vytvářet rekonstrukce (trojúhelníkové sítě), které interpolují vstupní množinu bodů. Další podmínkou pro využití více typů rekonstruktorů je práce nad stejnou množinou vstupních bodů. Rekonstruktory tedy nesmí provádět žádnou filtraci ani decimaci vstupní množiny bodů, jinak nebude možné výsledky z různých rekonstrukcí vzájemně kombinovat. Pokud je tedy nutné množinu vstupních bodů předzpracovat, musí být předzpracování provedeno mimo samotné rekonstruktory, kterým je pak předán výsledek. Ani použití dalších rekonstruktorů nezajistí vygenerování počáteční generace čítající stovky až tisíce jedinců. Navrhovaný způsob získání dalších triangulaci je masivní spuštění genetických operátorů nad stávající množinou rekonstrukcí, která může být pro zajímavost doplněna o několik náhodných triangulací. Tímto způsobem je možné vygenerovat zbývající triangulace vstupní generace, z nichž podle kvality genetických operátorů budou část tvořit přímo rekonstrukce.

5.2.2. Přizpůsobení úlohy genetickým algoritmům

Poté co je připravena strategie pro vytvoření počáteční generace, může být uvažován způsob použití genetických algoritmů. Využití nejjednodušší verze genetických algoritmů se nezdá možné, protože zřejmě existuje jen jediný způsob zápisu trojúhelníkové sítě ve formě bitového řetězce. Tento způsob předpokládá zápis trojúhelníkové sítě v podobě indexového pole, do kterého se postupně zapisují indexy vrcholů jednotlivých trojúhelníků. Toto pole je poté převedeno na bitovou reprezentaci (tedy každý index je reprezentován bitově), která je

použita jako chromozom. Tento způsob reprezentace se nejeví příliš výhodný, protože je stále nutné udržovat sémantiku jednotlivých indexů. Lokus pak nepředstavuje pozici v chromozomu, ale pozici na úrovni celých indexů a tedy bitová reprezentace vůbec není potřeba. Pokud by sémantika indexů nebyla udržována, docházelo by při křížení k roztržení bitové reprezentace indexu a následnému složení indexu nového, který by ovšem mohl být mimo hranice vstupní množiny bodů. Vzniklá triangulace by tím pádem nemohla existovat.

Chromozom může tedy být reprezentován řetězcem indexů. Pokud máme připravenou počáteční generaci těchto chromozomů, může začít výpočet genetickými algoritmy. Genetické operátory jsou prakticky stejné jako operátory z nejjednodušší verze genetických algoritmů akorát má každý gen chromozomu bohatší výběr alel.

Poslední možné přizpůsobení úlohy si lze představit jako tři samostatné indexové pole, které tvoří triploid. Trojice alel genů na odpovídajícím lokusu ze všech tří chromozomů představuje trojúhelník. Tato formalizace úlohy je zřejmě nejsložitější a její funkčnost značně nepravděpodobná. Pokud by operátor křížení byl konstruován podobně jako při křížení diploidů, docházelo by při reprodukci k příliš velkým změnám a prakticky žádná část stávající trojúhelníkové sítě by nebyla zachována.

5.2.3. Přizpůsobení genetických algoritmů úloze

Další možnost využití genetických algoritmů předpokládá zachování stávající reprezentace stavu úlohy a úpravu genetických operátorů tak, aby byly možné pracovat s touto reprezentací. Operátor selekce může být opět zachován. Operátor mutace může být reprezentován jednoduchou změnou v trojúhelníkové síti, např. prohozením náhodně zvolené hrany. Nejsložitější je opět úprava operátoru křížení. Křížení lze např. implementovat jako náhodně generovanou rovinu procházející množinou vstupních bodů, která rozdělí trojúhelníkové sítě obou křížených chromozomů na tři části. Trojúhelníky nad rovinou, trojúhelníky pod rovinou a trojúhelníky protnuté rovinou. Protnuté trojúhelníky jsou v obou chromozomech odstraněny. Následuje rekombinace, kdy jsou prohozeny části křížených chromozomů. Jedná se metodu postavenou na DeWallově cross-overu [23]. Rekombinace je zakončena spojením těchto částí, která může být provedena některou robustní metodou pro vyplňování děr v trojúhelníkových sítích a spojování oddělených částí sítí, např. [25].

5.2.4. Problémy při řešení úlohy

Popis úpravy úlohy se doposud zaměřoval jen na reprezentaci chromozomů a genetických operátorů. Zatím nebyla diskutována kvalita vzniklé trojúhelníkové sítě. Cílem úlohy je získat lepši rekonstrukci, než byly rekonstrukce vložené do počáteční generace. Tento záměr představuje značný problém a silně koliduje s jednoduchostí popisu chromozomu. Rekonstrukce musí tvořit manifold a to musí být bráno v úvahu ohodnocovací funkcí. Tato funkce musí silně znevýhodňovat nebo rovnou zavrhovat triangulace, které obsahují různé typy defektů, tj. např. duplicitní trojúhelníky, protínající se trojúhelníky, degradované trojúhelníky, špatné vějíře trojúhelníků u některého ze vstupních bodů, více než dva sousedící trojúhelníky na některé hraně atd. To znamená, že při ohodnocování každého jedince dané generace musí být možné všechny tyto informace zjistit. Proto není možné použít jednouché reprezentace chromozomů z kapitoly 5.2.2. Tyto reprezentace sebou nenesou topologickou informaci, potřebnou k vyhodnocení těchto testů. Zásadní nevýhodou genetických algoritmů tak zůstává časová a paměťová náročnost, která je neúnosně zvýšena udržováním topologické informace pro jednotlivé chromozomy.

5.3. Využití znalostí

Na počátku řešení této práce bylo zamýšleno využití znalostí pro zcela nový proces rekonstrukce. Znalosti, které by byly vstupem do rekonstrukce by se omezily na přibližné tvary některých částí objektu, které by před optimalizací nadefinoval uživatel. Zamýšlena byla aplikace čtyř různých tvarů (definičních těles): roviny, obalové koule, obalového elipsoidu a obalového válce. Nejedná se tedy o žádný produkční systém nýbrž explicitní zadání obalových tvarů některých částí předpokládaného povrchu. Nejjednodušší aplikace spočívala v použití jediného definičního tělesa, kterým by byla rovina. Na tuto rovinu by se promítly vrcholy a udělala by se rovinná triangulace. Tedy rekonstrukce objektu ve 2,5D. Pro 3D objekty se vykonala podobná operace se složitějšími obalovými tělesy. Každá část rekonstruovaného povrchu by byla uzavřena alespoň do jednoho obalového tělesa. Během rekonstrukce by se v jednotlivých tělesech opět vytvořil průmět bodů na plášť obalového tělesa a triangulace by byla řešena jako rovinný problém v souřadnicovém systému pláště obalového tělesa (sférické souřadnice, válcové souřadnice, atd.) Rovinná triangulace by představovala průmět prostorové triangulace. Po rekonstrukci jednotlivých částí by se identifikovaly body a s nimi incidující trojúhelníky, které byly součástí více než jednoho obalového tělesa. Nad těmito oblastmi by byla zavolána jiná heuristická metoda (genetické

algoritmy nebo simulované žíhání), které by vytvořila napojení oblastí. Dodatečné vylepšení této metody by umožňoval přidat k obalovému tělesu i znaménko, které by identifikovalo, zda se na jeho plášti budou promítat a rekonstruovat části, které leží uvnitř nebo vně. V případě roviny by znaménko určovalo průmět bodů před nebo za rovinou.

Další nápady se snažily o získání „obrazů“ rekonstruovaného objektu v podobě 2D triangulací na 3D datech. Proces může být nastíněn na problému rekonstrukce fasád. Fasády nasnímané na technické univerzitě v Gratzu jsou téměř 2,5D objekty, které trpí velmi špatným vzorkováním a tudíž je stávající rekonstruktor není schopný správně rekonstruovat (Obr. 2-1). Nový rekonstruktor by k tomuto problému přistupoval pomocí projekčních rovin, na kterých by byly vytvářeny rovinné triangulace prostorové množiny vstupních bodů. Tyto projekční roviny by byly postupně umísťovány v různých pozicích, takže by se získaly různé triangulace vstupní množiny z různých pohledů. Tímto způsobem by bylo velice lehké získat velké množství různých triangulací stejné datové množiny, které by pak mohly sloužit jako vstup pro genetické algoritmy, které by se z pohledů snažily rekonstruovat původní povrch.

Vylepšení této metody předpokládá chytřejší triangulaci. Nejprve by byly body seřazeny podle vzdálenosti od projekční roviny a poté by začala triangulace inkrementálním vkládáním. Body by do triangulace byly vkládány podle jejich vzdálenosti od řídící roviny. Pokud by se vkládal nějaký bod do již triangulované oblasti, znamenalo by to, že bod leží až za daným trojúhelníkem. Zde by mohla být aplikována rozdílná kriteria posuzování tohoto jevu, která by rozhodla, zda bod do triangulace přidán bude či ne. Jako kriterium by mohl sloužit obvod trojúhelníku, ve kterém se nově vkládaný bod nalézá (příliš velký obvod by znamenal vložení bodu) nebo odchylka ve vzdálenosti od řídící roviny oproti vrcholům trojúhelníku (pokud by byla menší než prahová hodnota, bod by byl přidán). Tato metoda by v kombinaci s genetickými algoritmy pravděpodobně měla šanci fungovat.

Původní snaha byla vydat se touto cestou, ale vzhledem ke kolizi se zadáním, přílišnou náročností a nejistotou výsledku byla tato snaha počátkem únoru ukončena a začalo se od začátku na aplikaci stochastických metod.

Při problému optimalizace si můžeme znalosti představit např. v podobě definování hran, rohů či okrajů objektu. Tyto možnosti však v řešené úloze nebyly využity. Automatická detekce těchto jevů a jejich následné zvýraznění (optimalizaci) vyžaduje do procesu optimalizace

zapojit rozpoznávání. Rozpoznávání by mělo velmi ztíženou úlohu, protože hrany jsou většinou rekonstruovány velmi špatně a tak není co rozpoznávat.

5.4. Stochastické metody

Nejjednodušším řešením úlohy je využití stochastických algoritmů. Novým cílem úlohy se po zamítnutí práce se znalostmi stala implementace simulovaného žíhání, která by zlepšovala rekonstruovanou síť. Princip simulovaného žíhání je velmi jednoduchý, takže se úloha zpočátku jevila jako bezproblémová, ovšem problémy velmi brzy nastaly.

Při testování kritérií, které budou popsány dále v této kapitole bylo zjištěno, že simulované žíhání nepřináší požadované výsledky. Původní úsudek předpokládal, že je to špatnými ohodnocovacími metodami a tak byly vytvářeny další kritéria. Později byla provedena důkladná analýza metody systémem, který byl nazván Visual Debugger. Při této analýze se ukázalo, že problém není v kritériích, ale v simulovaném žíhání a jeho aplikaci. Princip simulovaného žíhání v počátečních fázích optimalizace přijímá i nekvalitní stavy. Míra přijetí těchto nekvalitních stavů je dána teplotou žíhání a spád teploty je dán koeficientem. Běžně se uvádí, že by koeficient měl být okolo 0,95. Toto tvrzení pro optimalizaci trojúhelníkových sítí neplatí. Důsledek tohoto nastavení je přijímání velkého počtu nesprávných stavů, které přivedou trojúhelníkovou síť do takového stádia, že už na ní kritéria nestačí a k optimalizaci nedojde.

Proto byla implementována druhá metoda, která představuje nejjednodušší verzi stochastického gradientního algoritmu. Tato metoda běží v pevně daném počtu kroků, ve kterých náhodně vybírá hranu, která bude prohozena. Pokud je nová triangulace kvalitnější než stávající, je tato triangulace přijata a nahradí stávající triangulaci. V opačném případě se pokračuje algoritmus s původní triangulací. Tato metoda ukázala, že některá z navrhovaných kritérií skutečně dokáží zlepšit rekonstruovaný povrch. Novým cílem úlohy se tak stalo zvýrazňování špatně rekonstruovaných hran.

Další testy kritérií a obou metod dokázaly tato tvrzení. Kritéria, která vykazovala na standardních rekonstrukcích při použití se stochastickým gradientním algoritmem dobré výsledky, byla použita i pro optimalizaci modelů, které předtím prošly simulovaným žíháním. Získané výsledky jasně ukázaly, že kritéria nejsou schopná z těchto stavů přejít ani zpět

k počátečnímu stavu, natož ho zlepšit. Poslední test upravoval hodnoty koeficientu pro úpravu teploty při simulovaném žíhání. S nízkými hodnotami koeficientu (např. 0,1) se algoritmus simulovaného žíhání začíná chovat celkem rozumně a produkuje výsledky podobně stochastickému gradientnímu algoritmu. Bohužel s touto hodnotou ztrácí tento algoritmus většinu ze své podstaty.

