

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

Diplomová práce

Rekonstrukce povrchů geometrických
objektů z roztroušených bodů

Plzeň, 2002

Michal Varnuška

Surface reconstruction of geometrical objects from scattered point data

There is a wide range of applications, such as solid modeling, computer graphics or computer vision, for which surface interpolation or approximation from scattered data points in space is important. Depending on the field of application and related properties of the data, many algorithms were developed in the past. This diploma thesis describes in the theoretical part the most popular of these algorithms and especially two methods of Nina Amenta (the CRUST algorithms, based on the Delaunay triangulation). In the practical part the implementation, the results and problems of the algorithms by Nina Amenta are described.

Poděkování

Na tomto místě bych chtěl poděkovat vedoucí diplomové práce Doc. Dr. Ing. Ivaně Kolingerové za odbornou pomoc, vedení při zpracování této práce a hlavně trpělivost. Dále patří poděkování mým rodičům, příbuzným a přátelům, kteří mě podporovali při studiu.

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

V Plzni dne

.....

Obsah

1.	Úvod.....	7
1.1.	Popis problému.....	7
1.2.	Struktura diplomové práce.....	9
2.	Definice důležitých pojmů.....	10
2.1.	Konvexní obálka.....	10
2.2.	Delaunayova triangulace.....	11
2.3.	Voronoiův diagram.....	12
2.4.	Střední osa („medial axis“)	13
2.5.	Algoritmická složitost.....	14
3.	Existující algoritmy pro rekonstrukci.....	15
3.1.	Dělení prostoru – povrchové metody.....	15
3.1.1.	Algoritmus „Algorri a Schmitt“.....	16
3.1.2.	Algoritmus „Hoppe et al.“.....	16
3.1.3.	Algoritmus „Bajaj, Bernardini et al.“.....	17
3.1.4.	Algoritmus „ α -shape (Edelsbrunner a Mücke)“.....	18
3.1.5.	Algoritmus „Normalizovaná síť (Attali)“.....	19
3.2.	Dělení povrchu – objemové metody.....	21
3.2.1.	Algoritmus „Boissonnat“.....	21
3.2.2.	Algoritmus „Isselhard, Brunett a Schreiber“.....	22
3.2.3.	Algoritmus „ γ -indikátor (Veltkamp)“.....	22
3.2.4.	Algoritmus „Schreiber a Brunett“.....	23
3.3.	Znaménková funkce.....	24
3.3.1.	Algoritmus „Roth a Wibowoo“.....	25
3.3.2.	Algoritmus „rekonstrukce pomocí střední osy (Bittar et al)“.....	25
3.4.	Warping.....	26
3.4.1.	Volná deformace prostoru („Spatial free form warping“)	27
3.4.2.	Algoritmus „Algorri a Schmitt 2“.....	28
3.4.3.	Algoritmus Kohonenova charakteristická mapa („Kohonen feature map“, Baader a Hirzinger).....	28
3.5.	Inkrementální konstrukce povrchu.....	30
3.5.1.	Povrchově orientovaný algoritmus „Boissonnat“.....	30
3.5.2.	Algoritmus „Mencl a Müller“.....	30
3.6.	Clustering.....	32
3.6.1.	Algoritmus „Fua a Sander“.....	32
3.6.2.	Algoritmus „Mencl a Müller“.....	32
3.7.	Další algoritmy.....	33
4.	CRUST algoritmus.....	34
4.1.	Vzorkovací kritérium.....	34
4.2.	Dvouprůchodový „CRUST“ algoritmus.....	35
4.2.1.	2D případ.....	35
4.2.2.	3D případ.....	36
4.2.3.	Odhad normálových vektorů a filtrování.....	38
4.2.4.	Extrakce manifoldu.....	39
4.3.	Jednoprůchodový „CRUST“ algoritmus.....	40
5.	Implementace.....	44
5.1.	Delaunayova tetrahedronizace.....	46
5.2.	Výpočet pólů.....	47
5.3.	Druhá tetrahedronizace.....	49

5.4.	Generování primárního povrchu	50
5.4.1.	Dvouprůchodový algoritmus.....	50
5.4.2.	Jednoprůchodový algoritmus	51
5.5.	Extrakce manifoldu	53
5.5.1.	Pomocné struktury	53
5.5.2.	Vyhledání startovacích trojúhelníků	55
5.5.3.	Vlastní extrakce.....	55
5.6.	Nalezení a triangulace děr.....	58
5.7.	Generování finálního povrchu.....	59
6.	Experimenty a výsledky	60
6.1.	Analýza algoritmické složitosti.....	60
6.2.	Numerická robustnost	61
6.3.	Experimenty	61
6.3.1.	Dvouprůchodový algoritmus.....	61
6.3.2.	Jednoprůchodový algoritmus	64
7.	Závěr	67
7.1.	Seznam literatury	68
8.	Přílohy	71
8.1.	Uživatelská dokumentace samostatného programu	71
8.1.1.	Hlavní okno	72
8.1.2.	Nastavení parametrů výpočtu.....	73
8.1.3.	Nastavení parametrů vizualizace.....	74
8.1.4.	Povolení zobrazování různých dat	75
8.1.5.	Informační okno	75
8.1.6.	Okno o průběhu výpočtu.....	76
8.1.7.	Renderer	77
8.2.	Uživatelská dokumentace MVE modulu	79
8.3.	Výstupy	80

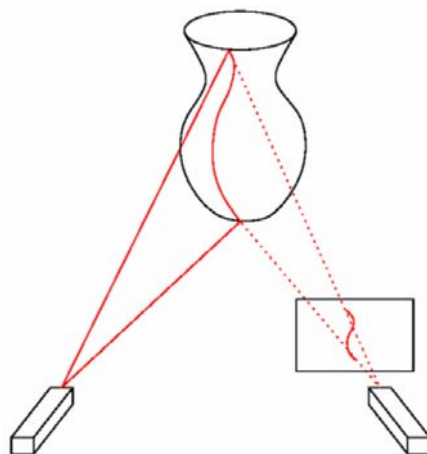
1. Úvod

1.1. Popis problému

Existuje velké množství aplikací v oblasti CAD (Computer Added Design), počítačové grafiky či výpočetní geometrie, pro které je potřebné pracovat s modely reálných objektů.

Častým způsobem získávání modelů těchto objektů je jejich digitalizace pomocí různých zařízení. Tato zařízení se dělí do dvou velkých skupin, bezkontaktní či kontaktní. Pro kontaktní metody se používají různé druhy senzorů. Mezi bezkontaktní patří různé 3D scannery, pracující na principu laserového či sonického scanování.

Na obr. 1-1 je vidět princip bezkontaktní digitalizace pomocí vzdálenostního scanneru [41]. Zdroj produkuje nějaký druh záření na objekt, který je scanován. Na těleso může být záření promítáno buď bod po bodu, tedy se dvěma stupni volnosti (paprsek se pohybuje nahoru a dolů, či doleva a doprava), či jako kontura s jedním stupněm volnosti (kontura se pohybuje doleva a doprava). Senzor, často tvořený CCD prvky, snímá odražené záření z objektu. Software, dodávaný se zařízením, vypočítá v systému souřadnic zařízení vzdálenost k objektu, která se dá ze znalosti pozice zdroje a snímače převést do 3D souřadnice bodu kontaktu paprsku s objektem.



Obr. 1-1 : Princip 3D scanneru (nalevo zdroj, vpravo senzor).

Přístroje pro digitalizaci si můžeme zjednodušeně představit jako černou skříňku, která má na vstupu reálný objekt a na výstupu produkuje množinu 3D bodů.



Obr. 1-2 : Proces digitalizace.

Tuto množinu S můžeme matematicky popsat jako:

$$\forall p \in S : p = [x, y, z]; x, y, z \in R$$

Nyní máme tedy k dispozici množinu bodů, které leží na povrchu digitalizovaného objektu. Mnoha aplikacím ale tento popis nepostačuje, proto je třeba nějakým způsobem spočítat, jak povrch objektu vypadal, a nahradit množinu bodů např. množinou trojúhelníků s určitými vlastnostmi. Je jasné, že tato nová množina bude pouze interpolací či aproximací tvaru původního objektu, protože pro přesný popis bychom potřebovali nekonečně jemné vzorkování.

Právě vzorkování je velice důležité z hlediska výsledné rekonstrukce, přičemž pro její úspěšnost je tato část kritická. Podobně jako se ve spektrální analýze používá Nyquistovo kritérium¹, tak i v oblasti rekonstruování jsou určité požadavky na hustotu a pravidelnost vzorkování. V případě nedodržení těchto požadavků pak nemusí být výsledný model korektní. Mohou v něm např. vznikat díry nebo naopak překrývající se simplex.

Výše popsany proces rekonstrukce se dá jednoduše znázornit tímto schématickým popisem:

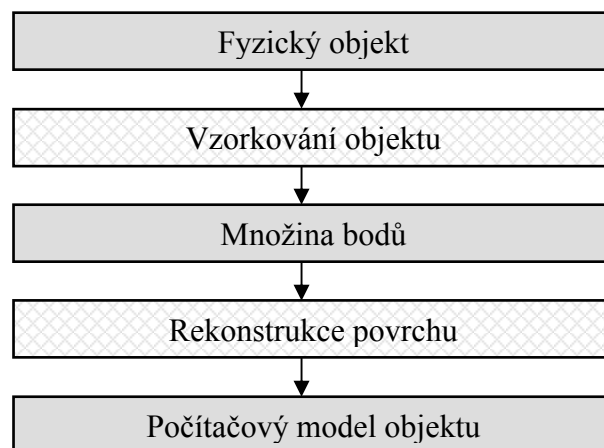


Schéma 1-1 : Převod fyzického objektu do počítačového modelu.

Po získání počítačového modelu objektu bývá pak velice často tento model tvořen sítí trojúhelníků, která se pak může v závislosti na druhu aplikace upravit, např. převodem

¹ Nyquistovo kritérium je maximální frekvence ve frekvenčním spektru frekvenčně omezené funkce (funkce, jejíž amplitudové spektrum je konečné). Tato frekvence se značí f_{max} a vztah vzorkovací frekvence f_s a Nyquistova kritéria f_{max} uvádí Shannovův vzorkovací teorém : signál spojité v čase je plně určen posloupností vzorků odebíraných ve stejných intervalech Δx , je-li jejich frekvence $f_s = 1/\Delta x$ větší než dvojnásobek nejvyšší frekvence v signálu f_{max} .

do parametrických ploch. To je již ale záležitost jiná, tato diplomová práce se zabývá pouze procesem rekonstrukce povrchu. Tuto úlohu můžeme stručně popsat jako relaci:

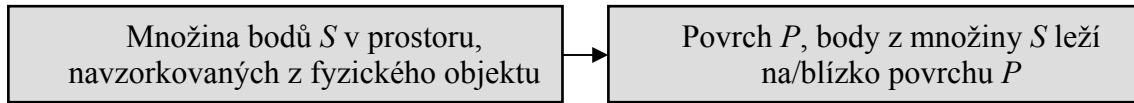


Schéma 1-2 : Relace mezi vstupními a výstupními daty

1.2. Struktura diplomové práce

Text celé práce je rozdělen na několik částí. První částí (teoretickou) je kapitola 2, která obsahuje definici základních pojmů a kapitola 3, kde jsou uvedeny hlavní algoritmy používané pro rekonstrukci. Velkou pomoc při přípravě této kapitoly poskytl článek Mencla a Müllera publikovaný na Eurographics 98 [40].

V další části (kapitola 4) je proveden rozbor dvou algoritmů od Niny Amenty, jednorůchodového a dvouřůchodového „crust“ algoritmu, v třetí části (kapitola 5 a 6) je pak popsána jejich implementace a rozbor výsledků.

Závěr je uveden v kapitole 7. V přílohách je pak uživatelský manuál vytvořeného programu, modulu do MVE a ukázky rekonstruovaných datových množin.

2. Definice důležitých pojmů

V této kapitole jsou popsány pojmy a definice, se kterými budeme pracovat v dalším textu s výjimkou, že definice, vyskytující se pouze u jednotlivých algoritmů, budou popsány přímo v místě použití.

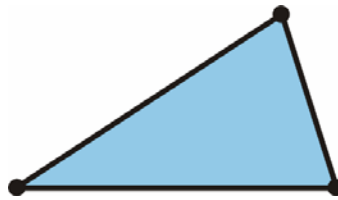
Jak již bylo uvedeno dříve, **vstupní množinu** S můžeme popsat jako množinu bodů v prostoru R^d , která diskrétně popisuje spojitý model. V našem případě je $d=3$ a každý bod p z množiny S bude dán třemi souřadnicemi v prostoru $[x, y, z]$.

$$\forall p \in S : p = [x, y, z]; x, y, z \in R$$

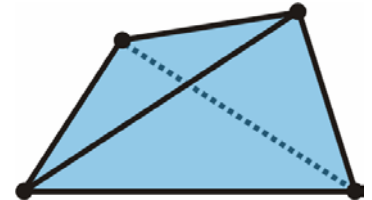
D-Simplex je takový útvar v R^d , pro který platí, že je definován pomocí $d+1$ bodů a sám se nachází v R^d . Z této definice plyne, že pro R^1 bude simplexem úsečka, pro R^2 trojúhelník a pro R^3 tetrahedron.



Obr. 2-1: 1-simplex.



Obr. 2-2: 2-simplex.



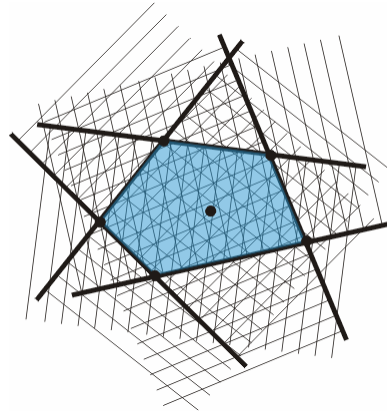
Obr. 2-3: 3-simplex.

2.1. Konvexní obálka

Konvexní obálka $CH(S)^2$ množiny bodů S je průnik všech poloprostorů, které obsahují S . Matematicky je možné jí popsat následovně: množina bodů v R^d je konvexní, pokud pro libovolné 2 body p, q množiny leží úsečka pq zcela uvnitř množiny. Konvexní obálka množiny bodů S v R^d je hranice nejmenší konvexní množiny, která obsahuje S .

$$CH(S) : \{\forall p, q \in S, S \in R^d : \overline{pq} \in A, A = \min\}$$

² Zkratka $CH(S)$ pochází z anglického překladu – convex hull. Někdy se lze setkat s výrazem $Conv(S)$.

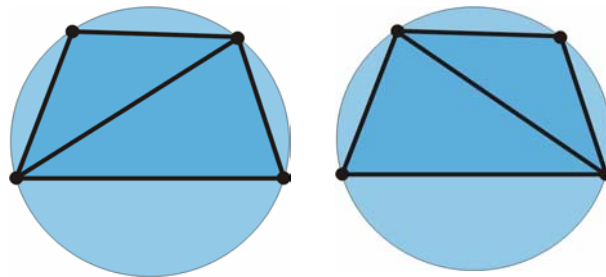


Obr. 2-2 : Konvexní obálka množiny 6 bodů (znázorněná jako průnik polopřímek, konvexní obálka je vyplněná oblast, body jsou v průsečíku hraničních přímek polopřímek a uvnitř).

2.2. Delaunayova triangulace

Obecně úloha triangulace je následující : Necht' je dána množina S bodů v R^d . Spojte body neprotínajícími se úsečkami tak, že každá oblast uvnitř konvexní obálky je d -simplex. Samozřejmě, že takto definovaných triangulací může vzniknout velké množství, nicméně existují určité druhy triangulací, kdy vzniklé simplexy splňují jistá kritéria. Často používaný druh triangulace se nazývá **Delaunayova³ triangulace** (pro R^3 se někdy používá název **Delaunayova tetrahedronizace**) a značí se $DT(S)$.

Hlavní výhodou této triangulace je, že maximalizuje minimální úhel. Laicky řečeno, vzniklé simplexy jsou co možná nejrovnostřednější. Při vytváření se používá kritérium opsané kružnice⁴, které říká, že simplex splňuje kritérium maximálního minimálního úhlu, pokud uvnitř kružnice opsané simplexu není další bod. Toto již přináší první problém. Stačí si představit $d+2$ bodů (a více), které budou ležet na kružnici. Potom triangulace v této části nebude jednoznačná (viz obr. 2-5).



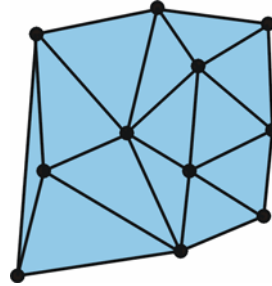
Obr. 2-3 : Nejednoznačnost Delaunayovy triangulace.

Shrnutí hlavních vlastností Delaunayovy triangulace :

³ Název Delaunayova triangulace je po jejím tvůrci, ruském matematikovi, jehož jméno bylo Boris Delaunay, který ji popsal poprvé v roce 1934.

⁴ Pro dvourozměrný případ je to kružnice opsaná, pro třírozměrný pak koule opsaná. V anglické literatuře se používají pojmy circumcircle a circumsphere. Delaunayova triangulace je rozšiřitelná i do větších dimenzí a toto kritérium samozřejmě platí stále. Otázkou je, jak se nazývá kružnice či koule v dalších dimenzích. Proto používám pojem kružnice i pro větší dimenze.

- Minimalizuje poloměr kružnice opsané a maximalizuje minimální úhel (ačkoliv neminimalizuje maximální úhel), tj. negeneruje příliš „hubené“ trojúhelníky, které mohou být problematické při zobrazování a dalším zpracování.
- Pokud žádné čtyři body neleží na kružnici, je DT jednoznačná (viz obr. 2-6).
- Hranice $DT(S)$ je $CH(S)$.



Obr. 2-4 : Ukázka Delaunayovy triangulace množiny 11 bodů.

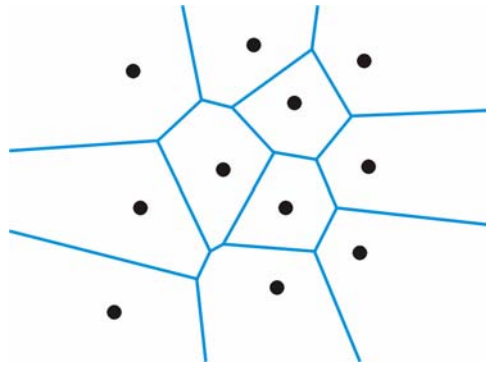
2.3. Voronoiov diagram

Pro každý bod $p \in S$ definujeme $V(p)$ jako **Voronoiovu buňku** bodu p . Pro všechny body $x \in R^d$ pak platí, že pokud se nacházejí v této buňce příslušející bodu p , pak eukleidovská vzdálenost bodu x a bodu p je menší nebo rovna vzdálenosti od x k jakémukoliv jinému bodu z množiny S . Matematicky se dá definovat takto :

$$V(p) = \{x : |p - x| \leq |p_j - x|, \forall x \in R^d, \forall p_j \neq p, p \in S, p_j \in S\}$$

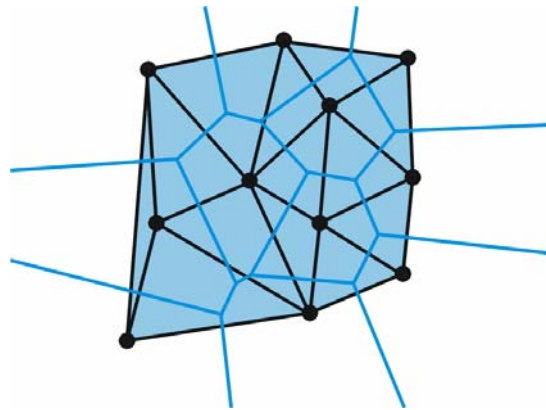
Množina všech Voronoiových buněk (tedy buněk každého bodu z množiny S) pak definují **Voronoiov diagram** $VD(S)$. Vlastnosti Voronoiova diagramu (příklad obr. 2-7) a jeho buněk jsou následující :

- Každá buňka je konvexní (v R^3 je to tedy konvexní polyhedron)
- V každé buňce se nachází jen jeden bod.
- Buňky ležící na $CH(S)$ mohou být neuzavřené.
- Pokud žádné $d+2$ body neleží na kružnici, mají uzly stupeň $d+1$.
- Uzel diagramu je geometrické místo, ke kterému mají stejnou vzdálenost nejméně $d+1$ body.
- Hrana diagramu je geometrické místo, ke kterému mají stejnou vzdálenost d bodů.
- Je-li p_i nejbližší soused p_j , pak tyto body mají společnou hranu ve Voronoiově diagramu.



Obr. 2-5 : Voronoiův diagram množiny 11 bodů.

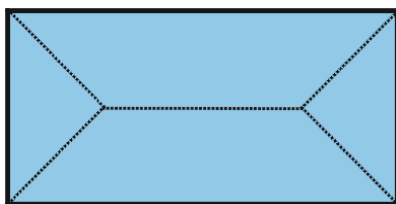
Další výhodnou vlastností VD je, že je duální k Delaunayově triangulaci. Body budou sdílet společnou hranu (stěnu) v triangulaci, pokud jejich Voronoiovy buňky sdílejí společnou hranu v VD . Obrázek 2-8 pak ukazuje na společném příkladu tuto dualitu (též obr. 2-7 a 2-6).



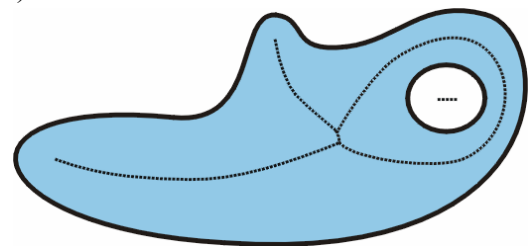
Obr. 2-6 - Dualita Voronoiova diagramu a Delaunayovy triangulace.

2.4. Střední osa („medial axis“)

Střední osa $d-1$ dimenzionálního povrchu v R^d je množina bodů, které mají na povrchu více než jeden nejbližší bod (viz obr. 2-9 a 2-10).



Obr. 2-8: Střední osa obdélníka.



Obr. 2-8 : Střední osa hladkého objektu.

Pokud se blíže podíváme na obrázek 2-9, zjistíme, že v případě ostrých vrcholů prochází střední osa těmito vrcholy (ostrý vrchol je místo, kde existuje pouze C^0 spojitost).

2.5. Algoritmická složitost

Algoritmická složitost algoritmu [URL5] vypovídá o rychlosti růstu časové spotřeby algoritmu vzhledem k velikosti N vstupní množiny. To, co se bere za jeden prvek $n_i \in N$ vstupní množiny, závisí na typu úlohy. Pro rekonstrukci povrchů z roztroušených bodů platí, že prvek množiny je jeden bod.

Pokud by bylo možné zpracovat všechny možné vstupy (permutací prvků vstupní množiny) algoritmu pro určité N , můžeme stanovit maximální, minimální a průměrný čas potřebný ke zpracování úlohy. Funkce $f_{max}(N)$, která přiřazuje maximální čas pro dané N , se nazývá složitost algoritmu pro nejhorší případ, funkce $f_{min}(N)$, která naopak přiřazuje minimální čas, se nazývá složitost algoritmu pro nejlepší případ.

