Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

DIPLOMOVÁ PRÁCE

Rekonstrukce povrchu objektů z řezů

Radek Sviták

Vedoucí diplomové práce: Prof. Ing. Václav Skala, CSc.
Studijní obor: Informatika a výpočetní technika
Zaměření: Počítačová grafika
Rok vydání: 2001

Děkuji rodičům, bratrovi a sestře za podporu a pomoc ve všech směrech.
Za odbornou pomoc při realizaci této práce děkuji Prof. Ing. Václavu Skalovi, CSc.

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

V Plzni dne Radek Sviták

Obsah

Kapitola 1

Úvod

Od doby, kdy se začaly používat technologie pro získání informací o vnitřní struktuře objektů bez jejich destrukce, jako např. v klinické medicíně počítačová axiální tomografie (CT), ultrazvuk, nukleární magnetická rezonance (NMR), konfokální mikroskopie (CM) apod., produkující popis vnitřní struktury objektu ve formě sad rovnoběžných řezů, objevuje se požadavek na zpětnou rekonstrukci povrchu pro lepší pochopení jeho tvaru. Také vytahovací techniky v CAD konstruují výsledný povrch ze série řezů.

Existuje mnoho publikací, článků a dalších materiálů zabývajících se problematikou rekonstrukce povrchu z řezů. V teoretické části této práce se pokusíme vytvořit výčet problémů vyskytujících se v této oblasti a přehled nejpoužívanějších přístupů a technik k jejich řešení.

Dále se tato práce bude hlouběji zabývat problematikou umělého výpočtu řezů trojrozměrných těles a problémem rekonstrukce povrchu z ortogonálních sad řezů. K této úloze naopak nebyla nalezena žádná referenční literatura.

V praktické části se budeme snažit implementovat vybrané algoritmy a pokusíme navrhnout jejich vylepšení.

1.1 Abstract

This document presents the problems that concern computation of slices of three-dimensional objects and surface reconstruction from the parallel slices and from the orthogonal set of slices as well.

In the theoretical part, the most well-known and generally used algorithms solving these problems are introduced as well as their advantages and disadvantages.

The practical section then describes the implementation of chosen algorithms including the pros and cons of these solutions.

In conclusion, the reached results are introduced and discussed. And the future work is presented as well.

1.2 Rozvržení dokumentu

Tato diplomová práce je rozčleněna do sedmi kapitol. Po úvodní části se budeme zabývat výpočtem řezů objektů zadaných trojúhelníkovou sítí. Ve třetí kapitole se ponoříme do problematiky rekonstrukce povrchu z paralelních a z ortogonálních řezů. Čtvrtá kapitola se bude zbývat metodami, které byly vybrány k implementaci jednotlivých úloh. V páté kapitole uvedeme dosažené výsledky. Kapitola šest je závěr. Zde budou dosažené výsledky

zhodnoceny a také zde bude uveden výčet směrů budoucí práce v této oblasti. Sedmá kapitola je seznam použité literatury.

Kapitola 2

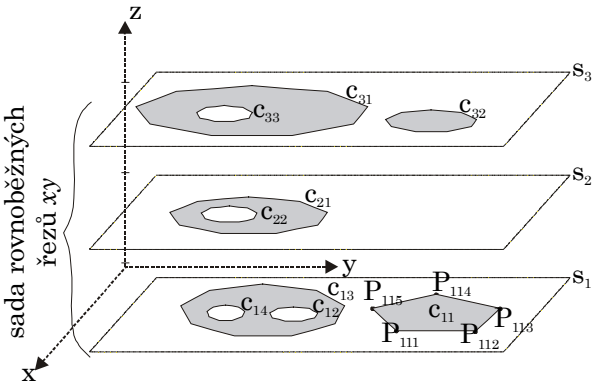
Výpočet řezů

Potřeba zabývat se umělým výpočtem řezů vznikla z důvodu omezené, v případě ortogonálních řezů vůbec žádné, dostupnosti vstupních dat pro řádné testování algoritmů rekonstrukce povrchu.

V této kapitole se budeme zabývat problematikou výpočtu řezů, resp. jednotlivých kontur, vstupních trojúhelníkových sítí. Nejprve bude uvedena použitá terminologie, dále popíšeme možné metody a jejich vlastnosti.

2.1 Sady řezů

Sada řezů je jedním ze způsobů hraniční reprezentace objektů (těles). Implicitně se pod pojmem sada řezů rozumí množina řezů S, obvykle rovnoběžných s dvojicí os kartézského souřadného systému (viz obr. 2.1), přičemž každý řez se skládá z množiny *kontur*, které odpovídají průniku hranice tělesa a roviny řezu.



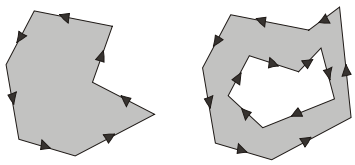
Obrázek 2.1: Sada řezů $S = \{s_1, s_2, s_3\}$ rovnoběžných s osami x a y

2.1.1 Kontury

Kontura je orientovaný jednoduchý polygon $c_i = \{p_1, p_2, ..., p_n\}$, kde i je index kontury v rámci řezu s_j a p_k, $k = 1..n$ jsou vrcholy polygonu, představující průnik tělesa a roviny řezu. Dále předpokládáme, že se kontury v rámci jednoho řezu neprotínají.

Ačkoliv to ve většině publikací není explicitně uvedeno, kontura se uvažuje jako *uzavřený* polygon, čili obecný mnohoúhelník. Toto omezení je logické, uvědomíme-li si, že v praxi kontura vzniká průnikem roviny a povrchu typu *manifold*.

Jsou definovány dva typy kontur: *Vnější kontura* a *díra*. Obvykle se předpokládá, že vnitřek tělesa leží nalevo od čáry, takže vnější obrysy jsou orientovány proti a vnitřní ve smyslu směru pohybu hodinových ručiček (viz obr. 2.2).



Obrázek 2.2: Obvyklá orientace kontur, vpravo je kontura obsahující díru

2.2 Generování řezů trojúhelníkových sítí

Protože v praxi se nejčastěji pro reprezentaci povrchů používá plošková, konkrétně trojúhelníková, reprezentace povrchů z důvodů uvedených např. v [1], budeme se v následujících odstavcích zabývat výpočtem řezů právě trojúhelníkových sítí.

2.2.1 Trojúhelníkové sítě

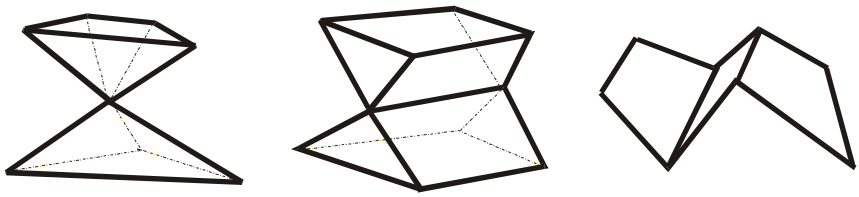
Garland [2] definuje model tělesa (resp. jeho povrch) jako množinu $M=(V, F)$, kde $V=(v_1, v_2 ..., v_n)$ je množina vrcholů a $F=(f_1, f_2, ..., f_m)$ množina trojúhelníků. Každý vrchol $v_i=[x,y,z]^T$ je dán třemi souřadnicemi v Eukleidovském prostoru $\mathbf{R}^3$. Každý trojúhelník $f_j=(k, l, m)$ je určen uspořádanou trojicí indexů k, l a m ($k\neq l$, $k\neq m$, $l\neq m$), přičemž platí, že $v_k\in V$, $v_l\in V$, $v_m\in V$.

Zde je důležité poznamenat, že v obecných trojúhelníkových sítích není známá sousednost jednotlivých trojúhelníků, tzn. nevíme, které trojúhelníky incidují s hranou aktuálního trojúhelníku. Úloha určení sousednosti je výpočetně náročný proces, proto v této práci znalost sousednosti nebudeme předpokládat.

2.2.2 Manifold a non–manifold povrchy

Jak bylo uvedeno v odstavci 2.1.1, při výpočtu řezů předpokládáme na vstupu trojúhelníkové sítě reprezentující povrchy typu manifold, v opačném případě by bylo nutné nejen definovat pojem *neuzavřené* kontury, ale také pracovat s konturami které se protínají.

Manifold, neboli „vyrobitelný“, se nazývá takové těleso v $\mathbf{E}^3$, jehož všechny vrcholy mají okolí topologicky ekvivalentní s otevřeným kruhem v $\mathbf{E}^2$, více v [3]. K takovému tělesu lze vytvořit rovinný model, což je graf $G=(V, H)$, jehož množina vrcholů V odpovídá množině vrcholů manifold tělesa a množina H odpovídá množině hran tělesa. To znamená, že takové těleso neobsahuje jevy jako sdílení vrcholu dvěmi částmi tělesa a hrany neincidující právě se dvěmi ploškami, viz obr. 2.3.



Obrázek 2.3: Ukázka non-manifold těles. Vlevo vrchol sdílený dvěmi částmi tělesa, uprostřed hrana incidující se čtyřmi ploškami, vpravo hrany incidijící jen s jednou ploškou

2.3 Problematika výpočtu řezů

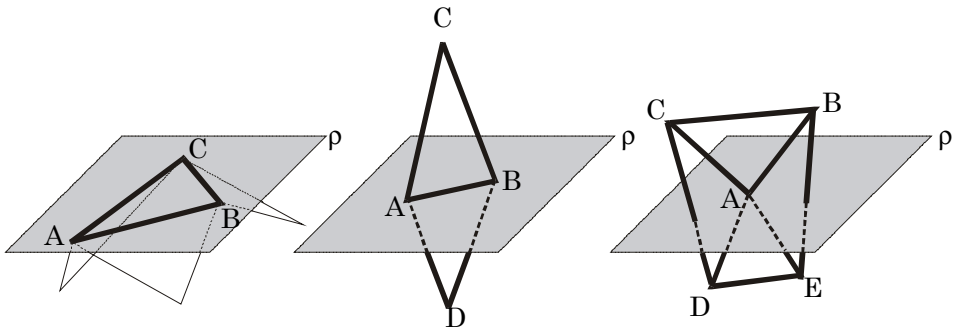
V následujících odstavcích budou popsány problémy při sestavování jednotlivých kontur řezů v útvary popsané v odstavci 2.1.1. Výpočet řezu trojúhelníkovou sítí bez znalosti sousednosti by se dal shrnout do třech kroků: Výpočet průniku trojúhelníkové sítě a roviny řezu (průsečnice), sestavení kontur (pospojování průsečnic v jednoduché uzavřené polygony), nastavení správné orientace kontur (viz odstavec 2.1.1, obr. 2.2).

2.3.1 Průnik trojúhelníku a roviny řezu

Průnik roviny řezu ρ a trojúhelníku *ABC* je buď prázdný (v případě, že všechny vrcholy A, B i C leží nad nebo pod rovinou řezu ρ) nebo je jím *průsečnice p*, úsečka která je základním stavebním prvkem kontur.

Ačkoliv je výpočet průsečnice *p* jednoduchou úlohou z analytické geometrie (stačí vypočítat její krajní body), je nutné si uvědomit, že v praxi mohou nastat 3 singulární případy, které není možné zanedbat:

- Trojúhelník ABC ležící v rovině řezu (obr. 2.4a) , tzn. každý vrchol A, B, C leží v rovině řezu. Takové případy jsou velmi časté u pravidelných geometrických těles. Z předpokladu manifold tělesa na vstupu, lze takovýto trojúhelník dále nezpracovávat, neboť o jeho příspěvek při stavbě kontur řezu ρ se postarají sousední trojúhelníky, incidující přes hrany AB, BC a AC.
- Právě jedna strana trojúhelníku ABC leží v rovině řezu (obr. 2.4b). Tento případ nastává zároveň s předchozím případem (díky předpokladu manifold tělesa na vstupu) ale také v případě, kdy dva trojúhelníky, neležící v rovině řezu, spolu sdílejí hranu, která leží v rovině řezu. Takovou hranu je nutné při stavbě kontur uvažovat pouze jednou.
- Právě jeden vrchol trojúhelníku ABC ležící v rovině řezu. Takový trojúhelník nepřispívá ke stavbě kontur tohoto řezu (přispívají trojúhelníky, které s tímto vrcholem incidují)



Obrázek 2.4 : Singulární případy pozice trojúhelníku vzhledem k rovině řezu ρ.
a) trojúhelník ABC ležící v rovině řezu, b) trojúhelníky ABC, ADB sdílející stranu AB, která leží v rovině řezu, c) troj. ABC, ADE sdílející vrchol A, který leží v rovině řezu.

2.3.2 Stavba kontur

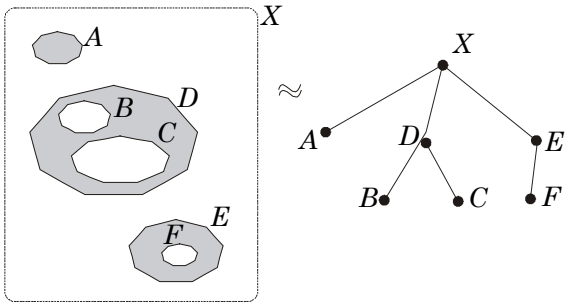
Vstupem této úlohy je množina průsečnic (úseček) $S = \{s_1, s_2, \ldots s_n\}$, která vznikne jako průnik roviny řezu ρ a trojúhelníkové sítě T (platí $S = \rho \cap T$). V tomto kroku se z vypočítaných průsečnic sestavují kontury. V podstatě vytváříme jednoduchý polygon tak, že k němu přidáváme úsečky z množiny S (které zároveň z množiny S odebíráme).

Úsečku $s_i \in S$ přidáme ke stávajícímu polygonu p právě tehdy, když platí právě jedna rovnost z následujících rovností: $A_p = A_i$, $B_p = A_i$, $A_p = B_i$, $B_p = B_i$, kde A_p, B_p jsou krajní stávající krajní body polygonu p a A_i, B_i krajní body úsečky s_i.

2.3.3 Orientace kontur

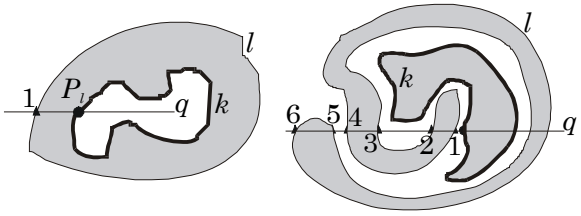
Nastavení orientace kontur podle odstavce 2.1.1, obr. 2.2 vyžaduje znalost hierarchie kontur $c_i \in C_\rho$ v rámci řezu ρ, neboť po sestavení kontur nevíme, zda-li kontura c_i představuje vnější konturu či díru.