5.5. Ohodnocení trojúhelníkové sítě

Pro využití mnoha metod umělé inteligence je nezbytné mít možnost ohodnotit kvalitu aktuální trojúhelníkové sítě, aby bylo možné následně k úloze přistupovat jako k optimalizaci ohodnocovaní funkce. Vzhledem k tomu, že trojúhelníkové sítě v úloze řešené touto prací reprezentují rekonstruovaný povrch, mělo by toto ohodnocení kvalitativně popisovat samotnou rekonstrukci. Sestavit vhodné kritérium, které by skutečně dokázalo kvalitativně rozlišit jednotlivé rekonstrukce je jednoznačně nejtěžší částí této práce.

Dále v textu je navrženo osm základních kritérií, ze kterých každé z nich umožňuje několik způsobů použití. Tyto kritéria byla navrhována hlavně pro operaci prohazování hran mezi dvěma sousedícími trojúhelníky, ale některé z nich by pravděpodobně bylo možné použít i pro jiné operace. Při započítání všech možností je k dispozici 38 kritérií pro ohodnocení aktuálního stavu trojúhelníkové sítě. Hlavní předpoklad pro správnou aplikaci mnoha ze zmiňovaných kritérií je jednotná orientace trojúhelníkové sítě a správný směr normál.

5.5.1. Možné optimalizace

V kapitole 3.1 byl popsán stavový model, který se používá v oblasti řešení úloh a který lze použít i pro optimalizace. Stavový prostor L optimalizace trojúhelníkové sítě představuje všechny možné konfigurace trojúhelníků nad danou množinou bodů v prostoru (počet prvků této množiny je neměnný). Je zřejmé, že již pro stovky bodů bude stavový prostor obrovský. Pro definici stavového modelu S optimalizace trojúhelníkové sítě zbývá definovat množinu elementárních operátorů Φ, které umožní přechody mezi jednotlivými stavy. Elementární operátory jsou pouze tři. Jedná se o operátor prohození hrany mezi dvěma sousedícími trojúhelníky, operátor odebrání trojúhelníku a operátor přidání trojúhelníku. Aplikací posloupnosti těchto operátorů lze přejít mezi jakýmikoliv dvěma stavy ve stavovém prostoru. V případě, že by byl přidán další požadavek, který by říkal, že se nesmí měnit počet trojúhelníků optimalizované sítě, stavový prostor by se značně zmenšil a zůstal by jen jeden

elementární operátor – prohození hrany mezi dvěma sousedícími trojúhelníky. Tento typ úlohy je řešen v této práci.

Optimalizace celé trojúhelníkové sítě může být problematická, protože jednotlivá ohodnocovací kriteria se obvykle zaměřují pouze na zvýraznění určité vlastnosti v trojúhelníkové síti. Je pravděpodobné, že aplikace takového kritéria na celou trojúhelníkovou síť zlepší požadované části, ovšem také způsobí mnoho problémů v místech, kde daná vlastnost není žádána. Jedná se o typický problém lokálních kriterií (viz. 5.2.2). Příkladem může být např. zvýraznění hran nebo naopak vyhlazení některých částí modelu. Proto je vhodné mít možnost vybrat část trojúhelníkové sítě a pustit danou optimalizaci jen na této části.

5.5.2. Možnosti ohodnocení

Vzhledem ke složitosti řešené úlohy, lze k ohodnocování aktuální trojúhelníkové sítě přistupovat dvojím způsobem. Běžný způsob předpokládá globální ohodnocovací kriterium celé trojúhelníkové sítě. To je nutné pro metody, které pracují s trojúhelníkovou sítí jako s celkem (např. genetické algoritmy). Druhý způsob je v oblasti optimalizace méně obvyklý. Tento způsob předpokládá pouze ohodnocení části, nad kterou je vykonávána daná operace, např. operátor prohození hran z předešlého příkladu – důležité je ohodnotit pouze okolí, které je touto operací upraveno. Takové ohodnocení lze chápat jako lokální ohodnocení.

Ohodnocování trojúhelníkové sítě lze provádět dílčími kroky, které jsou ve výsledku spojeny do jedné hodnoty. Pokud je s každým dílčím krokem spojena jeho váha, lze výsledné hodnocení spočítat jako váženou sumu dílčích hodnot. Při této operaci je třeba dávat pozor na rozsah hodnot jednotlivých dílčích kroků. To je možné předvést na jednoduchém příkladě, kdy se bude vypočítávat výsledné ohodnocení ze dvou kritérií. Jedno bude brát v úvahu úhly v radiánech a druhé délky hran. Je jasné, že první kriterium má rozsah možných hodnot z intervalu $<0;2\pi>$. Jaké hodnoty může nabývat druhé kriterium? Druhé kritérium může k výsledné hodnotě přispívat v řádech miliontin stejně jako v řádech milionů. Nepoměr mezi rozsahy hodnot obou kritérií může znehodnotit jejich váženou sumu. Tento nepoměr je možné řešit již zmíněnými váhami, ale to vyžaduje dopředu znát rozsahy dílčích hodnot. Systematické řešení tedy předpokládá normalizaci dílčích hodnot. K normalizaci lze použít průměr nebo medián možných hodnot. Normalizované hodnoty se tak budou s určitým rozptylem pohybovat okolo hodnoty jedna a vážená suma tak začne dávat smysl. Různá

nastavení kombinací základních osmi kritérií s různými váhami tak dávají k dispozici nespočetné množství možností pro ohodnocení aktuálního stavu. Další problém, který je třeba řešit je rozdíl mezi minimalizací a maximalizací stavu. V tomto řešení se při minimalizaci používají záporné hodnoty a při maximalizaci kladné hodnoty ohodnocovací funkce.

Implementace stochastických metod přichází s další možností řídit vyhodnocovací kritérium. Je možné nastavit prahové hodnoty úhlů, mezi kterými musí být úhel svíraný dvěma trojúhelníky sousedícími na hraně vybrané pro prohození. Pokud toto kritérium není splněno, prohození nebude provedeno. Výhoda tohoto postupu spočívá v tom, že reaguje na aktuální stav trojúhelníkové sítě.

5.5.3. Úhlové kritérium

První navrhované kritérium pro ohodnocování při prohazování hran je nejkomplikovanější, protože umožňuje dvanáct různých způsobů jak ho spustit. Základní dělení představuje typ úhlu, který bude ohodnocován. Prvním typem je ohodnocení vnitřních úhlu, který svírají sousední trojúhelníky (*Full Angle*), druhým typem pak ohodnocení úhlů, které svírají normály trojúhelníků. Základním rozdílem mezi těmito typy je rozsah hodnot a interpretace hodnot. Ohodnocení vnitřního úhlu, který svírají dva sousedící trojúhelníky může nabývat hodnoty z intervalu $<0;2\pi>$, přičemž hodnoty 0 a 2π lze považovat za chybu v trojúhelníkové síti, naopak hodnota π říká, že trojúhelníky leží na stejné rovině. Tento postup měl sloužit ke skutečné maximalizaci nebo minimalizaci úhlů v trojúhelníkové síti. Úhly, které svírají normály trojúhelníků mohou nabývat hodnot z intervalu $<0;\pi>$, přičemž obě tyto hodnoty znamenají, že trojúhelníky leží v rovině. Úhel mezi normálami tedy představuje odchýlení od plochy. Toto dělení je dále doplněno o možnost maximalizace nebo minimalizace vybraných úhlů. Poslední dělení vyjadřuje s jakými úhly se bude pracovat. K dispozici jsou tři varianty, které jsou odvozeny od faktu, že při prohazování hran mezi dvěmi trojúhelníky je ovlivněno až šest trojúhelníků. Dva přímo sousedí s danou hranou a každý z nich může mít na zbylých hranách jednoho souseda (je počítáno s tím, že vstupní síť je manifold). Základní varianta tedy vyhodnocuje jen úhel mezi trojúhelníky (*Only Edge*), které incidují s prohazovanou hranou. Doplňková varianta pracuje se všemi ostatními úhly, kromě úhlů mezi těmito dvěma trojúhelníky (*Only Adjacencies*). Poslední varianta pracuje se všemi úhly (*Full Evaluation*). V případě, že je úhlů více, bere se jako výsledek kriteria jejich průměr.

Kombinace minimalizace úhlů, který svírají normály řešeného jen na hranách se sousedy nebo na všech hranách představuje nejlepší kritéria, které byly vytvořeny. Výsledky dosažené s tímto ohodnocováním lze nalézt v příloze 9.5. Velkým překvapením je, že tyto kritéria fungují vcelku dobře na celém tělese, tj. hrany sou zvýrazněny, ale nedochází k deformacím plochých oblastí.

5.5.4. Hranové kritérium

Hranové kritérium se snaží maximalizovat minimální délku hrany v trojúhelníkové sítí nebo naopak minimalizovat maximální délku hrany v trojúhelníkové síti. Jedná se spíše o kritérium vhodné do rovinných úloh, takže nepřináší žádné výsledky.

5.5.5. Obvodové kritérium

Obvodové kritérium umožňuje čtyři způsoby vyhodnocování. Jedná se o maximalizaci nebo minimalizaci celkového obvodu trojúhelníků zúčastněných u prohazování hrany. Druhé dvě možnosti představují maximalizaci minimálního obvodu trojúhelníku v trojúhelníkové síti nebo minimalizace maximálního obvodu trojúhelníku v trojúhelníkové síti. Poměrně dobré výsledky generovala celková minimalizace a minimalizace maximálního obvodu. Výsledky jsou ovšem horší než výsledky úhlového kritéria.

5.5.6. Povrchové kritérium

Povrchové kritérium nabízí stejné možnosti jako obvodové kritérium, ale jeho výsledky jsou prakticky nepoužitelné.

5.5.7. Kritérium vnitřních úhlů

Kritérium vnitřních úhlů pracuje na Delaunayovském principu. Snaží se maximalizovat minimální vnitřní úhel trojúhelníků v trojúhelníkové síti nebo naopak minimalizovat maximální vnitřní úhel trojúhelníků v trojúhelníkové síti. Stejně jako hranové kritérium nepřináší žádné výsledky. Jeho aplikace je zajímavá jen v případě, že je s nízkou vahou použito jako doplňkové kritérium k některému dalšímu. V nerozhodných situacích tak může být vybrána varianta, ve které mají trojúhelníky lepší tvar.

5.5.8. Normálové kritérium

Normálové kritérium si bere částečně inspiraci z úhlového kritéria. Toto kritérium vůbec nebere v úvahu prohazovanou hranu, zajímá se jen o normály zbylých hran obou trojúhelníků

incidujících s prohazovanou hranou. Pro každý trojúhelník je vypočítán úhel, který svírají normály jeho zbylých hran. Tím jsou získány dvě hodnoty, nad kterými lez provádět celkem šest operací. Jedná se o globální minimalizaci a maximalizaci součtu těchto hodnot, o minimalizaci maximální či maximalizaci minimální. Poslední dvě hodnoty představují minimalizaci nebo maximalizaci jejich rozdílu. Ani jedna z těchto možností nepřináší dobré výsledky.

5.5.9. Kritérium normály na hraně

Další hranové kritérium se naopak stará jen o prohazovanou hranu. V tomto kritériu se určí úhel mezi normálou hrany a normálami koncových vrcholů této hrany. Vznikají opět dvě hodnoty. Toto kritérium lze globálně maximalizovat nebo minimalizovat. Bohužel ani tyto možnosti nepřináší zajímavé výsledky.

5.5.10. Kritérium normály ve vrcholu

Poslední kritérium je založené na normálách trojúhelníků incidujících s prohazovanou hranou a na normálách vrcholů, které jsou proti této hraně. Pro každý z trojúhelníků se určí úhel mezi normálou trojúhelníku a normálou vrcholu, tj. vzniknou dvě hodnoty, se kterými je možné nakládat stejně jako u normálového kritéria. Toto kritérium přináší zajímavé výsledky při použití globální minimalizace nebo minimalizace maxima. Výsledky jsou kvalitativně někde mezi úhlovým kritériem a obvodovým kritériem.