Abychom ale byli schopni určit složitost algoritmu pro jakékoliv N , musíme použít asymptotické verze těchto funkcí. To znamená, že budeme předpokládat, že N roste k nekonečnu a že existuje taková limita, pro kterou platí

$$\lim_{N \rightarrow \infty} f_{max}(N) = O(c + g(N)) = O(g(N))$$

Konstanta c omezuje funkci shora, ale pro tuto notaci je irelevantní, protože její zanedbání absorbuje rozdíly mezi druhy počítačů a implementační detaily. Z toho vyplývá, že výsledek této analýzy algoritmu má úplnou platnost pro dostatečně vysoká N , proto se tato metodologie nazývá **asymptotická složitost algoritmu**. Konstanta c je ale ještě příčinou jednoho jevu. Pokud budeme mít dva různé algoritmy s různou časovou složitostí, pak může platit, že algoritmus s větší složitostí bude pro malá N rychlejší než algoritmus s menší složitostí. Je to díky tomu, že konstanty se mohou lišit.

Stejně tak můžeme postupovat i pro funkci $f_{min}(N)$, jejíž asymptotická složitost se značí $\Omega(N)$. Celková složitost je pak omezena těmito dvěma funkcemi. Někdy se uvádí též značení $\Theta(N)$, které se používá pro očekávanou složitost. Tyto algoritmy jsou omezené funkcemi $c_1 f(N) \leq f_{avg}(N) \leq c_2 f(N)$ zdola i shora. Nicméně analýza této složitosti je poměrně obtížná, protože je třeba zvolit pravděpodobnostní rozložení odpovídající řešenému problému, což je ne vždy jednoznačné. Proto se často používá odhad na základě implementace.

3. Existující algoritmy pro rekonstrukci

Problém rekonstrukce je stále předmětem mnoha výzkumů. Bylo vyvinuto mnoho metod, které mohou být rozděleny do několika tříd (dělení není ostré, mnoho metod má vlastnosti několika tříd současně) a jsou založeny na:

- Dělení prostoru („*spatial subdivision*“)
- Znaménkové funkce („*distance function*“)
- Warpingu
- Inkrementální konstrukci povrchu („*incremental surface construction*“)

Následující stránky pak uvádějí stručný přehled nejdůležitějších metod a algoritmů. Jelikož většinou nemají své názvy, označuji je jmény jejich autorů tak, jak jsou uváděny v referencích na ně (obrázky u algoritmů jsou převzaty z článků autorů).

3.1. Dělení prostoru – povrchové metody

Základní vlastností algoritmů, které patří do této třídy, je, že ohraničující obálka (jakákoliv) vstupní množiny bodů S je rozdělena do nezávislých podprostorů. Existuje několik postupů pro různé aplikace, jak prostor rozdělit. Typickým příkladem je dělení pravidelnou mřížkou nebo adaptivní dělení pomocí „octree“ či nepravidelné dělení tetrahedrony. Některé z těchto dělení lze využít i pro řešení problému rekonstrukce. V tomto případě pak po rozdělení prostoru najdeme buňky, které se vztahují k povrchu popsaném vstupní množinou S a z těchto buněk extrahujeme povrch.

Výběr buněk může být proveden dvěma způsoby, buď se budeme orientovat na hledání povrchu⁵ nebo objemu. Základní koncepce povrchových metod je ukázána na schématu 3-1 :

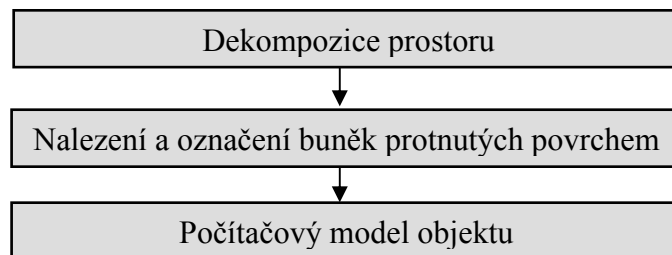


Schéma 3-1 : Koncepce rekonstrukce pomocí dělení prostoru (povrchové metody).

⁵ V anglické literatuře nazývané jako „surface based“ a „volume based methods“

3.1.1. Algoritmus „Algorri a Schmitt“

V prvním kroku tohoto algoritmu [1] je nalezena obálka tvaru kvádrů kolem vstupních bodů a prostor uvnitř je rozdělen do pravidelné pravoúhlé mřížky (voxelová mřížka). Ve druhém kroku jsou extrahovány pouze ty voxely, které obsahují nejméně jeden bod ze vstupní množiny. Třetí krok pak rozdělí voxely na čtyřstěny a vybere pouze ty čtyřstěny, které obsahují bod, jako první aproximaci povrchu.

Pokud je třeba vytvořit přesnější reprezentaci povrchu, pak se povrch, nyní reprezentovaný množinou tetrahedronů, převede do trojúhelníkové sítě rozdělením každého tetrahedronu na dva trojúhelníky. Tato síť je pak vyhlazena „**depth-pass**“ filtrem, který přiřazuje novou pozici každému vrcholu trojúhelníku (původní tetrahedron měl vstupní bod ve svém objemu a ne na povrchu, proto se nyní musí dopočítat pozice trojúhelníků tak, aby procházely bodem). Tato pozice je spočítána jako vážený průměr jeho staré pozice a pozice sousedů.



Obr. 3-1 : Kroky algoritmu (zleva – množina vstupních bodů, extrahované voxely, první aproximace povrchu, výsledná trojúhelníková síť).

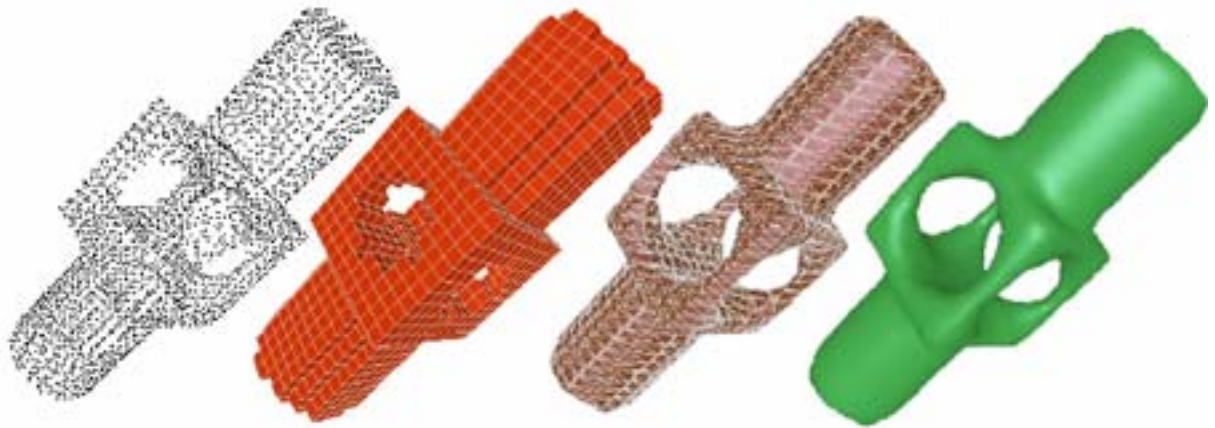
3.1.2. Algoritmus „Hoppe et al.“

Další možnost rekonstrukce [2] [3] [4] je založena na kombinaci metody hledání povrchu a vyhodnocení znaménkové vzdálenostní funkce definované pro každý bod. Tato funkce nabývá kladné, záporné či nulové hodnoty pro body nacházející se vně, uvnitř či na hranici objektu [2]⁶.

V prvním kroku je opět vytvořeno pravidelné dělení prostoru na voxely jako v předchozím algoritmu. Další krok pak vybere pouze ty voxely, které mají na opačných stranách různou hodnotu znaménkové funkce. Díky tomuto kritériu odpadne velké množství pro další zpracování nepotřebných voxelů, protože povrch musí procházet právě těmito vybranými buňkami.

Po tomto vyhodnocení je použit jako třetí krok standardní algoritmus „**Marching cubes**“ [5] pro získání povrchu nabývajícího nulové funkční hodnoty. Nevýhodou tohoto postupu je, že původní body nemusí procházet povrchem a tedy získaný model je pouze aproximací povrchu. Navíc dochází k zahlazení ostrých hran, což nemusí být vždy žádoucí. Odstranění těchto nevýhod bylo dosaženo teprve za cenu zvýšených výpočetních nároků [3] a později i vylepšeným generováním povrchu [4].

⁶ Výpočet této funkce je ukázán v kapitole 3.3.



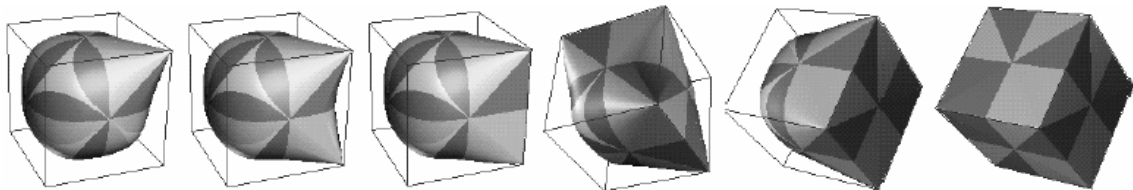
Obr. 3-2 : Kroky algoritmu (zleva – vstupní množina bodů, voxelizace prostoru, síť získaná aplikací alg. Marching cubes, výsledný povrch).

Hlavní výhody tohoto algoritmu jsou, že dokáže zpracovat velké vstupní množiny, není příliš citlivý na šum v datech a dá se u něj kontrolovat složitost výstupu pomocí zvětšování či zmenšování velikosti voxelové mřížky. Na druhou stranu, právě tato mřížka způsobuje jeho největší nevýhodu, algoritmus není adaptivní na měnící se hustotu vzorkování. Další nevýhoda pak souvisí s metodou Marching cubes, která může produkovat trojúhelníky s nevyhovujícím tvarem.

3.1.3. Algoritmus „Bajaj, Bernardini et al.“

Společnou vlastností předchozího a tohoto algoritmu [6] je využití stejné znaménkové funkce. Hlavním rozdílem je přístup k dělení prostoru, které je nyní adaptivní a tedy nepravidelné. Navíc je první aproximace povrchu spočtena již během fáze preprocessingu pomocí tzv. „ α -solid“ (zatímco „ α -shape“ (viz 3.1.4) je spočten za pomoci tzv. mazací koule na každém vstupním bodě, tato mazací koule se nyní aplikuje z vnějšku konvexní obálky a získáme tím vnější obrys tělesa)

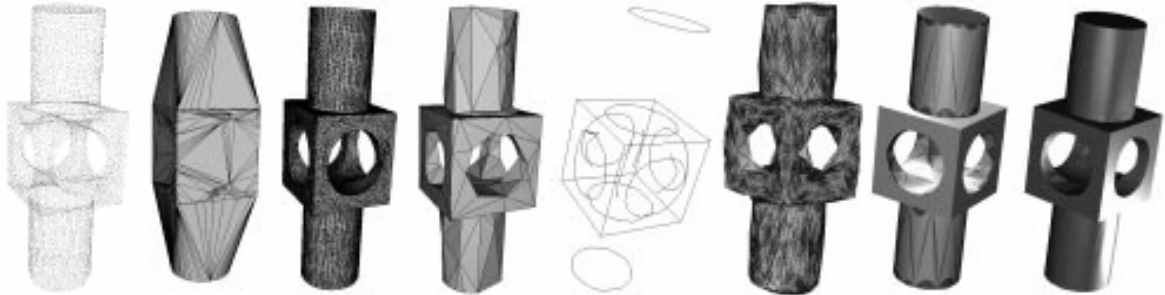
S pomocí znaménkové funkce je prostor inkrementálně rozdělen do tetrahedronů, přičemž první tetrahedron je vybrán tak, aby obklopoval všechny vstupní body. Kontrolou znaménka znaménkové funkce bodů získáme tetrahedrony, které leží na povrchu. Pro každý z nich je pak spočítána aproximace povrchu, který tetrahedronem prochází. Navíc je ale spočtena ještě chyba aproximace, která je využita v dalším kroku. Pokud je totiž chyba příliš velká, pak se vloží ještě střed tetrahedronu (definovaný jako střed koule opsané) do již vytvořené tetrahedronizace, přičemž rekurzivně opakujeme tento postup, dokud není chyba aproximace zanedbatelná. Výsledný povrch je pak ještě převeden na Beziérovky plochy (s C^1 spojitostí).



Obr. 3-3 : Modelování jednoduchou α -záplatou (zleva – interpolace vrcholu, interpolace dvou vrcholů, interpolace hrany, interpolace hrany a vrcholu, interpolace stěny, α -záplata degenerovaná na krychli).

Co je ještě důležité říci, je, že vkládané tetrahedrony musí splňovat Delaunayovu podmínku, tedy každý vložený tetrahedron musí mít prázdný vnitřek koule tomuto tetrahedronu opsané (bez dalších bodů).

Bernardini pak tento algoritmus dále rozšířil. Protože „ α -solid“ bývá poměrně složitý, zahrnul do vlastního algoritmu i jeho redukci. Ta je provedena těsně před poslední fází algoritmu, tedy před převodem na Beziérové plochy.



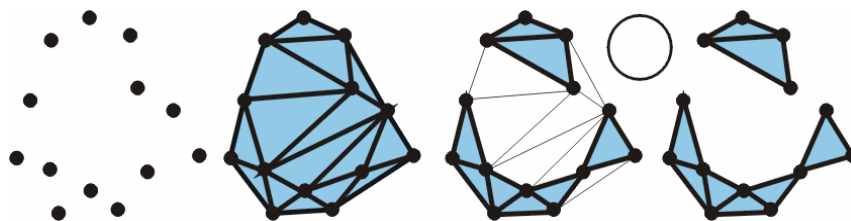
Obr. 3-4 : Alg.“Bajaj, Bernardini“ (zleva – vstupní množina, Delaunayova triangulace, α -solid, zjednodušená síť, dále pokračuje proces převodu na parametrické plochy - vyhledané hrany, vylepšená síť, α -záplaty, rekonstruovaný model).

3.1.4. Algoritmus „ α -shape (Edelsbrunner a Mücke)“

Tento algoritmus [7] [8] [9] dělí prostor adaptivně stejně jako v předchozím případě. Dekompozice je docílena pomocí Delaunayovy tetrahedronizace. Tetrahedronizace množiny bodů v prostoru je vlastně dekompozice konvexní obálky množiny do tetrahedronů, které mají ve svých vrcholech tyto body.

Druhým krokem je pak smazání tetrahedronů, trojúhelníků a hran z tetrahedronizace použitím tzv. „ α -balls“ (koule o poloměru α , tento parametr je dalším vstupem metody). Každý tetrahedron, trojúhelník nebo hrana je smazán v případě, že poloměr nejmenší možné koule opsané je větší než poloměr „ α -ball“. Výsledek je pak nazván „ α -shape“ a je tvořen souborem tetrahedronů, trojúhelníků a hran. Pokud bude parametr α roven nekonečnu, pak „ α -shape“ bude obsahovat celou tetrahedronizaci, pokud naopak tento parametr bude roven 0, pak výstupem budou pouze vstupní body.

Jako poslední krok se pak provede extrakce trojúhelníků z „ α -shape“ pomocí následující heuristiky : trojúhelník náleží povrchu v případě, že nejméně jeden ze dvou „ α -balls“ interpolující trojúhelník neobsahuje žádné další body ze vstupní množiny.



Obr. 3-5 : Kroky algoritmu „ α -shape“ v 2D (zleva – vstupní množina bodů, Delaunayova triangulace, trojúhelníky smazané α -kružnicí, hrany smazané za pomoci dvoubodové podmínky). Velkým problémem této metody je, že parametr α musí být určen experimentálně. Pokud je příliš malý, pak výsledný povrch je málo odlišný od konvexní obálky, naopak

pokud je příliš velký, pak je povrch fragmentován do mnoha oddělených částí. Algoritmus není navíc vhodný pro množiny, jejichž hustota vzorkování se mění.

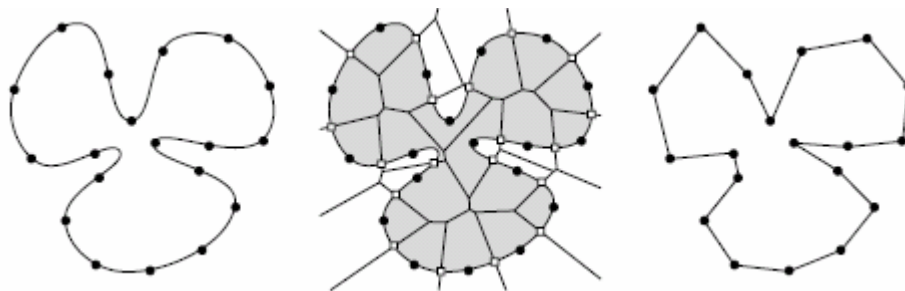


Obr. 3-6 : Kroky algoritmu „ α -shape“ v 2D. (zleva –Delaunayova tetrahedronizace vstupní množiny bodů, dále ukázka vlivu velikosti parametru α na rekonstrukci pomocí koule znázorňující velikost parametru α (část koule vždy napravo od obrázku).

3.1.5. Algoritmus „Normalizovaná síť (Attali)“

K dělení prostoru v tomto algoritmu [10] se opět používá Delaunayova triangulace. Dále se používá tzv. **normalizovaná síť**, která je obsažena v Delaunayově triangulaci. Je tvořena hranami, stěnami a tetrahedrony, jejichž duální Voronoiovy elementy protínají povrch tělesa.

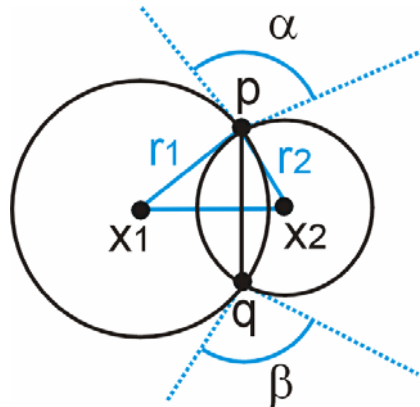
Ve dvourozměrném případě se normalizovaná síť křivky c skládá ze všech hran mezi dvojicemi bodů dané vstupní množiny bodů (vzorkující křivku c), které představují hranu Voronoiova diagramu protínající křivku c . Dobrou vlastností (nazývá se **r-regularita**) normalizované sítě je, že pro širokou třídu křivek si normalizovaná síť uchovává topologické vlastnosti originální křivky.



Obr. 3-7 : Rekonstrukce křivky (zleva : křivka s navzorkovanými body, normalizovaná síť, rekonstruovaná křivka)

Pro rekonstrukci křivky c jsou hrany příslušející rekonstruované síti zkoumány v závislosti na úhlu mezi průsečíky dvou možných kružnic okolo Delaunayovy hrany.

Úhel mezi kružnicemi je definován jako menší z úhlů mezi průsečíky dvou tečných rovin v průsečíku kružnic (kružnice mají dva průsečíky, tedy menší z dvou úhlů). Tato charakteristika je užitečná, protože Delaunayova kružnice má sklon být na hranici objektu tečnou. Rekonstruovaná síť se skládá ze všech hran, pro které platí, že příslušející Delaunayovy kružnice mají úhel menší $\pi/2$ (viz obr. 3-8). Pokud je vzorkovací frekvence dostatečně vysoká, pak rekonstruovaná síť je identická s normalizovanou sítí.



Obr. 3-8 : Výpočet úhlu kružnic. Hrana pq je z Delaunayovy triangulace, body x_1, x_2 jsou středy Delaunayovských kružnic, r_1, r_2 jejich poloměry. Úhly α a β jsou dva možné úhly mezi kružnicemi.

Jestliže pro dvourozměrný případ platí, že normalizovaná síť je korektní rekonstrukcí křivek tvořící obrys mající vlastnost **r-regularity**, přímé rozšíření do třetího rozměru není možné. Důvod je ten, že některé Delaunayovy koule mohou protínat povrch bez toho, aby byly na povrchu tangenciální. Proto může obsahovat normalizovaná síť ve třech dimenzích i díry.



Obr. 3-9 : Množina vstupních bodů a rekonstruovaný povrch

Pro překonání tohoto problému se používají dvě heuristiky pro filtraci. První je triangulace hranic děr v trojúhelníkové síti, pokud budeme brát v potaz pouze trojúhelníky obsažené v Delaunayovy tetrahedronizaci.

Druhá heuristika je založena objemově. Slučuje tetrahedrony a vytváří různé objemové reprezentace vstupní množiny dat. Množiny sloučených objektů jsou inicializovány všemi tetrahedrony a doplnkem konvexní obálky. Krok slučování je prováděn zpracováním Delaunayovských trojúhelníků se zmenšujícím se poloměrem. Pokud aktuální trojúhelník rozděluje dvě odlišné množiny, tak jsou sloučeny, pokud splňují následující podmínky:

- nezmizí žádný trojúhelník z normalizované sítě
- slučování nezpůsobí izolaci bodů uvnitř sjednocení těchto množin

Výsledný povrch je pak tvořen hranicí výsledných množin.

3.2. Dělení povrchu – objemové metody

Tyto algoritmy také obsahují hlavní tři kroky (viz schéma 3-2), které jsou podobné povrchovým metodám. Mnoho z těchto objemově orientovaných metod je založeno na Delaunayově tetrahedronizaci vstupní množiny bodů S . Některé metody eliminují tetrahedrony nacházející se vně předpokládaného tělesa, jiné zase používají Voronoiův diagram pro popsání výsledného tělesa pomocí skeletonu.

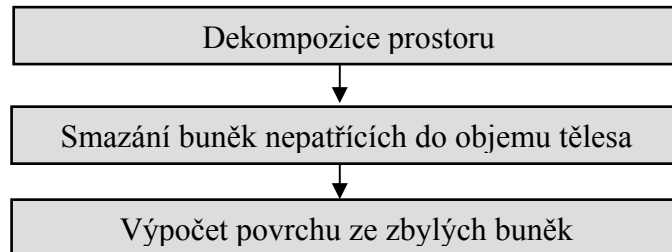


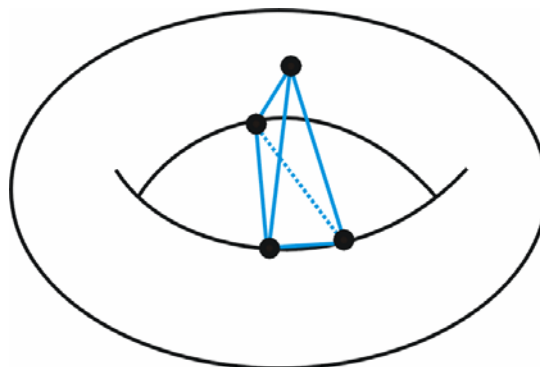
Schéma 3-2 : Koncepce rekonstrukce pomocí dělení prostoru (objemové metody)

Hlavním rozdílem od povrchových metod je ten, že je získána objemová reprezentace na rozdíl od povrchové.