Určení hierarchie kontur v rámci jednoho řezu vyžaduje vytvořit tzv. *strom vnoření,* obr 2.5, popsaný detailně v [4]. K vytvoření stromu je třeba určit, která kontura je vnější a která díra. Využijeme přitom faktu, že kontura je dírou, leží-li uvnitř jiné kontury.



Obrázek 2.5: Řez s konturami a odpovídající strom vnoření (vpravo). X je pomyslná kontura, která slouží jako kořen stromu

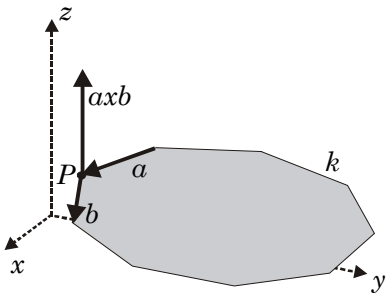
K určení, zda kontura k leží uvnitř kontury l se používá známá technika *Ray-Crossing*. V prvním kroku zjistíme nejlevější průsečík přímky q s konturou k. Označme jej P_l. V druhém kroku zjistíme počet n_l průsečíků přímky q s konturou l nalevo od P_l. Platí, že k je uvnitř l právě tehdy, když n_l je liché číslo, viz obr. 2.6.



a) b)

Obrázek 2.6: a) počet průsečíků přímky q s konturou l nalevo od P_l je 1 (lichý), tedy k je díra, vpravo je počet sudý, tedy k je vnější kontura.

Pokud již víme, je-li kontura k díra či vnější kontura, je ještě nutné polygon kontury odpovídajícím způsobem zorientovat. Test na orientaci polygonu provedeme například tak, že v jednom z extrémních bodů (např. bod s jednou nejmenší souřadnicí) vytvoříme vektory hran polygonu, které vektorově vynásobíme. Směr výsledného vektoru udává orientaci kontury, viz obr. 2.7.



Obrázek 2.7: Určení orientace kontury k v extrémním bodě P pomocí orientace vektoru $\boldsymbol{a}$ x $\boldsymbol{b}$

Kapitola 3

Rekonstrukce 3D povrchu z řezů

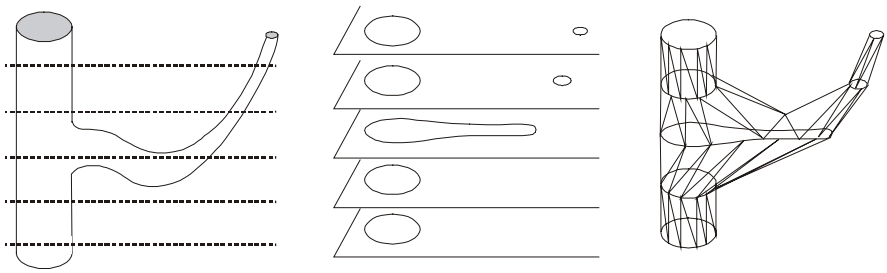
V této kapitole bude uveden výčet problémů a publikovaných metod pro rekonstrukci povrchu těles ze vstupních sad paralelních a ortogonálních řezů. K vytvoření přehledu problematiky rovnoběžných řezů byly použity práce [1, 10, 11]. Pro problematiku rekonstrukce povrchu z ortogonálních řezů se nepodařilo najít žádné reference.

3.1 Rekonstrukce z paralelních řezů

Oplášťovací algoritmy jsou publikovány zhruba od poloviny 70.let [5], jako výchozí prameny k této problematice lze doporučit publikace [6, 4, 7, 8, 9], pro získání přehledu lze použít [10].

Konstrukce povrchu z množiny řezů není v obecném případě triviální a deterministická úloha, neboť při řezání objektů dochází ke ztrátě informací o obejmu mezi řezy, které při zpětné rekonstrukci chybí. A tak je výsledkem rekonstrukce pouze hrubá aproximace povrchu.

Obrázek 3.1 představuje typickou ukázku problematiky rekonstrukce povrchu objektu z paralelních řezů. Demonstruje, že vždy jde pouze o odhad výsledného povrchu. V místech kde osa objektu svírá s rovinou řezu pravý úhel je rekonstrukce nejvěrnější. S rostoucí odchylkou od kolmého směru klesá relativní hustota vzorků (přesnost vzorkování) a zvyšuje se zkreslení tvaru, stejně jako se stává obtížnější i rozpoznání vzájemné korespondence kontur, které je nutnou podmínkou správného opláštění.



a) b) c)

Obrázek 3.1: a) vstupní objekt, b) sada jeho řezů, c) odhad povrchu rekonstrukcí z paralelních řezů

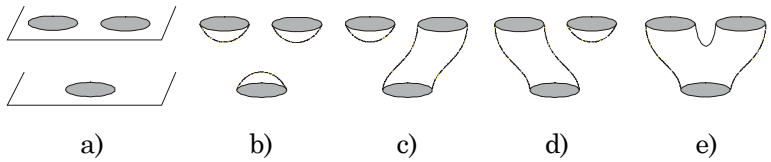
V [11] jsou algoritmy pro rekonstrukci povrchu z řezů rozděleny podle použitého principu na *objemové* a *povrchové*.

- Objemové metody – vyžadují vzdálenost mezi řezy shodnou nebo alespoň porovnatelnou s hustotou vzorků v řezu. Sadu řezů „vloží do“ prostorové mřížky a generují povrch standardními algoritmy pro hledání isoploch (Marching cubes, apod.). Čím větší je však vzdálenost mezi sousedními řezy, tím více tento přístup selhává, neboť je založen na překrývání promítnutých kontur.
- Povrchové metody – pracují přímo s řezy a interpolují postupně dvojice sousedních korespondujících kontur pásy trojúhelníků.

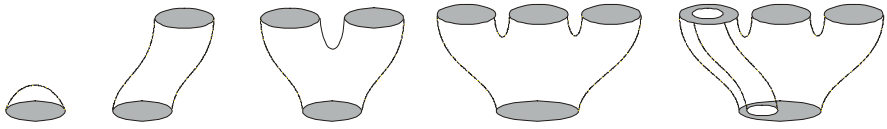
V této práci se zaměříme na skupinu metod s povrchovým přístupem. V dalších odstavcích budou popsány základní problémy a principy.

3.1.1 Problém korespondence

Určení korespondence (vzájemné přiřazení) kontur je klíčovým problémem, na kterém závisí výsledná rekonstrukce povrchu. Zjišťování, které kontury spolu topologicky sousedí, tj. patří stejnému objektu, je neúplně definovaný úkol, který lze automaticky řešit jen v případech, kdy známe další informaci o struktuře vzorkovaného objektu (viz obr. 3.2). Jiný postup je třeba volit např. při rekonstrukci několika kulových útvarů a jiný při rekonstrukci objektu se stromové strukturou.



Obrázek 3.2: Klasická ukázka pouhého odhadu skutečného povrchu, pokud nemáme další informaci o tvaru tělesa mezi řezy.



Obrázek 3.3: Konfigurace kontur ve dvojici sousedních řezů vede k různým způsobům větvení. a) víko, b) jednoduché opláštění 1:1, c) větvení 1:2, d) vícenásobné větvení 1:n, e) obecné vícenásobné větvení s dírami m:n.

Proto se tento problém řeší heuristikami nebo přímo interakcí s uživatelem. Výsledkem této fáze je obvykle lokalizace míst, kde dochází k větvení spolu s informací o typu větvení, viz obr. 3.3. Následuje výčet a popis vlastností publikovaných heuristik a technik používaných k automatickému určování korespondence kontur.

Plocha překrytí kontur

Nejjednodušší a současně nejpoužívanější heuristikou je dostatečná plocha překrytí kontur v sousedních řezech. Překrytí je definováno jako velikost oblasti průniku kolmé projekce kontur. Poměr velikosti plochy překrytí k ploše větší z kontur, případně ploše vzniklé jejím sjednocením, porovnáváme se zadanou konstantou. Tento postup je použitelný pouze pro jednoduché tvary objektů při dostatečně malé vzdálenosti řezů. Pokud zpracovávaná data

nejsou dostatečně hustá pro překrytí obrysů stejného objektu v sousedních řezech, pak tyto metody vytvářejí více objektů.

Velikost překrytí lze hrubě odhadnout plochou průniku obdélníkových obálek (bounding boxes), nicméně s rostoucí členitostí a protáhlostí je odhad stále nepřesnější. Přesnější je rasterizovat kontury do dvou bitových map a po vyplnění zjistit velikost průniku logickou operací AND jako počet společných pixelů.

Zobecněné válce

Nejsou-li data vhodná pro použití překrývacího kritéria, tj. má–li těleso např. stromovou strukturu kruhového či oválného průřezu, lze využít metodu zobecněných válců, která používá globálnější pohled na datovou množinu, a kterou navrhl Soroka [12] v roce 1981.

Tyto válce tvoří řezy ve tvaru elips, jejichž středy leží přibližne na přímkové ose (odchylují se o méně než zadanou chybu). K popisu jednotlivých elips stačí úhel natočení hlavní poloosy, souřadnice středu a délky poloos A, B.

Metoda, kterou uvedl Mayers [4] v roce 92, pracuje ve třech krocích. V prvním se z elips sestaví válcové úseky, které se ve druhé etapě pospojují do objektů. Ve třetí fázi jsou nalezena větvení. Snahou metody je vytvořit minimální počet co nejdelších válcových úseků.

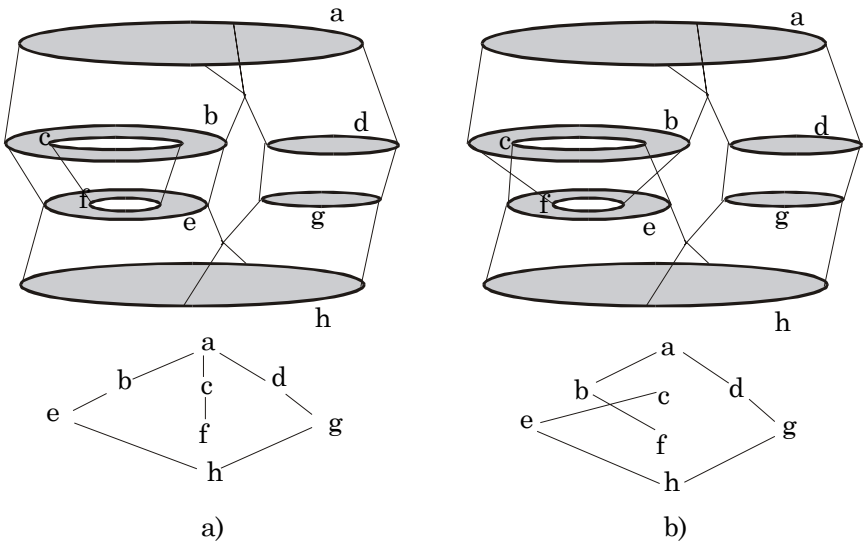
Uvedený postup je vhodný k rekonstrukci vláknitých či větvících se struktur bez cyklů, tj. takových, u nichž po rozvětvení nedochází k opětovnému spojení větví. Slabým místem metody je první krok, v němž může dojít k propojení vzdálených a vzájemně posunutých elips a k následnému šíření chyby.

Strom minimálního pokrytí v grafu kontur

Potíže s propagací chyby vzniklé v prvním kroku metody zobecněných válců vedly k návrhu metody pracující globálněji [4]. V první fázi se sestaví graf možných spojnic elips (graf přiřazení, viz obr. 3.4) tak, aby každé elipse odpovídal jeden uzel a každé možné spojnici jedna hrana. Za možné spojnice se považují všechny dvojice elips v sousedních řezech.

Hrany grafu jsou ohodnoceny cenou $c(i, j) = (x_i - x_j)^2 + (y_i - y_j)^2 + (A_i - A_j)^2 + (B_i - B_j)^2$, kde i a j jsou propojené elipsy, x a y souřadnice jejich středů a A a B délky jejich hlavní a vedlejší poloosy.

Poté se v grafu najde strom minimálního pokrytí, tj. strom, který obsahuje všechny uzly a jen takové hrany, aby součet jejich cen byl minimální (minimal spanning tree). Tento strom se rozdělí na nevětvící se úseky a poté se naleznou místa větvení.



Obrázek 3.4: Ukázka platného a neplatného grafu přiřazení. a) platný graf – existuje těleso, které lze popsat tímto grafem, b) neplatný graf přiřazení – vede ke spojení vnější kontury *b* s dírou *f* a díry *c* s vnější konturou *e*.

Metoda dává pro přirozeně větvené (acyklické) objekty lepší výsledky než předchozí, protože při globálním výpočtu stromu minimálního pokrytí nedochází k šíření lokálních chyb. Je ale stejně jako předchozí metoda zobecněných válců nevhodná k hledání objektů s cykly a může vytvářet chybné hrany, když propojí oddělené objekty, jejichž kontury jsou blízko.

3.1.2 Problém opláštění a větvení

V této fázi již známe vzájemné přiřazení kontur a úloha nyní zní nalézt jejich mezirezové propojení, vytvořit povrchovou (trojúhelníkovou) síť z jejich bodů. Tento problém lze také formulovat jako interpolování množiny rovinných křivek plochou.

Opět je třeba si uvědomit, že v obecném případě jde pouze o odhad skutečného tvaru původního tělesa, neboť nemáme informace o tvaru objektu mezi řezy. Základním požadavkem je vygenerování nedegenerované sítě, tj. sítě bez nedoléhajících a bez vzájemně se protínajících trojúhelníků.

Interpolační algoritmy se liší zejména tím, jak složité tvary a kombinace kontur umějí zpracovat. Protože je úloha větvení (branching problem) symetrická k úloze spojování (stačí zaměnit pořadí řezů), uvažujeme jen případy větvení, viz obr. 3.3.

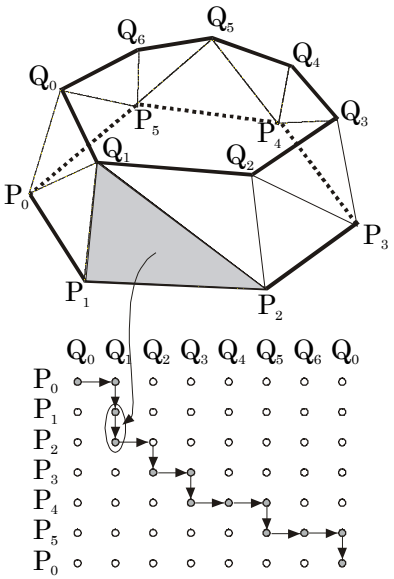
Základní algoritmy umějí propojit (opláštit) dvě konvexní, případně i konkávní kontury (spojení 1:1) a plošně triangulovat víko (triangulace víka je v obecném případě netriviální úloha triangulace nekonvexního polygonu s dírami). Složitější algoritmy zvládají větvení jednoduché (1 : 2, tzv. bifurkace), vícenásobné (1 : n), či obecné (m : n). Samostatným problémem je existence děr.