5.5.11. Testy jednotlivých kritérií

Jelikož možností je hodně, obsahuje tento text ve své příloze jen ukázku výstupů, která dopadla skutečně dobře. Prozkoumání dalších možností je možné přímo v aplikaci, která může sloužit i jako prohlížeč .tri souborů. Pro všechny zde zmiňované možnosti kritérií, je na přiloženém DVD uložen výstup simulovaného žíhání s koeficientem 0.9 a stochastického algoritmu pro datovou množinu Mechpart.

6. Implementace

Výsledná aplikace (nazvaná Optimizer) implementující simulované žíhání a stochastický horolezecký algoritmus byla implementována na platformě Microsoft .NET Framework 2.0. Tento výběr byl učiněn na základě dvouletých zkušenostem s tímto prostředím, které ukázaly jednoduchost a rychlost vývoje aplikací. Aplikace pro .NET Framework lze vyvíjet v různých programovacích jazycích. Zvolen byl jazyk C#.NET 2.0, který byl vytvořen přímo na míru .NET Frameworku. Jazyk C# je syntakticky velmi podobný jazykům C++ a Java. Navíc lze použít kvalitní vývojové prostředí Microsoft Visual Studio 2005 Professional[1] a výborný manuál MSDN, které jsou na Katedře informatiky a výpočetní techniky Západočeské univerzity studentům k dispozici v rámci programu MSDNAA zdarma.

Programy vytvořené na platformě .NET Framework patří do skupiny tzv. řízeného kódu, stejně jako např. programy vytvořené na platformě Java. Tato prostředí neumožňují přímou manipulaci s pamětí známou např. z jazyku C nebo C++. O čištění paměti se na těchto platformách stará tzv. *garbage collector*. Řízený kód dále disponuje různými vnitřními ochranami, jako jsou např. kontrola mezí polí nebo silné typování. Tyto vlastnosti značně usnadňují vývoj aplikací, protože programátor se může více soustředit na řešený problém. Nevýhodou těchto prostředí je rychlost. Dodatečné testy a správa paměti, které jsou dělány automaticky ubírají aplikaci výkon, takže výborně napsaná aplikace v C#.NET bude vždy maximálně stejně rychlá jako průměrně napsaná aplikace v C++.

Pro vykreslování datových množiny byla vybrána řízená verze multimediálního balíku Microsoft DirectX 9.0c. Tato řízená verze se obvykle označuje Managed DirectX nebo též MDX. Vzhledem k přechodu z .NET Framework 1.1 na .NET Framework 2.0 začala vznikat nová verze tohoto balíku označovaná Managed DirectX 2.0 Beta. Tato verze byla vybrána pro tuto aplikaci, protože využívala nové jazykové konstrukce, dostupné v C#.NET 2.0. Tento výběr lze považovat za velmi nešťastný. Jelikož se jedná o beta verzi, je mnoho metod v jakémsi rozpracovaném stavu, který se projevuje formou parametrů, které metody požadují.

[1] Přiložené DVD obsahuje odlehčenou verzi tohoto vývojového prostředí nazvanou Microsoft Visual C# 2005 Express Edition, která je distribuována zdarma a instalaci běžné distribuce .NET Framework 2.0 i .NET Framework 2.0 SDK nutné pro vývoj a ladění aplikací.

Tyto parametry většinou odporují principu řízeného kódu. Jedná se např. o požadavky na velikosti datových typů v bytech. Pro získávání některých informací o používání nových metod bylo nutné účastnit se beta testovacího programu. Další problém představovaly některé metody známé z předešlé verze nebo z neřízené verze DirectX, které v nové verzi chyběly[1] a bylo nutné vývojový team žádat o jejich přidání. Vývoj této verze Managed DirectX byl bohužel v průběhu prvního pololetí roku 2006 ukončen, protože její vývojový team byl přesunut na projekt XNA (jiné řízené rozhraní). Důsledkem toho je, že tato verze Manager DirectX není součástí standardní distribuce. Pro její používání a tedy i pro používání je nezbytné mít nainstalované DirectX SDK[2].

6.1. Rozbor vstupních dat

Vstupní data jsou aplikaci předávána ve formě .tri souboru, který obsahuje jen souřadnice vrcholů a indexy trojúhelníků. Zde číhá první úskalí úlohy, které si vyžádalo značné programátorské úsilí. Soubor neobsahuje normály, které jsou klíčové pro většinu ohodnocovacích kritérií. S normálami je to větší problém než se na první pohled zdá, protože výstup rekonstrukce může obsahovat defekty, kdy nad existujícími dírami přechází vnitřní povrch do vnějšího povrchu. Řešení orientace trojúhelníků prohledáváním do hloubky není nad takovým modelem možné. Tento problém je ekvivalentní Mobiovu pásu. Jediné možné řešení, je nechat rozhodnout uživatele. To znamená, že je třeba naimplementovat výběr trojúhelníků.

6.2. Asynchronní operace

Některé operace jsou řešeny asynchronně. Asynchronní operace je reprezentována jednotlivými úkony, které se během ní mají provést. Výsledek jednoho úkonu je přitom předáván následujícímu úkonu. Tímto postupem je řešeno například načítání souboru, kdy se nejprve načte .tri soubor, poté se nad tri souborem vytvoří plná topologie a nakonec je sestaven OctTree. Výhoda asynchronní operace spočívá v tom, že program po celou dobu reaguje na události. Asynchronní operaci je možné zrušit a je možné vidět její průběh. To je důležité především u optimalizace, která může trvat řádově desítky minut.

[1] Ve skutečnosti většinou nechyběly, jen byly ve zdrojových kódech Managed DirectX 2.0 označeny jakou soukromé (private).
[2] Vhodné DirectX SDK, které tuto verzi obsahuje lze nalézt na přiloženém DVD.

6.3. Zobrazovací systém

Zobrazovací systém je postaven na knihovně, kterou vytvořil Chad Vernon. Tato knihovna je volně k dispozici na jeho webových stránkách [web5], kde lze nalézt i sadu velice užitečných tutoriálů, které vysvětlují jednotlivé části knihovny. Knihovnu bylo nutné značně upravit, protože její původní použití směřuje do oblasti tzv. *First-person* her. To má poměrně daleko k vizualizaci technických dat. Velkou změnou tak muselo především projít ovládání a využívání okna aplikace. Objektový model grafického jádra lze nalézt v přílohách na Obr. 9-. Velkou výhodou tohoto systému je jeho schopnost se přizpůsobit aktuální grafické kartě, která je v počítači k dispozici. Samozřejmě další nakládání s grafickými daty je již na samotném programátorovi, takže podpora starších grafických karet, které umožňují pouze 16ti bitové indexy do indexových polí (např. testovací sestava tři, viz. příloha 9.4) musela být implementována zvlášť.

Zvláštní pozornost si zaslouží interakce, která je řešena přes vykreslování modelu do offscreen bufferu. Každý trojúhelník je vykreslován jinou barvou, což umožňuje jeho jednoznačné určení. Barva je přímo mapována na index trojúhelníku v trojúhelníkové síti. Při kliknutí myší tedy stačí zjistit barvu na daném pixelu v offscreen bufferu a je tím okamžitě určen index vybraného trojúhelníku. Obrázek offscreen bufferu lze v aplikaci získat přes nabídku Debug.

6.4. Kontrola kvality sítě

Pro udržení kvality sítě a její manifoldnosti je nezbytné provádět při každém novém prohazování hran testy, které zjistí zda nedošlo k poškození sítě. Celkem jsou prováděny čtyři testy, jejichž vhodné umístění dokáže včas rozpoznat špatné prohození hrany a ušetřit tím mnoho dalších výpočtů.

6.4.1. Dvojitá hrana

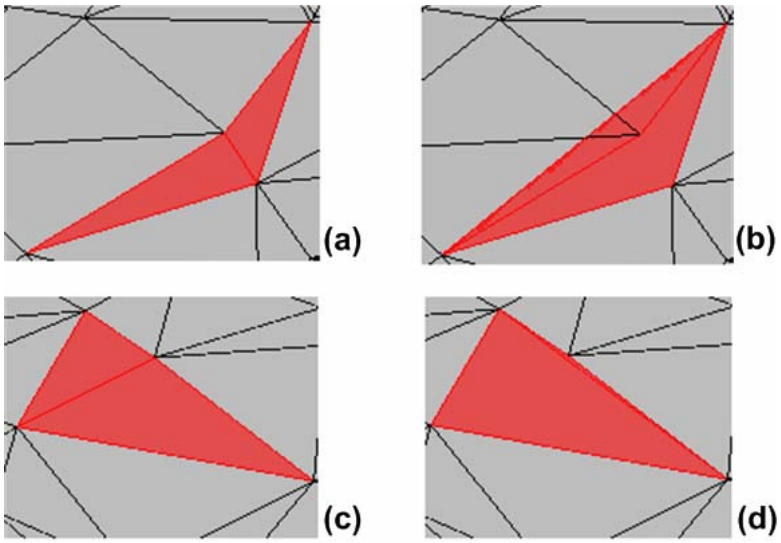
Nejjednodušší test kontroluje zda při prohození hrany nevznikla tzv. dvojitá hrana. Dvojitá hrana představuje situaci, kdy jsou mezi dvěma vrcholy dvě stejné hrany. Test je velice jednoduchý, stačí zkontrolovat jaké hrany vedou do jednoho z koncových vrcholů prohazované hrany a zda se tam nevyskytuje nějaká, která by měla stejný druhý koncový vrchol jako aktuální hrana. Test lze připojit hned na začátek operace prohození hrany.

6.4.2. Degradace trojúhelníku

K degradaci trojúhelníku dochází pokud je prohazovaná hrana mezi dvěma trojúhelníky, které u jednoho vrcholu hrany mají pravý úhel. Pokud dojde k pohození této hrany, jeden z trojúhelníků degraduje na pouhou úsečku. Tato situace se detekuje porovnáním dvojnásobku nejdelší hrany trojúhelníku s jeho obvodem. Tento test lze provádět až po operaci prohození hrany a jako jediný nemůže být vypnut, protože je klíčový pro další funkčnost metod.

6.4.3. Překrytí trojúhelníků

K překrytí trojúhelníků dochází za stejné situace jako k degradaci trojúhelníků, akorát úhly, které jsou u jednoho vrcholu hrany nejsou pravé, ale tupé. Po prohození hrany dojde k situaci, kdy jeden trojúhelník leží na druhém. Pro detekci této situace se testuje úhel, který svírají normály těchto trojúhelníků. Test je prováděn současně s testem na degradaci trojúhelníku.



Obr. 6-1 (a)(b) Problém překrytí trojúhelníků,
(c)(d) problém degradace trojúhelníku

.

6.4.4. Protnutí trojúhelníků

Největším problém při prohazování hran představoval test na protnutí dvou trojúhelníků. Bylo nutné implementovat efektivní test, který by zjistil zda po prohození hrany nedojde k protnutí jiného trojúhelníku. Pro testování se používá metoda překrytí intervalů jejíž popis lze nalézt v [17]. Kód k tomuto testu je možné najít na [web6]. Tento kód však bylo nutné značně

upravit, protože dostupná implementace počítala s testováním samostatného trojúhelníku, kdežto v tomto případě bylo potřeba testovat trojúhelník, který je prvkem trojúhelníkové sítě. Původní test posuzoval každého souseda, který s testovaným trojúhelníkem sdílel hranu nebo vrchol za protínající trojúhelník. Další problém, který bylo třeba řešit byla mohutný pokles rychlosti výpočtu, díky velkému množství prováděných testů. Tento problém byl vyřešen zavedením datové struktury OctTree nad trojúhelníkovou sítí. Vzhledem k náročnosti tohoto testu je jeho výpočet prováděn až ve chvíli, kdy je skutečně rozhodnuto, že prohození hrany bude akceptováno všemi ostatními částmi metody.

6.5. Oktalový strom

Oktalový strom (OctTree) je datová struktura využívaná pro dělení 3D prostoru, jejíž popis lze nalézt v [17]. V této aplikaci bylo OctTree původně používáno pro interakci, protože výběr trojúhelníků byl prováděn metodou odvozenou od Ray-Castingu. Později se pro OctTree našlo lepší uplatnění. OctTree je stavěno nad trojúhelníkovou sítí a jeho využití spočívá v redukci počtu trojúhelníků, které je třeba testovat na protnutí při operaci prohazování hrany. Pro ladění umožňuje aplikace zobrazit celý oktalový strom, případně jen jeho uzly nebo listy, které se týkají aktuálního výběru trojúhelníků.