3.2.1. Algoritmus „Boissonnat“

Prvním krokem algoritmu [12] je opět Delaunayova triangulace množiny vstupních bodů. Pak jsou smazány z triangulace konvexní obálky tetrahedrony, které mají pouze dvě strany, pět hran a čtyři body, nebo jednu stranu, tři hrany a tři body. Díky tomuto eliminačnímu pravidlu je tento algoritmus použitelný pouze pro povrchy bez děr.

Pro pomoc při rozhodování, který tetrahedron smazat, se používá pomocná hodnota, která je vypočítána jako maximální vzdálenost stran tetrahedronu ke středu koule opsané. Je velice vhodná, protože ploché tetrahedrony z tetrahedronizace mají sklon být vně tělesa a v oblastech s vysokým detailem. Algoritmus zastaví v případě, že všechny body leží na povrchu nebo že smazání tetrahedronu s nejvyšší rozhodovací hodnotou výrazně nezlepší součet přes všechny tetrahedrony incidující s hranicí polyhedronu.



Obr. 3-10 : Příklad tetrahedronu, který nemůže být odstraněn eliminačním pravidlem.

3.2.2. Algoritmus „Isselhard, Brunett a Schreiber“

Nevýhodou předchozí metody je, že neumí pracovat s objekty obsahujícími díry. Proto byla tato metoda upravena a vylepšena [13]. První krok je stejný, tedy Delaunayova triangulace, v dalším kroku je však upraveno eliminační pravidlo. Eliminovány jsou navíc tetrahedrony s jednou stranou a čtyřmi body, třemi stranami, s čtyřmi stranami v případě, že žádný bod nezůstane izolován.

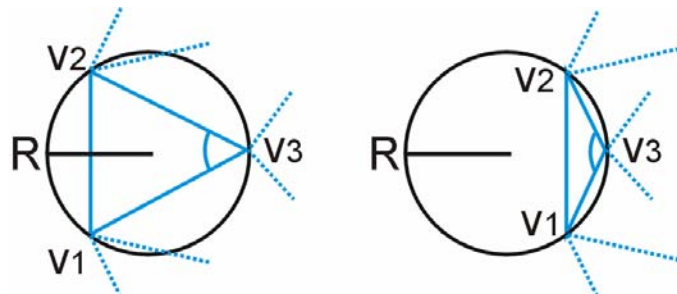
Eliminační proces je kontrolován pomocí eliminační funkce, která je definována jako maximální hodnota rozhodovací funkce zůstávajících odstranitelných tetrahedronů. Pozorováním této funkce si můžeme povšimnout skokových změn. Ty jsou patrné např. v případě, že bylo dosaženo požadovaného tvaru.

Algoritmus zastaví v případě, že bylo dosaženo ve všech bodech povrchu. Pokud se tak nestalo, pak jsou odstraněny další tetrahedrony pro získání ostrých rohů. Pro tento případ je pak eliminační pravidlo použito v původní podobě i za cenu, že nebudou odhaleny díry v tělese.

3.2.3. Algoritmus „ γ -indikátor (Veltkamp)“

Pro popis této metody [14][15] je třeba vysvětlit si pár pojmů. Už v názvu metody se používá výraz **γ -indikátor**, což je hodnota asociovaná s koulí procházející třemi body polyhedronu, přičemž **γ -indikátor** může nabývat jak kladných, tak záporných hodnot. Jeho absolutní hodnota je spočtena jako $1 - r/R$, kde r je poloměr kružnice opsané těmto třem bodům a R je poloměr hraničního tetrahedronu (koule opsané). Bude mít zápornou hodnotu, pokud střed koule je uvnitř a kladnou, pokud střed je vně polyhedronu. Je třeba poznamenat, že **γ -indikátor** je nezávislý na velikosti hraničního trojúhelníku (či tetrahedronu). Proto je adaptivní k oblastem, kde se mění hustota vstupní množiny bodů. Odstranitelná stěna je stěna s kladným **γ -indikátorem**.

Prvním krokem je spočtení Delaunayovy triangulace. V druhém kroku vytvoříme množinu, kde budou obsaženy všechny odstranitelné tetrahedrony, přičemž si je uložíme v seřazeném pořadí jejich **γ -indikátoru**. Odstranitelné tetrahedrony jsou stejného typu jako u Boissonatova algoritmu. Nejprve je odstraněn tetrahedron s největším **γ -indikátorem** a je upravena hranice. Tento proces probíhá do té doby, dokud všechny body neleží na hranici, nebo dokud není prázdná množina odstranitelných tetrahedronů.



Obr. 3-11 : Dvě ukázky γ -indikátoru pro dvourozměrný případ.

Hlavním kladem tohoto algoritmu je, že je adaptivní vzhledem k hustotě vzorkování. Nevýhodou, stejně jako u Boissonatova algoritmu, že neumí zpracovat objekty s dírami.



Obr. 3-12 : Průběh algoritmu „ γ -indikátor“ (zleva : konvexní obálka, dvě mezifáze algoritmu, výsledný povrch).

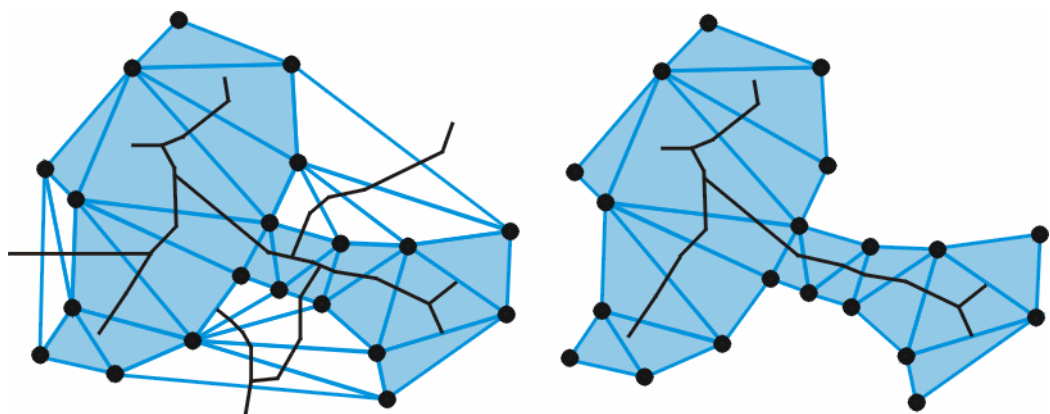
3.2.4. Algoritmus „Schreiber a Brunett“

Tento algoritmus [16] využívá na rozdíl od předchozích Voronoiův diagram⁷. Algoritmus je zde vysvětlen pro dvourozměrný případ, přičemž zobecnění do třetího rozměru je jednoduché. První krok spočívá ve spočtení tohoto diagramu (či Delaunayovy triangulace a z ní se získá s využitím duality Voronoiův diagram). Z tohoto diagramu se spočte MST (minimum spanning tree), což je podgraf diagramu, pro který platí, že obsahuje všechny uzly diagramu (Voronoiovy uzly) a délka tohoto grafu je nejmenší ze všech možných podgrafů diagramu.

Ve druhém kroku je aplikována na tento podgraf strategie prořezávání stromu, která rozdělí strom do mnoha podstromů reprezentujících region definovaný disjunktní množinou trojúhelníků duální k uzlům podstromů. Pravidla prořezávání jsou následující:

- budou odříznuty všechny hrany, pro které koncový bod není obsažen v kružnici opsané duálnímu trojúhelníku u druhého koncového bodu
- hrana bude odstraněna v případě, že její délka je kratší než střední hodnota poloměru obou kružnic opsaných duálnímu trojúhelníku

, přičemž počet oříznutých hran může být kontrolován s použitím délky hrany jako parametru.



Obr. 3-13 : Vlevo je Delaunayova triangulace polygonu (vyplněný) společně s konvexní obálkou a MST (černá čára), vpravo rekonstruovaný polygon po klasifikaci na vnější a vnitřní polygony.

⁷ Což není na druhou stranu zas tak velký rozdíl, jelikož Voronoiův diagram je duální k Delaunayově triangulaci (či tetrahedronizaci).

Výsledné regiony jsou rozděleny na vnější a vnitřní. Právě pro potřeby nalezení vnitřních regionů je ještě do množiny regionů přidána konvexní obálka. Na ní se začíná s klasifikací, protože pokud si vezmeme region mající svou část na konvexní obálce, pak právě ze znalosti konvexní obálky můžeme zjistit jeho polohu a pokračovat pak podobně na jeho sousedy.

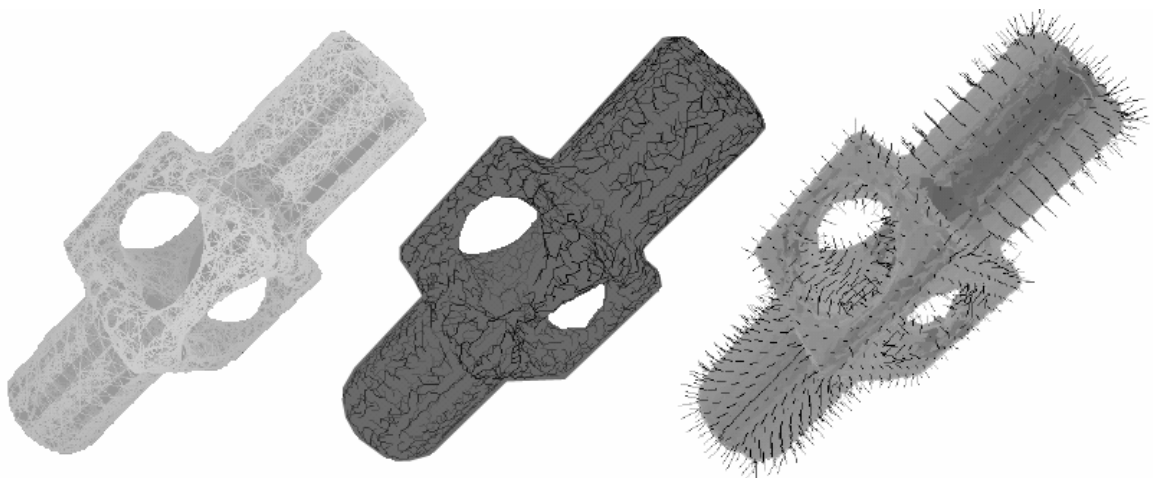
3.3. Znaménková funkce

Tato funkce popisuje nejkratší vzdálenost od bodu k povrchu. Pro uzavřená tělesa může být vzdálenost záporná či kladná v závislosti na tom, jestli bod leží uvnitř či vně tělesa. V předchozích algoritmech se tato funkce používala jako pomocná společně s dělením prostoru a spočítala se následujícím postupem.

Na začátku se spočte pro každý bod p_i vstupní množiny S odhad tečné roviny. Ta je získána tak, že se proloží rovina metodou nejmenších čtverců nejbližšími k sousedy bodu p_i . Protože potřebujeme zjistit také znaménko funkce vzdálenosti v případě uzavřeného povrchu, musíme roviny orientovat stejným směrem za použití **Riemannianova grafu**. Uzly tohoto grafu jsou umístěny ve středu centroidů k bodů použitých pro výpočet tečných rovin. Dva uzly i, j grafu jsou spojeny hranou (i, j) tehdy, pokud jeden uzel je v k -sousedství druhého uzlu. Tyto hrany se po tomto postupu výpočtu nacházejí blízko povrchu tělesa. Každé hraně je ještě přiřazena váha jako jedna mínus absolutní hodnota skalárního součinu dvou normál v koncových uzlech hrany (definované jako normála tečné roviny, ze které tento uzel vznikl). Orientace rovin je zajištěna tak, že se vezme orientace první (jakékoliv) roviny a procházením **EMST** výsledného **Riemannianova grafu** se podle ní orientují další roviny.

S použitím první aproximace povrchu pomocí tečných rovin a jejich správné orientace je spočtena vzdálenost (se znaménkem) bodu a nejbližšího středu tečné roviny (uzel grafu použitý pro výpočet tečné roviny).

Tento výpočet není jediným způsobem, jak spočítat znaménkovou vzdálenostní funkci. Další možností je zkonstruovat nějakým způsobem hrubý povrch tělesa a vzít jako tuto funkci vzdálenost bodů a povrchu, přičemž tento způsob se používá pro získání lepší aproximace povrchu.



Obr. 3-14 : zleva Riemannianův graf středů centroidů tečných rovin , EMST středů centroidů tečných rovin, znaménková funkce (znázorněna pomocí vektorů)

3.3.1. Algoritmus „Roth a Wibowoo“

Tento algoritmus [17] počítá vzdálenost v jednotlivých bodech od dané voxelové mřížky. V těchto bodech jsou navíc spočítány normály z tělesa tak, že nalezneme dva nejbližší body mřížky a pomocí nich normálu spočteme. Jejich orientace, která je potřeba pro výpočet vzdálenostní funkce, se dá zjistit následovně. Představme si voxelovou mřížku a šest základních směrů, které v ní existují – $\pm x$, $\pm y$, $\pm z$. Pokud se podíváme z velké vzdálenosti podél každého směru, potom voxely, které uvidíme, musí být na povrchu tělesa a orientace jejich normál je proti směru, kterým se díváme. Normály jsou tímto způsobem určeny pro body viditelné na povrchu, pro další body jsou pak určeny díky sousednosti. Tato heuristika funguje pouze v případě, že rekonstruujeme těleso bez děr.

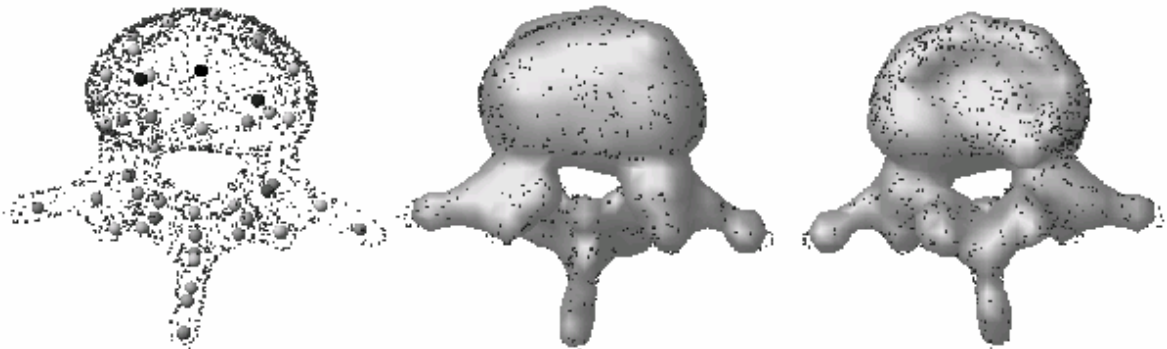
Hodnota znaménkové vzdálenostní funkce v bodě voxelové mřížky je spočtena váženým průměrem ze znaménkových vzdáleností každého bodu z 8-okolí voxelu. Znaménková vzdálenost k bodu s normálou je eukleidovská vzdálenost do tohoto bodu s kladným znaménkem v případě, že úhel mezi normálou a vektorem do vrcholu voxelu přesáhne 90 stupňů.

3.3.2. Algoritmus „rekonstrukce pomocí střední osy (Bittar et al)“

Metoda [18] se skládá ze dvou kroků, výpočtu střední osy a výpočtu implicitního povrchu. Střední osa se spočítá z voxelizace dané množiny bodů. Voxely obsahující body dané množiny jsou pak na hranicích rekonstruovaného tělesa. Eliminaci voxelů začneme na krajních voxelích a postupujeme až do té doby, dokud voxely neobsahují zadané body. Pak zbylé voxely budou buď na povrchu tělesa nebo uvnitř tělesa. Vzdálenostní funkce je potom propagována z hraničních voxelů směrem dovnitř objemu, přičemž na hranicích bude mít tato funkce nulovou hodnotu. Voxely s největší hodnotou budou příslušet střední ose.

Povrch tělesa je spočten pomocí distribuce středů koulí po střední ose. Poloměr koule je roven vzdálenosti přiřazené jejímu středu na střední ose. Pro každou kouli je definována funkce („**field function**“), která dovoluje spočítat skalární pole pro každý bod v prostoru. Hodnota funkce pro celou množinu koulí je získána jako suma této funkce pro každou kouli. Implicitní povrch je definován jako izoplocha „**field**“ funkce, obsahuje tedy všechny body, pro které má tato funkce konstantní hodnotu.

Jelikož je tento proces výpočtu časově náročný, používá se speciální strategie, která omezuje počítání pouze na body s vhodnou pozicí.



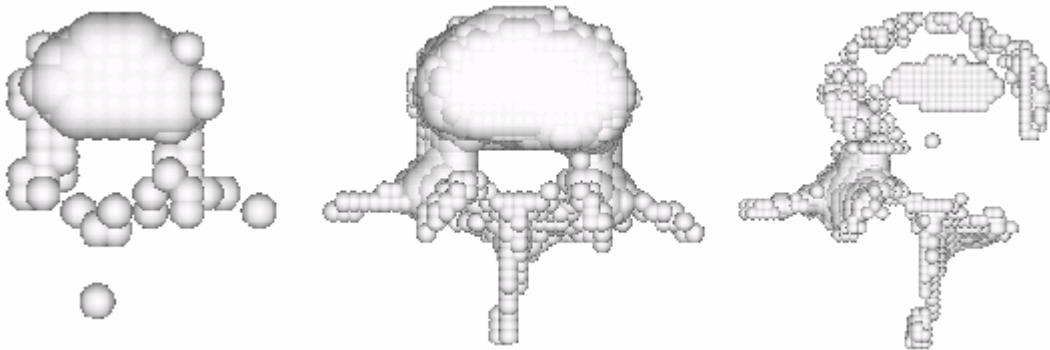
Obr. 3-15: Rekonstrukce povrchu pomocí střední osy (zleva : množina bodů společně s vybranými body na střední ose, rekonstruovaný povrch zepředu a zezadu).

Tvar výsledného povrchu je silně závislý na typu funkce. Například „**sharp**“ funkce zachovává detaily, zatímco „**soft**“ funkce detaily zhlazuje (viz obr. 3-16). Také spojitost výsledného tělesa může být ovlivněna tvarem funkce.



Obr. 3-16 : Příklad vlivu funkcí. „Soft“ funkce vlevo spojuje dvě koule dohromady, „sharp“ funkce vpravo naopak zachová detaily.

Díky voxelizaci je rozhodujícím bodem, jak přesně jsme schopni spočítat střední osu (viz obr. 3-17). Pokud je rozlišení malé, potom nebudou drobné detaily reprezentovány přesně. Čím více budeme rozlišení střední osy zjemňovat, tím kvalitnější detaily budou. Nicméně neplatí, že pokud bychom dosáhli nekonečně jemného vzorkování střední osy, budeme mít povrch rekonstruovaného tělesa přesný. Rozpadnou se totiž části povrchu díky tomu, že vstupní množina bodů je vzorkována také v určitém rozlišení.



Obr. 3-17 : Rozlišení střední osy (zleva : hrubé, střední, jemné – části povrchu se rozpadnou na nespojitě části).

3.4. Warping

Rekonstrukce povrchu založená na Warpingu⁸ znamená, že se pokusíme nějakým způsobem deformovat počáteční povrch tak, abychom dosáhli dobré aproximace povrchu zadaného vstupní množinou bodů. Představme si např. jako počáteční povrch trojúhelníkovou síť kolem nějaké množiny bodů (jakákoliv síť obklopující množinu). Pro všechny vrcholy trojúhelníkové sítě nějakým způsobem zjistíme, se kterými body povrchu rekonstruovaného tělesa korespondují a body ze sítě posuneme do těchto míst na povrchu. Důsledkem toho se nám počáteční trojúhelníková síť zdeformuje do nového tvaru, který se bude blížit původnímu tvaru povrchu tělesa.

Z tohoto jednoduchého přiblížení se dá odvodit, že tato skupina metod je dobrá pro případy, kdy je počáteční hrubá aproximace již dána, jelikož to značně ulehčuje nalezení korespondujících bodů.

⁸ Ve slovníku je ke slovu warping přiřazeno několik významů, z nichž významově nejbližší je deformace, zkroucení, zprohýbání či zborcení.

Základní metody warpingu jsou již poměrně staré, např. Murakiho „**blobby model**“ [19] pro aproximaci 2.5D sítí, či „**deformable superquadrics**“ Terzopoulose, Witkina a Kasse [20] [21].

Müllerova metoda [22] extrahuje uzavřený geometrický model ze vstupní množiny. Začíná s jednoduchým modelem, který je ale již topologicky blízko rekonstruovanému objektu a deformuje tento model, že se buď zvětšuje nebo zmenšuje tak, aby obsáhl objekt obsažený uvnitř. Je vytvořena množina jednoduchých neprotínajících se polyhedronů, která je buď obsažena uvnitř objektu nebo tento objekt obklopuje. S každým vrcholem polyhedronu je asociována funkce, která sjednocuje cenu lokální deformace spjatou s vlastnostmi jednoduchých polyhedronů a spojitost mezi šumem a význačnými rysy objektu. Pokud budeme minimalizovat tato omezení, pak dosáhneme efektu podobného nafukování balónu uvnitř nějakého dutého tělesa nebo vysávání vzduchu z igelitového sáčku, ve kterém je těleso umístěno.

Naprosto odlišný přístup warpingu je postup tzv. modelování orientovaných částic („**oriented particles**“), navržený Szeliskim a Tonesenem [23]. Každý částice má několik parametrů, které jsou upravovány postupně během simulace modelování. Modelováním interakce navzájem mezi částicemi je povrch vytvářen vztahy mezi těmito částicemi (odpuzování x přitahování). Pro každý vzorek (bod vstupní množiny) je vytvořena jedna částice s vhodnými parametry. Prostor mezi body (respektive částicemi) je vyplněn zvětšujícími se částicemi daleko od izolovaných bodů a hran. Poté, co takto dostaneme hrubou aproximaci, jsou vkládány další částice pro vyhlazení povrchu.

V následujících třech odstavcích pak podrobněji popíši tři algoritmy s odlišnými základními postupy založenými na čistě geometrickém a fyzikálním přístupu a za použití umělé inteligence.

3.4.1. Volná deformace prostoru („**Spatial free form warping**“)

Hlavní myšlenka spočívá v deformaci celého prostoru, ve kterém je objekt určený k procesu warpingu obsažen. Deformace prostoru je definována konečnou množinou vektorů skládajících se z počátečního a koncového bodu, z kterých je počítáno prostorové vektorové pole („**displacement spatial vector field**“) interpolací za použití jedné z metod interpolace roztroušených bodů. Těchto metod samozřejmě existuje velké množství, je však dobré si vybrat tu metodu, která poskytuje pro daná data nejvhodnější výsledky.