Toroidní graf

Opláštění lze stejně jako určení korespondence kontur uvedené v odstavci 3.1.1 chápat jako úlohu řešitelnou pomocí aparátu teorie grafů. Takto problém opláštění dvou kontur formuloval Keppel [5] v roce 1975. Problém konstrukce povrchu mezi dvěma konturami byl převeden na problém nalezení cyklu s minimální cenou v tzv. toroidním grafu (obr. 3.5).

Množina uzlů toroidního grafu odpovídá všem možným úsečkám propojující vrcholy kontur v sousedních řezech (tj. každý vrchol s každým). Množina hran představuje trojúhelníky, které vzniknou ze dvojice úseček se společným vrcholem. Aby ve výsledné síti nevznikaly vzájemně se protínající trojúhelníky, jsou v grafu povoleny pouze hrany vodorovné a svislé. Prohledávání grafu lze zjednodušit, pokud nalezneme počáteční „nejlepší" spojnici (např. úsečka P_1Q_1 na obr. 3.5). Cesta v toroidním grafu pak v této spojnici začíná a končí.

Grafové algoritmy sice dávají optimální výsledek, ale u kontur více vrcholy jsou příliš výpočetně náročné ($O(n^2)$), proto se nejčastěji používají heuristické metody, které nezkoumají celý obvod kontury naráz a nesnaží se propojit dvojici kontur optimálně, ale jen tak, aby nově přidávaný trojúhelník vyhověl použitému cenovému kritériu. V následujícím odstavci bude uveden výčet a stručný popis používaných kritérií.



Obrázek 3.5: Převod problému plátování dvou obrysů na prohledávací problém. Hrany spojující sousední body jedné kontury se nazývají úseky obrysu. Hrany spojující vrchol jedné kontury s vrcholem druhé kontury jsou nazývány překlenutí. Uzly toroidního grafu představují překlenutí, hrany odpovídají vytvořeným trojúhelníkovým plátům, tzn. pláty jsou reprezentovány orientovanými hranami mezi uzly grafu. Plátování je vytvořeno nalezením nejlevnějšího cyklu v grafu.

3.1.3 Kritéria pro stanovení ceny trojúhelníku

Cenu lze stanovit mnoha způsoby. Nejpoužívanější jsou lokální metriky, které se vyhodnocují pouze na základě informací o právě vytvářeném trojúhelníku. Kritéria, která budou dále uvedena, lze detailně studovat v [4].

Maximální objem

Vyhodnocuje se jako příspěvek záplaty (resp. objemu klínu, který tvoří trojúhelníková záplata a spojnice těžišť kontur) k oplášťovanému objemu, který je určen dvojicí rovnoběžných řezných rovin a trojúhelníkovými záplatami. U konkávních částí kontur se naopak snažíme tento objem minimalizovat. Algoritmus, který by využíval tohoto kritéria, musí rozlišovat konvexní a konkávní části kontur.

Minimální povrch

Toto kritérium je snadno vyjádřitelné, neboť jde o plochu vytvářeného trojúhelníku, ale nehodí se pro tzv. patologické případy, kdy jsou kontury vůči sobě značně posunuty a docházelo by k chybnému propojení napříč tělesem (obr. 3.6). Řešením je kontury normalizovat, tzn. například tak, aby se překrývala jejich těžiště a byly zhruba stejně velké.

Směr přiřazení

Toto kritérium zvýhodňuje ty spojnice kontur, jejichž směr se příliš neliší od směru spojnice těžišť. Je vhodné pro konkávní kontury, ale jsou zde problémy se silně konkávními nebo zavinutými tvary kontur.

Minimální délka spoje (překlenutí)

Délka spoje je kritérium, které je nejvíce lokální, neboť hodnotí pouze délku přidávané hrany spojující kontury v sousední dvojici řezů. Před jeho aplikací je nutná normalizace kontur (posun a zvětšení) jako v případě minimálního povrchu.

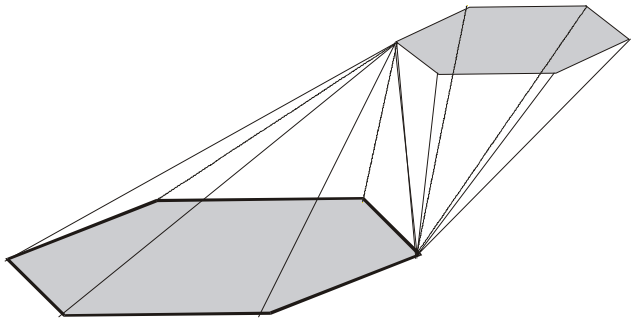
Proces normalizace před výpočtem plátování mapuje obrysy z jejich ohraničujících obdélníků na jednotkový čtverec a převádí tyto čtverce pro každý obrys tak, aby byly vystředěny do počátku roviny. Vypočtené plátování je poté použito na původní obrysy.

Spojování poměrných úseků

V tomto případě je poloha vrcholů spojovaných kontur brána relativně vůči jejich počátkům a jsou preferovány „vertikální“ spojnice, tj. spojnice bodů relativně stejně vzdálených od počátku.

Globální metriky

Jednotlivé trojúhelníky jsou na počátku ohodnoceny mírou „zásluh“ (zásluha = 1 / délka spoje). Následuje relaxační fáze, při níž se zásluhy sousedících trojúhelníků vzájemně dolaďují. Trojúhelník, jehož zásluhy přitom klesnou k nule, je odstraněn.



Obrázek 3.6: Příklad chybného opláštění při použití lokálních metrik bez předchozí normalizace kontur

Pro všechny uvedené metriky lze najít případy, kdy selhávají. Například kritérium minimálního povrchu a kritérium minimální délky spoje může vést ke konstrukci dvojice jehlanů dotýkajících se jedinou hranou, přestože bychom očekávali vznik zešikmeného hranolu (obr. 3.6), v jiných případech může dojít k k vytvoření zkrouceného pláště. Proto se

těmto případným jevům předchází normalizacemi (posunutí kontur aby měla těžiště nad sebou, změna měřítka, případné pootočení)

3.1.4 Jednotlivé případy při oplášťování, problém větvení

V této časti uvedeme základní metody řešící jednotlivé případy, které mohou nastat při generování povrchu mezi dvěma řezy.

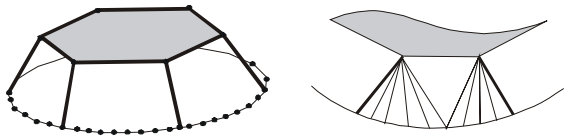
Víko a dno

Jako *víko* označujeme situaci, kdy kontura nemá pokračování v následujícím řezu. Obvykle se taková kontura trianguluje některým z algoritmů pro triangulaci obecných mnohoúhelníků (v obecném případě je nutné počítat i s existencí díry). Analogický problémem je *dno*.

Dvojice kontur (1 : 1)

V tomto případě se v prvním kroku algoritmu nalezne „nejlepší“ spojnice dvou vrcholů (každý z jedné kontury). V dalších krocích se postupně přidávají další spojnice, přičemž jsou vybírány s využitím kritérií uvedených v odstavci 3.1.3.

Lze použít i další vylepšení např. v případě, že kontury mají výrazně odlišný počet vrcholů. V první fázi spojíme všechny vrcholy kontury s menším počtem vrcholů s jim nejbližšími vrcholy na kontuře s větším počtem vrcholů (obr. 3.7a). Ve druhé fázi již pracujeme pouze v takto ohraničených částech (obr 3.7b).

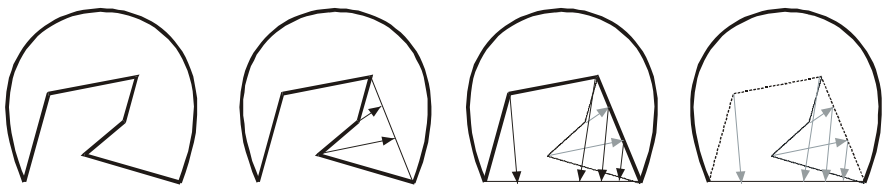


a) b)

Obrázek 3.7: a) spojení vrcholů kontury s menším počtem vrcholů s, podle daného kritéria, nejlepšími vrcholy kontury s větším počtem vrcholů

Dále lze vylepšit práci s nekonvexními konturami převodem na konvexní. Algoritmus nejprve zkonstruuje konvexní obálku kontury a rekurzivně zpracovává nekonvexity, tj. ty části hranice, které na konvexní obálce neleží.

U varianty pracující s toroidním grafem lze při zpracovávání nekonvexních úseků výhodně používat kritérium minimalizace objemu.



a) b) c) d)

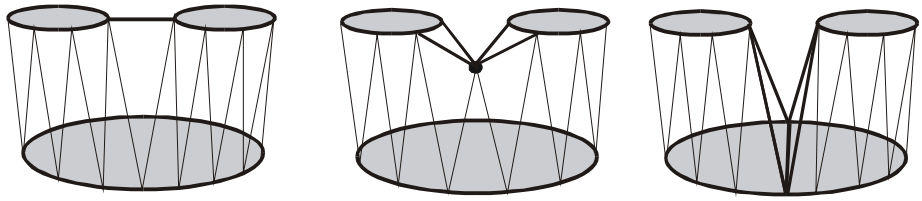
Obrázek 3.8: Přemísťování vrcholů v konkávní části kontury na konvexní obálku při opláštění nekonvexních kontur. a) konkávní kontura, b,c) rekurzívní přemístění nekonvexit na obálku, d) kontura upravená k triangulaci.

Dvoufázový algoritmus rekurzívně přemísťuje body z nekonvexit na konvexní obálku a s touto novou konturou provede generování povrchu metodou pro konvexní kontury. Tím se určí topologie vrcholů. Vrcholy i se spojnicemi pak vrátí na původní místa, čímž je zachována geometrie. V každé úrovni rekurze jsou všechny nekonvexity vyjmuty, je u nich vypočítána konvexní obálka a tím nalezeny nekonvexity. Vrcholy se přemístí na odpovídající úsečku KO aktuální úrovně tak, aby byla zachována jejich relativní poloha na obvodu nekonvexity. Celou situaci lze sledovat na obrázku 3.8.

Oba uvedené algoritmy spojují nekonvexní kontury, aniž by na některé z nich musely přidat nějaký nový bod. Je však dokázáno (Gitlin a kol.), že opláštění trojúhelníkovou sítí obsahující pouze původní vrcholy obecně nemusí existovat, protože se trojúhelníky sítě mohou vzájemně protínat a vzniká degenerovaný povrch.

Jednoduché větvení (1 : 2) – bifurkace

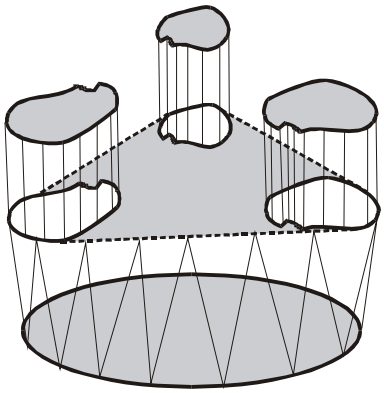
Tento problém lze řešit mnoha způsoby. K nejjednodušším patří prosté spojení dvojice kontur v nejbližších vrcholech (viz obr. 3.9a). Po přidání spojnice v obou směrech, vznikne z dvojice kontur jedna nekonvexní kontura, k opláštění použijeme lib. algoritmus z předchozího odstavce.



a) b) c)

Obrázek 3.9.: Tři základní způsoby opláštění bifurkací: a) spojení kontur v nejbližších vrcholech, b) přidání mezilehlého bodu, c) rozdělení spodní kontury na dvě části

Přesnější propojení (z pohledu uživatele) vznikne, snažíme-li se zachovat úžlabí mezi konturami a do prostoru mezi rovinami řezů vložíme pomocný bod (obr. 3.9b). Další možností je rozdělení spodní kontury na dvě části osou spojnice (obr. 3.9c).



Obrázek 3.10: Opláštění vícenásobného větvení 1:3 nejjednodušším způsobem, který produkuje nepřirozeně velkou rovnou plochu

Obecné větvení (1 : n)

Lze jej vyřešit převedením na jednodušší případ spojení dvojice kontur (1 : 1). Příklad větvení 1 : 3 je na obrázku 3.10. Nejjednodušší je rozdělit větvení na tři nezávislé části. Na spojení 1 : 1 ve směru větvení, na jednoduché spojení konvexní obálky se spodní konturou a na plošnou triangulaci nově přidané mezičezové kontury. Plošnou triangulací však vzniká nepřirozeně velká rovná plocha. V literatuře lze nalézt způsoby jak tento nedostatek potlačit.

Obecné větvení (m : n)

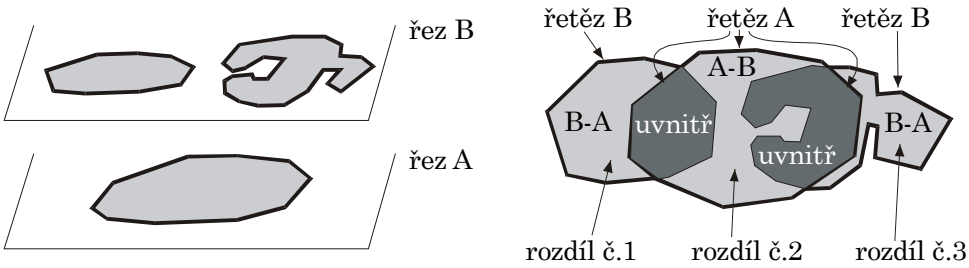
Lze jej převést přidáním jedné či více kontur v pomocných mezirovinách na případ 1 : n. Postup vytvoření mezilehlé kontury lze nalézt např. v [6].

3.1.5 Olivův algoritmus

Univerzální oplášťovací algoritmus navrhl Oliva [8], který převedl prostorový problém hledání spojnic kontur v sousedních řezech do rovinné úlohy. Jeho algoritmus konstruuje nedegenerovaný povrch adaptivním přidáváním mezilehlých řezů (kontur).

Korespondence kontur

Algoritmus určí korespondenci kontur (topologii) na základě existence jejich překrytí v kolmém průmětu.



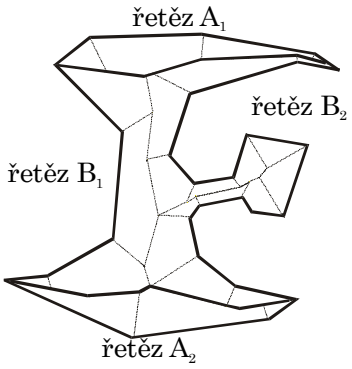
Obrázek č. 3.11: Vstupní sada kontur (vlevo), jejich kolmý průmět a výpočet oblastí překrytí a rozdílů (vpravo)

Opláštění kontur

Vychází z předpokladu, že překrytí kontur je ve skutečnosti v místě vnitřku tělesa a nebude tedy triangulováno.