6.5.1. Poloha trojúhelníku vůči kvádru

Základní operací pro vystavění OctTree nad trojúhelníkovou sítí je určení polohy trojúhelníku vůči kvádru. K tomu se používá metoda založená na dělící ose, kterou publikoval Alenině-Moller [17]. Zrojový kód této metody lze nalézt na [web7]. Tento zdrojový kód bylo nutné důkladně odladit, protože byl numericky nestabilní. Problémy dělaly trojúhelníky, které ležely na plášti hranolu. Obzvlášť vážná byla situace s trojúhelníky, které byly umístěny po obvodě OctTree, protože se mohlo stát, že nebyly součástí žádného listu a tím pádem se v OctTree vlastně ztratily. Bylo třeba přidat vstupní parametr konstrukce OctTree, který udává epsilón. Tato hodnota je pak používána pro porovnávání v těchto testech.

Výhody využití OctTree jsou vidět v posledním grafu přílohy 9.3, kde je zobrazena doba výpočtu optimalizace pro sto tisíc iterací v závislosti na parametrech OctTree. Tato doba se pohybovala mezi dva a půl až tři a půl minuty. Doba bez použití OctTree byla téměř jednu hodinu a deset minut.

6.5.2. Urychlení předzpracováním

Při stavbě OctTree nad trojúhelníky se nabízí možnost předzpracování, která značně zmenší počet testů polohy trojúhelníku proti kvádru. Toto předzpracování pro daný uzel OctTree spočívá v rozdělení prostoru do osmi subprostorů, které se sbíhají ve středu tohoto uzlu. Každý potomek tohoto uzlu tak bude právě v jedno z těchto subprostorů. Následně se pro každý trojúhelník vyhodnotí poloha jeho vrcholů vůči těmto subprostorům. Tím vznikne přístupový vektor do tabulky, která byla vytvořena po vzoru algoritmu Marching Cubes. V této tabulce je pro každý z 256 možných přístupových vektorů určeno, které potomky aktuálního uzlu je třeba testovat s právě zpracovávaným trojúhelníkem. Příloha 9.3 ukazuje grafy pro urychlení výstavby OctTree s využitím tohoto předzpracování a pro úsporu testů při použití předzpracování.

6.5.3. Úprava stromu po prohození hran

Nevýhoda použití OctTree jsou jeho statické vlastnosti. Při změně hrany v trojúhelníkové síti může dojít k přesunu trojúhelníků z některých uzlů. Na tuto operaci je nutné nějak rozumně reagovat. Po změně hrany nastává složitá operace rekonfigurace OctTree, která je ovšem lokalizovaná jen na zasažené uzly.

7. Závěr

V rámci této práce bylo prozkoumáno softwarové vybavení pro rekonstrukci trojúhelníkových sítí, které je dostupné na Západočeské univerzitě. Následně byly analyzovány metody umělé inteligence, které by mohly zlepšit výstup z tohoto programu.

Pro realizaci byla nakonec vybrána metoda simulovaného žíhání, která byla později doplněna i o stochastický horolezecký algoritmus. Zřejmě největším úspěchem práce je aplikace kritéria, které je schopné pracovat na celém objektu a zvýraznit jeho špatně rekonstruované hrany. Byly navrženy i některá další kritéria včetně systému pro vytváření vážených kombinací vytvořených kritérií. Možnosti aplikace tak dalece převýšily očekávání, přestože dosažené výsledky za očekáváním zaostávají.

V implementaci byly realizovány i některé další zajímavé postupy, jako je OctTree s preprocessingem, výběr trojúhelníků pomocí myši nebo vizuální ladění běhu optimalizace. Aplikace tak narostla do velkých rozměrů.

Tato práce měla velmi ambiciózní záměr, který jí dal neuvěřitelný rozsah. Tomu nasvědčuje i teoretická část tohoto dokumentu, která se nejprve zabývá rekonstrukcí povrchů a poté umělou inteligencí. Dá se tedy říci, že zde byly teoretické části dvě což vzhledem k časovému presu při dokončování práce negativně poznamenalo implementační část a uživatelskou dokumentaci, která by jinak byla obsahově mnohem bohatší a rozvinutější.

8. Literatura

[1] M. Varnuška, *Rekonstrukce povrchů geometrických objektů z roztroušených bodů*, Diplomová práce, Západočeská univerzita, Plzeň 2002

[2] M. Varnuška, *Surface reconstruction of geometrical objects from scattered points*, Disertační práce, Západočeská univerzita, Plzeň 2005

[3] N. Amenta, M. Bern, *Surface Reconstruction by Voronoi Filtering*, 1998

[4] N. Amenta, M. Bern, M. Kamvysselis, *A New Voronoi-Based Reconstruction Algorithm*, SIGGRAPH 1998, 415-421

[5] N. Amenta, Sunghee Choi, *One-Pass Delaunay Filtering for Homeomorphic 3D Surface Reconstruction*, TR99-08, 1999

[6] N. Amenta, S. Choi, R. K. Kolluri, *The Power Crust*, ACM 2001, ISBN: 1-58113-366-9/01/06

[7] B. Mederos, N. Amenta, L. Velho, L. Figueiredo, *Surface Reconstruction from Noisy Point Clouds*, Eurographics Symposium on Geometry Processing, 2005

[8] N. Amenta, S. Choi, T. Dey, N. Leekha, *A Simple Algorithm for Homeomorphic Surface Reconstrucition*, 16th Sympos. Computational Geometry, ACM, 2000

[9] T. Dey, J. Giesen, J. Hudson, *Delaunay Based Shape Reconstruction from Large Data*, Proc. IEEE Sympos. In Parallel and Large Data Visualisation and Graphics, 2001, pp. 19 - 27

[10] T. Dey, S. Goswami, *Tight Cocone: A Water-tight Surface Reconstructor*, Proc. 8th ACM Sympos. Solid Modelling Application, 2003, pp. 127 - 134

[11] T. Dey, S. Goswami, *Provable Surface Reconstruction from Noisy Samples*, Proc. of 20 annual symposium on Computational Geometry, SCG, 2004, pp. 330 – 339

[12] H. Hoppe, *Surface Reconstruction from Unorganized Points*, Doctoral Thesis, Department of Computer Science and Engineering, University of Washington, 1994

[13] J. Kohout, *Delaunay triangulation in parallel and distributed environment*, Disertační práce, Západočeská univerzita, Plzeň, 2005

[14] H. Xie, K. T. McDonell, H. Qin, *Surface Reconstruction of Noisy and Defective Data Sets*, IEEE Visualisation, 2004, ISBN: 0-7803-8788-0/04

[15] Y. Ohtake, A. Belyaev, H. P. Seidel, *An Integrating Approach to Meshing Scattered Point Data*, ACM 2005, ISBN: 1-59593-015-9/05/0006

[16] J. O'Rourke, *Computational Geometry in C*, Second edition, Cambridge University Press, 1998, ISBN: 0-521-64976-5

[17] T. Akenine-Möller, E. Haines, *Real-Time Rendering*, Second edition, A K Press, Ltd., 2002, ISBN: 1-56881-182-9

[18] M. Mitchell, *An Introduction to Genetic Algorithms*, The MIT Press, 1998, ISBN: 0-262-63185-7

[19] V. Mařík, O. Štěpánková, J. Lažanský a kol., *Umělá inteligence 1*, Academia Praha 1993, ISBN 80-200-0496-3

[20] V. Mařík, O. Štěpánková, J. Lažanský a kol., *Umělá inteligence 2*, Academia Praha 1997, ISBN 80-200-0504-8

[21] V. Mařík, O. Štěpánková, J. Lažanský a kol., *Umělá inteligence 3*, Academia Praha 2001, ISBN 80-200-0472-6

[22] S. Kirkpatrick, G. D. Gelatt, Jr., M. P. Vecchi, *Optimization by Simulated Annealing*, Science, Volume 220, 1983, pp. 671 – 680,

[23] I. Kolingerova, *Probabilistic Methods for Triangulated Models*, 8th International Conference on Computer Graphics and Artficial Intelligence 3IA 2005, Limoges, France, May 2005

[24] Y. Yu, *Surface Reconstruction from Unorganized Points Using Self-Organizing Neural Networks*, Proceeding of IEEE Visualization, 1999, 61-64

[25] A. Emelyanov, *Surface reconstruction form clouds of points*, Disertační práce, Západočeská univerzita, Plzeň, 2004

[web1] http://www.kiv.zcu.cz/studies/predmety/uir/

[web2] http://cgal.inria.fr/Reconstruction/

[web3] http://www.cse.ohio-state.edu/~tamaldey/cocone.html

[web4] http://genetika.wz.cz/pojmy.htm

[web5] http://www.c-unit.com/

[web6] http://jgt.akpeters.com/papers/Moller97/tritri.html

[web7] http://jgt.akpeters.com/papers/AkenineMoller01/tribox.html

9. Přílohy

9.1. Uživatelský manuál

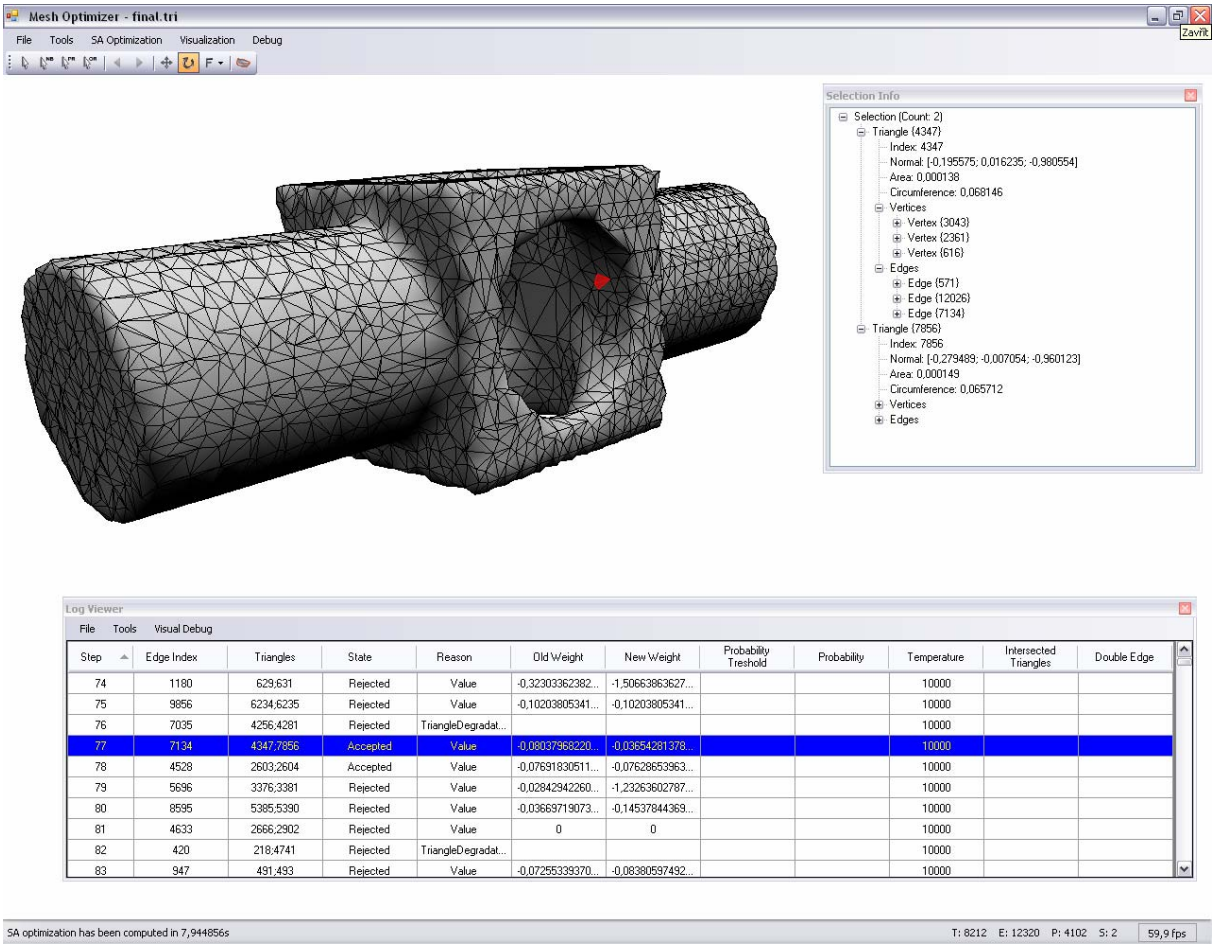
Při vytváření tohoto programu byla snaha o jeho srozumitelnost a jednoduchost, proto zde nebude popisován každý ovládací prvek, který je v programu obsažen. Těchto prvků je skutečně velmi mnoho a téměř všechny obsahují nějaký tooltip, který občas dá uživateli radu k čemu daný ovládací prvek slouží.