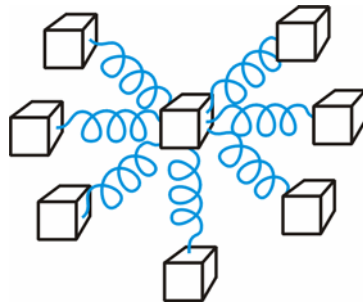
Výsledné vektorové pole přiřazuje každému bodu v prostoru jeho cílový bod. Pokud je toto pole aplikováno na každý vrchol počáteční sítě, pak se tato síť deformuje kolem objektu.

Výhoda tohoto postupu spočívá v tom, že je potřeba pouze malý počet vektorů pro důležité rysy objektu, jako jsou rohy či ostré hrany. Stále otevřenou otázkou je ale, jak nalézt tyto vektory automaticky.

3.4.2. Algoritmus „Algorri a Schmitt 2“

Tento algoritmus překládá danou aproximující trojúhelníkovou síť do fyzikálního modelu. Vrcholy sítě jsou interpretovány jako hmotné body a hrany jsou nahrazeny pružinami. Každý uzel (tedy hmotný bod) výsledné sítě pružin (viz obr. 3-18) je přiřazen svému nejbližšímu bodu dané množiny S vstupních bodů. Uzly a pružiny jsou vybrány tak, že trojúhelníková síť je deformována směrem ke vstupní množině.

Model může být vyjádřen jako lineární soustava diferenciálních rovnic stupně dva, která může být řešena iterativně. Efektivita je zajištěna rovnoměrným dělením prostoru do voxelové sítě a aproximováním počáteční trojúhelníkové sítě z této množiny voxelů způsobem popsáným v původní metodě autorů (viz.výše) [1].



Obr. 3-18 : Znárodnění sítě pružin příslušející jednomu z bodů (ve středu)

3.4.3. Algoritmus Kohonenova charakteristická mapa („Kohonen feature map“, Baader a Hirzinger)

Kohonenova⁹ charakteristická mapa je další metoda [24] [25] [26] rekonstrukce povrchu pomocí warpingu. Dá se popsat jako dvourozměrné pole neuronů. Každý neuron u_j má svůj váhový vektor w_j . Na počátku jsou tyto vektory naplněny náhodnými normalizovanými hodnotami (délka vektoru je jedna). V průběhu trénování neuronů (rekonstrukce) jsou vstupy neuronů naplněny vstupními daty, které pozměňují váhové vektory. Každý vstupní vektor i je poskytnut neuronu j , takže výstup je dán jako skalární součin vektorů w_j a i .

$$o_j = \vec{w}_j \cdot \vec{i}$$

Neuron generující největší odezvu o_j je se nazývá centrum budící oblasti („**centre of excitation area**“). Váhy tohoto neuronu a jeho sousedů jsou aktualizovány následujícím výrazem :

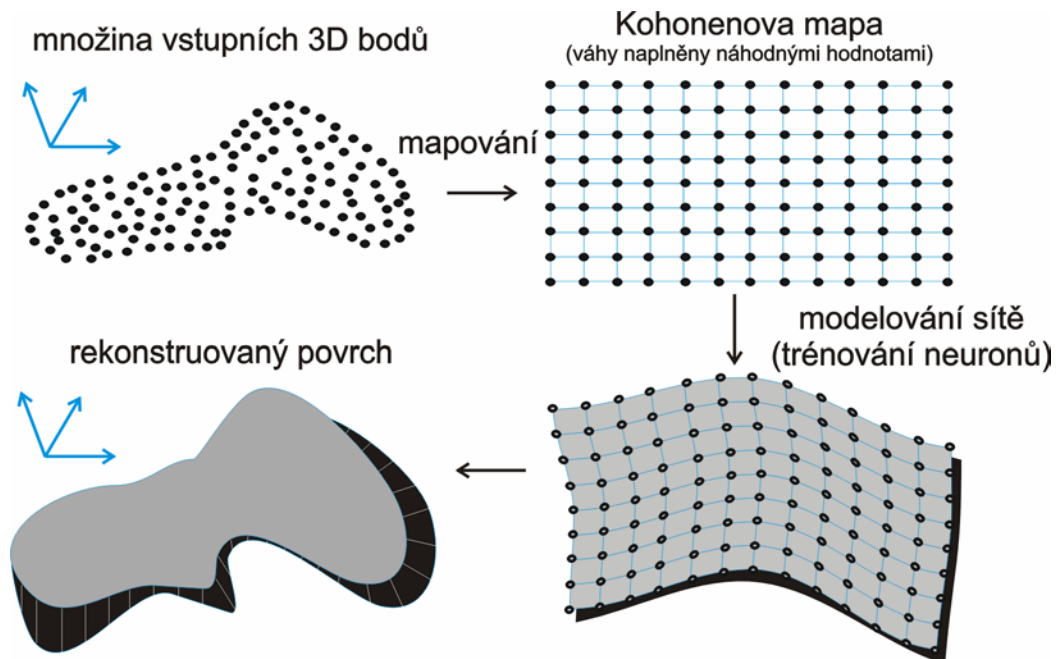
$$\vec{w}_j(t+1) = \vec{w}_j(t) + \varepsilon_j (\vec{i} - \vec{w}_j(t))$$

Po této úpravě musí být váhové vektory znova znormalizovány. Hodnota

$$\varepsilon_j = \eta \cdot h_j$$

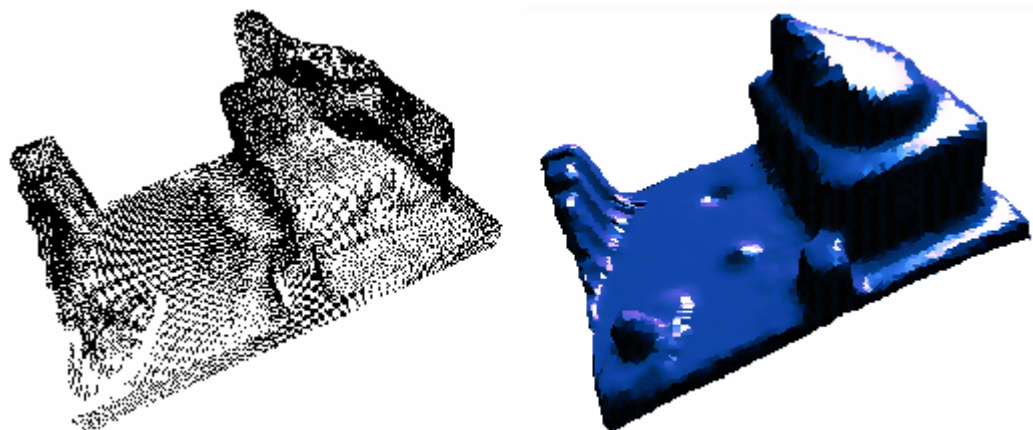
⁹ Kohonenova síť je neuronová jednovrstvá síť s dopředným šířením s učením bez učitele, přičemž výstupní hodnota neuronů je definována jako vzdálenost mezi vstupním a váhovým vektorem. Další použití tohoto druhu sítě je v oblasti analýzy dat, shlukování či vytváření sémantických map.

obsahuje dvě konstanty : konstantu popisující váhu učení η a konstantu vlivu na nejbližší sousedy h_j , takže neurony vzdálené od aktuálního neuronu jsou pozměněny pouze málo.



Obr. 3-19 : Průběh algoritmu.

Algoritmus má jednu nepěknou vlastnost. Pokud vstupní data zcela neodpovídají neuronové síti, je možné, že některé neurony mohou být hodně vzdáleny od centra budících oblastí. Proto musí být aktualizovány pouze za pomoci nejbližších sousedů, což není tak efektivní pro umístění neuronů blízko skutečného povrchu. Proto Baader a Hirzinger představili způsob reverzivního trénování. Na rozdíl od klasického způsobu, kdy pro každý vstupní bod je vytvořen a aktualizován korespondující neuron, pracuje toto trénování následovně: pro každý neuron u_j je zjištěna část vstupních dat s největším vlivem a ta je využita pro aktualizaci tohoto neuronu. Ve své implementaci používají autoři kombinaci obou způsobů pro trénování sítě.



Obr. 3-20 : Výsledek rekonstrukce (zleva : množina bodů, rekonstruovaný povrch).

3.5. Inkrementální konstrukce povrchu

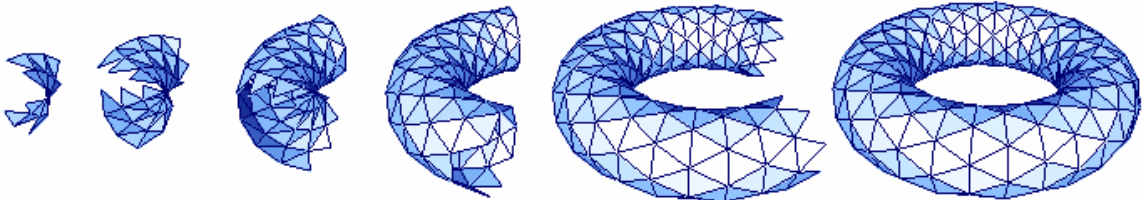
Hlavní myšlenka inkrementální konstrukce povrchu je vytvoření interpolujícího či aproximujícího povrchu přímo s pomocí určitých povrchově orientovaných vlastností vstupní množiny bodů. Například vytváření povrchu může začít na počáteční hraně, o které můžeme prohlásit, že v rekonstruovaném povrchu určitě bude. Z této hrany pak pokračujeme a spojujeme ji s dalšími body podle určitých kritérií tak, aby vznikaly trojúhelníky interpolující či aproximující skutečný povrch. Tento způsob používá např. Boissonnat ve svém algoritmu.

Jiná možnost je začít s globálním drátěným modelem povrchu a iterativně ho vyplňovat tak, aby vznikl kompletní povrch, což je hlavní myšlenka algoritmu od Mencla a Müllera.

3.5.1. Povrchově orientovaný algoritmus „Boissonnat“

Tento algoritmus [13] začíná na té hraně (tj. spojenci mezi dvěma body), která má nejmenší délku ze všech možných hran. Protože je nutné vytvořit pomocí této hrany první trojúhelník a dále pak připojovat další trojúhelníky formující povrch, je spočítána lokální tečná rovina založená na sousedních bodech této hraniční hrany. Sousední body jsou pak promítnuty na tangenciální rovinu.

Nový trojúhelník je pak vytvořen právě z těchto bodů podle následujícího kritéria: bod bude vybrán jako vrchol nového trojúhelníka, pokud maximalizuje úhel nad svou hraniční hranou. Laicky si to můžeme představit tak, že pokud se budeme nacházet v tomto bodu, tak uvidíme hranu pod maximálním úhlem. Algoritmus končí v případě, že již není možné vytvořit novou hranu.



Obr. 3-21 : Průběh rekonstrukce algoritmu.

3.5.2. Algoritmus „Mencl a Müller“

Tato metoda [27] [28] se skládá ze sedmi hlavních kroků:

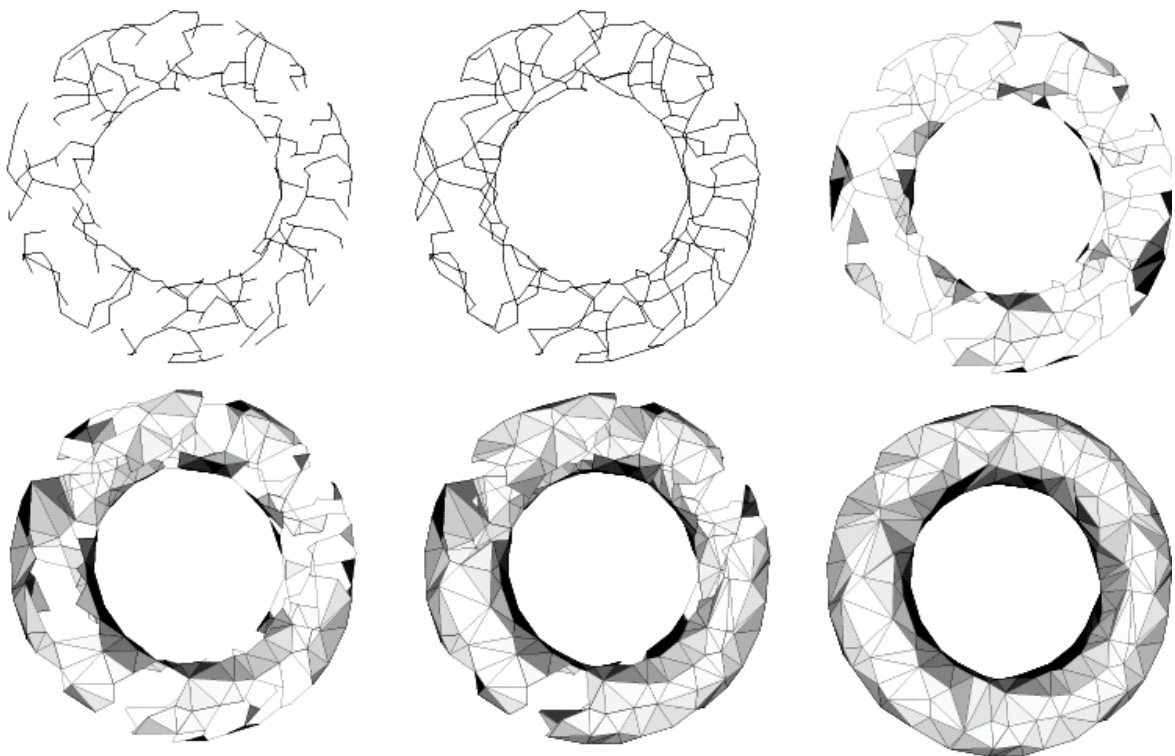
- Výpočet EMST (eukleidovský „**minimum spanning tree**“) ze vstupní množiny.
- Rozšíření grafu.
- Rozpoznání hlavních rysů.
- Extrakce nesouvisejících objektů.
- Spojení souvisejících objektů.
- Spojení sdružených hran v grafu.
- Vyplnění drátěného modelu trojúhelníky.

První dva kroky jsou zde proto, aby nám pomohly vytvořit tzv. graf popisu povrchu („**Surface Description Graph**“, **SDG**). Ten je vytvořen z **EMST** a jeho následným rozšířením, které spojuje hrany v listech grafu s jejich sousedy. Toto opatření zabrání vzniku protínajících se hran a úzkých trojúhelníků v dalších krocích. Dále je třeba ve třetím kroku rozpoznat důležité informace o možných strukturách vznikajících na povrchu. Stejně jako v procesu rozpoznávání objektů v rastrových obrázcích i zde se určují objekty typu hrana, cesta, prstenec bodů atd. Rozpoznávání je heuristické a v procesu rozpoznávání se aplikují různá pravidla. Např. objekt typu prstenec je rozpoznán na základě těchto pravidel :

- Úhel mezi sousedními body musí být větší než 135° .
- Normály bodů prstence jsou podobné.
- Projekce neprodukuje protínající se hrany.
- Rotace bodů je vždy stejným směrem.

Poté jsou tyto oblasti rozpojeny a/nebo spojeny v závislosti na různých pravidlech, abychom získali vhodný popis objektů nacházejících se ve vstupní množině (krok 4, 5). V posledním kroku před triangulací je graf ještě více rozšířen spojením sdružených hran v grafu s ohledem na určitá omezení. Nakonec jsou do takto popsaného povrchu vloženy trojúhelníky za pomoci systému pravidel pro získání povrchu s vysokou přesností.

Mencel a Müller vymysleli způsob rozpoznání charakteristických rysů pro vylepšení přesnosti **SDG** v závislosti na předpokládaném povrchu objektu. Myšlenka je v možnosti integrace různých druhů rozpoznávacích algoritmů v hlavním algoritmu, zatímco se zachovává strukturální konzistence **SDG**.



Obr. 3-22 : Rekonstrukce anuloidu (odshora zleva: EMST vstupní množiny bodů, SDG povrchu, průběh rekonstrukce po nalezení 100, 250, 450 trojúhelníků, celková rekonstrukce).

Na rozdíl od ostatních metod, tento algoritmus vytváří po částech lineární povrch, který přesně interpoluje vstupní množinu. Může také pracovat s datovými množinami, které mají velké změny v hustotě vzorkování. Toho se může využít tak, že není třeba nadměrně vzorkovat ty části povrchu, které mají jen malé změny v křivosti. Rekonstrukce ostrých rohů v umělých či syntetických objektech je také bezproblémová, stejně jako rekonstrukce neorientovatelných povrchů, jako např. Möbiova páska¹⁰. Nevýhodou však je, že většina použitých pravidel je heuristických.

3.6. Clustering

Někdy se může stát, že vstupní množina neobsahuje pouze jeden objekt. Výše popsané algoritmy však budou mít s tímto případem problémy, protože nebudou schopny tento případ rozeznat (např. budou brát mezeru mezi objekty jako efekt podvzorkování, takže mohou být objekty, v místě, kde jsou si nejbližší, spojeny). Tento problém může být odstraněn segmentováním nebo seskupováním („clustering“) vstupní množiny bodů do podmnožin bodů, které budou příslušet určité komponentě.

3.6.1. Algoritmus „Fua a Sander“

Tento algoritmus [29] [30] [31] se skládá z tří kroků. V prvním kroku je okolo každého bodu iterativně vytvořena kvadratická záplata a tento bod je pak přesunut na tuto záplatu. Dalším dodatečným efektem tohoto kroku, kromě vytvoření množiny lokálních záplat, je vyhlazení dané množiny bodů.

Poté, co je první krok dokončen, data vytvářejí zatím ještě neregulární vzorkování povrchu ležícího pod nimi. Ve druhém kroku jsou proto přesunuty body společně se svými lokálními záplatami na pozici v regulární síti (dané původní množinou bodů).

V třetím kroku je provedeno povrchově orientované shlukování. Je vytvořen graf, jehož uzly tvoří body po korekci (přesunutí na regulární síť) v předchozím kroku. Hrana v grafu je mezi dvěma uzly přidána v případě, že jejich záplaty jsou shodné (shodnost v tomto případě není absolutní, je použita určitá tolerance). Spojené komponenty grafu pak definují shluk (cluster) ve vstupní množině. Pak každý z těchto clusterů může být rekonstruován za použití některého z výše uvedených algoritmů.

3.6.2. Algoritmus „Mencl a Müller“

Tato metoda [27] [28] má opět tři hlavní kroky. Nejprve jsou vyšetřeny všechny možné hrany v závislosti na své délce. Pokud délka hrany přesáhne jistou hodnotu lokální hustoty (zjištěnou za pomoci lokální hustoty bodů na obou koncových bodech), hrana bude smazána. Pokud hrana spojuje důležité regiony, pak samozřejmě v grafu zůstane. Tyto regiony často představují důležité struktury povrchu a v dalším kroku je nutné na ně brát ohled.

¹⁰ August Ferdinand Möbius, německý astronom a matematik (geometrie, topologie, atd...), popsal tzv. Möbiovu pásku, která představuje vlastně dvourozměrný povrch s pouze jednou stranou. Dá se získat ve třídídimenzionálním prostoru, když si vezmeme pruh papíru, pootočíme na jednom konci o 180 stupňů a spojíme.

V druhém kroku jsou všechny hrany spojující důležité regiony prozkoumány mnohem větším detailu. V závislosti na informaci o strukturách obsažených v těchto regionech se rozhodne, jestli hrany je spojující skutečně tak mají být, pokud ne, jsou smazány.

V posledním kroku je pak prozkoumány sousední oblasti kolem každého regionu. Je spočteno nejlepší spojení (nová hrana) mezi sousedními regiony a je vloženo do grafu popisujícího povrch.

3.7. Další algoritmy

Celá tato kapitola se zabývala základním teoretických popisem různých druhů algoritmů. Samozřejmě existují i další algoritmy, ale většina z nich bude nějakým způsobem vycházet z těchto popsanych, ať již budou provádět vylepšení jejich nedostatků, nebo použijí již prověřené teoretické části pro vytvoření jiného. Další kapitoly se pak budou zabývat popisem a implementací dvou algoritmů Niny Amenty založených na Voronoiově diagramu. Sama jejich myšlenka vychází z algoritmu Attaliho normalizované sítě (kapitola 3.1.5).

4. CRUST algoritmus

Jako hlavní algoritmus této diplomové práce jsme vybrali „CRUST“¹¹ algoritmus vytvořený Ninou Amentou [URL1]. Tento algoritmus můžeme zařadit podle předchozího dělení do třídy algoritmů pracujících s dělením prostoru a používajících objemový přístup.

Hlavním důvodem, proč jsme se na tento algoritmus zaměřili je, že pro dělení prostoru se používá Delaunayova triangulace, jejíž implementaci v Delphi vytvořenou vedoucí této diplomové práce jsem měl k dispozici. Dalším, neméně důležitým důvodem je ten, že práce Niny Amenty byly prvním algoritmem, kde byla do jisté míry garantována funkčnost i teoretickými důkazy, i když pouze pro hladké plochy a při splnění určitých podmínek na vzorkování bodů. Dalším důvodem je, že jde o metody poměrně nové a zachycující současný stav znalostí problematiky. V neposlední řadě pak články Niny Amenty jsou ve srovnání s některými jinými prameny ještě přijatelně srozumitelné, zejména „praktičtější“ verze pro SIGGRAPH [32].

Původní algoritmus, který je popsán v [32] (společně s důkazy pak v [33]) pracuje dvouprůchodově, potřebuje tedy zpracovat dvě Delaunayovy triangulace. Posléze se nám ještě podařilo nalézt od stejné autorky novější algoritmus [34], který potřebuje pouze jednu triangulaci. Základ tohoto druhého algoritmu byl vytvořen společně s T.K.Deyem a je založen na jeho „COCONE“ algoritmu [36], [37], [URL2].

Pro vlastní implementaci jsem vyzkoušel obě metody. Přestože pracují na podobném principu, objevují se u nich různé problémy a každá z těchto metod je v něčem lepší a v něčem horší. Nicméně se dá u obou metod říci, že pokud budou data splňovat poměrně tvrdé požadavky autorů (viz níže), pak teoretický popis algoritmu zajišťuje, že povrch bude správně rekonstruován. V následujícím textu je pak uveden popis obou metod, jejich implementace a výsledky.

Popis algoritmů omezím na části důležité pro jejich implementaci. Přesná teoretická znění jsou uvedena v materiálech ([32] [33] [34]) a jsou hodně obtížné na porozumění. Definice konvexní obálky, Voronoiova diagramu a Delaunayovy triangulace jsou uvedeny v kapitole 2.

4.1. Vzorkovací kritérium

Co je pro obě metody společné, je vzorkovací kritérium, které „musí“ být splněno, aby rekonstrukce proběhla korektně a její správnost byla teoreticky podložena.

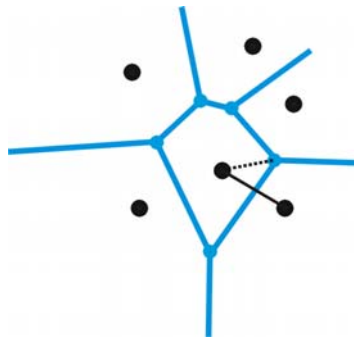
¹¹ Crust znamená v češtině skořápka, kůra či povlak.