Po již zmíněném kolmém průmětu se vypočítají vzájemné rozdíly kontur (viz obr. 5.28). Část hranice, která patřila konturám v jednom řezu, se označí jako řetěz A, část patřící konturám v druhém řezu jako řetěz B.

Pokud se nejedná o víko nebo dno a pokud nejsou kontury tvarem i polohou totožné, je rozdíl tvořen minimálně jedním řetězem A a jedním řetězem B. Další postup se liší podle počtu řetězů (totožné kontury se jednoduše triangulují jako 1 : 1).



Obrázek 3.12: Kontura s vytvořenou ABN (*Angular Bisector Network*)

Metoda využívá síť ABN (Angular Bisector Network) k rozdělení složitých mnohoúhelníků na jednodušší. ABN je síť úseček, ležících na osách úhlů sousedních či protilehlých hran kontur. Skutečnost, že úsečky leží na osách úhlů, vychází z předpokladu, že mezilehlé kontury vytváříme vždy uprostřed mezi rovinami originálních řezů. Při kolmém průmětu se takto vytvořená pomocná kontura promítne právě do os.

Po rozdělení původní kontury na části (v místě mezilehlého řezu, tj. části ABN společné dvojici řetězů A a B) se každá takto vzniklá zpracovává samostatně, viz obr. 3.12. Podle kritéria minimální délky spoje se postupně triangulují jednotlivé buňky sítě (oblasti, vymezené hranami ABN a hranami řetězu (A nebo B)).

Velkou výhodou algoritmu je, že vytváří nedegenerovaný povrch, tj. zaručuje, že se vygenerované trojúhelníky neprotínají a že v opláštění nevzniknou nespojitosti způsobené nedolehnutím trojúhelníků. Dokáže zpracovat i tělesa obsahující cykly, např. toroid.

Nevýhodou je, že nerozpoznává korespondenci kontur, pokud se nepřekrývají, a vytváří chybná propojení, když se překryjí kontury spolu nesouvisející.

3.2 Rekonstrukce z ortogonálních řezů

V odstavci 3.1 bylo uvedeno, že přístupy k rekonstrukci povrchu z paralelních řezů se dají rozdělit do dvou skupin, na *povrchové* a *objemové*. Základním nedostatkem výsledných rekonstrukcí získaných povrchovými metodami je velká nepřesnost v místech, kde je osa objektu více odchýlena od směru kolmého k rovině řezu a také vysoký stupeň odhadu tvaru v procesu plátování (odstavec 3.1.4).

Nedostatkem objemových metod je požadavek velké hustoty řezů. Ten je v praxi mnohdy nesplnitelný, neboť není např. možné vystavovat pacienta záření po dobu, než se získá dostatečný počet řezů.

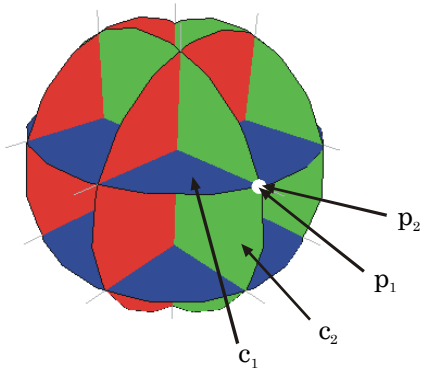
Rekonstrukce povrchu z ortogonálních řezů je určitým kompromisem mezi výše uvedenými přístupy. Není kladen požadavek na vysokou hustotu řezů a přitom jsou eliminovány případy, kdy je nutné odhadovat tvar objektu, odchýlí-li se jeho osa od směru kolmého k rovině řezu, neboť v tomto případě začíná být kolmá k rovinám ortogonální sady řezů.

3.2.1 Definice pojmů

V tomto odstavci uvedeme definice termínů se kterými budeme v dalším textu pracovat. Význam jednotlivých pojmů lze zároveň sledovat na obrázcích 3.13 - 15.

Systém ortogonálních řezů

Systémem ortogonálních řezů nazveme množinu $\boldsymbol{S} = \{S_{xy}, S_{yz}, S_{zx}\}$, kde *Sxy* je sada řezů objektu s rovinami řezů rovnoběžnými s osami x a y kartézského souřadného systému (v dalším textu je budeme zkráceně nazývat *xy*-řezy), S_{yz} je sada *yz*-řezů a S_{zx} je sada *zx*-řezů, přičemž alespoň dvě sady jsou neprázdné, tj. obsahují nenulový počet řezů.



Obrázek 3.13: Systém ortogonálních řezů, $p_1 \in c_1$ je uzlový bod kontury c_1, $p_2 \in c_2$ je uzlový bod kontury c_2, platí relace $\psi_c(c1, c2)$, $\omega(p_1, p_2)$

Relace ortogonální řezy

Relaci mezi řezy s_a a s_b $\psi_s(s_a, s_b)$, $s \in S_a$,, $s \in S_b$ nazveme *relace ortogonálních řezů*, jestliže platí že S_a a S_b jsou sady ortogonálních řezů jejichž roviny spolu svírají pravý úhel.

Relace ortogonální kontury

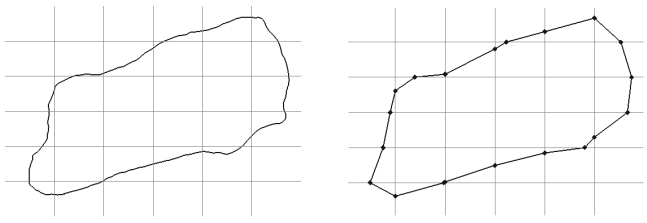
Relaci mezi konturami c_a a c_b $\psi_c(c_a, c_b)$, $c_a \in s_a \in S_a$, $c_b \in s_b \in S_b$ nazveme *relace ortogonálních kontur*, platí-li relace $\psi_s(s_a, s_b)$.

Uzlový bod

Průsečík dvou ortogonálních kontur nazveme *uzlový bod*. Platí věta, že pro každý uzlový bod p, existuje v systému ortogonálních řezů takový bod q, že platí $\omega(p, q)$.

Relace kontaktu

Relaci mezi dvěma uzlovými body kontur p_a a p_b $\omega(p_a, p_b)$, $p_a \in c_a$, $p_b \in c_b$ nazveme *relace kontaktu* (ortogonálního), platí-li $p_a \equiv p_b$ a relace $\psi(c_a, c_b)$.



a) b)

Obrázek 3.14: Zjednodušená kontura (vpravo)

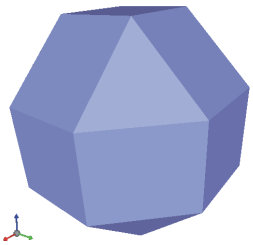
Zjednodušená kontura

Konturu tvořenou pouze uzlovými body budeme nazývat *zjednodušená kontura*.

Jednoduchá, úplná rekonstrukce

Při *jednoduchá rekonstrukci* povrchu pracujeme pouze se zjednodušenými konturami. Předpokládáme přitom, že nahrazením původního tvaru kontury úsečkou mezi jednotlivými uzlovými body nebude výsledný povrch příliš degradovaný (ukázka jednoduché rekonstrukce, viz obr. 3.15).

Naopak při *úplné rekonstrukci* povrchu zohledňujeme tvaru kontur mezi jednotlivými uzlovými body. Tento postup je vhodnější, v případě nízkého počtu řezů v jednotlivých sadách (např. v případě obr. 3.15).



Obrázek 3.15: Ukázka jednoduché rekonstrukce povrchu ze systému ortogonálních řezů z obr. 3.13.

3.2.2 Výpočet uzlových bodů

V tomto kroku se průchodem všech kontur vytvářejí uzlové body (definice uzlového bodu, viz předchozí odstavec) například algoritmem založeným na stejné myšlence jako Cohen – Sutherlandův algoritmus pro ořezávání.

Postupně kráčíme po kontuře (polygon) a kdykoliv překročíme hranici aktuálního bloku, jsou vypočítány souřadnice uzlového bodu jako průsečíku segmentu kontury a roviny odpovídajícího ortogonálního řezu, viz obr 3.14. Složitost algoritmu lze tedy odhadnout jako O(n), kde n je počet bodů na kontuře, resp. v celém systému řezů. Samozřejmě, že rychlost závisí na tvaru tělesa.

Pro potřeby jednoduché rekonstrukce je možné vrcholy kontury mezi jednotlivými uzlovými body odebrat.

3.2.3 Vytváření relací kontaktu uzlových bodů

Jestliže máme vypočítané uzlové body, můžeme přistoupit k procesu vytváření relací kontaktu. Tzn., že naším úkolem je pro každý uzlový bod najít takový uzlový bod na ortogonální kontuře, který představuje průsečík těchto dvou kontur.

Metoda A

Pro každý uzlový bod kontury se prochází všechny uzlové body v konturách ortogonálního řezu, než dorazíme k vrcholu, který splňuje relaci kontaktu (definice viz odstavec 3.2.1).

Výhody této metody jsou jednoduchost a paměťová nenáročnost, nevýhodou je zvýšená časová složitost výpočtu (toto řešení vede na složitost $O(n^2)$).

Hledání kontaktů je možné urychlit pomocí ohraničujících indexů bloků, tzn. pokud je v ortogonálním řezu více kontur, nebudou se procházet všechny. Toto urychlení se ovšem neprojeví u málo členitých objektů, jejichž řezy sem skládají z malého počtu kontur.

Metoda B

Tato metoda je založena na představě prostorové mřížky, kterou tvoří systém ortogonálních řezů. Vytvoříme si pomocnou třírozměrnou matici s rozměry odpovídajími počtům řezů v jednotlivých sadách.

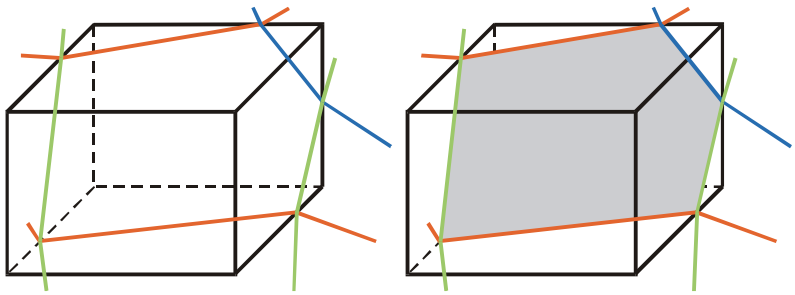
Před spuštěním procesu hledání kontaktů přiřadíme každému prvku matice seznam uzlových bodů, se kterými odpovídající blok, který v systému řezů odpovídá pozici prvku v matici, inciduje. Při hledání kontaktu se budou procházet pouze tyto seznamy.

Nevýhodou tohoto řešení je zvýšená paměťová náročnost, výhodou je vysoká rychlost výpočtu.

3.2.4 Jednoduchá rekonstrukce povrchu

Systém rovin ortogonálních řezů vytváří prostorovou mřížku v obecném případě s buňkami ve tvaru kvádru (dále budeme místo kvádr používat termín blok). Kontury incidují se stěnami bloků, jejich uzlové body s hranami bloků. Tím dostáváme množinu bloků mající na svých hranách body, které představují vrcholy plátů.

To připomíná situaci jako v metodách extrahujících povrch isoploch z volumetrických dat (pravidelná mřížka). Např. ve třetím kroku známého algoritmu *Marching Cubes* se na hranách krychle s ohodnocenými vrcholy (uvnitř / venku) interpolací počítají vrcholy budoucích plátů.

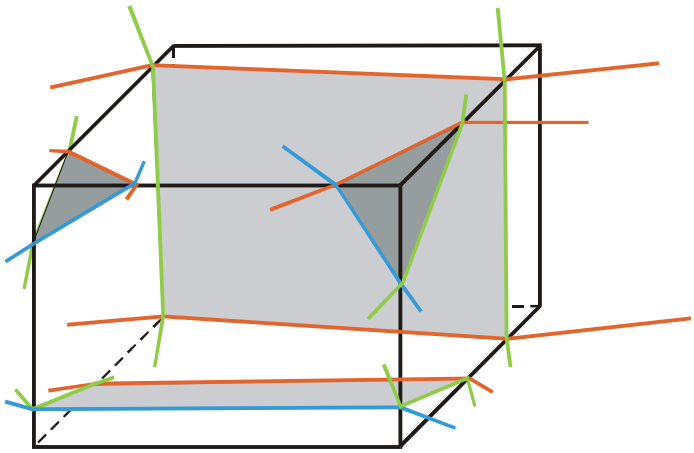


Obrázek 3.16: Ukázka bloku, který vzniká v systému ortogonálních řezů. Vlevo lze pozorovat kontury a jejich uzlové body incidující s tímto blokem, vpravo je vzniklý plát

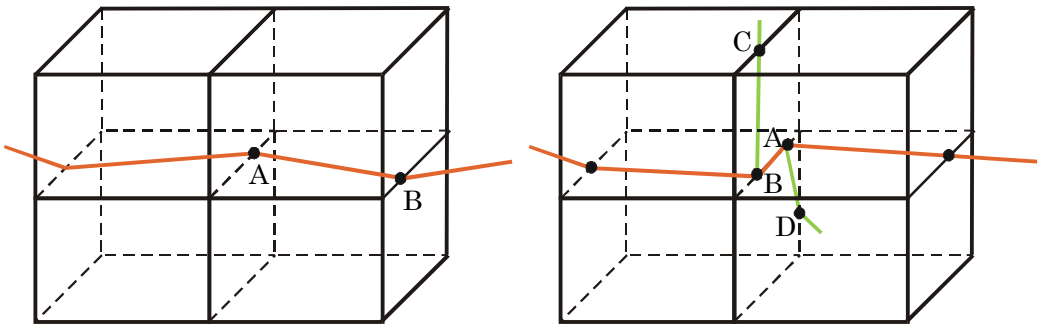
Pokud by v každém bloku vznikaly pouze pláty (tzn. konfigurace ohodnocení vrcholů by znamenala, že blokem prochází povrch), které vznikají při extrakci isoploch z volumetrických dat [1], úloha rekonstrukce povrchu z ortogonálních řezů by se zredukovala na úlohu výpočtu uzlových bodů (na hranách jednotlivých bloků) a následné nasazení libovolného algoritmu pro převod trojrozměrných objemových dat na trojúhelníky [1].

To však, jak ukazuje obrázek 3.17, není možné neboť v jednom bloku může vzniknout situace, kdy mají všechny čtyři vrcholy bloku stejné ohodnocení (např. uvnitř),

což je pro výše uvedené algoritmy signál, že se nejedná o blok, jímž prochází povrch, nýbrž o blok *uvnitř* nebo *mimo* těleso.