Pro spuštění programu je nutné mít nainstalovánu platformu .NET Framework 2.0 a správnou verzi DirectX SDK. Instalace obojího lze nalézt na přiloženém DVD. Instalace samotného programu probíhá prostým zkopírováním z DVD.

Po spuštění programu se uživateli ukáže jednoduché okénko v němž je v tuto chvíli nejdůležitější položka nabídky `File` a `Open`. Po načtení souboru se zpřístupní velké množství dalších položek v nabídkách a panelu nástrojů, které umožní uživateli začít práci. Pro výběr trojúhelníků a interakci s modelem slouží panel nástrojů, který se nachází pod hlavním menu. Pro uložení souboru, ukončení aplikace nebo uložení obrázku slouží nabídka `File`. Pro úpravu normál a nastavení OctTree slouží nabídka `Tools`. Nabídka `SAOptimization` spouští proces optimalizace trojúhelníkové sítě podle zadaných kritérií. V nabídce `Visualisation` je možné zapnout co se má uživateli zobrazovat, případně v podnabídce `Options` je možné některé vlastnosti vizualizačního jádra. Většina vlastností, které jsou v těchto nabídkách k dispozici odpovídá tématům probraným v druhé polovině tohoto textu. Některé položky nabídek jsou závislé na výběru trojúhelníků.

Poslední nabídkou je `Debug`. Tato nabídka sloužila především k ladění vlastností aplikace. Obtížné ladění představovala především interakce, OctTree a samozřejmě samotné simulované žíhání. Poslední dvě položky této nabídky umožňují přístup k tzv. `Log Viewer`. Log Viewer je část tohoto programu, která procházet jednotlivé kroky, které provedla stochastická metoda při svém vyhodnocování. Pro využití této metody je nutné nejprve spustit optimalizaci se zapnutou volbou `Use Full Logging`. Při tomto způsobu práce se v průběhu optimalizace zaznamenává každý krok, který algoritmus provedl. Po dokončení výpočtu je

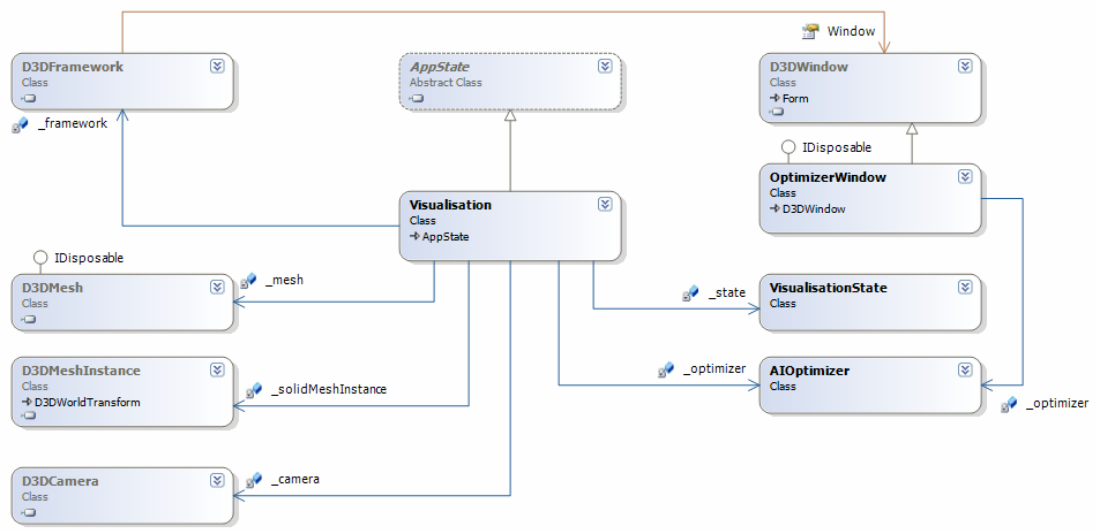
celý průběh uložen do souboru log.xml v adresáři aplikace. Tento soubor je pak možné otevřít v Log Vieweru, zvolením `Open Last`. Procházením načteného seznamu je možné sledovat jednotlivé kroky optimalizace, případně kontrolovat jednotlivé testy. Po vybrání řádku v Log Vieweru se trojúhelníková síť zpětně transformuje do stavu, ve kterém byla v daném řádku a trojúhelníky, mezi kterými je aktuálně zpracovávaná hrana jsou označeny. Pro práci s označenými trojúhelníky lze ještě z nabídky Debug vybrat položku `Selection Info`, která umožní zjistit veškeré dostupné informace o vybraných trojúhelnících (Obr. 9-1). Jelikož jak Log Viewer tak Selection Info dialog jsou určeny hlavně pro ladění a testování, je třeba si dávat pozor jaké množství dat jim je předáváno. Klasický případ, který aplikaci na nemalou chvíli zmrazí je otevření Selection Info dialogu a zadání příkazu `Select All` z nabídky `Tools`.



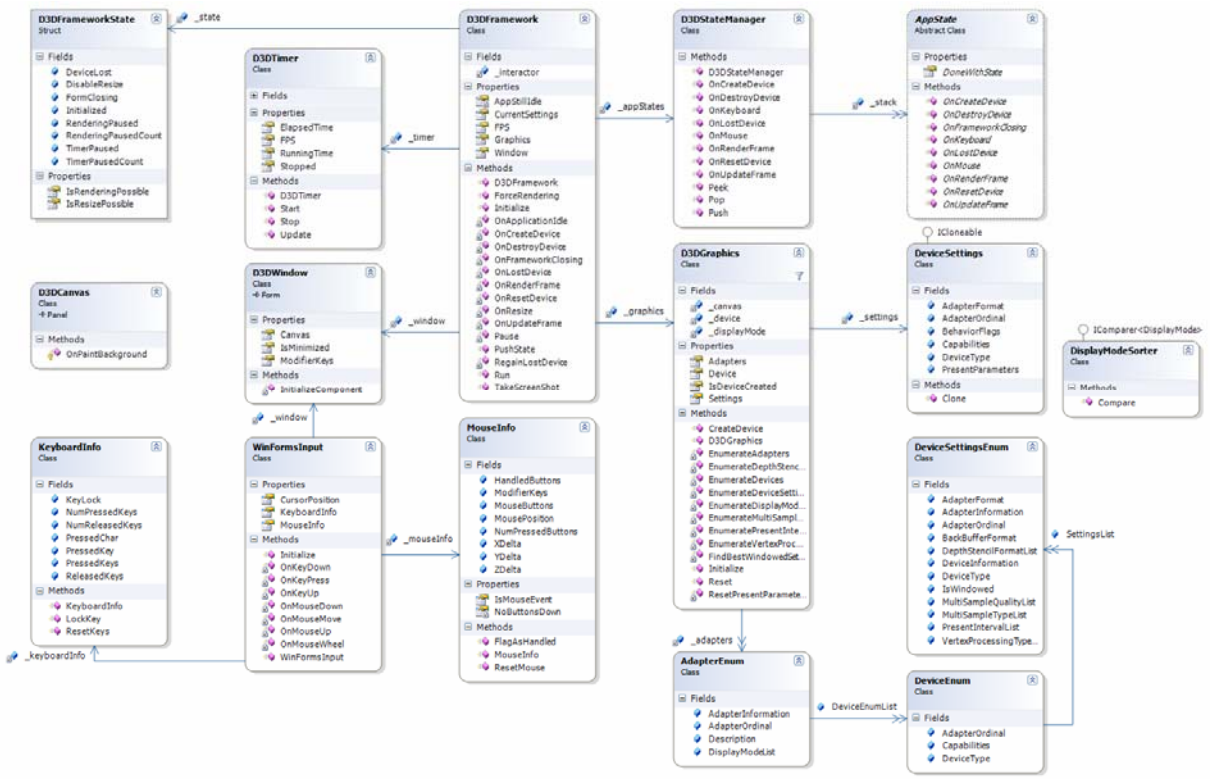
Obr. 9-1 Ukázka aplikace se zapnutým vizuálním laděním. Dole je vidět Log Viewer, ve kterém je vybrán 77. krok při optimalizaci. Zobrazovaný model je upraven tak, aby tomuto kroku odpovídal. Dále jsou vybrány trojúhelníky, se kterými se v tomto kroku pracovalo. Detailní informace o těchto trojúhelnících je zobrazena v okně Selection Info.

9.2. DSL diagramy

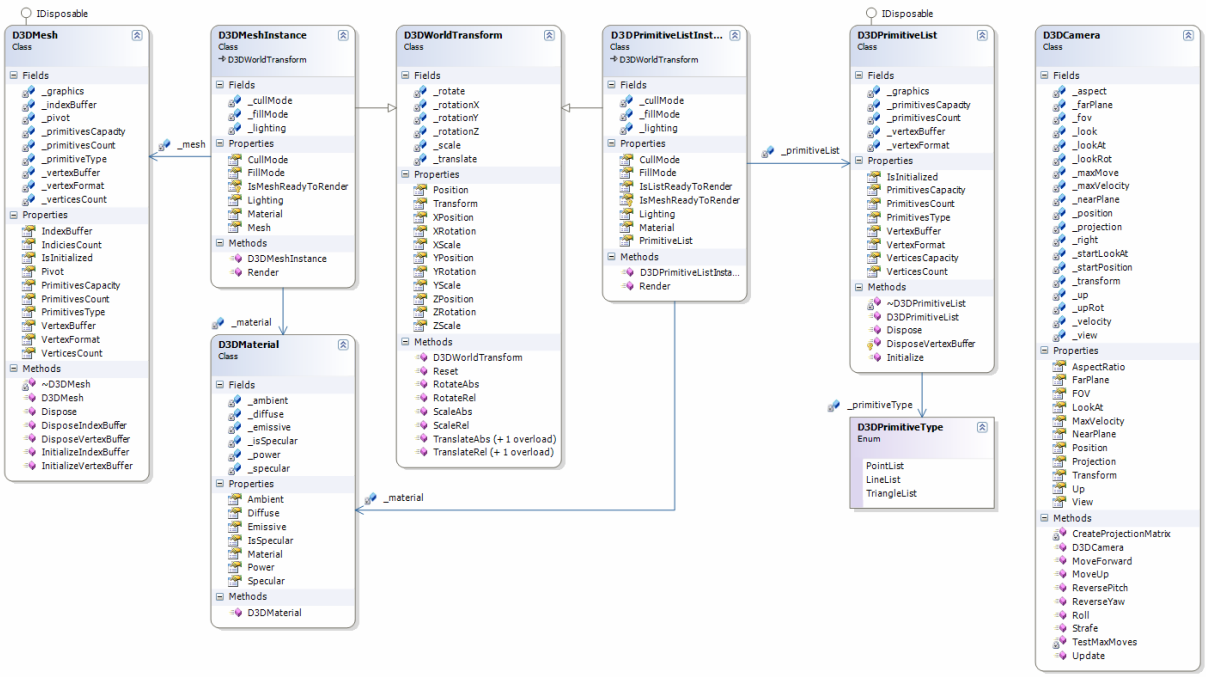
Standardní popis objektového modelu je prováděn UML diagramy tříd. UML (Unified Modeling Language) ovšem neumožňuje modelovat některé jazykové konstrukce jazyka C#. Proto jsou pro diagramy tříd použity DSL (Domain Specific Language) diagramy, které přestavují specializaci UML na jazyk C#.