„Dobrý“ vzorek je takový vzorek, pro který platí, že jeho vzorkovací hustota je úměrná vzdálenosti ke střední ose. Vzorek S se nazývá r -vzorek („ r -sample“), pokud eukleidovská vzdálenost jakéhokoliv bodu $p \in S$ k nejbližšímu bodu je nejvíce r -krát vzdálenost k nejbližšímu bodu na střední ose.

Pro $r < 0.06$ platí, že rekonstrukce proběhne korektně. Nicméně naše data nikdy tuto podmínku nesplňovala a autorka se také zmiňuje o tom, že úspěšně je možné rekonstruovat i data, kdy r je několikanásobně větší.

Výhodou tohoto kritéria je, že nebere v úvahu distribuci bodů. Navíc bere v potaz křivost povrchu, protože střední osa je blíže k povrchu, kde je jeho křivost větší. U ostrých rohů a vrcholů se pak bude střední osa dotýkat povrchu. To je také důvod, proč tyto metody nejsou příliš dobré pro povrchy s těmito vlastnostmi.

Výpočet tohoto kritéria není příliš obtížný, stačí si vzpomenout na definici Voronoiovy buňky, která již tvoří ve své podstatě střední osu sousedních vzorků. Stačí proto vzít vrchol této buňky, který je nejbliže k bodu, pro který kritérium počítáme (je to určité zjednodušení, přesně by to měla být kolmice na nejbližší hranu z VD).



Obr. 4-1: Výpočet vzorkovacího kritéria (čárkovaně = vzdálenost k nejbližšímu bodu V.buňky, plná čára = vzdálenost k nejbližšímu bodu).

4.2. Dvouprůchodový „CRUST“ algoritmus

4.2.1. 2D případ

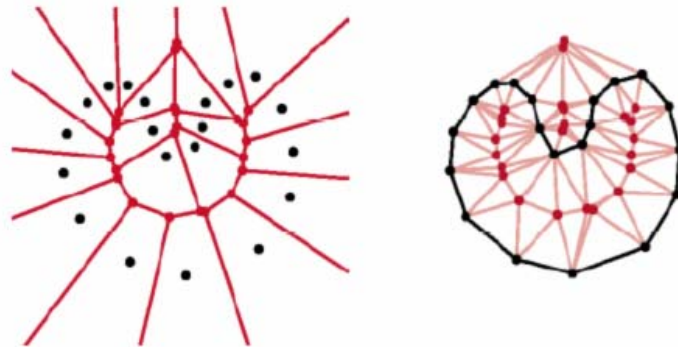
Algoritmus popíše nejprve pro 2D případ [35], protože je jednodušší pro pochopení. V tomto případě bude „crust“ grafem na množině bodů vstupní množiny S . Dá se definovat následovně: hrana e bude náležet grafu v případě, že kružnice opsaná této hraně nebude obsahovat jak žádný další bod ze vstupní množiny S , tak žádný další bod z množiny V (tvořené vrcholy Voronoiových buněk). Pro „crust“ platí následující teorém :

„Crust“ r -vzorku z hladkého povrchu F pro $r < 0.25$ spojuje pouze sousedící body tvořící F .

Pokud množina S je tedy r -vzorek (pro malé r), vrcholy Voronoiových buněk (tvořící množinu V) budou aproximovat střední osu a jakákoliv kružnice opsaná hraně mezi dvěma nesousedícími vrcholy bude obsahovat nějaký další bod buď z množiny S , nebo z množiny V .

Výše uvedená definice „crustu“ pak vede na následující algoritmus. Nejprve spočteme Voronoiův diagram množiny S a všechny vrcholy buněk dáme do množiny V . Dále spočítáme Delaunayovu triangulaci $S \cup V$. „Crust“ pak bude obsahovat ty hrany z Delaunayovy triangulace, jejichž kružnice opsané nebudou obsahovat jakýkoliv jiný bod z množiny $S \cup V$.

Dá se jednoduše říci, že „crust“ bude podmnožina Delaunayovy triangulace. Vrcholy z množiny V se používají pro odfiltrování nechtěných hran. Tato technika se nazývá **Voronoiovo filtrování**.

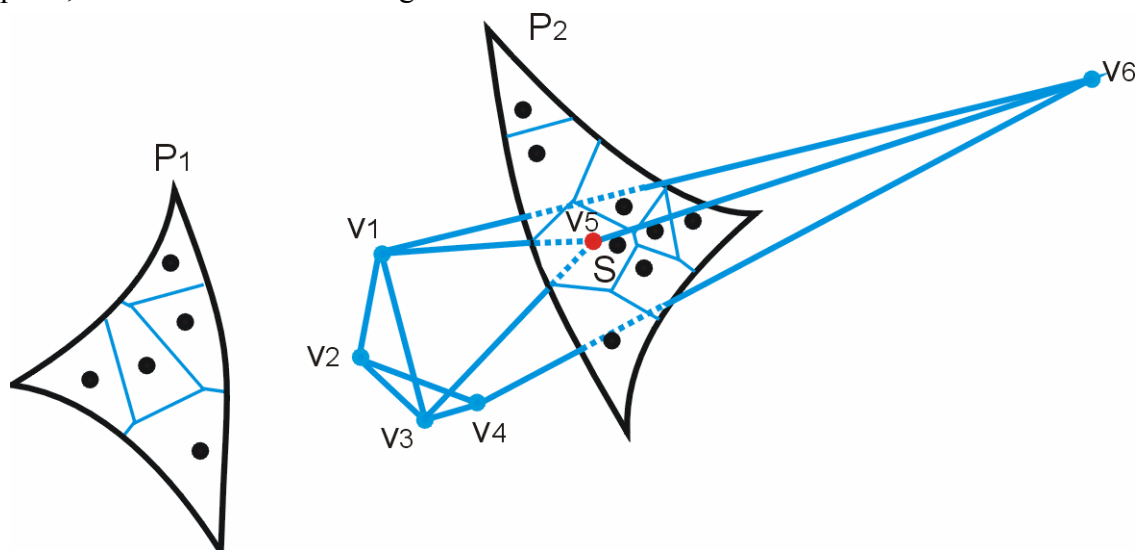


Obr. 4-2 : Voronoiovo filtrování ve 2D (převzato z [32]).

4.2.2. 3D případ

Jednoduchý algoritmus pro Voronoiovo filtrování bohužel neplatí pro 3D, protože již není pravda, že všechny vrcholy Voronoiových buněk pro dostatečně dobře navzorkovaný povrch leží blízko střední osy. Na druhou stranu, dá se najít mnoho vrcholů, pro které to platí.

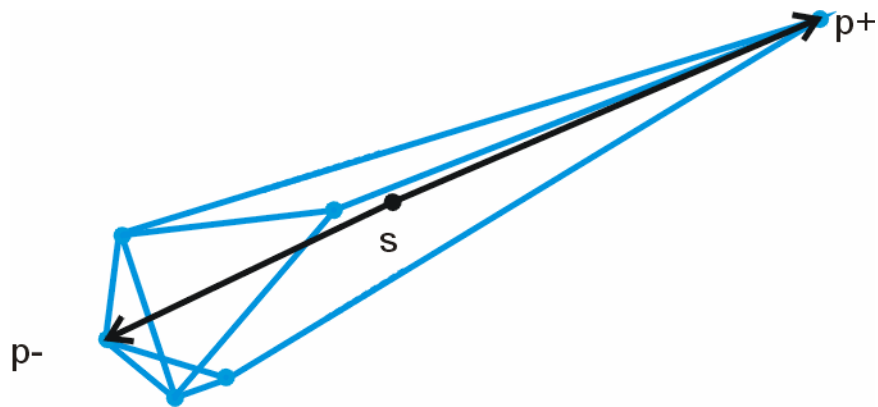
Představme si Voronoiovu buňku V s bodu s (bod nejblíže bodu V_5) jako na obrázku 4-2. Bod s je obklopen dalšími body z povrchu a buňka V je ohraničena protínajícími se rovinami oddělující jí od sousedních buněk, přičemž každá rovina je téměř ortogonální k povrchu. Proto je buňka dlouhá, úzká a zhruba ortogonální k povrchu. Dále bude platit, že tato buňka bude i ortogonální vzhledem ke střední ose.



Obr. 4-3 : Ukázka Voronoiovy buňky bodu s ve 3D na povrchu P_2 . Vrcholy buňky leží blízko střední osy (čtyři vrcholy V_1, V_2, V_3, V_4 uprostřed), blízko centra křivosti (bod V_6 vpravo nahoře) a jeden bod V_5 v ekvidistanční vzdálenosti od 4 bodů u povrchu. Povrch P_1 je proti povrchu P_2 a střední osa prochází mezi nimi

Tato úvaha pak vede k následujícímu algoritmu. Místo toho, abychom brali v úvahu všechny vrcholy Voronoiových buněk ve Voronoiově filtraci, vezmeme pro každý bod s pouze dva tyto vrcholy. Nejvzdálenější vrchol nazveme p^+ a budeme ho nazývat **kladný pól** Voronoiovy buňky (viz obr. 4-4). Protože ale nevíme, jaká strana povrchu bude orientována ven, potřebujeme vzít ještě co možná nejvzdálenější vrchol na druhé straně povrchu (pro implementaci stačí, aby se jako kladný pól vzal nejvzdálenější vrchol buňky a jako záporný pól nejvzdálenější bod s negativním skalárním součinem s kladným pólem). Tento vrchol nazveme **záporný pól** buňky a budeme ho značit p^- (na obrázku 4-2 by to byl bod V_2 a kladný pól by byl bod V_6). Dále můžeme ještě definovat vektory n^+ a n^- , které budou ukazovat z bodu s do jejich pólů p^+ a p^- .

Díky tomu budeme nyní pro každý bod mít dva póly, z kterých je možné odvodit přibližný průběh povrchu tímto bodem. Výjimku budou tvořit pouze body na konvexní obálce, pro které stačí mít pouze jeden pól (který bude vždy směrem dovnitř objektu), jelikož je pro ně známa orientace povrchu.



Obr. 4-4 : Póly p^+ a p^- Voronoiovy buňky bodu s . Vektor sp^+ se značí n^+ a vektor sp^- se značí n^- .

Pokud nyní spočteme Delaunayovu triangulaci původní množiny bodů S a jejich pólů, měli bychom získat triangulaci, která bude mít tetrahedrony takové, že jejich vrcholy budou tvořeny vždy body z množiny S a jejich póly. Extrakcí trojúhelníků, které mají vrcholy tvořeny pouze z množiny S , pak získáme trojúhelníkovou síť interpolující povrch. Výše uvedené myšlenky můžeme shrnout:

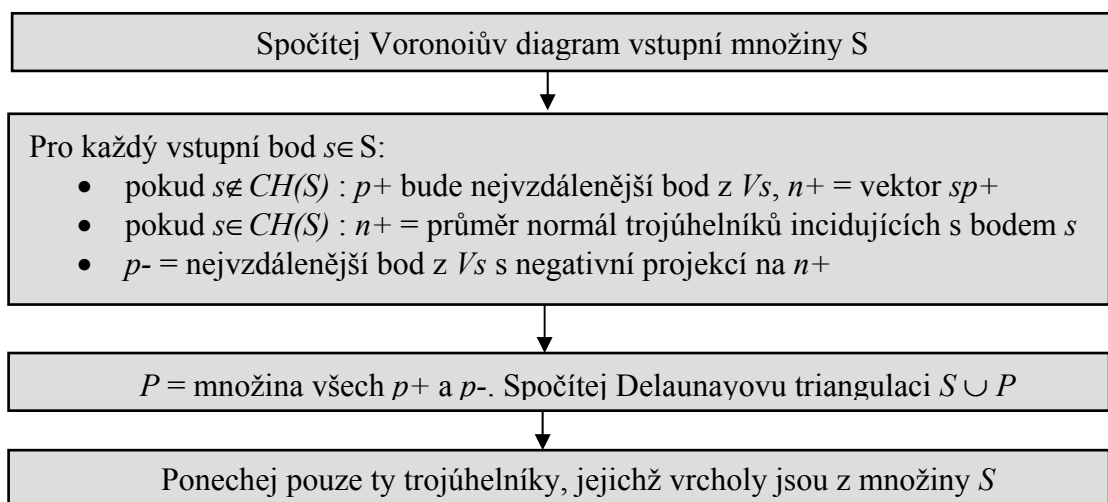


Schéma 4-1 : Dvouprůchodový „crust“ algoritmus

Na tomto základním algoritmu je krásné, že nepotřebuje žádné další dodatečné parametry pro výpočet a pracuje automaticky. Výstup tohoto algoritmu je třídímní „crust“, který je tvořen množinou trojúhelníků, jež se geometricky přibližují původnímu povrchu.

Necht' S je r -vzorek hladkého povrchu F , přičemž $r < 0.06$. Pak „crust“ obsahuje množinu trojúhelníků tvořících síť topologicky ekvivalentní F . Každý bod z „crustu“ leží nejvýše ve vzdálenosti $5r \cdot d(p)$ od nějakého bodu p z F , kde $d(p)$ je vzdálenost od bodu p ke střední ose.

Bohužel „crust“ není většinou tvořen jen správnými trojúhelníky. Často se také objevují všechny čtyři trojúhelníky velmi tenkého povrchového tetraedronu, takže „crust“ netvoří manifold. Vizuálně je to ale již model, se kterým je možné pracovat.

Autorka se ještě zmiňuje o druhém způsobu počítání pólů, kdy se jako druhý pól bere druhý nejvzdálenější vrchol Voronoiovy buňky bez ohledu na jeho pozici. Tento výpočet jsem vyzkoušel, nicméně jsem ho nezahrnul do konečného programu, protože primární povrch obsahoval mnohem větší množství trojúhelníků a extrakce manifoldu proběhla pouze s velkými obtížemi (mnoho překrývajících se trojúhelníků). Sama autorka o něm píše, že tento způsob není teoreticky podložen a proto rekonstrukce, která dopadla překvapivě dobře, může být náhodná.

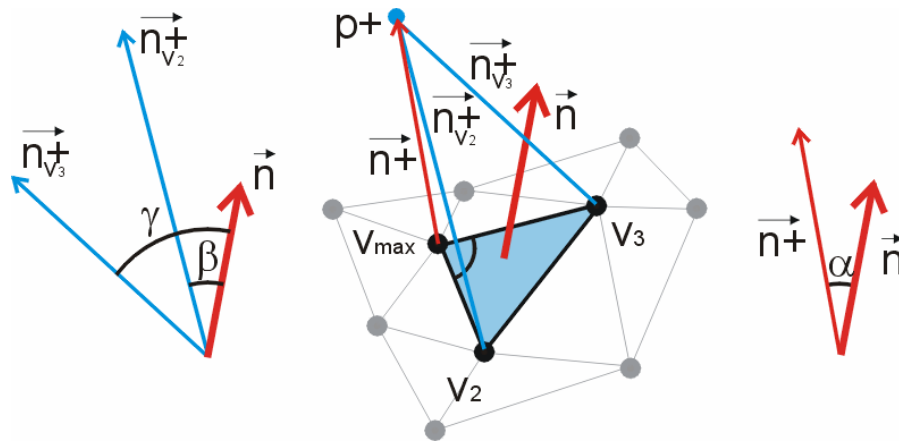
4.2.3. Odhad normálových vektorů a filtrování

Abychom mohli vytvořit po částech lineární manifold, který konverguje k rekonstruovanému povrchu, potřebujeme ještě zavést zvláštní filtrování, jelikož Voronoiovo filtrování může vytvořit např. velmi úzké trojúhelníky, které jsou kolmé k povrchu. V předchozích odstavcích jsem se již zmínil o tom, že normály k povrchu n^+ a n^- spočtené pomocí pólů jsou téměř ortogonální na povrch. Chyba (rozdíl úhlu skutečné normály a této normály) je lineární v r .

Toho nyní můžeme využít a zrušit trojúhelníky, jejichž normály se budou příliš lišit od normál n^+ a n^- . Nejprve je nutné zjistit největší úhel v trojúhelníku a vrchol v_{max} , ve kterém tento úhel je. Pro tento vrchol zjistíme jeho pól p^+ a vektor n^+ z bodu do pólu. Dále určíme vektory pro ostatní vrcholy trojúhelníků tak, že od vrcholů vedeme vektory k již určenému pólu p^+ ve vrcholu v_{max} (na obr. 4.4. vektory nazvané n_{v3^+} a n_{v2^+})

Pro potřebu filtrace je ještě nutné určit úhel θ , o který se může normála trojúhelníka od normál ve vrcholech lišit (viz obr. 4-5). Tento úhel je třeba určit experimentálně, ale je závislý na r . Trojúhelník odstraníme v případě, že jeho normála se bude lišit minimálně o úhel θ ve vrcholu v_{max} a minimálně o $2,2\theta$ v ostatních vrcholech (na obr. 4.4 tedy $\alpha > \theta \wedge \beta > 2.2\theta \wedge \gamma > 2.2\theta$).

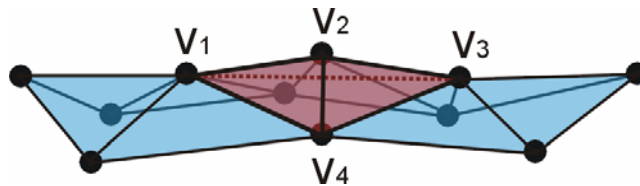
Sama autorka ale uvádí, že se jedná o poměrně nebezpečný krok, protože nesprávnou volbou parametrů se můžeme připravit o větší množství trojúhelníků, než je třeba. Tento jev se projevuje hlavně na hranicích tělese, nebo na místech, kde jsou ostré vrcholy.



Obr. 4-5: Filtrace pomocí normál. U vrcholu v_{\max} se nachází největší úhel v trojúhelníku, proto vybereme jeho pól $p+$. Od ostatních vrcholů v_2 a v_3 pak vedeme vektory do $p+$ a zkontrolujeme úhel mezi normálou trojúhelníka n a těmito vektory.

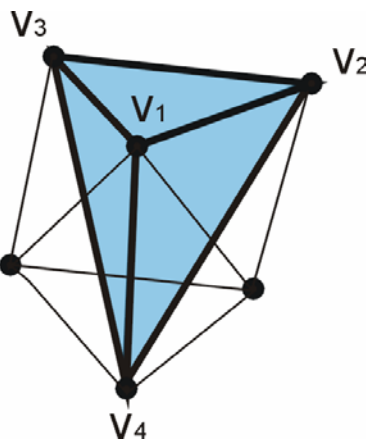
4.2.4. Extrakce manifoldu

Posledním krokem k získání manifoldu je jeho extrakce. Tento krok zajišťuje, že získáme povrch, který je topologicky stejný jako rekonstruovaný povrch. Nyní kopírují trojúhelníky extrahované Voronoiovým a normálovým filtrováním povrch pouze geometricky. Stále se zde zatím nacházejí trojúhelníky, které do manifoldu nepatří a kterých je třeba se při extrakci vyvarovat, např. velmi úzké tetrahedrony mohou mít i po filtrování všechny čtyři stěny v povrchu (viz obr. 4-6).



Obr. 4-6 : 4 trojúhelníky z tetrahedronu v_1, v_2, v_3, v_4 , které zůstanou i po filtraci normál a tvořící bublinu na povrchu.

Nejprve se musí orientovat všechny trojúhelníky. Začneme na nějakém bodu ležícím na konvexní obálce, protože na té můžeme jednoduše spočítat orientaci povrchu procházejícího tímto bodem:



Obr. 4-7 : Tetrahedron na konvexní obálce, na straně v_1, v_2, v_3 nemá souseda.

Tetrahedron z Delaunayovy triangulace nebude mít na konvexní obálce souseda, proto můžeme jednoduše určit stěnu (v_1, v_2, v_3) , která je na konvexní obálce. Směr

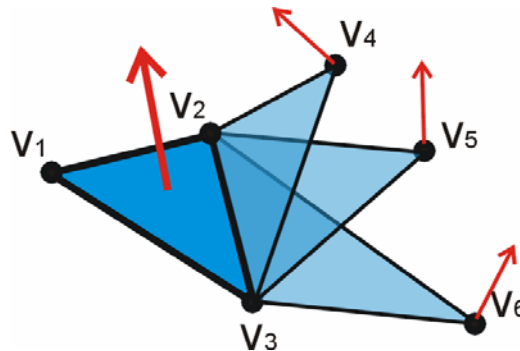
normálového vektor této stěny spočteme pomocí bodu V_4 . Pokud skalární součin normály a vektoru z jakéhokoliv vrcholu V_1, V_2, V_3 do V_4 bude kladný, pak oba vektory ukazují stejným směrem (tedy dovnitř objektu) a normálový vektor musíme obrátit, aby ukazoval směrem ven z objektu.

Podobným způsobem pak můžeme pokračovat dále vyhledáváním do hloubky. Vezmeme si trojúhelníky, které incidují s některou hranou již zpracovaného trojúhelníka a normály naorientujeme tak, aby ukazovaly do stejného směru.

V průběhu zpracování můžeme ovšem narazit na ostrou hranu. To bude taková hrana, pro kterou platí, že trojúhelníky na této hraně svírají úhel větší než $3\pi/2$, a tuto hranu je nutné smazat.

Poslední krok pak tvoří vlastní extrakci. Budeme postupovat po jednotlivých stěnách a hledat k nim sousedy, které mají nejbližší k povrchu.

Příklad si můžeme ukázat na obrázku níže. Trojúhelník V_1, V_2, V_3 je již zpracovaný, tzn. má správně orientovanou normálu směrem ven z tělesa a leží na povrchu. Na hraně V_2, V_3 je třeba se rozhodnout, který trojúhelník přijmout dál. Protože ale požadujeme, aby další trojúhelník byl opět co nejbližší skutečnému povrchu, vybereme takový trojúhelník, jehož normála bude svírat se zpracovaným trojúhelníkem nejmenší úhel.



Obr. 4-8: Vyhledání dalšího trojúhelníka na povrchu

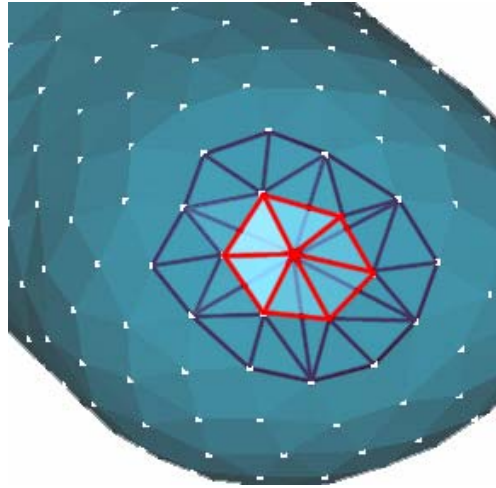
Extrakce manifoldu patří ale mezi nejslabší místa celého algoritmu a jsou s ní největší problémy. Blíže se jí pak budu věnovat ještě v popisu implementace.