Obrázek 3.17: Ukázka možného případu incidence bloku s konturami a jejich uzlovými body. V tomto případě vznikají v bloku 4 pláty.



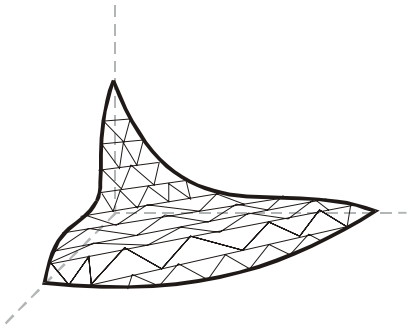
a) b)

Obrázek 3.18: Vlevo jednoduchý úsek, kdy oba uzlové body A i B incidují se čtyřmi bloky a segment AB se dvěma. Vpravo je složitější situace – body A i B sice incidují se čtyřmi bloky, segment AB však také

Princip metody je založen na předpokladech, že každý uzlový bod kontury inciduje se čtyřmi bloky a každý segment (úsek) zjednodušené kontury inciduje se dvěma (nad a pod rovinou řezu) nebo (ve složitějším případě) se čtyřmi bloky, viz obr. 3.18.

Proces plátování pracuje ve dvou fázích. V první fázi generuje pláty se segmenty incidujícími právě se dvěma bloky (nad a pod rovinou řezu), přičemž pro přechod na ortogonální kontury z důvodu nalezení vrcholů plátu využívá relace kontaktu.

Ve druhé fázi se řeší plátování kolem segmentů, které incidují se čtyřmi bloky. V tomto případě je nutné přidávat nové hrany (v situaci na obr. 3.18b jsou to hrany AC a BD), aby nedocházelo k špatným triangulacím, které by měly za následek díry ve výsledném povrchu.



Obrázek 3.19: Příklad plátování při úplné rekonstrukci

3.2.5 Úplná rekonstrukce povrchu

Úplná rekonstrukce je v základním principu je shodná s metodou jednoduché rekonstrukce, až v okamžiku plátování se začne pracovat i s vrcholy kontur ležících mezi jednotlivými uzlovými body, viz obr. 3.19. Úloha plátování přechází v úlohou triangulace prostorového polygonu.

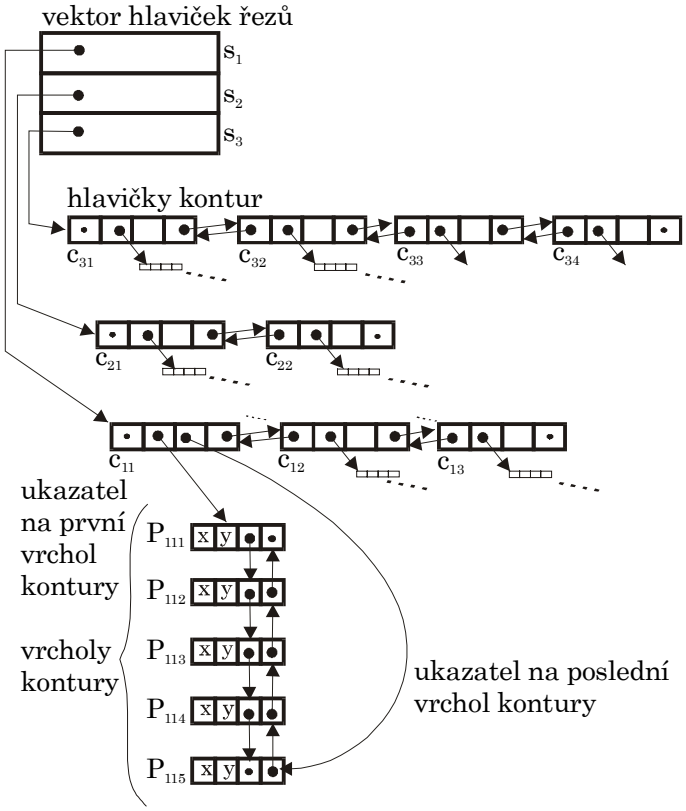
Kapitola 4

Realizace

V této kapitole bude uveden popis implementovaných algoritmů a datových struktur použitých při realizaci algoritmů pro výpočet řezů trojúhelníkových sítí a pro rekonstrukci povrchu z řezů. Dále budou diskutovány vlastnosti a omezení zvoleného řešení.

4.1 Výpočet řezů

Vstupem úlohy je trojúhelníková síť uložená ve struktuře T_Triangle_Mesh (popis viz příloha C) . Dále pak počet a rozmístění jednotlivých rovin řezů. Z důvodů uvedených v odstavci 2.2, je kladen požadavek, aby vstupní síť splňovala vlastnosti manifold těles.



Obrázek 4.1: Ukázka reprezentace řezů použitými datovými strukturami. Struktura představuje uložení sady řezů z obr. 2.1.

4.1.1 Datové struktury

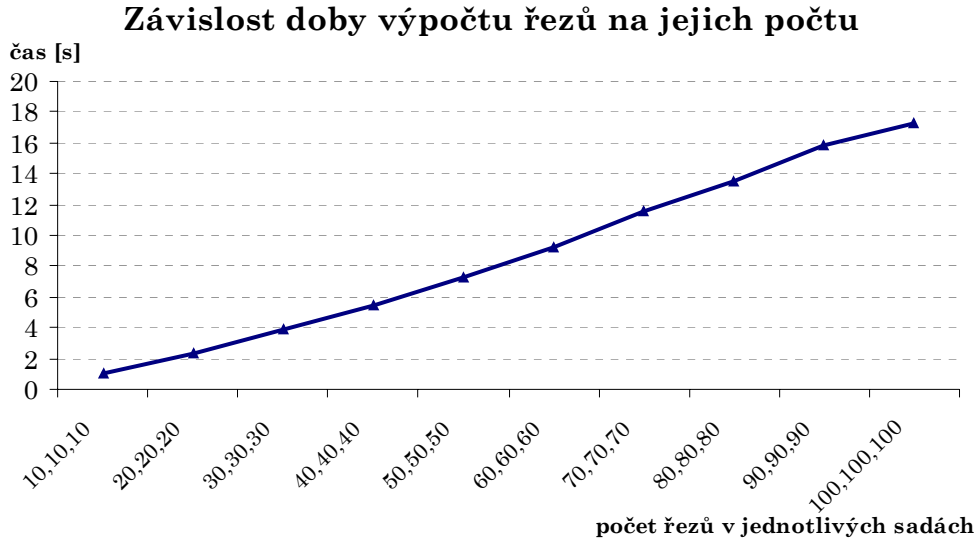
Protože z počtu trojúhelníků ve vstupní síti a počtu a rozmístění jednotlivých řezů nelze určit počet vygenerovaných kontur (ani počet vrcholů tvořících tyto kontury), pro uložení řezů jsme zvolili dynamické datové struktury. Na obr. 4.1 je uveden příklad použití datových struktur pro sadu řezů z obr. 2.1.

4.1.2 Implementovaný algoritmus

Algoritmus výpočtu řezů pracuje na principech popsaných v odstavci 2.3. Proces výpočtu průsečnic prochází celou množinu trojúhelníků, přičemž pro každý trojúhelník zjišťuje, zda-li nemá neprázdný průnik s některou rovinou, případně s množinou rovin řezů. Využíváme přitom toho, že roviny řezů jsou seřazené podle jejich polohy. Pro vyhledání roviny, která má průnik s trojúhelníkem, používáme *binární* vyhledávání [13]. Proto je složitost této části algoritmu $O(n\log(m))$, kde n je počet trojúhelníků vstupní sítě a m je počet řezů.

Proces sestavování kontur pracuje dvoufázově. V první fázi se vypočítána průsečnice ukládá do struktury řezu, přičemž se snažíme jí připojit k již existující části kontury. Pokud se připojení nezdaří je založena další kontura (část kontury). Po proběhnutí první fáze se tedy řez skládá z částí kontur.

Ve druhé fázi jsou z částí kontur sestavovány *uzavřené* kontury. Dvě části jsou spojeny v jednu, jsou-li shodné v jednom nebo v obou (uzavření kontury) krajních bodech. Algoritmus hledá nejlepší spojení, tzn. např. pro první část prochází všechny zbylé části kontur v řezu (přičemž *část kontury* se od *kontury* odlišuje tím, že není uzavřená). Složitost tohoto kroku je tedy $O(n\log n)$, kde n je nyní počet částí kontur.



Graf 4.1: Závislost doby výpočtu na počtu řezů v jednotlivých sadách. Byl řezán objekt `Bone.tri` složený ze 137072 trojúhelníků. Při počtu řezů 100 v každé sadě (poslední sloupec) zabíraly datové struktury přibližně 9 MB. Údaje byly naměřeny na počítači s jedním procesorem Intel Pentium II 433 MHz se 128 MB RAM.

V dalších krocích se vytváří hierarchie kontur v řezu (podle principu popsaného v odstavci 2.3.3). Tento krok má složitost $O(n\log n)$, kde n je nyní počet kontur v řezu. Nastavení výsledné orientace již probíhá v lineárním čase v závislosti na počtu kontur.

Po skončení výpočtu je nutné vygenerované řezy uložit do SLC struktur pro reprezentaci sad řezů. Popis struktury SLC, viz příloha C.

4.1.3 Paralelizace

Úloha výpočtu řezů v jednotlivých sadách je velmi dobře paralelizovatelná (lze zpracovávat každou sadu zvlášť, a také v rámci jedné sady lze počítat zvlášť jednotlivé řezy) bez nutnosti použití synchronizačních technik pro ošetření kritických sekcí [9]. Proto je výpočet řezů v jednotlivých sadách rozdělen mezi vlákna a zpracováván paralelně (pozn. pseudoparalelně na počítači s jedním procesorem).

4.1.4 Systémy pro výpočet řezů

V praxi existuje mnoho způsobů jak reprezentovat prostorová tělesa. Mezi nejpoužívanější patří:

- polygonální reprezentace,
- implicitní plochy,
- CSG stromy,
- volumetrická data (data uložená v prostorových mřížkách).

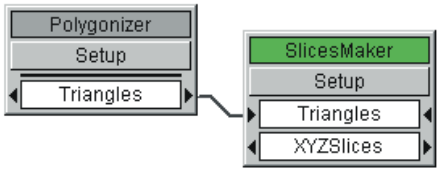
Pokud bychom potřebovali vytvářet řezy těles zadaných všemi výše uvedenými reprezentacemi, máme na výběr dvě možnosti jak tento problém vyřešit. První možnost je vytvořit optimální algoritmy generující řezy tělesa pro každou reprezentaci. Vyvinout takové metody vyžaduje úsilí, důvodem pro toto řešení však může být požadavek na vysokou rychlost výpočtu.

Druhou možností je využít publikované, případně již implementované algoritmy pro převod výše uvedených reprezentací na trojúhelníkové sítě a používat pouze jeden algoritmus pro generování řezů.

Prostředí MVE (Modular Visualization Environment) v současné době obsahuje moduly pro generování trojúhelníkového povrchu implicitních funkcí i volumetrických dat.

Systém pro výpočet řezů implicitních funkcí

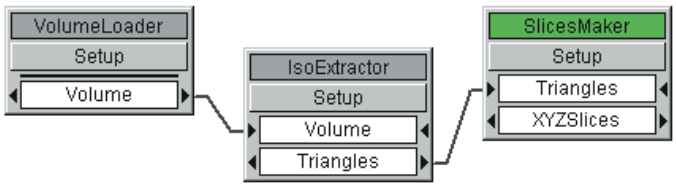
Použijeme modul *Polygonizer* pro vytvoření povrchu implicitních funkcí se strukturou trojúhelníkové sítě. Schéma propojení modulů viz obr. 4.2.



Obrázek 4.2. Schéma propojení modulů pro generování řezů implicitních funkcí

Systém pro výpočet řezů volumetrických dat

Pro vytvoření isoplochy trojúhelníkové struktury z volumetrických dat, která odpovídá zadanému prahu je v MVE k dispozici modul *IsoExtractor*. Pro načítání volumetrických dat lze použít modul *VolumeLoader*. Schéma propojení modulů, viz obr. 4.3.



Obrázek 4.3: Schéma propojení modulů pro výpočet řezů volumetrických dat.

4.2 Rekonstrukce povrchu

Vstupem úlohy je systém ortogonálních řezů $\boldsymbol{S} = \{S_{xy}, S_{yz}, S_{zx}\}$, sady S_{xy}, S_{yz}, S_{zx} jsou uloženy ve struktuře SLC pro reprezentaci sad řezů. Pokud je neprázdná právě jedna sada (konfigurace $\{S_{xy}, 0, 0\}$, $\{0, S_{yz}, 0\}$, nebo $\{0, 0, S_{zx}\}$) vstupního systému S, je spuštěn proces rekonstrukce povrchu z paralelních řezů, v ostatních případech (kromě konfigurace {0, 0, 0}) se spouští rekonstrukce z ortogonálních řezů.

4.2.1 Datové struktury

Protože se v průběhu rekonstrukce povrchu vyskytují případy, kdy je nutné odebírat či přidávat další body do kontury (např. uzlové body při rekonstrukci z ortogonálních řezů) nebo kontury dělit na části (při řešení problému bifurkací), byla zvolena dynamická datová struktura. Protože však k výše uvedeným případům, kdy je třeba zasahovat do kontur a měnit jejich složení nedochází příliš často, byla navržená kompromisní optimalizovaná homogenní datová struktura pro ukládání seznamu zřetězených prvků, viz obr. 4.4.