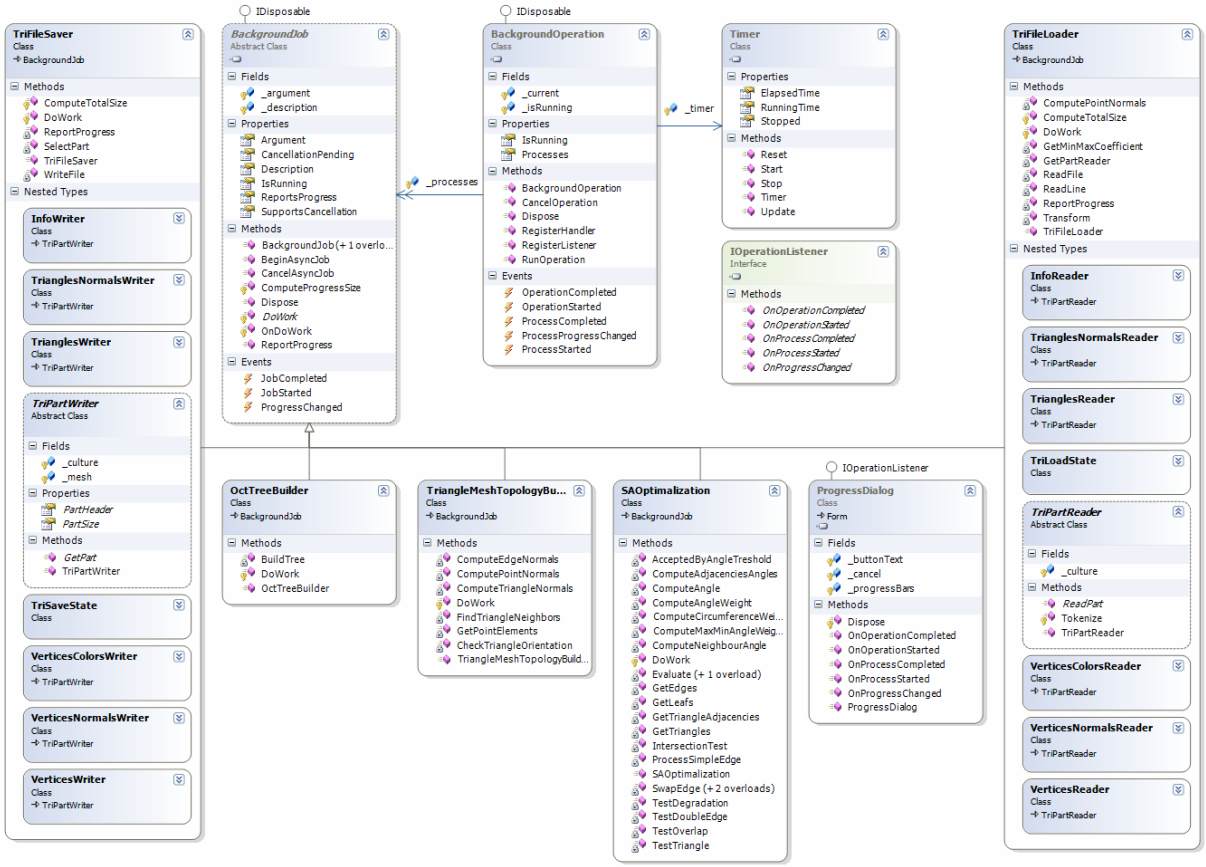
Obr. 9-2 Základní zjednodušené schema aplikace Optimizer



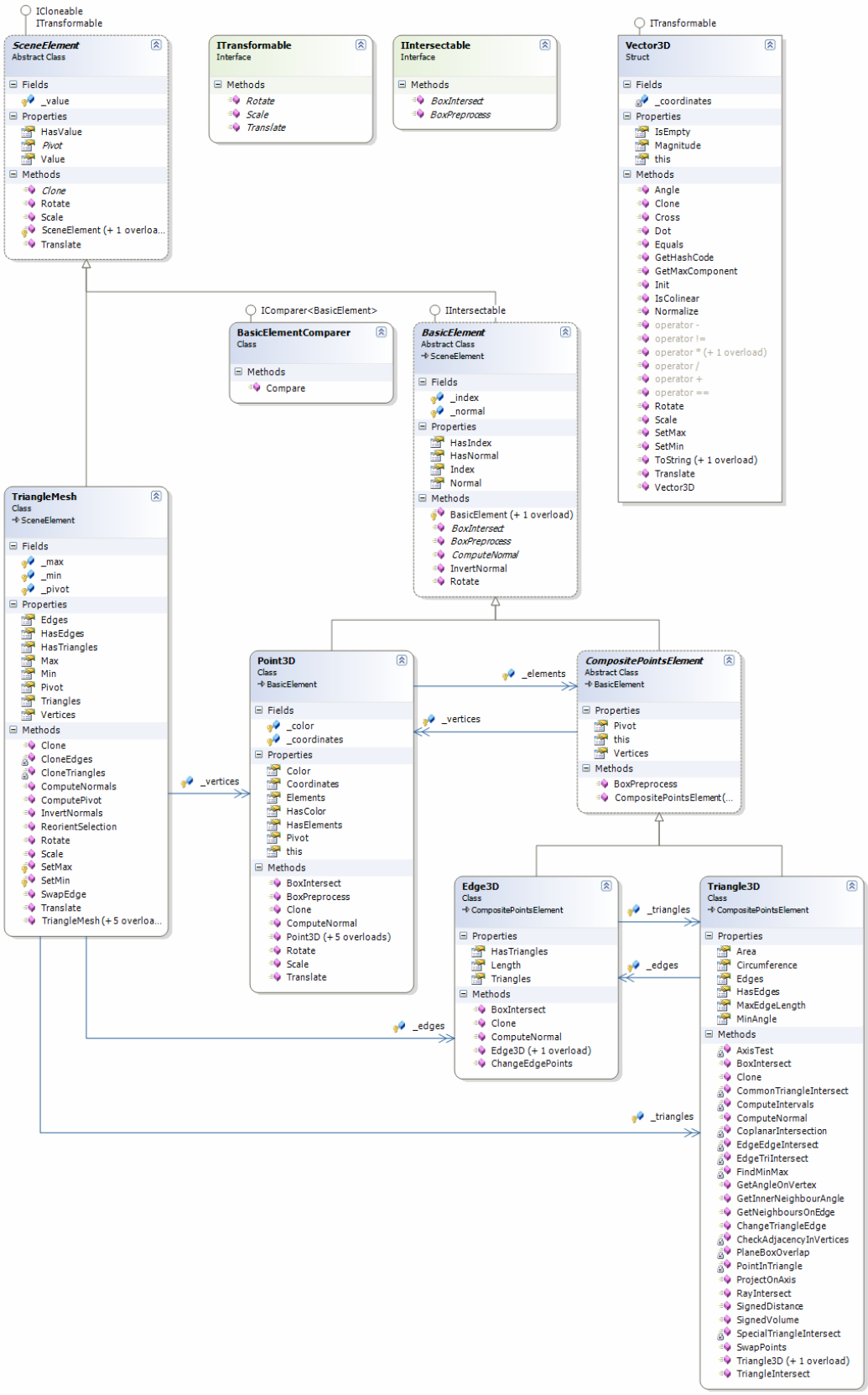
Obr. 9-3 Objektový model vizalizačního modulu.



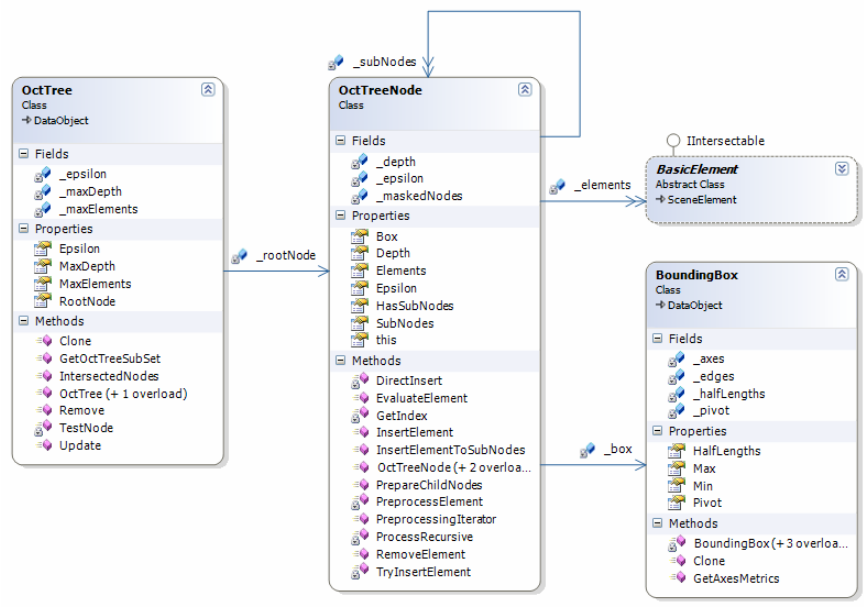
Obr. 9-4 Objektový model zobrazovaných elementů.



Obr. 9-5 Objektový model pro asynchronní operace.

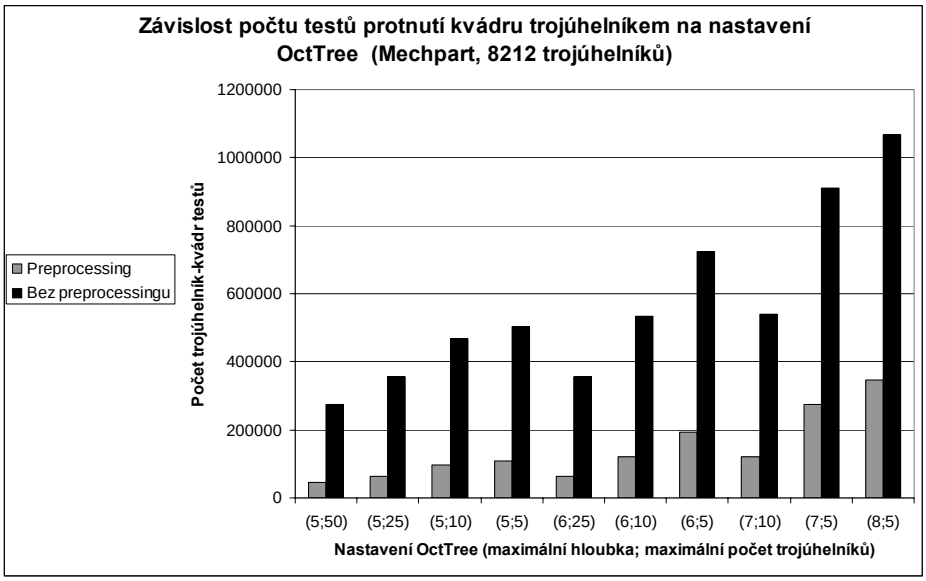


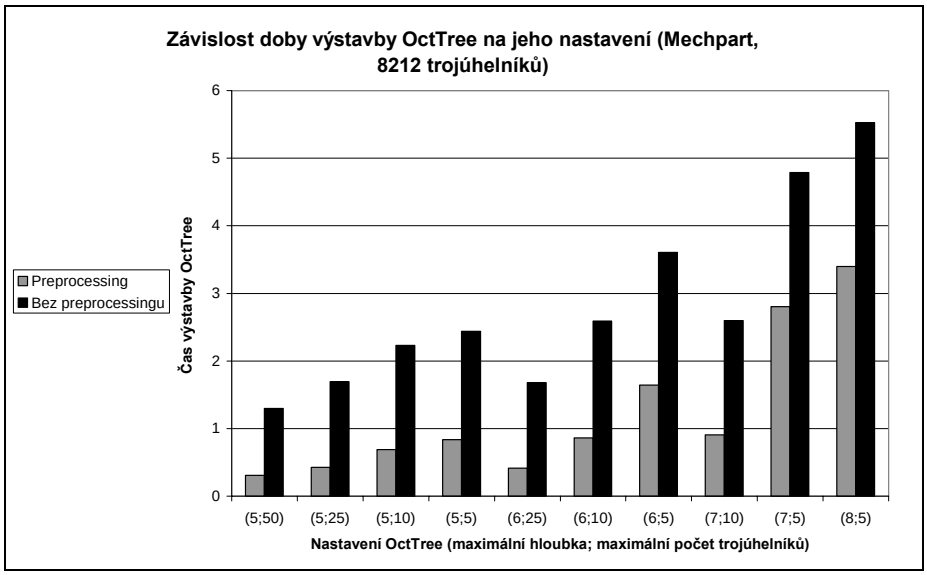
Obr. 9-6 Objektový model reprezentace trojúhelníkové sítě s plnou topologií.