4.3. Jednoprůchodový „CRUST“ algoritmus

Hlavní nevýhodou dvouprůchodového „crust“ algoritmu je, že potřebuje dvě Delaunayovy tetrahedronizace pro extrakci vhodných trojúhelníků. Přestože nejlepší algoritmy pro Delaunayovu tetrahedronizaci mají algoritmickou složitost $O(N)$ (pro očekávaný případ), druhá tetrahedronizace probíhá s přibližně s trojnásobkem počtu bodů první triangulace a časový nárůst je značný.

Nicméně už první triangulace obsahuje trojúhelníky, které patří do povrchu (viz obr. 4-9). Proto by bylo dobré nalézt takové kritérium, kterým bychom mohli jednoduše určit ty simplex, kterých se to týká.

V kapitole 4.2.2 jsme ukázali (intuitivně), že Voronoiovy buňky bodů ležících na povrchu jsou dlouhé, úzké a téměř ortogonální k povrchu (viz obr. 4-10). Toto pozorování lze využít a na jeho základě vyvinout takovou heuristiku, která by nám nechala jen ty trojúhelníky z tetrahedronů, které budou tvořit povrch.



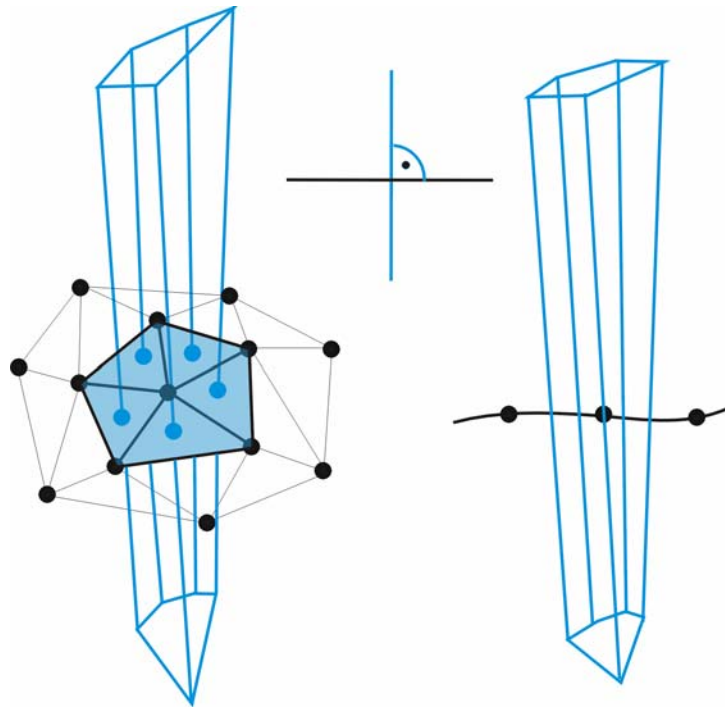
Obr. 4-9: Rekonstruovaný povrch a tetrahedrony z první tetrahedronizace. Červené trojúhelníky jsou správné trojúhelníky z tetrahedronizace incidující s bodem uprostřed, zbytek trojúhelníků jsou pozůstatky tetrahedronů (jejich další stěny), které musíme správně rozpoznat a dále s nimi nepracovat.

Algoritmus je založen na následujícím teorému. Necht' T je množina trojúhelníků povrchu P , přičemž T splňuje tyto tři podmínky:

- T obsahuje pouze všechny trojúhelníky, jejichž duální Voronoiovy hrany protínají P .
- Každý trojúhelník v T je malý (poloměr kružnice opsané tomuto trojúhelníku je mnohem menší než vzdálenost vrcholů ke střední ose)
- Všechny trojúhelníky v T jsou „ploché“ (normály trojúhelníků mají pouze malé odchylky od normál povrchu ve vrcholech trojúhelníka),

Za předpokladu, že povrch P je hladký a vzorkování dostatečně husté, první podmínka zajišťuje, že T obsahuje po částech lineární manifold homeomorfní k P . Druhá a třetí podmínka pak říká, že jakýkoliv manifold M , extrahovaný z T , obsahující všechny body ze vstupní množiny, a pro který platí, že všechny sousední trojúhelníky mají k sobě tupý úhel, musí být homeomorfní k P .

Množinu T můžeme spočítat následovně. Po Delaunayově tetrahedronizaci máme k dispozici ke každému bodu množinu tetrahedronů (a tedy i množinu trojúhelníků), přičemž povrch bude procházet některými jejich stěnami. Pro každý bod p můžeme zjistit, jaká bude normála povrchu v tomto bodě, stejným postupem jako ve dvouprůchodové verzi. Vezmeme si nejvzdálenější vrchol v_p Voronoiovy buňky a nazveme ho pólem buňky. Vektor od bodu p k pólu v_p pak bude představovat normálový vektor n_p povrchu v tomto bodě.



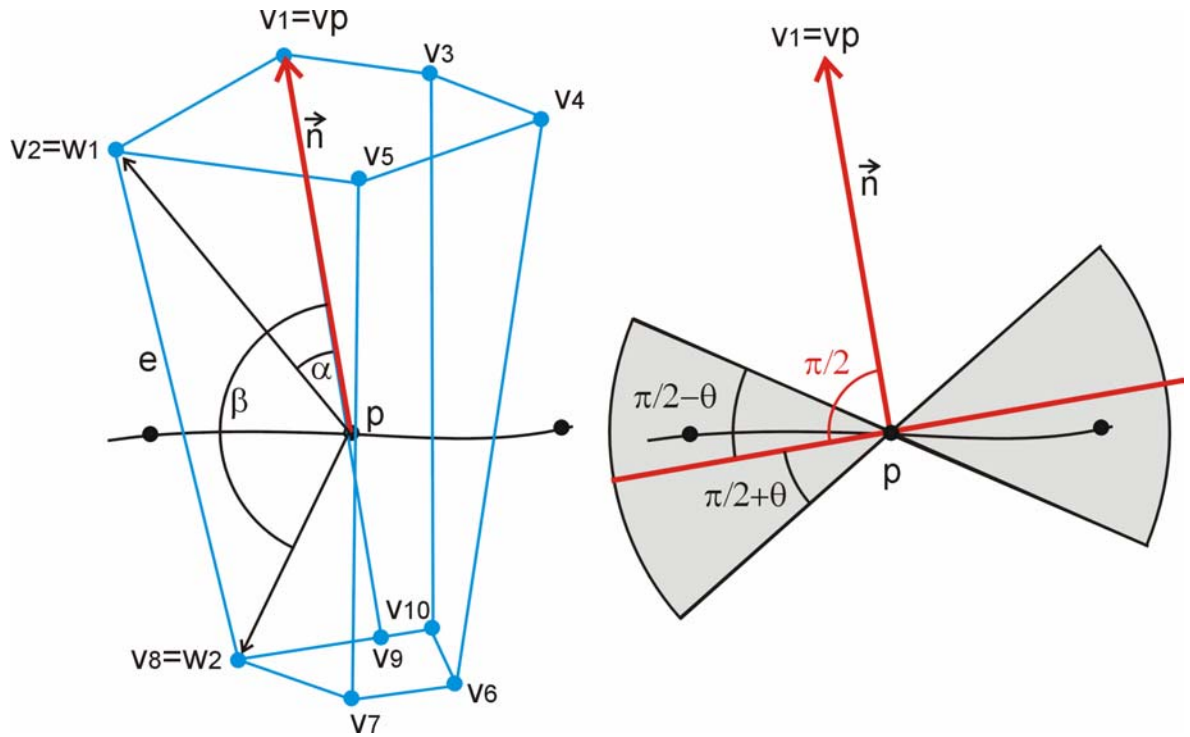
Obr. 4-10 : Vlevo typický příklad Voronoiovy buňky, vpravo její průmět (s vykresleným průběhem povrchu uprostřed)

Jelikož nyní máme ke každému bodu jeho přibližnou normálu k povrchu, můžeme poměrně jednoduše zjistit trojúhelníky tvořící povrch (viz obr. 4-11). Jakýkoliv trojúhelník tvořící stěnu tetrahedronu má ve VD duální hranu e . Aby byl trojúhelník povrchový, musí tato hrana procházet povrchem. Koncové body hrany e označíme w_1 a w_2 . Pak spočítáme úhly vektorů $\alpha = \angle w_1 p, n_p$ a $\beta = \angle w_2 p, n_p$ a zjistíme, jestli interval $\langle \alpha, \beta \rangle$ protíná interval $\langle \pi/2 - \theta, \pi/2 + \theta \rangle$. Pokud toto bude platit pro všechny tři body trojúhelníka incidujícího s hranou e , pak tento trojúhelník bude patřit do množiny T . Úhel θ je brán jako vstupní parametr algoritmu.

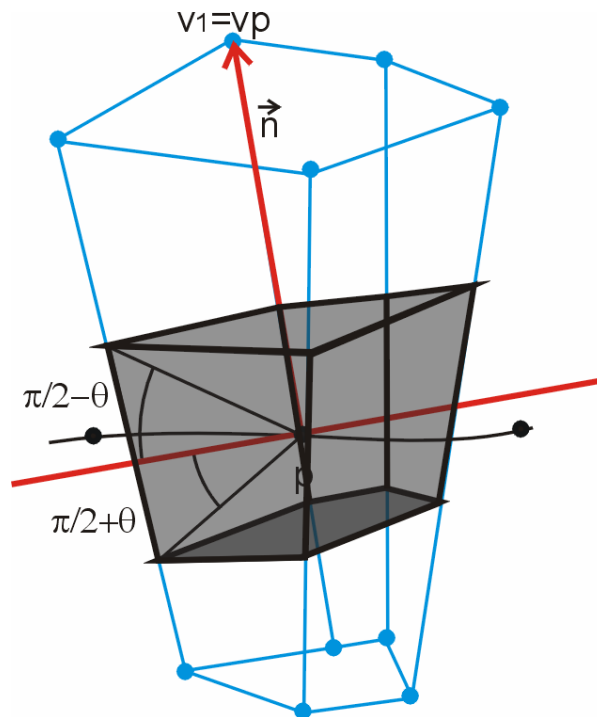
Po získání množiny T se dostaneme do stejné fáze jako ve dvouprůchodovém algoritmu po filtraci trojúhelníků pomocí normál a můžeme aplikovat stejný postup extrakce manifoldu.

Již v úvodu této kapitoly jsem uvedl, že tento jednorůchodový algoritmus pracuje podobně jako Deyův „COCONE“ algoritmus. Dey používá název „COCONE“ pro třírozměrný útvar vytvořený stejně jako na obrázku 4-11 vpravo (viz obr. 4-12).

Ve své práci šel Dey ale ještě mnohem dále než Amenta a v závislosti na tvaru Voronoiovy buňky dokázal vypracovat strategie rozpoznávání důležitých míst, jako jsou hranice objektu či detekce podvzorkování. Nicméně tyto strategie patří již mimo tuto diplomovou práci a budou brány v potaz v případě další práce na tomto algoritmu.



Obr. 4-11 : Postup výpočtu filtrace trojúhelníka (pro jeho jeden bod p).



Obr. 4-12 : Deyův COCONE.

5. Implementace

Implementace obou „CRUST“ algoritmů proběhla výhradně pomocí vývojového prostředí Borland Delphi 5.0 firmy Inprise pod operačním systémem Microsoft Windows NT. Jedním z důvodů volby tohoto programovacího jazyka (přestože ho mnozí „programátoři“ neuznávají) je má dobrá znalost tohoto jazyka a snadnost práce v prostředí a jednoduchost ladění. Navíc obsahuje velice dobře pracující kompilátor a optimalizátor, takže rychlost kódu generovaného tímto prostředkem se plně vyrovná jakýmkoliv dalším kompilátorům. Má také vynikajícím způsobem zpracovanou knihovnu ovládacích a vizualizačních komponent¹², proto vývoj vstupně-výstupních částí aplikací je velice rychlý a programátor může většinu času věnovat řešenému problému.

Jedním z bodů zadání této diplomové práce byla implementace těchto algoritmů do systému MVE [URL3], který byl vyvinut na Západočeské Univerzitě v Plzni týmem počítačových grafiků¹³ [URL4]. Celý systém je založen na používání jednotných datových struktur a poskytování předepsaných funkcí v rámci modulů. Právě tyto moduly tvoří páteř tohoto systému, jelikož v sobě implementují výkonnou část systému. Mohou být tří druhů, vstupní, výstupní a výpočetní. Vstupním modulem může být např. generátor bodů či modul pro načítání trojúhelníkových sítí, výpočetním modulem pak může být např. modul počítající tetrahedronizaci nad danou množinou bodů nebo decimující trojúhelníkovou síť. Výstupy z těchto modulů je pak možné pomocí výstupního modulu uložit do souboru v různých formátech.

Programátorsky jsou pak moduly řešeny jako DLL¹⁴ knihovny (v jedné knihovně může být i více modulů). Ty jsou při startu systému registrovány MVE editorem tak, že jsou od nich vyžádány seznamy funkcí a podle nich editor zjistí, jaké moduly knihovny obsahují. Ty jsou pak dostupné z editoru a uživatel je může libovolně podle svého uvážení používat.

Hlavní výhodou tohoto systému je právě jednoduchost práce. Uživatel se nemusí o nic starat a nemusí mít žádné vědomosti o vnitřní práci modulů. Důležité pro něj je pouze znát vstup, výstup a činnost modulu. Pak může jednotlivé moduly podle svého uvážení pospojovat v runtime editoru tak, aby byl splněn jeho požadavek.

Implementaci obou „crust“ algoritmů jsem provedl objektově, protože obě metody mají hodně společného a sdílí část kódu. Hlavní třídou je třída *TCRUST*, jejíž konstruktor má jako parametr název souboru s daty. Tato třída dále zapouzdřuje hlavní datové typy a

¹² Knihovna VCL - Visual Component Library.

¹³ Viz. <http://herakles.zcu.cz/people.php>

¹⁴ DLL - Dynamic Link Library.

metody pro nahrání dat ze souboru, výpočtu tetrahedronizace (převzato, viz kap. 5-1), extrakci manifoldu a retriangulaci děr. Odvozené třídy od ni jsou pak třídy *TCRUST_ONE_PASS* pro implementaci jednorůchodového algoritmu a třída *TCRUST_TWO_PASS* pro implementaci dvouprůchodového algoritmu. Pro urychlení výpočtů s vektory jsem použil navíc vlastní knihovnu *U_FPU* poskytující základní vektorové funkce napsané v assembleru matematického procesoru.

V této kapitole se však budu snažit odprostit od jakýchkoliv jazykově (programátorsky) závislých konstrukcí a veškerou implementaci popíši slovně (pro lepší orientaci ve zdrojových textech však budu uvádět v závorkách metodu či funkci, která daný výpočet provádí). Samotný popis spustitelného programu a modulu do MVE bude pak uveden v přílohách, společně s ukázkou různých vstupů a výstupů

Celý průběh výpočtu je možné rozdělit do několika fází (podkapitol), které budou mít podobný charakter jako u popisu algoritmu v kapitole 4. Dvouprůchodový algoritmus bude mít takovouto strukturu :

- Tetrahedronizace vstupní množiny
- Výpočet $p+$ a $p-$ pólu.
- Druhá tetrahedronizace vstupní množiny společně s póly.
- Generování primárního povrchu a filtrování pomocí normál.
- Extrakce manifoldu.
- Nalezení a triangulace děr v manifoldu.
- Generování finálního povrchu.

Jednorůchodový algoritmus má podobnou strukturu, proto nebudu zvlášť popisovat každý algoritmus, ale u odpovídajících podkapitol uvedu pouze místa, které se liší :

- Tetrahedronizace vstupní množiny
- Výpočet $p+$ pólu.
- Generování primárního povrchu a úhlo-hranové filtrování.
- Extrakce manifoldu.
- Nalezení a triangulace děr v manifoldu.
- Generování finálního povrchu.

V původních algoritmech se však nevyskytoval předposlední bod, tedy nalezení a triangulace děr. Tento bod jsem dodal z toho důvodu, že manifold může obsahovat po své extrakci díry, které je možné díky existující sousednosti jednoduše nalézt a pokusit se je retriangulovat. U jednorůchodové verze jsem zavedl navíc úhlo-hranové filtrování jako heuristiku, která může podstatně vylepšit výsledek.

Veškeré výpočty jsem se snažil dělat pomocí polí hodnot a indexů na ně. Např. všechny datové struktury, ve kterých se objevují vrcholy tetrahedronů či trojúhelníků, mají v sobě pouze odkazy na reálné souřadnice těchto simplexů. Má to několik obrovských výhod, pokud není nutné pracovat přímo se souřadnicemi, můžeme pracovat pouze s indexy, což je mnohem rychlejší a navíc nebudeme mít problémy s omezenou přesností reálných čísel.

5.1. Delaunayova tetrahedronizace

Programový kód této části programu byl převzat z práce Doc. Dr. Ing. Ivany Kolingerové, jelikož vytvoření rychlého a robustního kódu pro Delaunayovu tetrahedronizaci je samo o sobě záležitostí na diplomovou práci.

Pro konstrukci tohoto druhu tetrahedronizace se používá několik druhů algoritmů (blíže např. v [38] [39]). Poměrně jednoduchá je metoda inkrementální konstrukce, která je založena na vlastnosti prázdné Delaunayovy kružnice. Pokud ovšem neobsahuje účinné techniky pro vyhledávání, pak její algoritmická složitost je $O(N^3)$. Mnohem komplikovanější je použití techniky zametání, která je ovšem obtížně rozšiřitelná do 3D. Výhodou je ale její algoritmická složitost $O(N \log N)$. Oblíbenou metodou je i použití techniky Rozděli a Panuj („Divide and Conquer“), které pracuje na principu rekursivního rozdělení vstupní množiny, triangulace těchto podmnožin a jejich následného slučování. Algoritmická složitost je sice také $O(N \log N)$, ale je obtížná pro implementaci ve 3D, proto se také v praxi ve 3D téměř nepoužívá. Je také dokázáno, že problém Delaunayovy triangulace v dimenzi E^d se dá převést na problém konstrukce konvexní obálky v E^{d+1} a zpětnou projekcí pak získat triangulaci. Jako většina popsanych metod je ale díky implementačním problémům pro vyšší dimenze než E^2 nepoužitelná.

Naopak poměrně hodně užívanou metodou je metoda inkrementálního vkládání, přestože její algoritmická složitost pro nejhorší případ je $O(N^2)$ díky tomu, že před vložením nové hrany se musí otestovat její poloha vůči ostatním již přijatým hranám. Nicméně randomizací vstupní množiny a efektivním vyhledáváním se dá docílit algoritmu se složitostí $O(N \log N)$ až $O(N)$ v očekávaném případě.

Název této metody je podle způsobu konstrukce. Do již existující triangulace (na začátku se zvolí takový simplex, který obsahuje uvnitř všechny body vstupní množiny, přičemž na konci se pak smaže) se přidá bod a lokálně se triangulace upraví prohozením stěn simplexů tak, aby splňovala Delaunayovu podmínku.

Tento algoritmus je také použit v kódu Doc. Dr. Ing. Kolingerové. Jedinou nevýhodou tohoto kódu je paměťová náročnost (vnitřně se používá orientovaný acyklický graf (DAG), který značně urychluje vyhledání simplexu, do kterého se má nový bod vložit), která omezuje jeho použití na středně veliké množiny bodů (na 1GB operační paměti je přibližně 230 000 bodů).

Tato implementace má ještě jednu velkou výhodu. Je numericky robustní díky tomu, že pro kritické výpočty byla použita speciální numerická knihovna pro numericky stabilní geometrické predikáty [42] [43]. O málo se sice výpočet zpomalí (vůči verzi bez této knihovny), ale zpřesní se. Přesné rozhodování je kritické hlavně v místech, kde jsou body blízko u sebe a povrch, z kterého jsou navzorkovány, má malou křivost. Pak se zde budou tvořit tetrahedrony, které budou hodně tenké a díky nepřesné aritmetice může u nich dojít k lokálnímu porušení Delaunayovy podmínky (a dokonce pádu programu).

Vyzkoušel jsem tetrahedronizaci jednoho z testovacích souborů („hotdog.pts“, viz obr. 5-1) a verze se standardní aritmetikou generovala 31641 tetrahedronů, zatímco verze s přesnou aritmetikou 32096 tetrahedronů.



Obr. 5-1 : Ukázka vstupních dat (soubor hotdog.pts)

Jednoduše si ale můžeme tuto knihovnu představit jako černou skříňku, která má jako vstup množinu bodů v E^3 a výstupem bude množina tetrahedronů splňující Delaunayovu podmínku a adaptivně dělicí prostor. Navíc má tato výstupní množina spočtenou sousednost tetrahedronů, čehož můžeme využít v dalších výpočtech.

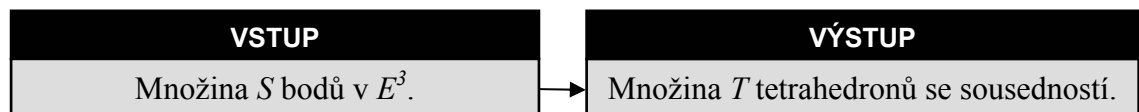
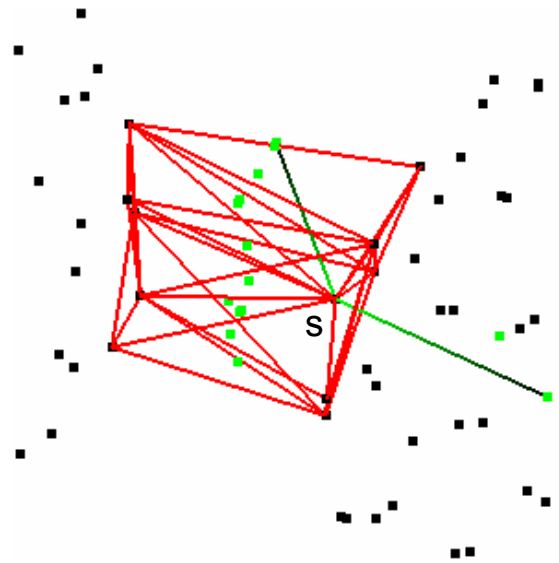


Schéma 5-1 : Vstup a výstup Delaunayovy tetrahedronizace

5.2. Výpočet pólů

Po získání tetrahedronizace T vstupní množiny S (viz obr. 5-2) musíme s její pomocí spočítat Voronoiův diagram V . Jelikož ale potřebujeme z tohoto diagramu pouze vrcholy Voronoiových buněk, nemusíme ho počítat celý. Jednoduše využijeme duality, která říká, že vrchol Voronoiovy buňky je místo, které má stejnou vzdálenost od 4 bodů (v E^3). Potom bude platit, že pro jakýkoliv bod $s \in S$ můžeme určit vrcholy V_{si} jeho Voronoiovy buňky V_s tak, že určíme pro každý tetrahedron s tímto bodem incidujícím (tzn. jeden vrchol tetrahedronu musí být bod s) střed koule opsané.