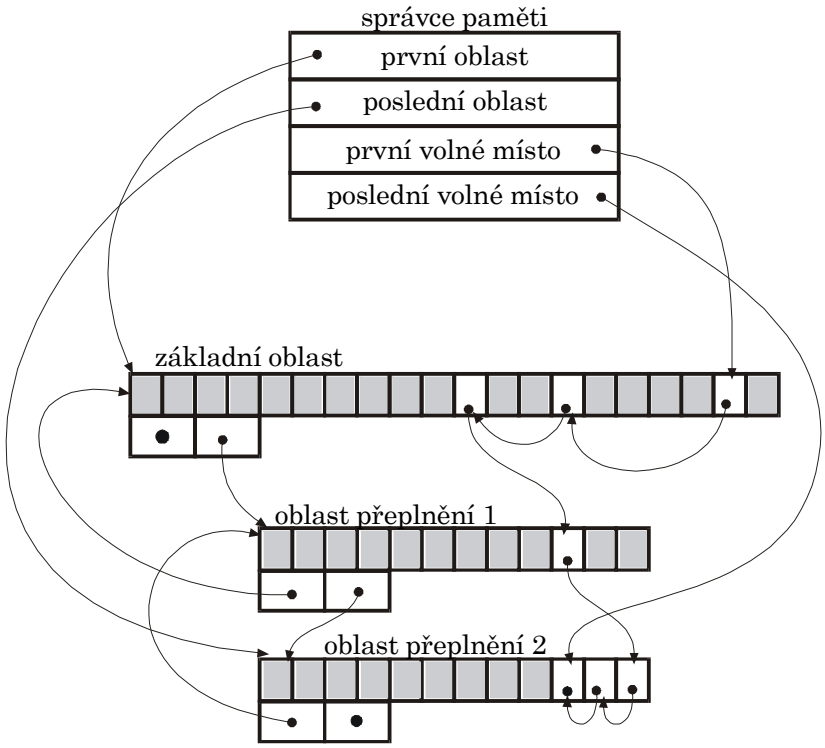
V prvním kroku se vytvoří vektor prvků požadované velikosti. Prvky jsou spolu propojeny v obousměrně zřetězeném seznamu, jak je běžně zvykem. Pokud se v průběhu výpočtu vyskytne požadavek na přidání prvku, alokuje se vektor velikosti (jakási oblast přetečení) např. jedné desetiny velikosti prvního vektoru do které je nový prvek uložen, přičemž se patřičně aktualizuje zřetězený seznam. Pokud po zaplnění první oblasti přetečení vznikne požadavek na přidání dalšího prvku, připojí se další oblast přetečení.

Správa celé struktury je v samostatné vrstvě, při používání zřetězeného seznamu v metodách rekonstrukce povrchu se používají běžné funkce pro práci se seznamy (přidej, odeber, předchůdce, následník, atd.).

Výhodou použití této struktury je optimalizovaný způsob získávání dynamické paměti nikoliv po drobných částech, ale po větších blocích. Dále pak jednoduchost, s jakou se seznamy používají (není třeba neustále implementovat metody pro práci se zřetězeným seznamem).

Nevýhodou je zvýšená paměťová náročnost, neboť je nutné uchovávat pomocné informace o struktuře. V případě že se prvky ruší, není možné odebírat oblasti přetečení, klesne-li počet prvků na úroveň velikosti základní oblasti. V takovém případě je nutné prvky uklidit z oblastí přeplnění do základní oblasti. Je však třeba si uvědomit, že po

provedení tohoto kroku se prvky v paměti přemísťují, tzn. není možné spoléhat na to, že používané přímé ukazatele na prvky seznamu zůstanou platné. Implementační detaily, viz příloha C.



Obrázek 4.4: Ukázka správy paměti navržené optimálně pro práci s konturami a jejich body. Základem je zřetězený seznam prvků, o jejich zřetězení se však *správce paměti* nestará. Ten pouze vrací ukazatele na volná místa, která má zřetězená v seznamu volných míst. V případě, že je tento seznam prázdný a vznikne požadavek na přidání prvku alokuje další oblast přeplnění.

4.2.2 Implementace algoritmu rekonstrukce povrchu z paralelních řezů

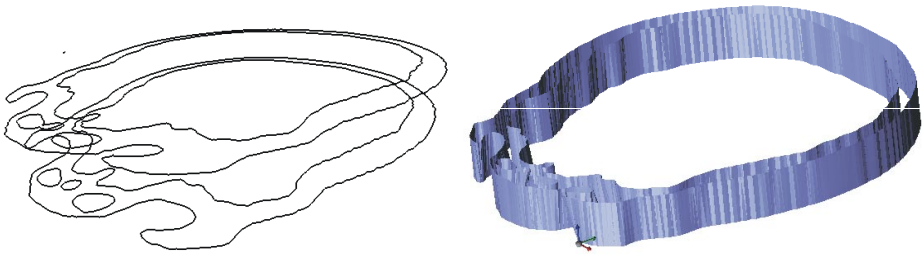
Pro rekonstrukci povrchu z paralelních řezů byl zvolen povrchový přístup pracující přímo s konturami, neboť rekonstrukcí povrchu z objemových mřížek se zabývají studie extrakce isoploch z volumetrických dat (algoritmy založené na metodách Marching Cubes, Marching Tetrahedra a dalších).

Problém korespondence

Problém korespondence je řešen na základě překrývání normovaných kontur v sousedních řezech, přičemž se nepracuje přímo s plochou kontury (v obecném případě nekonvexní mnohoúhelník), ale s čtyřúhelníkovými ohraničujícími obálkami (bounding box).

Problém větvení a plátování

Problematika větvení byla omezena na zpracování opláštění dvojice kontur (1 : 1), to znamená, že implementovaný algoritmus se v případě větvení kontury k rozhodne pro „nejlepší“ větev, se kterou konturu k spojí 1:1. Ostatní větve ponechá nespojené. Vycházíme přitom z faktu, že každé větvení je pouze odhadem tvaru objektu. Algoritmus neřeší plátování vík a den. Ukázka výsledku opláštění, viz obr. 4.5.



Obrázek 4.5: Vlevo dva rovnoběžné řezy lebkou člověka (objekt `CTMayo`), vpravo ukázka opláštění kontur implementovaným algoritmem.

Paralelizace

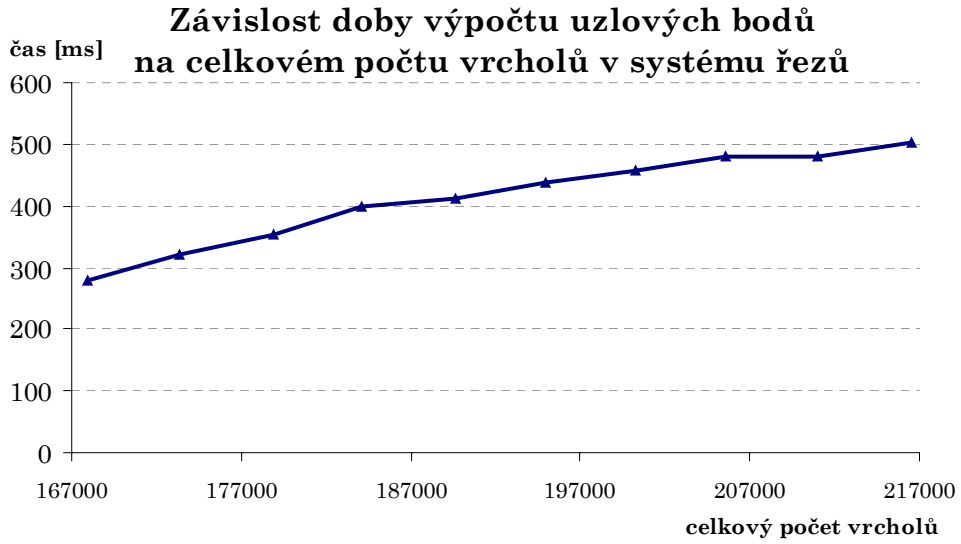
Protože zvolený postup rekonstrukce patří mezi lokální techniky, tj. vytváří se opláštění kontur vždy pouze mezi dvěma řezy, je úloha dobře paralelizovatelná. Každému výpočetnímu vláknu lze přidělit množinu řezů, které má rekonstruovat a není potřeba zabývat se kritickými sekcemi.

4.2.3 Implementace algoritmu rekonstrukce povrchu z ortogonálních řezů

Pro implementaci metod rekonstruující povrch z ortogonálních řezů byly použity principy naznačené v odstavci 3.2.

Výpočet uzlových bodů

Princip metody pro výpočet uzlových bodů kontury v daném systému řezů je uveden v odstavci 3.2.2. Graf 4.2 ověřuje její lineární složitost v závislosti na celkovém počtu vrcholů kontur v celém systému.

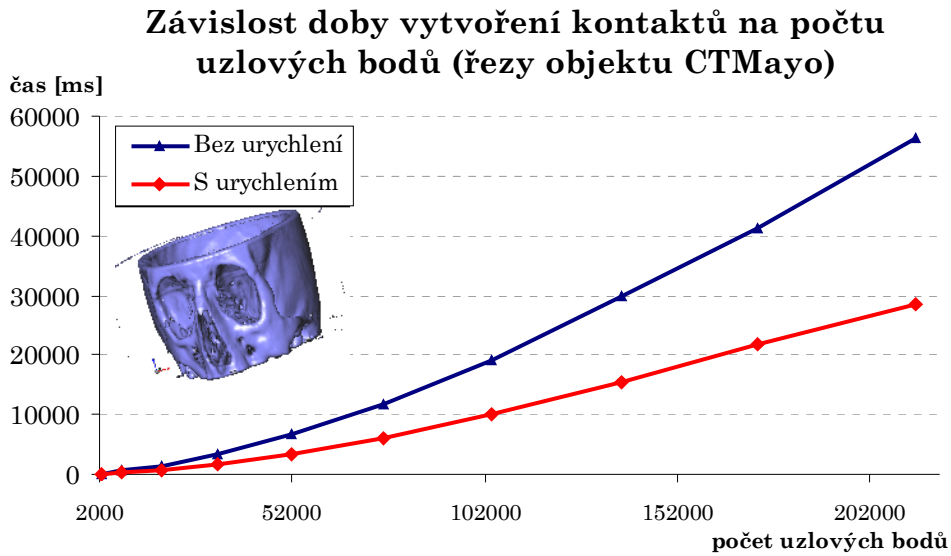


Graf 4.2: Ověření lineární složitosti algoritmu pro výpočet uzlových bodů

Hledání kontaktů

Pro implementaci úlohy vytvoření relace kontaktu uzlových bodů byla zvolena *metoda A*, popsaná v odstavci 3.2.3. Z hlediska časové náročnosti je tato část nejslabším místem procesu rekonstrukce, neboť všechny ostatní části mají složitost lineární.

Graf 4.3 ukazuje porovnání časové složitosti metody před a po implementaci urychlení (odstavec 3.2.3). Myšlenkou urychlení při hledání kontaktu pro uzlový bod p je procházet jen takové kontury v ortogonálním řezu, jejichž ohraničující obálka (bounding box) reprezentovaná indexy bloků inciduje s p. Toto urychlení se však projeví pouze u objektů složitějších tvarů, které mají v jednotlivých řezech větší počet kontur.



Graf 4.3: Porovnání doby nutné pro vytvoření kontaktů v závislosti na počtu uzlových bodů metodou bez urychlení s metodou s urychlením pomocí ohraničujících indexů bloků (analogie s ohraničující obálkou obdélníku tvaru – bounding box). Důležité je poznamenat, že urychlení bylo dosaženo proto, že řezy objektu CTMayo se skládají z více kontur.

Rekonstrukce povrchu

Byl implementován algoritmus pro jednoduchou rekonstrukci, pracující (princip viz odstavec 3.2.4) ve dvou fázích. V první fázi se plátují jednoduché a jednoznačné oblasti, které vytvářejí kontury v prostorové mřížce systému rovin řezů. Ve druhé fázi se algoritmus snaží najít řešení pro oblasti incidující se *složitými* segmenty (za složitý segment je považován segment incidující se čtyřmi bloky).

Nedostatkem implementovaného algoritmu je, že ve velmi složitých a singulárních případech je neúspěšný a na povrchu tělesa zanechává díry.

Paralelizace

Výpočet uzlových bodů a hledání kontaktů jsou dobře paralelizovatelné části, neboť pracují na jednotlivých konturách a proto je mohou jednotlivá výpočetní vlákna zpracovávat odděleně.

Ve fázi rekonstrukce povrchu se pracuje se všemi konturami najednou, jsou aktualizovány příznaky incidence apod. Pro paralelní běh bychom museli zavést

synchronizační mechanismy pro ošetření kritických sekcí. Paralelizace tohoto kroku však není nutná, neboť rychlost výpočtu rekonstrukce není slabým místem procesu.

4.3 Další zpracované úlohy

Pro práci s řezy bylo nutné vytvořit modul pro načítání řezů ve formátu SLC (viz příloha C). Pro pochopení případů, které nastávají na jednotlivých konturách a v blocích systému řezů byl vytvořen vizualizér řezů, který pro zobrazení využívá grafickou knihovnu *OpenGL*.

Vizualizer řezů má implementovány funkce pro zobrazení jednoho, všech nebo žádného řezu každé sady, umožňuje nastavení barev a tloušťky čar. Zobrazuje orientaci kontur, jednotlivé body kontur a mřížku, kterou vytváří systém rovin ortogonálních řezů.

4.4 Knihovna SlicesModules.dll

Všechny výše uvedené úlohy byly zpracovány jako moduly (oddělené funkce) a umístěny do dynamicky připojované knihovny `SlicesModules.dll`, která má implementované rozhraní pro export modulů (funkcí) do systému MVE (Modular Visualization Environment). Následuje výčet modulů se stručným popisem zpracovávané úlohy.

- *SlicesMaker* – výpočet, editor řezů,
- *SurfReconstr* – rekonstrukce povrchu z řezů,
- *SlicesLoader* – načítání řezů uložených ve struktuře SLC,
- *SlicesViewer* – zobrazování systému řezů.

Moduly jsou napsány v programovacím jazyce C++, knihovna sestavena v prostředí *Microsoft Visual C++*. Knihovnu lze používat v systémech s jádrem Win32 (Windows 95, 98, NT, 2000). Popis instalace knihovny je v příloze B, popis ovládání modulů, viz příloha A.

4.5 Samostatná aplikace „Slices.exe“

Aplikace **Slices.exe** slouží ke generování řezů vstupních trojrozměrných objektů reprezentovaných trojúhelníkovou sítí a pro rekonstrukci povrchu z vypočítaných nebo načtených řezů ve formátu SLC.

Pro načítání a zobrazování trojúhelníkových sítí využívá moduly *TriangleLoader* a *Renderer* z knihovny `Triangle_Modules.dll`, dodávané spolu s MVE. Ostatní funkce využívá z knihovny `SlicesModules.dll`.

Aplikace je, stejně jako knihovny, které používá, konstruována pro systémy s jádrem Win32 (Windows 95, 98, NT, 2000). Popis instalace a ovládání programu, viz příloha A.