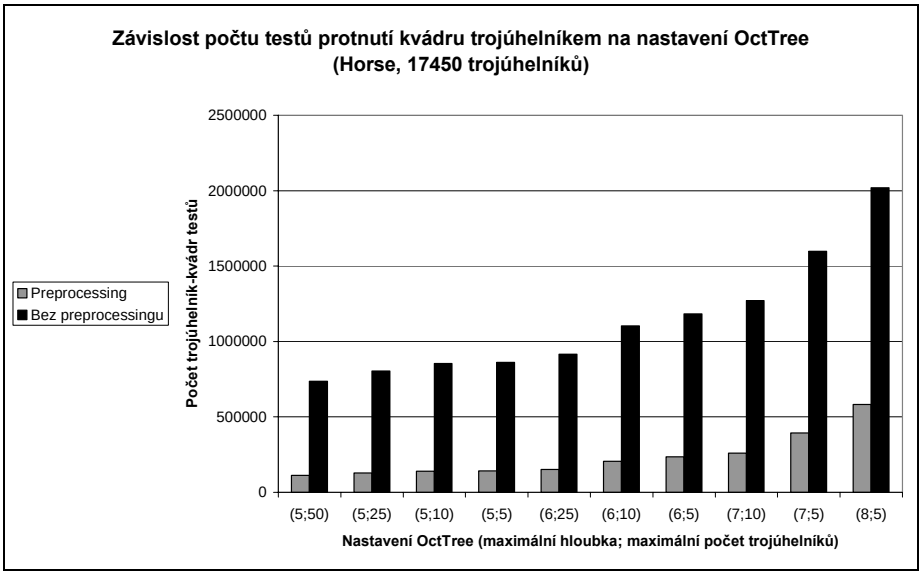


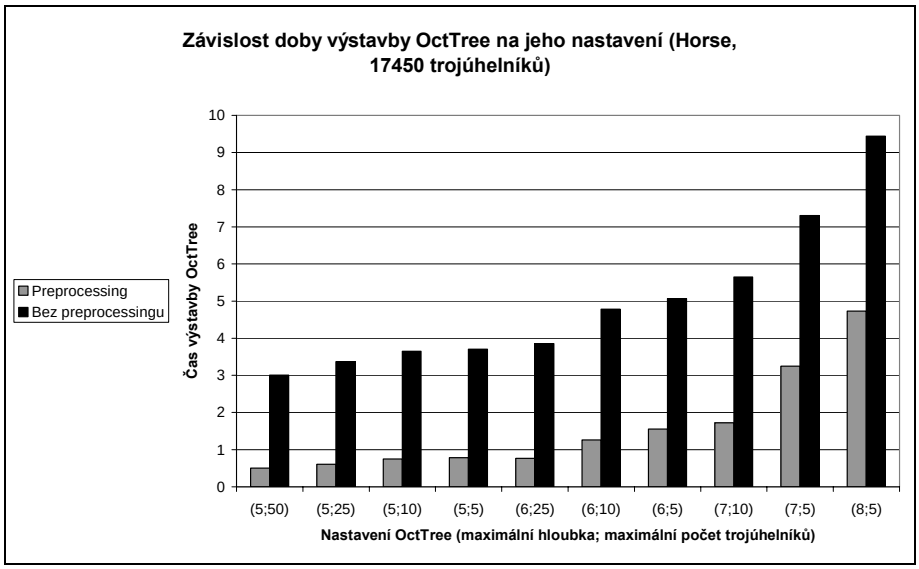
Obr. 9-7 Objektový model OctTree

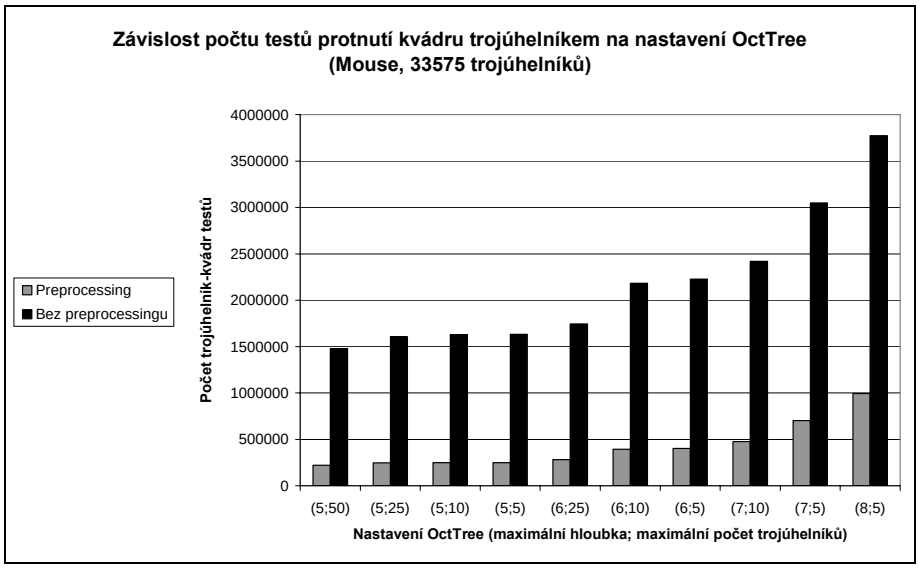
9.3. Grafy testů OctTree

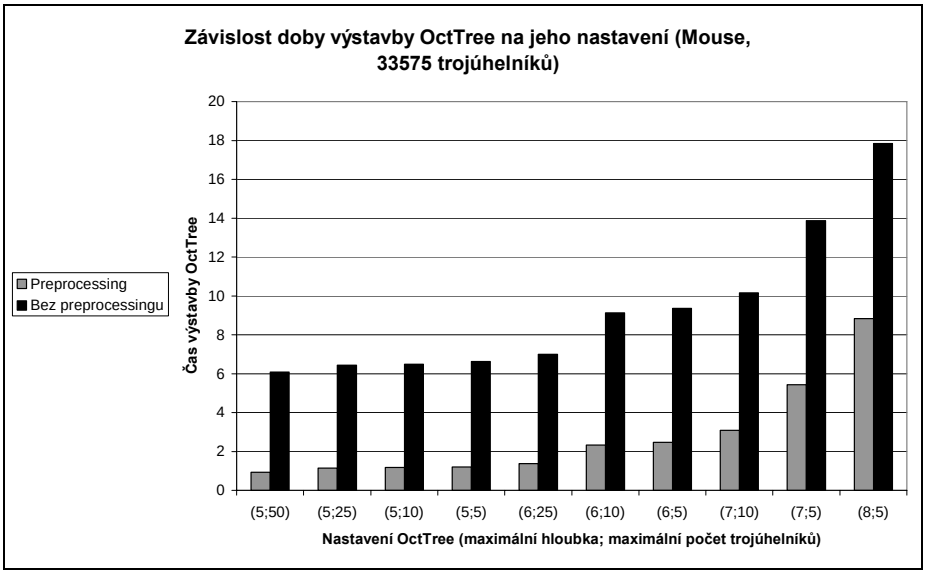


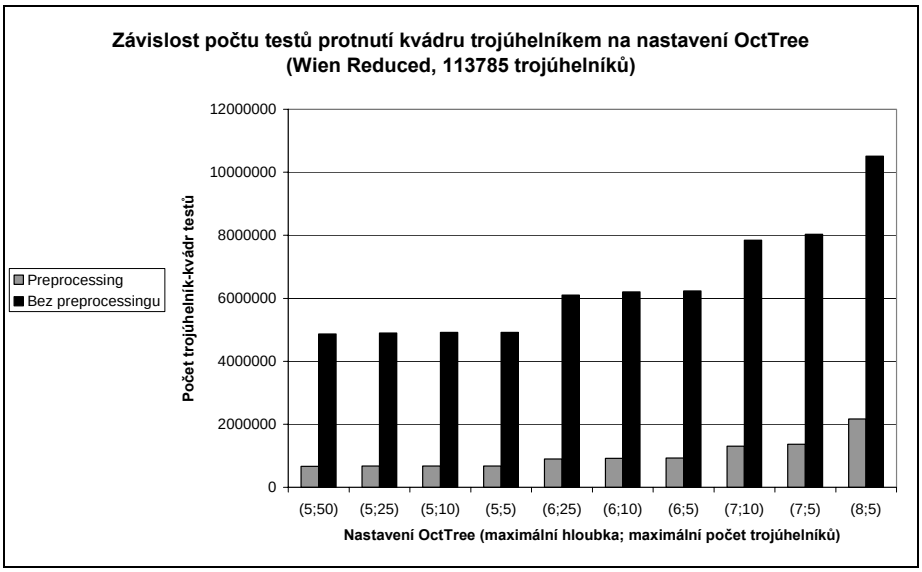


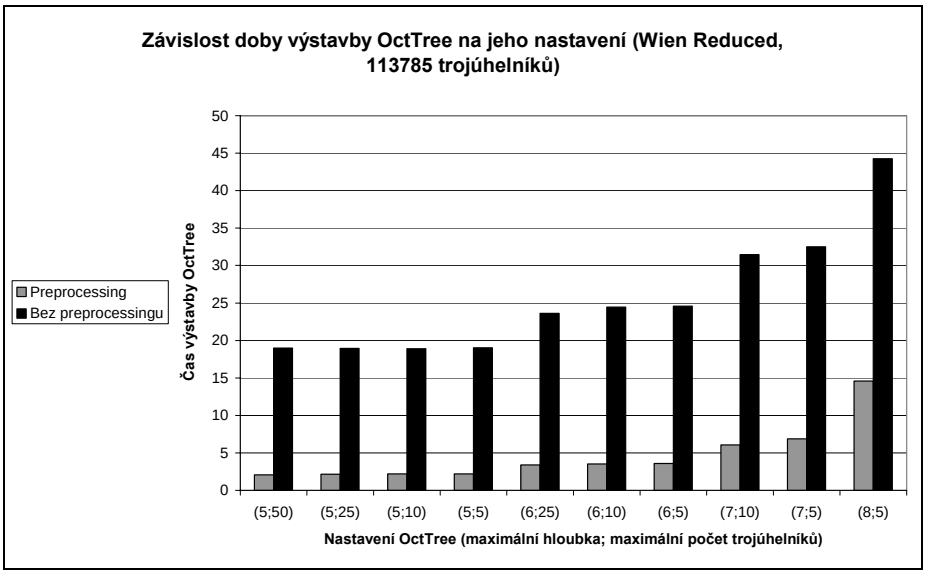


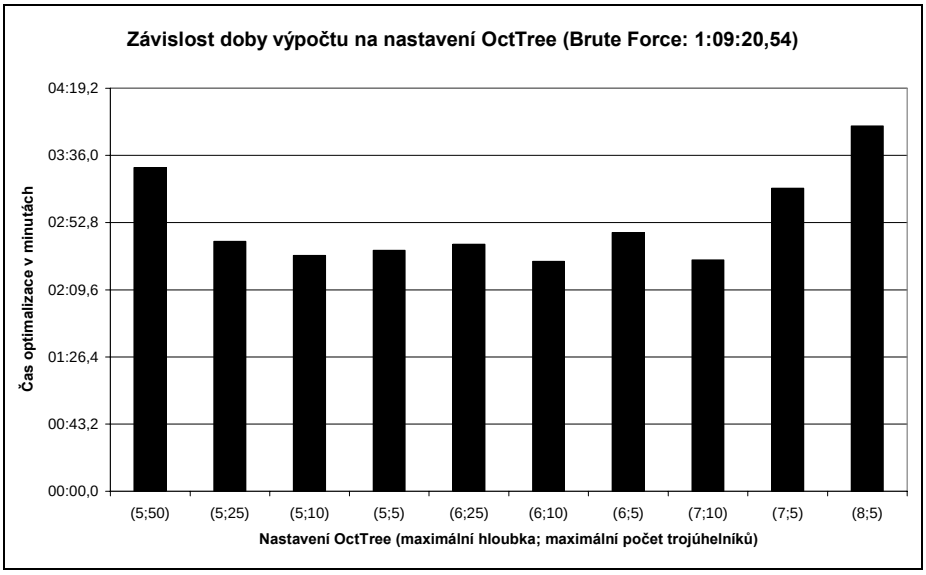

Závislost doby výstavby OctTree na jeho nastavení (Horse, 17450 trojúhelníků)
Preprocessing
Bez preprocessingu
Čas výstavby OctTree
0
1
2
3
4
5
6
7
8
9
10
(5;50)
(5;25)
(5;10)
(5;5)
(6;25)
(6;10)
(6;5)
(7;10)
(7;5)
(8;5)
Nastavení OctTree (maximální hloubka; maximální počet trojúhelníků)


Závislost počtu testů protnutí kvádru trojúhelníkem na nastavení OctTree (Mouse, 33575 trojúhelníků)
Preprocessing
Bez preprocessingu
Počet trojúhelník-kvádr testů
0
500000
1000000
1500000
2000000
2500000
3000000
3500000
4000000
(5;50)
(5;25)
(5;10)
(5;5)
(6;25)
(6;10)
(6;5)
(7;10)
(7;5)
(8;5)
Nastavení OctTree (maximální hloubka; maximální počet trojúhelníků)


Závislost doby výstavby OctTree na jeho nastavení (Mouse, 33575 trojúhelníků)
Preprocessing
Bez preprocessingu
Čas výstavby OctTree
0
2
4
6
8
10
12
14
16
18
20
(5;50)
(5;25)
(5;10)
(5;5)
(6;25)
(6;10)
(6;5)
(7;10)
(7;5)
(8;5)
Nastavení OctTree (maximální hloubka; maximální počet trojúhelníků)