Protože ale budeme s těmito středy koulí tetrahedronů opsaných v tomto kroku algoritmu intenzivně pracovat, je dobré si je předpočítat a uložit do pomocného pole (pole `TCRUST.fTetrahedron_Sphere`, metoda `TCRUST.Preprocess_Tetrahedron`). Dále ještě můžeme využít sousednosti v množině tetrahedronů T a urychlit vyhledávání tetrahedronů incidujících s bodem s tak, že si pro každý bod zjistíme jeden (jakýkoliv) tetrahedron, který s tímto bodem inciduje (pole `TCRUST.fTetra_For_Point_Array`, funkce `TCRUST.Prepare_Data`).



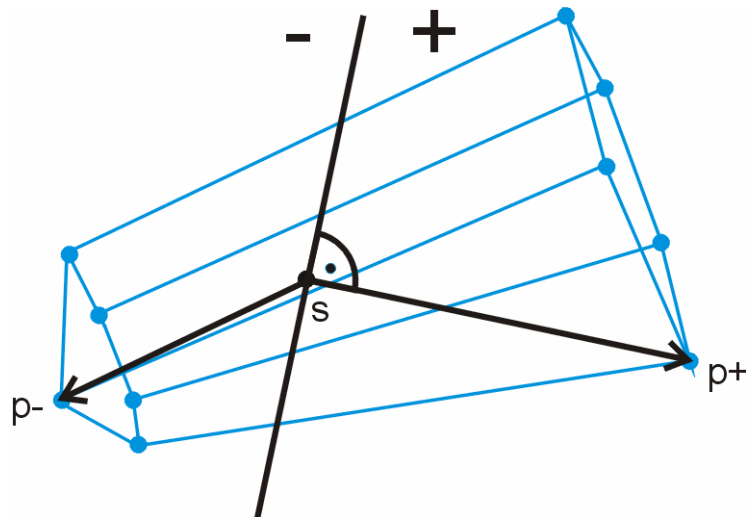
Obr. 5-2 : Detail (místo, kde se na obr. 5-1 obě části téměř dotýkají). Póly bodu s (zeleno-černě), incidující tetrahedrony (červeně), vrcholy Voronoiovy buňky (zeleně).

Poté můžeme již pro každý bod s začít vyhledávat incidující tetrahedrony (metoda *TCRUST_TWO_PASS.Compute_Poles*). Vezmeme si první (ve fázi preprocessingu spočtený) tetrahedron a uložíme si ho do pomocného pole. Poté prozkoumáme všechny stěny tetrahedronu a v případě, že stěna obsahuje jako jeden z vrcholů vrchol s , vstoupíme do tetrahedronu s touto stěnou sousedící a rekurzivně dále pokračujeme, dokud nenajdeme všechny tetrahedrony. Pomocné pole pomáhá ukládat již nalezené tetrahedrony a navíc si díky němu můžeme před tím, než vstoupíme do dalšího tetrahedronu, zkontrolovat, jestli už tento tetrahedron byl použit.

Po zjištění všech tetrahedronů incidujících s bodem s se již jednoduše naleznou všechny vrcholy Voronoiovy buňky bodu s (středů koulí tetrahedronům opsaných, spočtené ve fázi preprocessingu). Z nich nalezneme kladný pól $p+$ (nejvzdálenější vrchol buňky) a vektor $n+$ (vektor z bodu s do bodu $p+$) bodu s . Záporný pól $p-$ nalezneme tak, že vezmeme další nejvzdálenější vrchol buňky splňující podmínku, že skalární součin vektoru $n+$ a vektoru $n-$ (vektor z bodu s do $p-$) bude záporný (viz obr. 5-3).

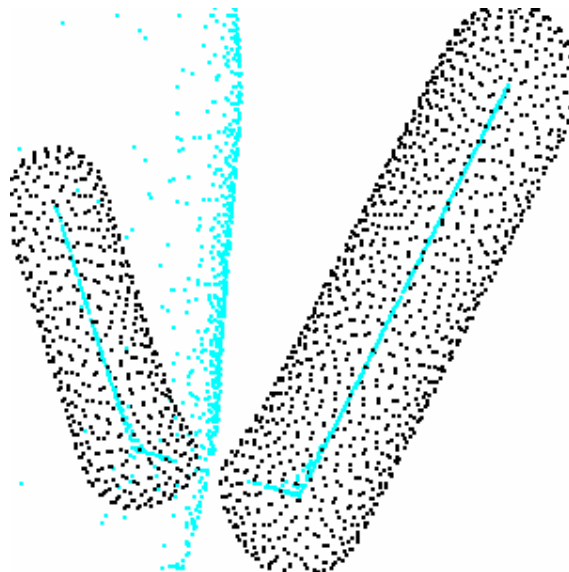
Pokud se bod s bude nacházet na konvexní obálce, pak to můžeme v tomto kroku jednoduše detekovat, protože se bod s bude nacházet na stěně tetrahedronu, která nebude mít souseda. Proto si nastavíme pro tento bod příznak, že leží na konvexní obálce.

V programu je navíc implementován výpočet hodnoty *r-vzorku* pro každý bod s . Pro tento výpočet potřebujeme znát nejbližší vrchol Voronoiovy buňky a nejbližší bod z množiny S . Ten získáme prozkoumáním vzdálenosti všech vrcholů tetrahedronů incidujících s bodem s a jelikož máme pro bod s spočítané i vrcholy buňky, jednoduše nalezneme nejbližší z nich (představuje vlastně nejbližší bod na střední ose). Celková hodnota *r-vzorku* pro celou množinu S je pak dána největší lokální hodnotou *r-vzorku* každého bodu.



Obr. 5-3 : Výpočet kladného a záporného pólu. Rovina kolmá na vektor $sp+$ v bodě s odděluje poloprostory, kde skalární součin vektoru $sp+$ a vektoru z bodu s bude kladný nebo záporný.

Pro jednorůchodový algoritmus funguje hledání pólu naprosto stejně, jenom nemusíme vyhledávat záporný pól $p-$. Proto z důvodu urychlení je metoda *Compute_Poles* v obou třídách implementována zvlášť.



Obr. 5-4 : Množina vstupních bodů (černě) a zobrazené póly (modře).

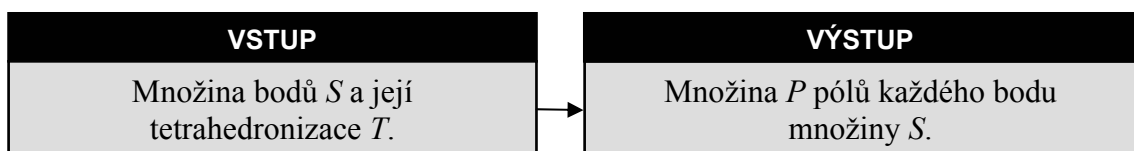


Schéma 5-2 : Vstup a výstup kroku výpočtu pólů

5.3. Druhá tetrahedronizace

V případě dvouprůchodového algoritmu je ještě třeba provést druhou tetrahedronizaci množiny $S \cup P$. Právě tato druhá tetrahedronizace nám zaručí, že většina nově

vzniklých tetrahedronů bude mít vrcholy z množiny S i z množiny P , která vlastně představuje střední osu.

Při přípravě dat (metoda *TCRUST_TWO_PASS.Create_Source_Point_2*) pro druhou tetrahedronizaci je třeba dát pozor na pořadí dat (nesouvisí s vlastní tetrahedronizací, protože interně se data stejně randomizují), protože na něm závisí rychlost dalšího kroku (generování primárního povrchu). Pokud bychom body vkládali do druhé tetrahedronizace náhodně, museli bychom při generování primárního povrchu použít test, jestli všechny body z právě zpracovávaného trojúhelníku patří pouze do množiny S , přičemž tento test by měl pro každý bod algoritmickou složitost $O(N)$. Pokud zachováme pořadí množin a napřed skutečně vložíme do tetrahedronizace množinu S a až po ní množinu P , stačí každý vrchol právě zpracovávaného povrchu srovnat s největším indexem množiny S , získáme tedy test s algoritmickou složitostí $O(1)$.

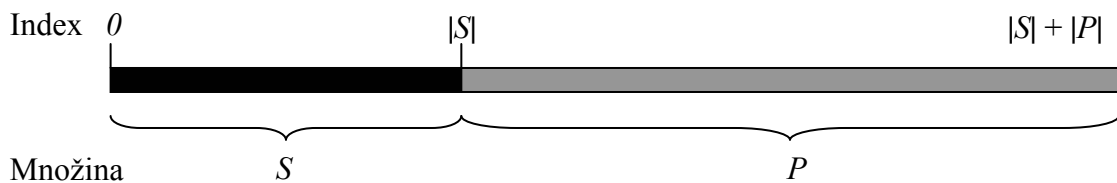


Schéma 5-3 : Příprava bodů pro druhou tetrahedronizaci.

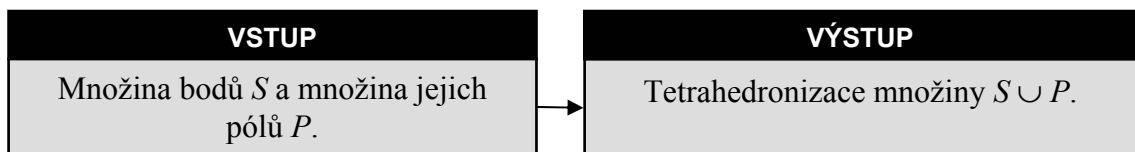


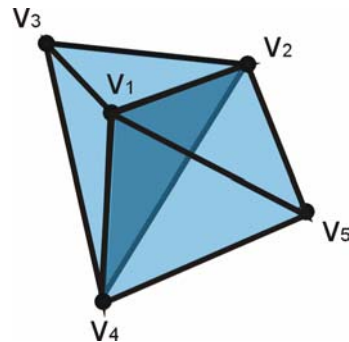
Schéma 5-4 : Vstup a výstup druhé Delaunayovy tetrahedronizace.

5.4. Generování primárního povrchu

5.4.1. Dvouprůchodový algoritmus

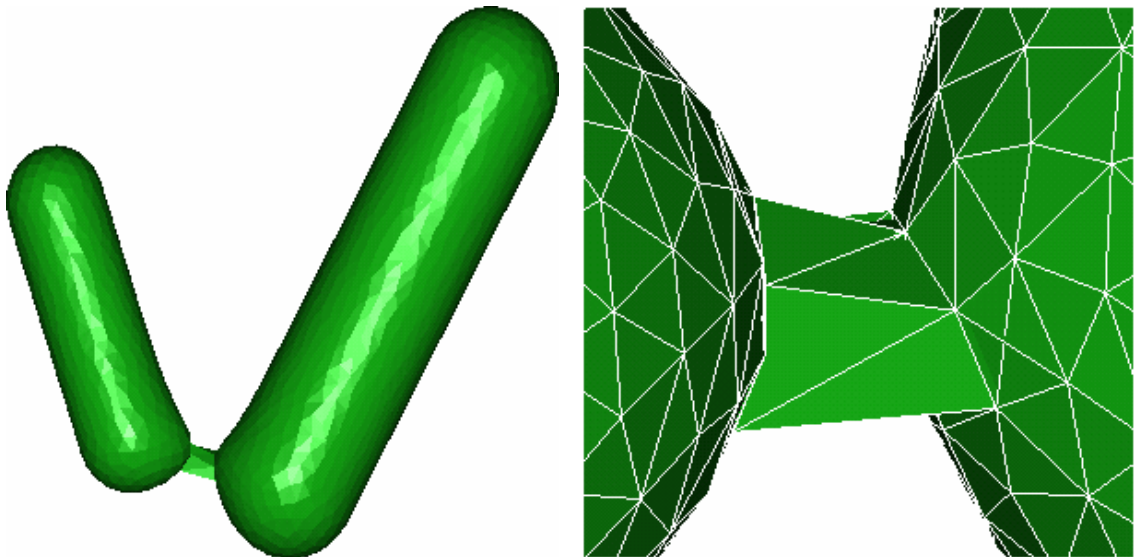
Po druhé Delaunayově tetrahedronizaci máme k dispozici množinu tetrahedronů, jejichž vrcholy patří buď do množiny P nebo do množiny S . Pro získání primárního povrchu (metoda *TCRUST_TWO_PASS.Compute_Object_Hull*) stačí otestovat všechny stěny tetrahedronů a vzít pouze ty trojúhelníky (tvoří stěny), které mají všechny vrcholy z množiny S .

Je třeba si dát ovšem pozor na to, že všechny trojúhelníky vyskytující se uvnitř konvexní obálky jsou v tetrahedronizaci dvakrát (viz obr. 5-5). Proto jsem při zpracování zavedl pro každý tetrahedron zvláštní příznak. Tetrahedrony jsou za sebou zpracovávány postupně jeden za druhým a po svém zpracování je tento příznak nastaven. Každý trojúhelník, před tím, než je testován, zda má všechny své vrcholy z množiny S , projde ještě testem na to, jestli druhý tetrahedron s ním incidující nebyl již zpracován. To zamezí vzniku duplicitních trojúhelníků v primárním povrchu a navíc urychlí tento krok algoritmu, protože se nemusí testovat větší počet trojúhelníků, než je třeba.



Obr. 5-5 : Trojúhelník V_1, V_2, V_4 uvnitř konvexní obálky je sdílen dvěma tetrahedrony V_1, V_2, V_3, V_4 a V_1, V_2, V_4, V_5 .

Pokud je zapnuta filtrace pomocí normál, musí ještě trojúhelník projít posledním testem. Ten je pouze aplikací matematických operací uvedených v kapitole 4.2.3 proto jej nebudu dále rozvádět (funkce `TCRUST_TWO_PASS.Compute_Object_Hull.Is_Triangle_Bad`).



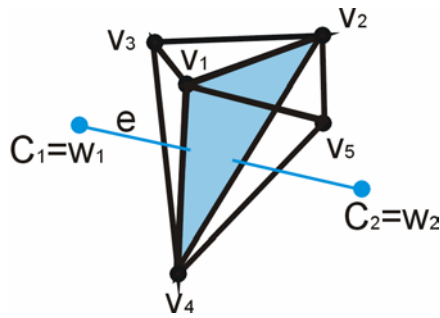
Obr. 5-6 : Primární povrch bez filtrace, vpravo detail nadbytečných trojúhelníků (po zapnutí filtrace tyto trojúhelníky zmizí).

5.4.2. Jednoprůchodový algoritmus

Podobně jako u dvouprůchodového algoritmu máme na vstupu tohoto kroku k dispozici množinu tetrahedronů, tentokrát však již z první tetrahedronizace. Primární povrch se bude skládat z některých stěn (trojúhelníků) těchto tetrahedronů.

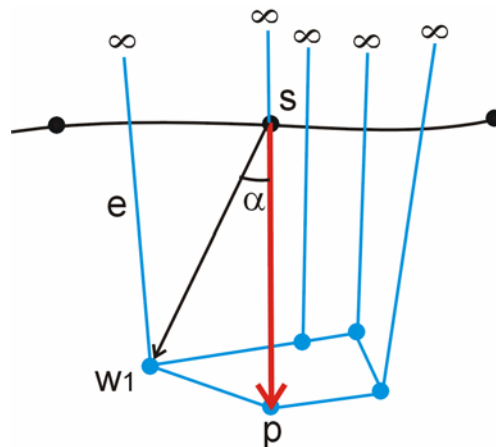
Abychom správně rozhodli, který trojúhelník patří povrchu, je třeba implementovat test popsany v kapitole 4.3. Každý trojúhelník má ve Voronoiově diagramu jednu duální hranu e (viz obr. 5-7), jejíž dva koncové body w_1 a w_2 jsou tvořeny středy koulí opsaných dvěma tetrahedronům trojúhelník sdílejících.

Středy jsou již předpočtené pro každý tetrahedron z fáze tetrahedronizace a preprocessingu, proto je nalezení hrany e rychlé. Pak pro tuto hranu aplikujeme test z kapitoly 4.3 a pokud bude trojúhelník správný, hrana e bude procházet povrchem (s určitou tolerancí θ).



Obr. 5-7 : Střed koule opsané tetrahedronu V_1, V_2, V_3, V_4 je C_2 , tetrahedronu V_1, V_2, V_5, V_4 je C_1 . Tyto středy jsou zároveň koncové body hrany e z VD trojúhelníku V_1, V_2, V_4 .

Je třeba si dát pouze pozor na tetrahedrony se stěnou na konvexní obálce (viz obr, 5-8). První vrchol w_1 hrany e z VD stěny (trojúhelníku) z DT bude tvořen středem koule opsané tomuto tetrahedronu, ale druhý vrchol se bude nacházet v nekonečnu (Voronoiova buňka nebude uzavřená).



Obr. 5-8 : Neuzavřená Voronoiova buňka bodu na konvexní obálce.

Druhý vrchol není ale třeba počítat (šel by spočítat přibližně jako hodně vzdálený bod, protože známe počáteční bod hrany a směrový vektor), protože je jasné, že testem projde. Proto stačí aplikovat na bod w_1 jednoduchý test, který nám řekne, zda úhel $\alpha = \angle sw_1, sp$ je menší než $\pi/2 + \theta$ (viz. obr. 4-11).

Někdy se ovšem může stát, že testem projdou i trojúhelníky do povrchu nepatřící (často trojúhelníky ležící na konvexní obálce). Pozorováním jsem zjistil, že tyto trojúhelníky jsou velice úzké a dlouhé. Proto jsem ještě zavedl úhlo-hranové filtrování, které dokáže tyto trojúhelníky odstranit. Bohužel je třeba zadat jako vstup tohoto filtrování dva parametry. Prvním parametrem je velikost minimálního úhlu v trojúhelníku. Pokud trojúhelník bude mít svůj minimální úhel větší než tento úhel, již se dál netestuje, protože nebude úzký. V opačném případě se ještě musí otestovat velikost jeho dvou hran u vrcholu s nejmenším úhlem. Pokud velikost obou hran bude větší, než vzdálenost nejbližšího bodu (ten je spočten z fáze výpočtu hodnoty r -vzorku) tomuto vrcholu vynásobená multiplifikátorem (druhý parametr filtru), trojúhelník bude navíc i dlouhý a musí se zrušit.

Díky těmto dvěma parametrům je možno vlastně kontrolovat, co to znamená, že trojúhelník je dlouhý a úzký. Avšak jako většina heuristik funguje tento filtr jen v některých případech, kdy může značně vylepšit povrch, zatímco v jiných případech může naopak smazat i správné trojúhelníky.

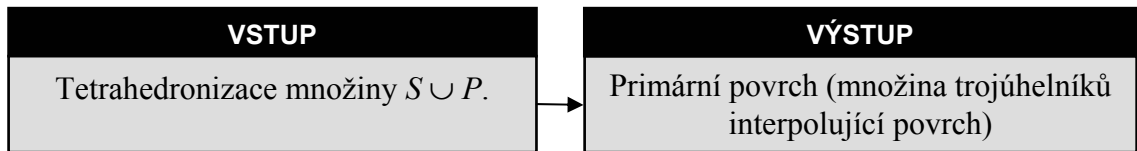


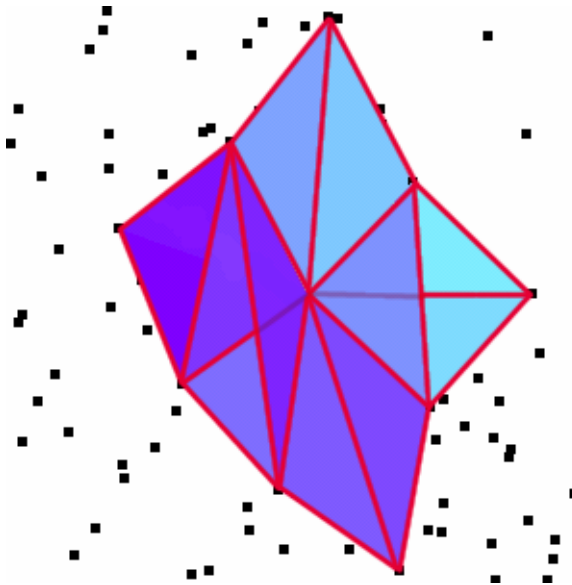
Schéma 5-5 : Vstup a výstup kroku generování primárního povrchu.

5.5. Extrakce manifoldu

Primární povrch vygenerovaný z tetrahedronů může však obsahovat ještě velké množství přebytečných trojúhelníků (viz obr. 5-9), navíc i ty dobré nemusí mít správnou orientaci (normály směrem ven z objektu). Proto je důležité primární povrch znova zkontrolovat, zajistit zrušení nadbytečných trojúhelníků a správnou orientaci zbylých (metoda *TCRUST.Create_Triangle_Mesh*).

Extrakce manifoldu se dá rozdělit na několik dílčích kroků :

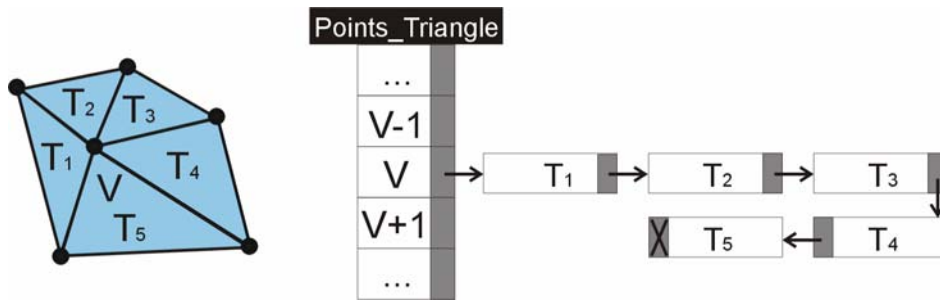
- Inicializace pomocných struktur.
- Vyhledání startovacích trojúhelníků.
- Vlastní extrakce vyhledáváním do hloubky.



Obr. 5-8 : Přebytečné (překrývající) se trojúhelníky (reálná data).

5.5.1. Pomocné struktury

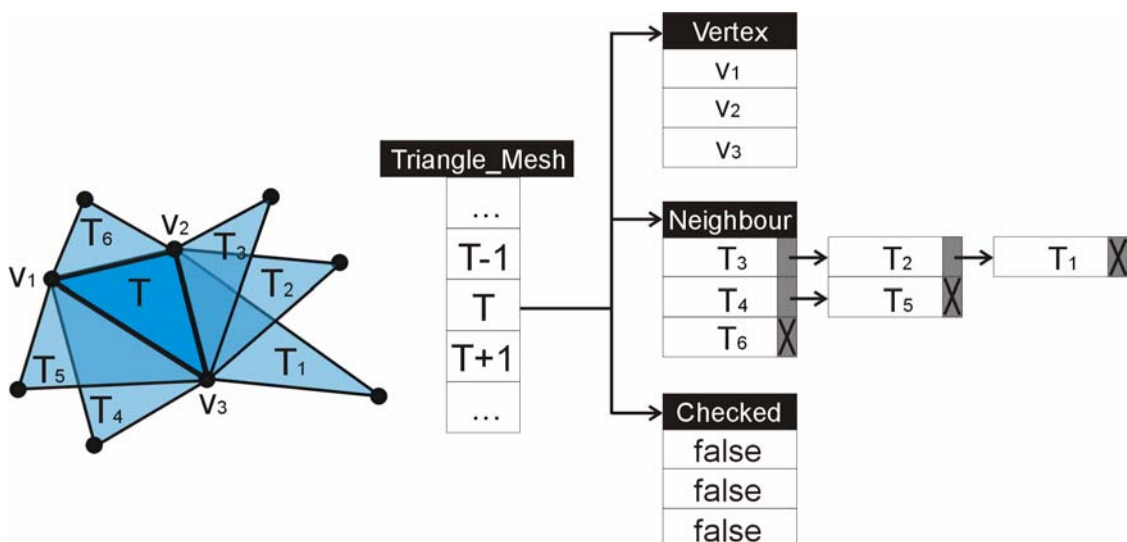
Pro urychlení se používají dvě pomocné datové struktury (struktura *Triangle_Mesh* a *Points_Triangle*). Před vlastním vyhledáváním se provede jejich inicializace (metoda *TCRUST.Find_Neighbours*). Struktura *Points_Triangle* (viz obr. 5-10) obsahuje pro každý bod množiny S spojový seznam trojúhelníků incidujících s tímto vrcholem (jeden z vrcholů trojúhelníka je tento vrchol). Struktura *Triangle_Mesh* (viz obr. 5-11) pak obsahuje pro každý trojúhelník soupis jeho vrcholů (položka *Vertex*), pomocný příznak pro vyhledávání děr (položka *Checked*) a hlavně spojový seznam všech sousedních trojúhelníků pro každou hranu (hrana je jednoznačně určena protějším vrcholem v trojúhelníku).