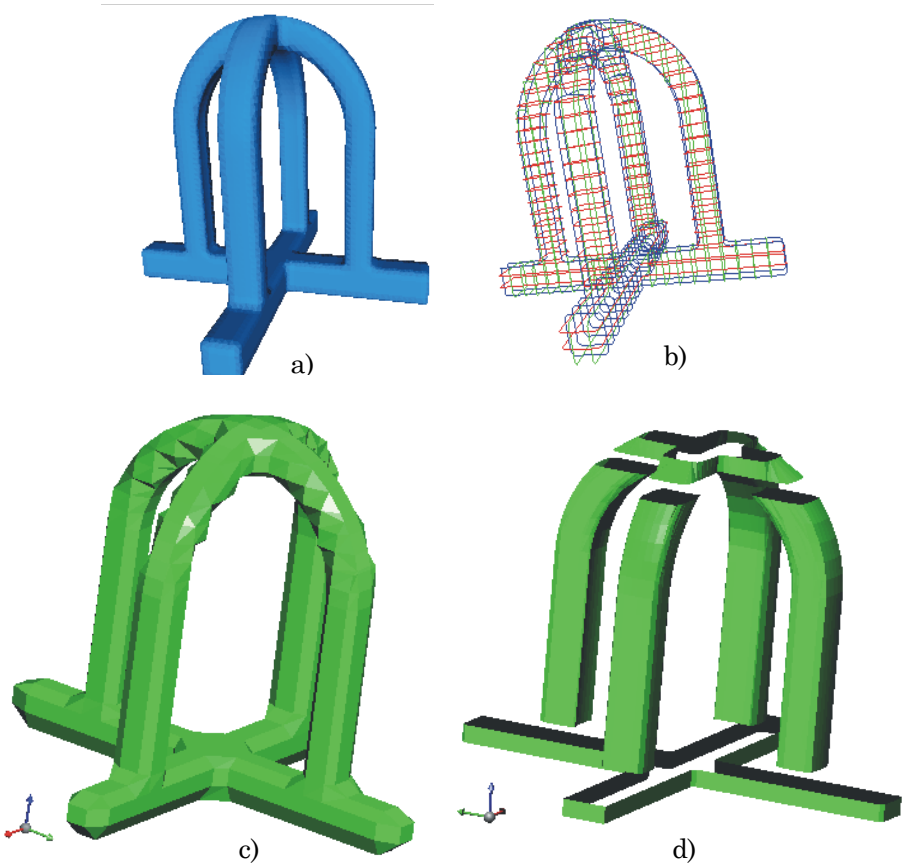
Kapitola 5

Dosažené výsledky

V této kapitole budou představeny výsledky, kterých bylo dosaženo implementací algoritmů pro výpočet řezů a rekonstrukci povrchu z řezů.

5.1 Objekt „Syn64“

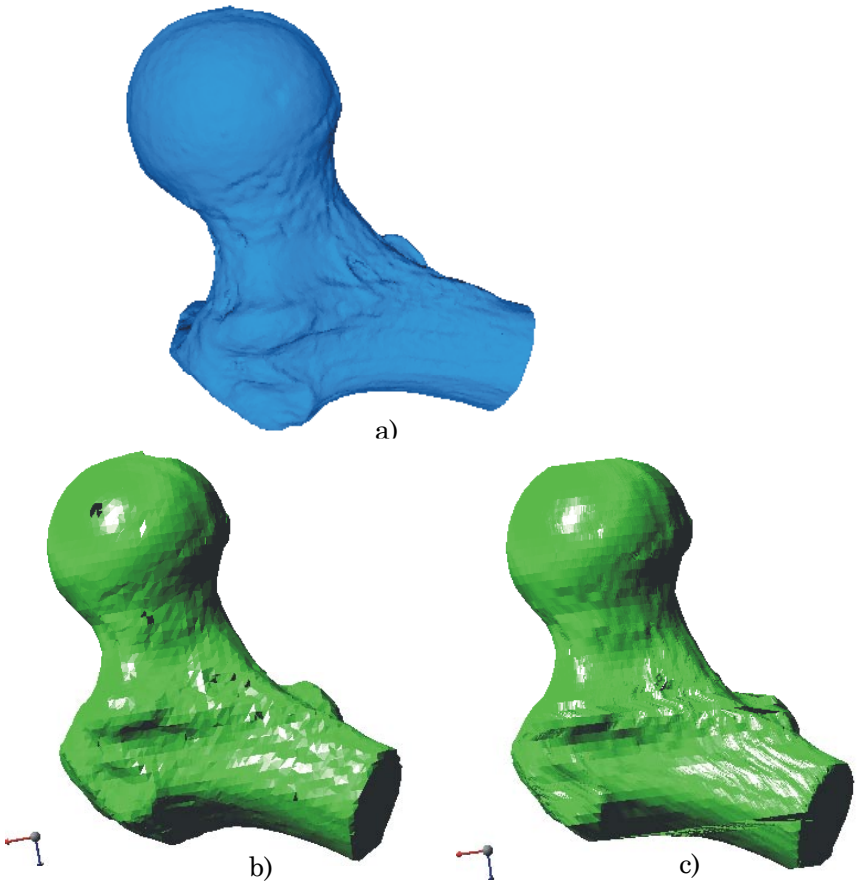
Objekt *Syn64* (14858 trojúhelníků, obr 5.1a) byl nařezán 20-ti řezy v každé sadě (obr 5.1b). Výsledný povrch jednoduché rekonstrukce z ortogonálních řezů tvoří 1924 trojúhelníků (obr 5.1c), v místech záhybů je dosti hrubý. Povrch rekonstrukce z paralelních řezů (7220 trojúhelníků, obr. 5.1c) je hladký jako původní povrch, metoda však neřeší plátování větvení kontur, vík a den.



Obrázek 5.1: a) objekt Syn64, b) systém sad řezů (každá sada obsahuje 20 řezů), c) rekonstrukce z ortogonálních řezů, d) rekonstrukce z paralelních řezů

5.2 Objekt „Bone“

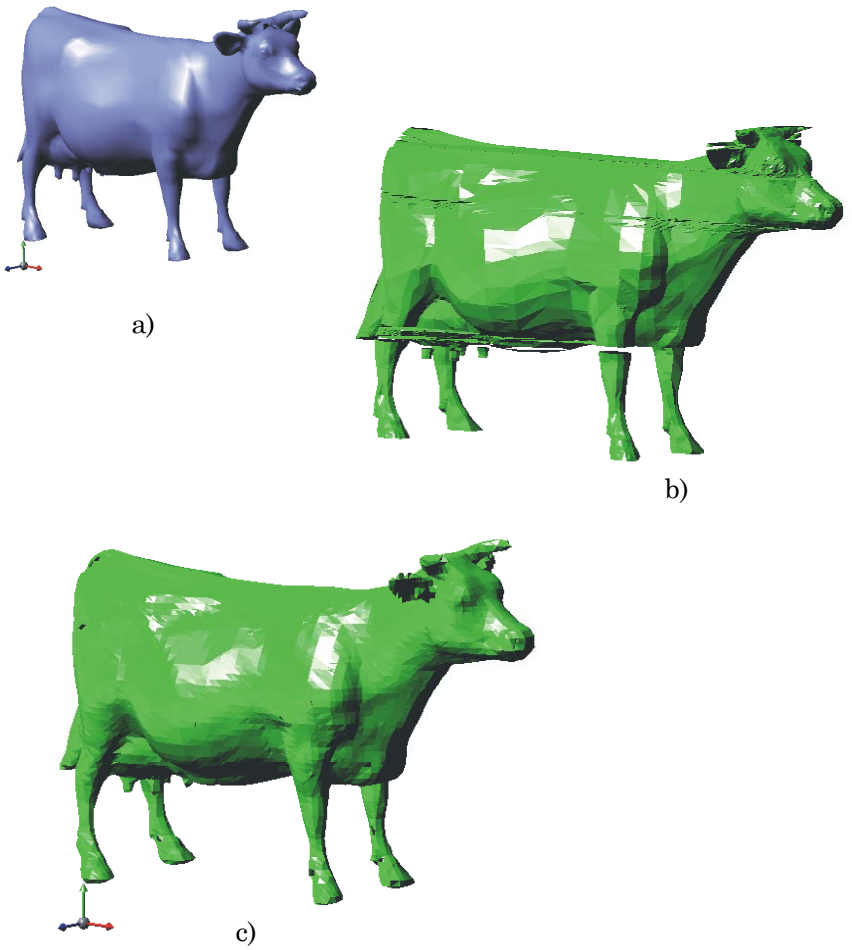
Objekt Bone (137072 trojúhelníků, obr. 5.2a) byl nařezán 50-ti řezy v každé sadě. Povrch získaný rekonstrukcí z ortogonálních řezů (20285 trojúhelníků, obr. 5.2b) obsahuje chyby (díry). Ve spodní části rekonstruovaného povrchu z paralelních řezů (54331 trojúhelníků, 5.2c) lze pozorovat zkroucení.



Obrázek 5.2: a) původní objekt Bone, b) rekonstruovaný povrch z ortogonálních řezů (50 řezů v každé sadě), c) povrch rekonstruovaný z paralelních řezů (50 řezů)

5.3 Objekt „Cow“

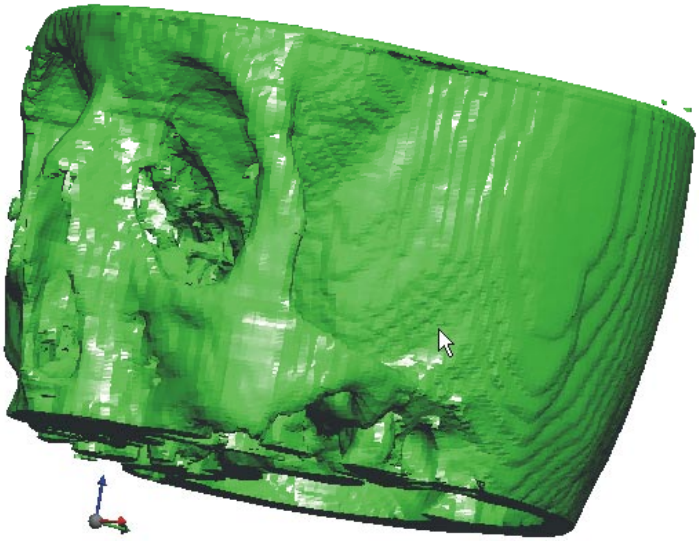
Objekt Cow (5782 trojúhelníků, obr. 5.3a) představuje objekt se složitějším tvarem. Rekonstruovaný povrch ze 100 paralelních řezů (32121 trojúhelníků, obr. 5.3b) vykazuje chyby špatného větvení (napojení těla a noh krávy) i kroucení povrchu. Povrch rekonstruovaný z ortogonálních sad řezů (40, 70, 60 řezů v jednotlivých sadách, 25324 trojúhelníků výstupní sítě, obr. 5.3c) se velmi blíží originálu, pouze v částech vysokých detailů (uši, rohy) je hrubý.



Obrázek 5.3: a) původní objekt Cow, b) povrch rekonstruovaný z paralelních řezů, c) povrch získaný z rekonstrukce z ortogonálních řezů

5.4 Objekt „CTHead“

Lidská lebka (objekt CTHead) představuje složitý objekt pro implementovanou metodu rekonstrukce povrchu z paralelních řezů, která v tomto případě vrací velmi degenerovaný povrch. Metoda rekonstrukce povrchu z ortogonálních řezů však dává velmi uspokojivé výsledky a výsledný povrch je (při počtu řezů 100 v každé sadě) téměř shodný s původním, viz obr. 5.4. Vstupní síť má 196000, výstupní 212473 trojúhelníků.



Obrázek 5.4: Povrch rekonstruovaný z ortogonálních řezů

5.5 Objekt „Teapot“

Tento objekt se nehodí pro rekonstrukci z ortogonálních řezů, neboť nesplňuje podmínku manifold tělesa (hrdlo je válec nulové tloušťky). Avšak řezům v xy-rovině toto nevadí a použitím rekonstrukce z paralelních řezů, získáme velice uspokojivý povrch, viz obr. 5.5. Vstupní síť má 160000, výstupní 71958 trojúhelníků.

Kapitola 6

Závěr

V následujících odstavcích budou zhodnoceny dosažené výsledky dosavadní práce a zároveň bude uveden výčet problémů a dalších úkolů, které by měly být v budoucnu vyřešeny.

6.1 Zhodnocení výsledků

Implementované metody pro výpočty řezů pracují uspokojivě a pokud vstupní těleso je manifold, jsou výsledné kontury bez negativních jevů typu křížení apod. Výpočet probíhá s optimální rychlostí.

Metoda pro rekonstrukci povrchu z paralelních řezů vykazuje všechny chyby, které jsme předpokládali (byly publikovány). Jedná se velice jednoduchou metodu, která není v obecném případě vhodná pro objekty, jejichž tvar se často větví a jejichž řezy obsahují konkávní nebo zavinuté kontury.

Implementovaný algoritmus jednoduché rekonstrukce z ortogonálních řezů má problémy s plátováním v singulárních případech, které nastávají na složitějších systémech ortogonálních kontur. Tím vznikají na površích nežádoucí díry. Výhodou algoritmu je, že nevytváří degenerovaný povrch.

6.2 Budoucí práce

Ukazuje se, že požadavek, aby vstupní těleso bylo manifold, je zbytečně silný, stejně jako není nutné pracovat pouze s uzavřenými konturami. Další implementace generátoru řezů by měly umožňovat vytváření i neuzavřených kontur. Bude potřeba změnit i strukturu pro ukládání řezů, neboť SLC struktura je navržena pouze pro uzavřené kontury. Stačilo by tedy požadovat, aby vstupní těleso neobsahovalo protínající se povrch.

V oblasti rekonstrukce povrchu z paralelních řezů, bylo publikováno mnoho algoritmů. Většina algoritmů má jistá omezení a nejsou vhodné pro obecné případy. Určitou odlišnost vykazuje Olivův algoritmus (odstavec 3.1.5). Bylo by vhodné tento algoritmus detailně nastudovat, implementovat a pokusit se jej vylepšit.

Zdá se, že má smysl se dále zabývat rekonstrukcí povrchu z ortogonálních řezů, neboť dosavadní výsledky ukazují, že výsledná rekonstrukce dobře aproximuje tvar původního tělesa.

Bude zřejmě výhodné pro rekonstrukci zvolit více objemový přístup, tzn. pracovat s prostorovou mřížkou, kterou tvoří roviny ortogonálních sad řezů, protože zamezuje šíření případné chyby mimo prostorovou buňku. Tím půjde také urychlit vyhledávání kontaktů,

současné nejslabší místo a celou složitost celého algoritmu tím stlačit na O(n), kde n je počet bodů v systému řezů.

Dále bude třeba přistoupit k řešení *úplné rekonstrukce*, tzn. využívat tvar kontur mezi jednotlivými uzlovými body. Aproximace nahrazením úsečkou způsobuje příliš hrubé výstupní povrchy.

Kapitola 7

Literatura

[1] Žára, J., Beneš, B., Felkel, P.: Moderní počítačová grafika, Computer Press, 1998.

[2] Garland, M: *Multiresolution Modeling: Survey & Future Opportunities*. SIGGRAPH 97 course notes, 1997.

[3] Ježek, F.: Geometrické a počítačové modelování, ZČU Plzeň, únor 2000.

[4] Meyers, D., Skinner, S., Sloan, K.: Surface from Contours. ACM Transaction on Graphics 11, 3, July 1992

[5] Keppel, E.: Approximating complex surfaces by triangulation of contour lines. "IBM Journal of Research and Development", 1975.