Závislost počtu testů protnutí kvádru trojúhelníkem na nastavení OctTree
(Wien Reduced, 113785 trojúhelníků)
Preprocessing
Bez preprocessingu
Počet trojúhelník-kvádr testů
12000000
10000000
8000000
6000000
4000000
2000000
0
(5;50)
(5;25)
(5;10)
(5;5)
(6;25)
(6;10)
(6;5)
(7;10)
(7;5)
(8;5)
Nastavení OctTree (maximální hloubka; maximální počet trojúhelníků)


Závislost doby výstavby OctTree na jeho nastavení (Wien Reduced,
113785 trojúhelníků)
Preprocessing
Bez preprocessingu
Čas výstavby OctTree
50
45
40
35
30
25
20
15
10
5
0
(5;50)
(5;25)
(5;10)
(5;5)
(6;25)
(6;10)
(6;5)
(7;10)
(7;5)
(8;5)
Nastavení OctTree (maximální hloubka; maximální počet trojúhelníků)


Závislost doby výpočtu na nastavení OctTree (Brute Force: 1:09:20,54)
Čas optimalizace v minutách
04:19,2
03:36,0
02:52,8
02:09,6
01:26,4
00:43,2
00:00,0
(5;50)
(5;25)
(5;10)
(5;5)
(6;25)
(6;10)
(6;5)
(7;10)
(7;5)
(8;5)
Nastavení OctTree (maximální hloubka; maximální počet trojúhelníků)

9.4. Testovací sestavy

Testovací sestava 1:

Procesor: AMD XP 2100+ (jádro Palomino)

Paměť: 1GB DRR 333MHz

Grafická karta: ATI Radeon 9500 128MB DDR

Disk: 60GB 7200 RPM

Testovací sestava 2:

Notebook Prestigio Nobile 157

Procesor: Pentium M 1,6GHz

Paměť: 512MB DDR 333MHz

Grafická karta: ATI Radeon 9700 M 64MB DDR

Disk: 40GB 5400 RPM

Testovací sestava 3:

Notebook Dell Latitude D610

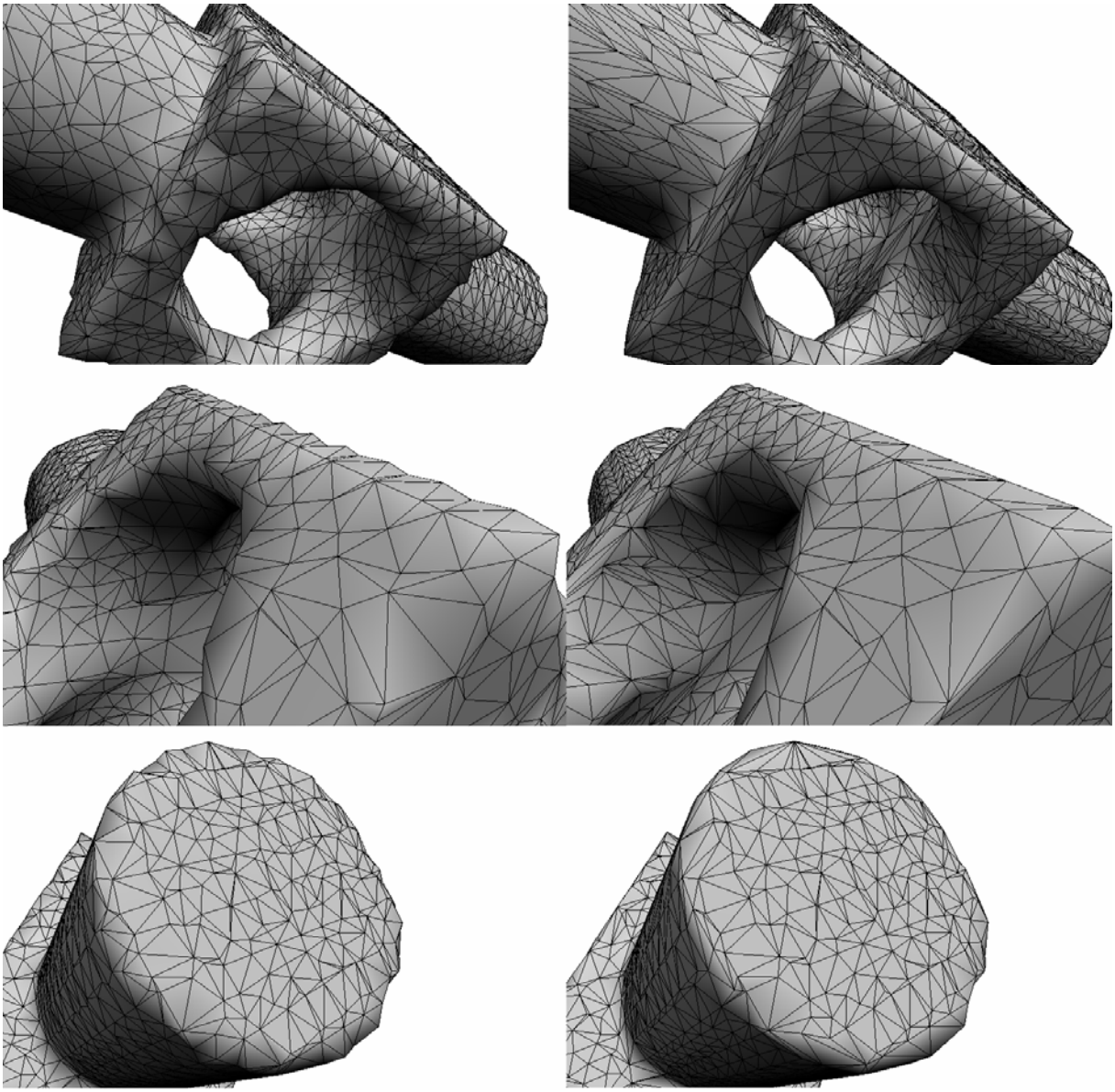
Procesor: Pentium M 1,73GHz (Dothan)

Paměť: 1GB DDR2 533MHz

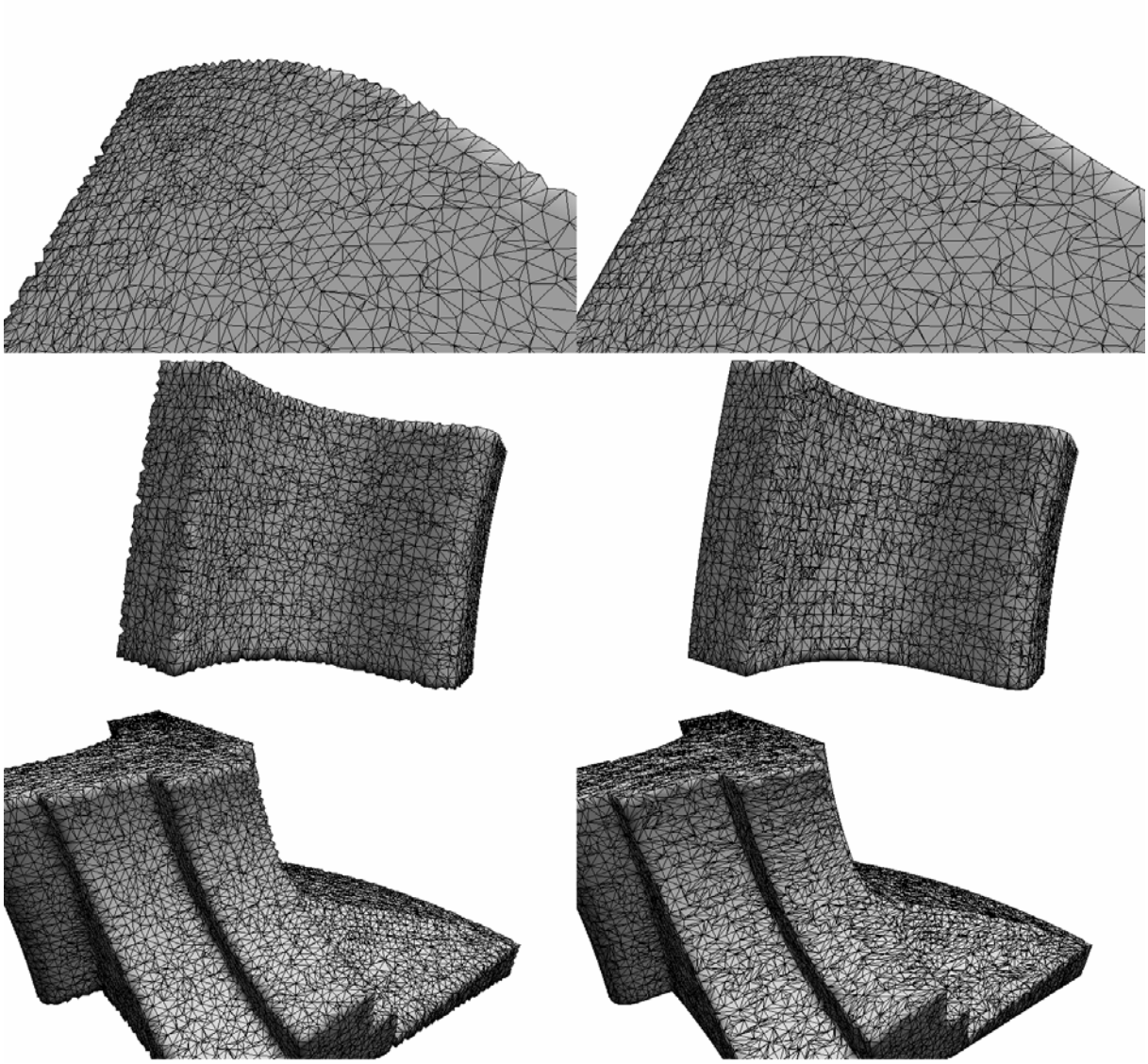
Grafická karta: Integrovaná Intel 915GM/GMS

Disk: 40GB 5400 RPM

9.5. Ukázky výstupu



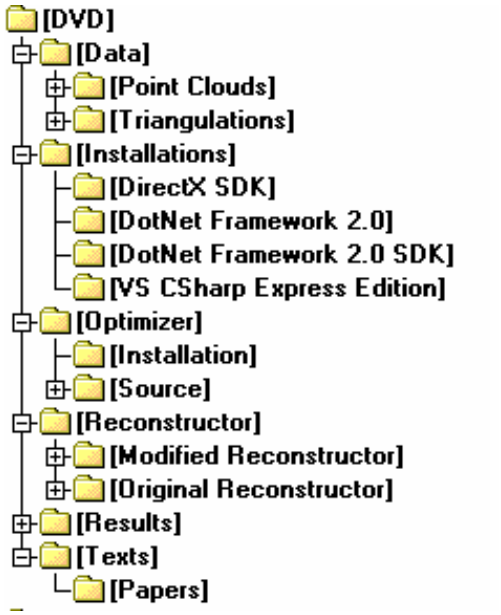
Obr. 9-8 Ukázky zvýraznění hran úhlovým kritériem pro datovou množinu mechpart.



Obr. 9-9 Ukázky zvýraznění hran úhlovým kritériem pro datovou množinu fandisk.

9.6. Obsah přiloženého DVD

Na přiloženém DVD je možné najít kompletní materiál k této práci. Adresářová struktura DVD je znázorněna na Obr. 9-10. Adresář Data obsahuje vstupní datové množiny pro program `Reconstructor`, které jsou obsaženy v adresáři `Point Clouds`. V adresáři `Triangulations` jsou hotové rekonstrukce, které představují vstup pro program Optimizer. Adresář `Installations` obsahuje kompletní softwarovou výbavu, která by mohla být potřeba v případě spouštění nebo úpravy programu Optimizer. Následují dva adresáře, které obsahují aplikace `Optimizer` a `Reconstructor`. Reconstructor existuje ve dvou verzích – základní verzi [2] a modifikované verzi, která ukládá do i trojúhelníkovou síť před extrakcí manifoldu. V adresáři `Results` se nalézají uložené trojúhelníkové sítě, které byly optimalizovány simulovaným žíháním a stochastickým gradientním algoritmem. V posledním adresáři se nalézají citované dokumenty a elektronická podoba této práce.



Obr. 9-10 Adresářová struktura přiloženého DVD.

Evidenční list

Souhlasím s tím, aby moje diplomová práce byla půjčována k prezenčnímu studiu v Univerzitní knihovně ZČU v Plzni.

Datum: Podpis:

Uživatel stvrzuje svým čitelným podpisem, že tuto diplomovou práci použil ke studijním účelům a prohlašuje, že ji uvede mezi použitými prameny.

Jméno	**Fakulta/katedra**	**Datum**	**Podpis**