[6] Ekoule A. B., Peyrin F. C., Odet C. L.: A triangulation algorithm from arbitrary shaped multiple planar contours. ACM transaction on graphics, April 1991.

[7] Bajaj, C., Coule, E., Lin, K.: Arbitrary Topology Shape Reconstruction from Planar Cross Sections, Publikováno na 4. SIAM konferenci, Nashville, Tennesse, November 1995.

[8] Oliva J.–M., Perrin, M., Coquillart S.,: 3D Reconstruction of complex polyhedral shapes from contours using a simplified generalized Voronoi diagram. Computer Graphics Forum, 1996.

[9] Ježek, K., Matějovic, P., Racek, S.: Paralelní architektury a programy. ZČU, Plzeň, 1997.

[10] Lederbuch, P.: Metody rekonstrukce povrchu z rovinných řezů (referát k rigorózní zkoušce), Plzeň, 1997.

[11] Felkel, P.: Algorithms for Computed Assisted Segmentation and Reconstruction of Three Dimensional Biomedical Data, ČVUT, Praha, November 1998

[12] Soroka, B. I.: Generalized Cones from Serial Sections. Comput. Graph. Image. Proc., 1981

[13] Rychlík, J.: Programovací techniky, Kopp, České Budějovice, 1995

PŘÍLOHA A

User's guide

This document serves as a user guide for the MVE modules *SlicesMaker*, *SlicesLoader*, *SlicesViewer* and *SurfReconstr*, which can be found in the `SlicesModules.dll` library. In the following paragraphs you will learn what purpose do these modules serve for as well as how to use them within the Modular Visualization Environment.

SlicesMaker module

The *SlicesMaker* was designed to compute 2D slices of the object's with triangle mesh surface. Therefore, the input of the *SlicesMaker* module is the Triangles MVE type. As the result of its computations, the module offers a sets of slices, so the *XYZSlices* MVE type is the module's output type.

Since the *SlicesMaker* module requires a triangle represented object, it may only follow modules that produce the objects (manifold objects) with triangle mesh structures. That is, for instance, the *TriangleLoader* module, the *Polygonizer* module or the *IsoExtractor* module. The examples of the possible schemes are in the Fig. A.1

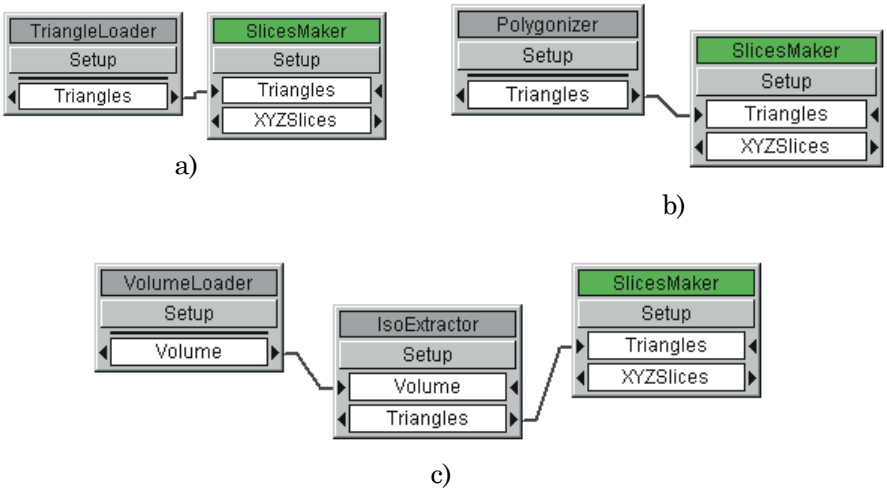


Figure A.1: Possible schemes to put data to the *SlicesMaker* module. a) slices of triangle meshes, b) slices of implicit functions, c) slices from volume data will be computed

It is possible to adjust the *SlicesMaker* module default parameters by clicking the Setup button on the middle icon on the MVE main screen before the whole schema is executed. In the setup window you can adjust the number of slices in each set, the orientation of the contours and the output directory for saving computed data (sets of slices).

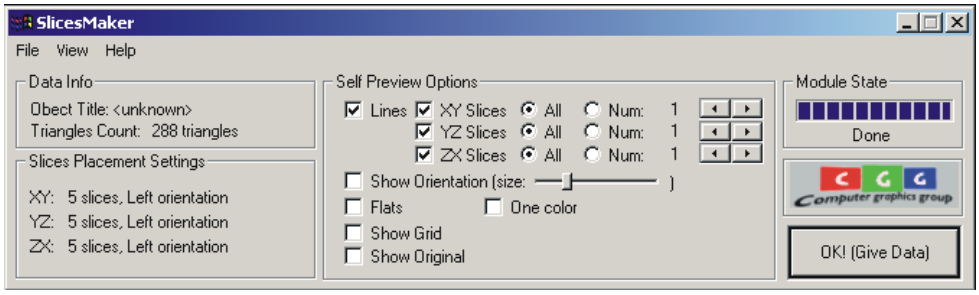


Figure A.2: Main window of the application. In panel *Self Preview Options* you can adjust the current mode of drawing the contours (similar to the *SlicesViewer* module).

In the figure A.2., there is the *SlicesMaker* main window. If you choose the *File->Save to SLC* menu option, the program will save the computed data to the file(s).

In the menu *View*, there you can see two options. First is for show the *Slices Setup Dialog* (this dialog we'll describe later). Second is for show the *Slices Preview window*, where you can watch the computed slices.

If you need to create special placement of the slices (not the default uniform placement), you have to use the *Slices Setup Dialog* (see fig. A.3; this dialog you can open from the *SlicesMaker* main dialog in the menu *View->Slices Setup Dialog*). In this dialog you can operate with individual slices (add or remove them) or you can work with the selection.

In the dialog, ther is the input object rendered (the blue one), the red lines represents the slices (their position) and the yellow bar represents the current selection of slices. The selection is created if there exists one slice (represented as red line) with enabled *Begin of the selection* check box and one another with selected *End of selection*.

If you want remove on slice from the set, you can do it in two ways. You can click on it with the mouse and then push the *Delete Slice* button. Or if you know the number of the slice (for example from the *Slices Preview Window*), you can put it into the edit box in the *Current Slice* panel. And than push the *Delete Slice* button.

If you want to add one single slice, push the button *Add Slice!*. The program will prompt you for the position of the slice.

If you want to remove more slices, you have to select them with the yellow bar. And then use the button *Delete Slices* in the *Operations in Selection* panel.

If you need to add more slices, you have to put the two border slices marks (the red lines), then create the selection and then choose *Add Slices* button. The entered count of slices will be added into this selection with the uniform placement.

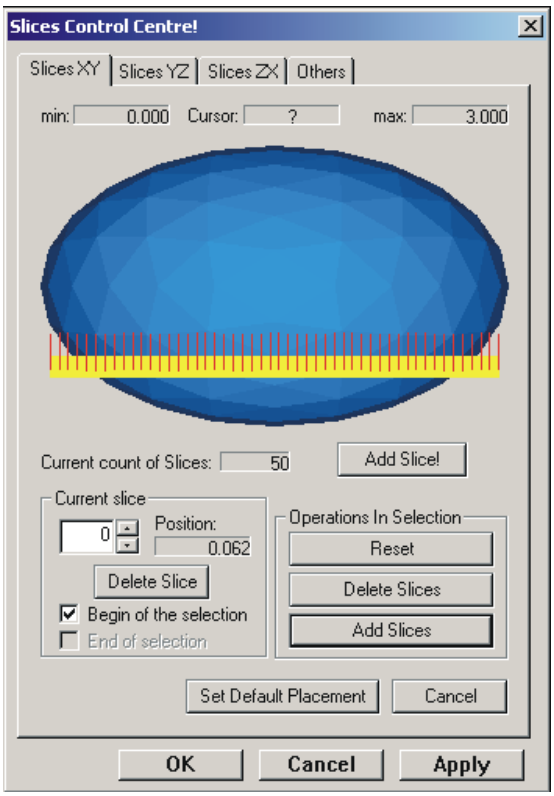


Figure A.3: Slices Setup Dialog. Here you can choose how many slices (and their position also) will be in each set.

SlicesLoader module

This module was created to load slices data with SLC structure. The output type of this module is the *XYZSlices* type. So, for example the *SlicesMaker* od *SlicesViewer* module can follow.

In the setup window of the *SlicesLoader* module enter the names of the files which contains the set of the slices in the SLC structure.

SlicesViewer module

This module serves to display the sets of slices. Therefore, the input of the *SlicesViewer* module is the *XYZSlices* type. In the fig. A.4 is described how to adjust the attributes of displayed contours. You use the mouse to move, rotate and scale with the object in the *SlicesViewer* window.

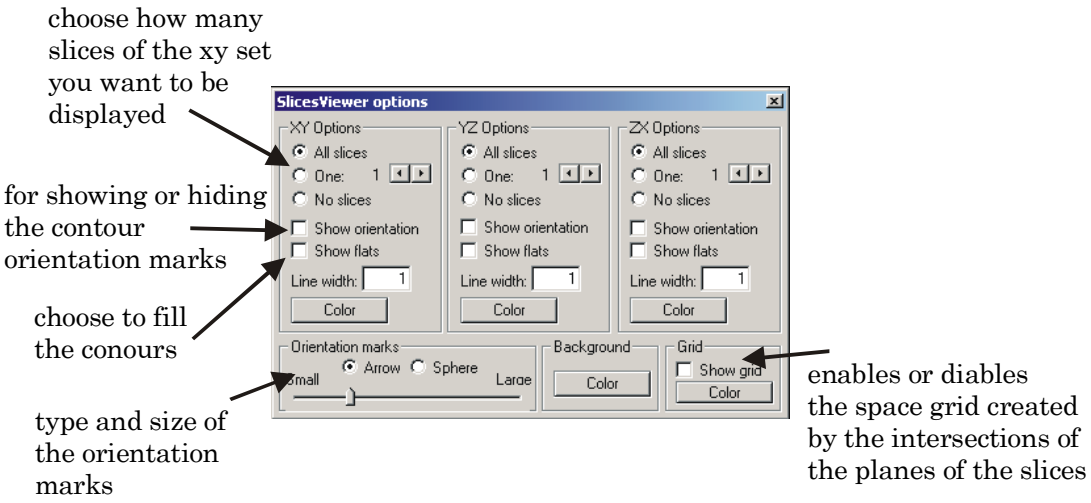


Figure A.4: Options panel for the *SlicesViewer* module

SurfReconstr module

This module was created to reconstruct the surface of the object represented by slices. Its input can be each module which produces 2D slices with the SLC structure. The output type of the *SurfReconstr* module is the *Triangles* type.

When you put just one set of slices on the input of *SurfReconstr* module, the reconstruction using the algorithm for surface reconstruction from parallel slices will start. If you put orthogonal sets of slices, the reconstruction process using the algorithms for surface reconstruction from orthogonal slices will start.

The Application Slices.exe

You don't have to use MVE to create slices from triangle meshes or to reconstruct surface from slices. You can use the `Slices.exe` application.

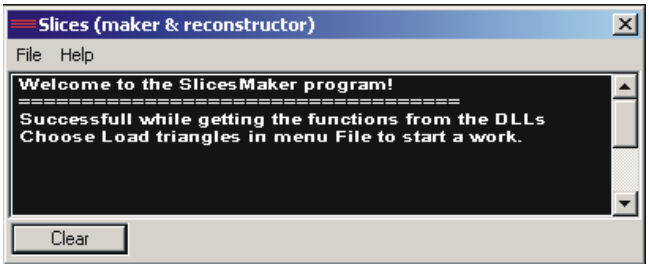


Figure A.5: Main window of the *Slices.exe* application

When you execute the `Slices.exe` application, the same window like in the fig. A.5 will be created. In the black message window you can see if the initialization of the application was successful. In menu *File* you can choose what to do.

If you choose *File->Load triangles* in the menu the program will prompt you to enter a file name with triangle mesh set. Then the application executes *SlicesMaker* module and the slices will be computed.

If you choose *File->Load* slices in the menu the program will prompt you to enter file name(s) with set(s) of slices. Than, the program executes immediately the

reconstruction procedure. You will be able to see the reconstructed surface in the well known Renderer module.

PŘÍLOHA B

Installation guide

The *SlicesMaker*, the *SlicesLoader*, the *SlicesViewer* and the *SurfReconstr* module are all stored within the `SlicesModules.dll` library. The modules do not require an special installation. Just to copy it to the directory with the other MVE modules. See the MVE documentation for details.

Application Slices.exe installation

The application uses functions from the `SlicesModules.dll` and `TriangleModules.dll` dynamic linked libraries. Therefore, the application has to be in the same directory with these libraries. `Slices.exe` is the *Win32* application, it does work in Windows 95, Windows 98, Windows NT and Windows 2000.

PŘÍLOHA C

Programmer's guide

The program is written in the C++ language using the MFC library. So the both, the application `Slices` and the dynamic linked library `SlicesModules.dll` are built in Microsoft Visual C++ programming tool.

SlicesMaker implemented model

- `SMaker.h`, `SMaker.cpp` contains class *CSlicesMaker* with methods to computing slices (server class),
- `SPCtrlCentre.h`, `SPCtrlCentre.cpp` contain class *CSPCtrlCentre* with methods for controlling the slices placement (server class). It also has routines for the communication with the user,
- `SSelfViewer.h`, `SSelfViewer.cpp` contain the class *CSelfViewer* with methods for displaying the slices, contours and their points (server class),
- `SMakerInterface.h`, `SMakerInterface.cpp` contain class *CSMakerInterface* representing the *SlicesModules* main window. This is the client class. From class *CSPCtrlCentre* it gets the slices placement and from class *CSlicesMaker* it gets computed slices. To the *CSelfViewer* it puts the computed slices to visualize.

SlicesLoader implemented model

- SLoader.h, SLoader.cpp contains the class *CSLoader* for loading the data stored in files.

SlicesViewer implemented model

- `SViewer.h`, `SViewer.cpp` contains the *CSViewer* class for displaying the slices and the contours

SurfReconstr implemented model

- `SurfReconstr.h`, `SurfReconstr.cpp` contains class *CSurfR* for surface reconstruction (server class)
- `RWizard.h`, `RWizard.cpp` contain class for communication with the user, it gets slices data form previous modules and puts them to the *CSurfR* class to create the surface reconstruction

Common files

- `Common.h`, `Common.cpp` contains class's definitions of the geometry primitives and of the improved memory management for representing dynamic list
- `types.h` contains types definitions common for all MVE modules
- `Triangle_types.h` contains definitions about the structure for giving triangle meshes between MVE modules (*T_Triangle_Mesh*)
- `slcio.h` contains definition of the *SLC* structure

Application Slices

It's very simple program using the functions from DLL libraries. Is linked statically with the MFC functions, to be more portable.