 Obr. 5-9 : Datová struktura *Points_Triangle*.

Vytvoření struktury *Points_Triangle* je jednoduché, stačí projít všechny trojúhelníky z primárního povrchu, vrcholy trojúhelníka vzít jako indexy do tabulky *Points_Triangle* a přidat prvek do spojového seznamu. Tak bude mít vytvoření struktury algoritmickou složitost $O(R)$, kde R je počet trojúhelníků.

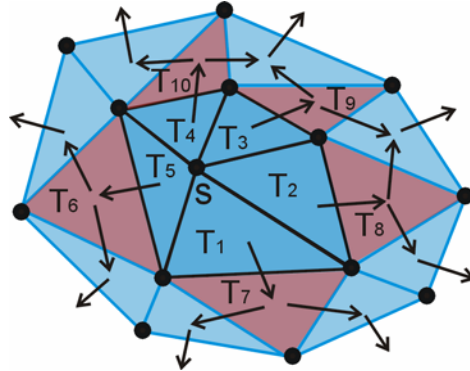
Pro vytvoření struktury *Triangle_Mesh* (v podstatě trojúhelníková síť s vícenásobnou sousedností) můžeme využít již existující struktury *Points_Triangle*. Pokud bychom síť vytvářeli přímo, algoritmus by měl složitost $O(R^2)$, protože pro každý trojúhelník bychom museli vyhledávat sousední trojúhelníky v celé síti. S pomocí *Points_Triangle* však algoritmická složitost klesne na $O(cN)$ ($N = |S|$ = počet bodů vstupní množiny), protože pro každý bod z množiny S budeme sousednost vytvářet pouze přes jeho incidující trojúhelníky (na obr. 5-12 bychom tedy vyhledávali sousedy pěti trojúhelníků $T_1 \dots T_5$).

Pokud označíme symbolem q počet trojúhelníků incidujících s nějakým bodem, pak konstanta c se bude rovnat $2 \cdot q^2$, protože musíme pro každý trojúhelník projít všechny jeho možné sousedy. Pro pravidelně navzorkované body však většinou inciduje každý bod s čtyřmi až osmi trojúhelníky a q pro nepravidelné vzorkování není q příliš vysoké.


 Obr. 5-10 : Datová struktura *Triangle_Mesh*.

5.5.2. Vyhledání startovacích trojúhelníků

Vlastní extrakce manifoldu funguje technikou prohledávání do hloubky. Při startu prohledávání musíme vyhledat v již existujícím primárním povrchu použitím pomocných struktur trojúhelníky, které jsou v pořádku, a o kterých můžeme stoprocentně říci, že patří do manifoldu (viz obr. 5-12).



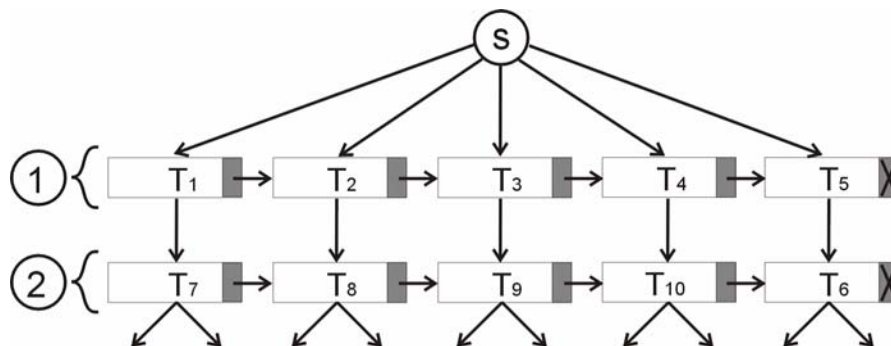
Obr. 5-11 : Extrakce manifoldu, trojúhelníky incidující s bodem s jsou startovací.

Startovací trojúhelníky nalezneme za pomoci tzv. deštníčku. Řekneme, že bod s má správný deštníček v případě, že incidující trojúhelníky jsou navzájem se nepřekrývající (nepřekrývá se jejich průmět na rovinu definovanou bodem s a jeho normálou). Normálu máme ale správně určenou pouze na bodech umístěných na konvexní obálce. U bodů uvnitř konvexní obálky nemůžeme určit, přestože máme přibližné normálové vektory povrchu určeny pomocí pólů, který směr ukazuje ven z povrchu.

Proto stačí najít na konvexní obálce takový bod, pro který platí, že má správný deštníček. Ve vlastní implementaci (funkce *TCRUST.Has_Point_Nice_Fan*) jsem ale použil podstatné zjednodušení, které využívá sousednosti. Pro bod s na konvexní obálce stačí zkontrolovat, jestli všechny trojúhelníky s ním incidující mají na hraně, kde jeden z vrcholů je bod s , maximálně jednoho souseda. V důsledku toho odpadne poměrně složité matematické testování.

5.5.3. Vlastní extrakce

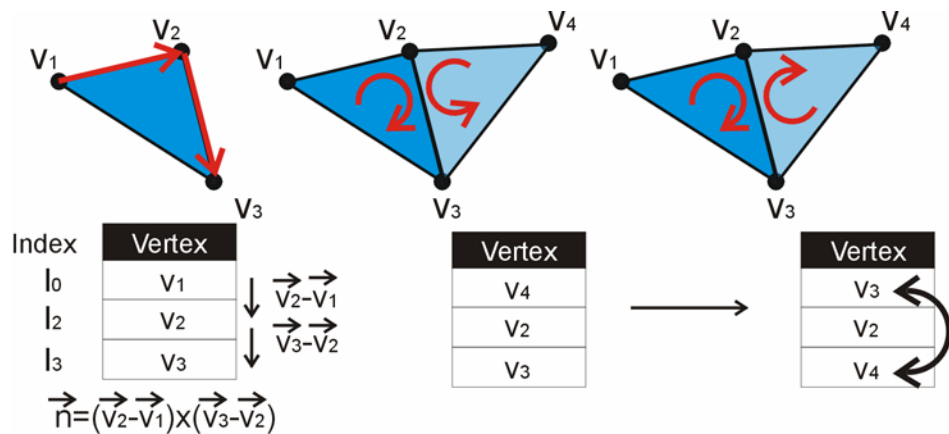
Po nalezení startovacích trojúhelníků určíme jejich orientaci tak, aby jejich normálové vektory ukazovaly ven z povrchu, nastavíme pro tyto trojúhelníky příznak, že patří do manifoldu, a uložíme je do pomocného spojového seznamu (představující vlastně jednu úroveň stromu). Strom nepotřebujeme mít k dispozici celý, stačí pouze aktuální úroveň a úroveň následující.



Obr. 5-12 : První dvě úrovně stromu pro příklad na obr. 5-12.

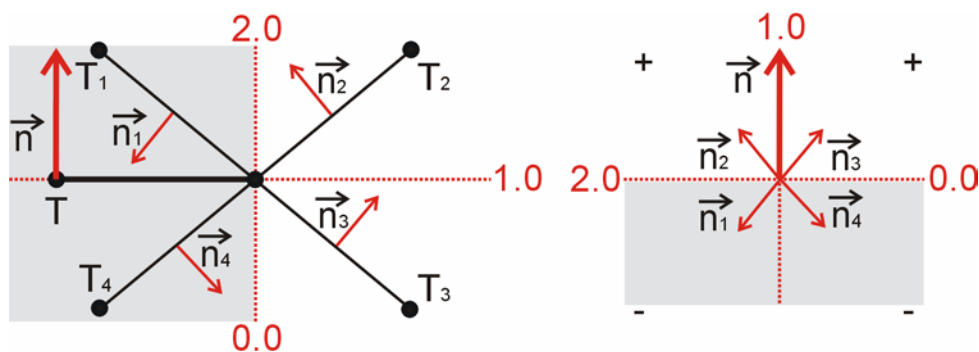
Po vytvoření úrovně stromu pak pro každý trojúhelník z této úrovně musíme naorientovat jeho ještě nezpracované sousedy (nemají nastavený příznak, že patří do manifoldu).

Orientace vychází vždy z již zpracovaného trojúhelníka (viz obr. 5-14). Vrcholy jsou uloženy v poli *Vertex* (ze struktury *Triangle_Mesh*) a výpočet normály probíhá tak, že se spočte vektorový součin vektorů *Vertex [1] - Vertex [0]* a *Vertex [2] - Vertex [1]* (tedy $V_2 - V_1$ a $V_3 - V_2$). Sousední trojúhelník však může mít pořadí vrcholů v poli *Vertex* takové, že tento výpočet bude generovat normálu ukazující opačným směrem. Proto je nutné prohodit v tomto poli vrcholy tak, aby normály ukazovaly stejným směrem. Po tomto prohození je ale nutné upravit ještě sousednosti v poli *Neighbours*.



Obr. 5-13 : Výpočet normál sousedního trojúhelníka (V_1, V_2, V_3 je již zpracovaný)

Po orientaci normál spočteme skalární součin normálových vektorů zpracovaného trojúhelníka a jeho sousedů (podle obr. 5-15). Tím dostaneme cosinus úhlu vektory sevřeného. Pokud znaménko cosinu bude záporné (T_1, T_2), trojúhelník nemusíme brát v úvahu, protože by na povrchu vznikla ostrá hrana. V případě kladného výsledku pak provedeme transformaci cosinu trojúhelníků umístěné ve stejném kvadrantu jako T_2 z intervalu $\langle 0.0, 1.0 \rangle$ na interval $\langle 1.0, 2.0 \rangle$. Pak vybereme jako správný trojúhelník ten, jehož „cosinus“ bude největší a nastavíme jeho příznak (funkce *TCRUST.Check_Triangle_Neighbours*).



Obr. 5-14 : Výpočet úhlů normál, vlevo pro trojúhelníky, vpravo pro jejich normály (T je zpracovaný trojúhelník s normálou n , T_i možná pozice sousedů, n_i jejich normály)

Zbylé trojúhelníky na hraně je pak nutné odstranit z celé triangulace (ze struktur *Triangle_Mesh* a *Point_Triangle*). Někdy se však může stát (zvláště, pokud nejsou body pravidelně navzorkované), že na hraně zpracovaného trojúhelníka bude jeden ze sousedů již označen (dostaneme se do něj přes jiný trojúhelník). Potom musíme celý test přeskočit a rovnou tento trojúhelník vzít jako správný.

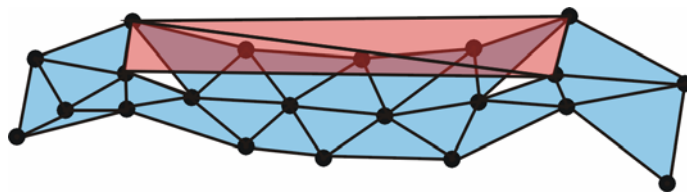
Nově označený trojúhelník vložíme do spojového seznamu další úrovně stromu a po ukončení prohledávání aktuální úrovně můžeme přejít na tuto další úroveň (viz obr. 5-16). Hledání končí v případě, že všechny trojúhelníky ze sítě mají nastaven příznak, že patří do manifoldu (trojúhelníky tam nepatřící jsou z triangulace v průběhu vyhledávání smazány).



Obr. 5-15 : Průběh extrakce, barevně jsou odlišeny různé úrovně stromu.

Bohužel tento krok algoritmu provázejí také největší problémy. Může se stát, že na určitých místech dojde při extrakci k lokálním chybám díky tomu, že z trsu trojúhelníků přiléhající k jedné hraně se vybere nesprávný trojúhelník (přestože podle kritéria výběru je správný).

Tyto špatné trojúhelníky (viz obr. 5-17) se objevují hlavně na povrchu, který má nejméně v jednom směru malou křivost. Při tetrahedronizaci zde vznikají velice úzké tetrahedrony, jejichž všechny stěny (trojúhelníky) projdou extrakcí primárního povrchu. Více se tento problém objevuje u dvouprůchodového algoritmu, jelikož díky dvojnásobné tetrahedronizaci je v primárním povrchu více trojúhelníků.



Obr. 5-16 : Transparentní trojúhelníky jsou nesprávné. Pod nimi vznikají v manifoldu díry.

Druhou chybou, která se může objevit, je prohození orientace trojúhelníků (viz obr. 5-18). Ta nastává např. v případě, že povrch není hladký a v některém svém místě má ostrý úhel. Často se objevuje v kombinaci s předchozí chybou, kdy na těchto místech může dojít k výběru špatného trojúhelníka (kritérium nepovoluje ostré úhly sousedních trojúhelníků) jdoucího dovnitř objektu.

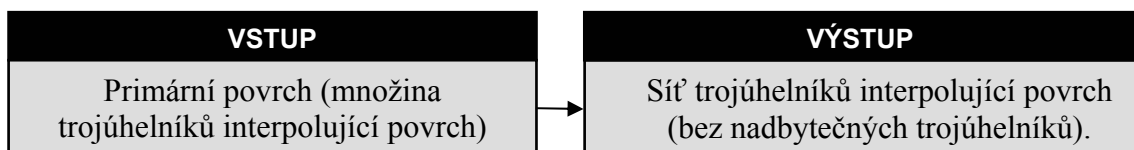
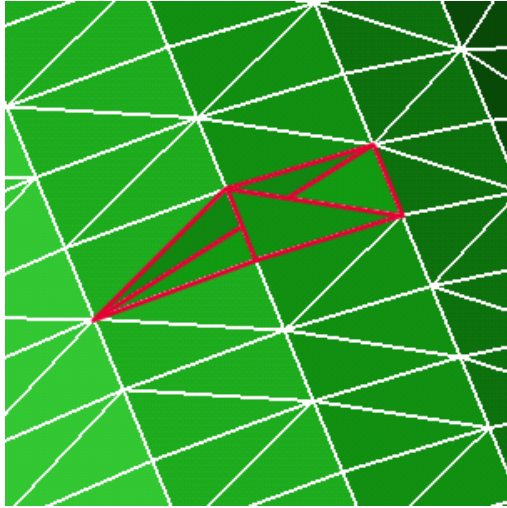
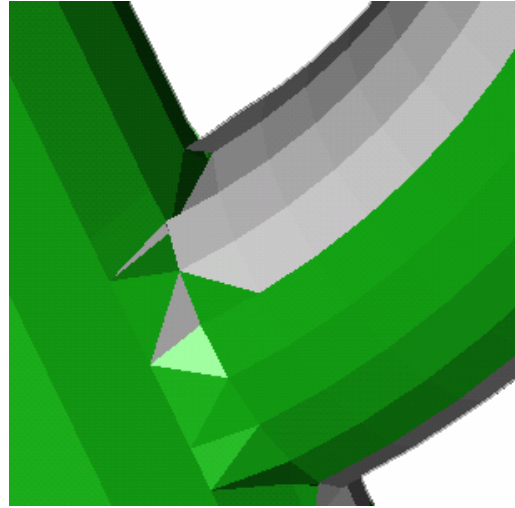


Schéma 5-6 : Vstup a výstup kroku extrakce manifoldu.



Obr. 5-17 : Překrývající se trojúhelníky



Obr. 5-18 : Prohození orientace.

5.6. Nalezení a triangulace děr

Oba původní algoritmy končí v okamžiku nalezení manifoldu. Nicméně se v něm mohou objevit díry, které vzniknou díky podvzorkování určitých oblastí nebo nevhodnou filtrací. Pozorováním jsem zjistil, že hodně těchto děr má jednoduchý tvar (např. chybějící trojúhelník), proto jsem do algoritmu zavedl jako poslední krok detekci těchto oblastí.

Opět je možné využít sousednosti, která proces detekce urychlí (metoda *TCRUST.Fill_Holes*). Prochází se celá síť a v případě nalezení trojúhelníka, který nemá na jedné své hraně souseda, se jeden z vrcholů této hrany označí jako startovací. V druhém vrcholu se pak hledá další trojúhelník, kterému také chybí u jedné z hran soused, a stejným způsobem se pokračuje dále až do okamžiku, kdy se vrátíme do startovacího vrcholu. Při detekci se ale může stát, že se narazí na trojúhelník, který nemá souseda na dvou hranách. Potom musíme pokračovat po takové hraně, která bude svírat s předchozí hranou nejmenší úhel, abychom dostali co možná nejmenší díru.

Detekované díry se mohou retriangulizovat, nicméně implementována je pouze retriangulace trojúhelníkové a čtyřúhelníkové díry¹⁵. Díry s větší velikostí (více hran než tři) jsou problémové, protože nemusí být konvexní. Navíc je otázka, jestli větší díry skutečně nemají v povrchu být.

Všechny nově vzniklé trojúhelníky se ukládají do zvláštního spojového seznamu, odkud se vloží do nové sítě až v posledním kroku při generování finálního povrchu.

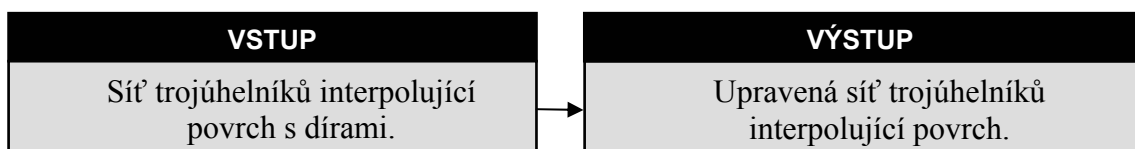
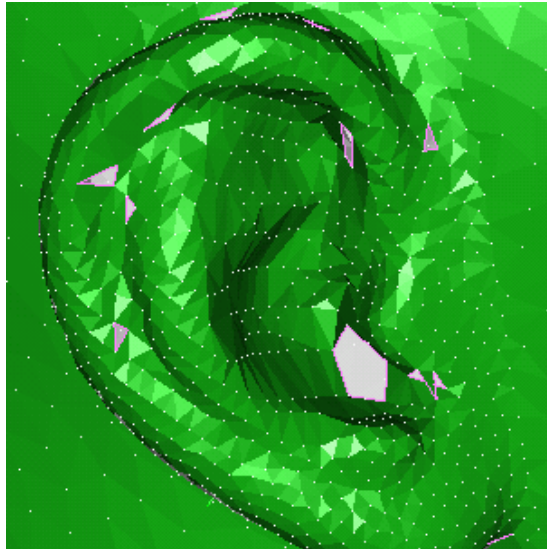


Schéma 5-7 : Vstup a výstup kroku hledání a triangulace děr.

¹⁵ Čtyřúhelníková díra také nemusí být vždy konvexní, ale pozorováním jsem zjistil, že tomu tak vždy je. Proto jsem implementoval i retriangulaci této díry, přestože by nemusela být vždy správně.



Obr. 5-19 : detekce děr

5.7. Generování finálního povrchu

Posledním krokem je vytvoření finálního povrchu. Ten je tvořen všemi zbylými trojúhelníky ze sítě a navíc trojúhelníky, které vzniknou při detekci děr. Sice by bylo možné nově vzniklé trojúhelníky vkládat do sítě rovnou, ale pro potřeby MVE se stejně musí použít pro uložení trojúhelníkové sítě jiná struktura, proto je lepší vytvořit celou síť znova.

Vytvoření probíhá podobným způsobem jako inicializace (vytvoření sítě s vícenásobnou sousedností) struktur v kroku extrakce manifoldu. Struktura *Points_Triangle* se použije stejně, ale místo struktury *Triangle_Mesh* pro vícenásobnou sousednost se použije struktura *PMVE_Triangle* z systému MVE.

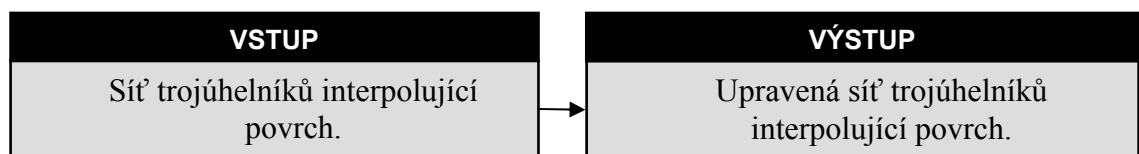


Schéma 5-8 : Vstup a výstup kroku generování finálního povrchu.

6. Experimenty a výsledky

Testování časové spotřeby algoritmů proběhlo na jednoprocessorovém počítači Intel Pentium III 450 MHz s 1 GB paměti pod operačním systémem Microsoft Windows 2000. Bohužel nebylo k dispozici smysluplné porovnání výsledků (časových) s Ninou Amentou, jelikož její testování proběhlo na strojích značně odlišných od našeho. Část testovacích dat byla vybrána tak, aby se shodovala s daty autorky, a zbytek tak, aby měla různou velikost (ale aby alespoň částečně splňovala kritérium hladkosti).

6.1. Analýza algoritmické složitosti

Částečná analýza byla již provedena v implementační části, proto zde provedu shrnutí dosažených výsledků. Nejprve je nutné ujednotit si značení :

- $N = |S|$ = počet bodů vstupní množiny S .
- T_1 = počet tetrahedronů po první tetrahedronizaci množiny S .
- T_s = průměrný počet tetrahedronů incidujících s jedním bodem.
- $N_p = |P|$ = počet pólů z první tetrahedronizace.
- T_2 = počet tetrahedronů po druhé tetrahedronizaci množiny $S \cup P$.
- R_1 = počet trojúhelníků primárního povrchu.
- R_2 = počet trojúhelníku manifoldu.

Použitá **Delaunayova tetrahedronizace** má očekávanou algoritmickou složitost $O(N \log N)$ (asymptotickou $O(N^2)$, nicméně díky randomizaci vstupu se blíží první uvedené hodnotě). Případní druhá tetrahedronizace pak probíhá přibližně s trojnásobkem bodů původní velikosti množiny.

Preprocessing hledání středů koulí opsaných tetrahedronů je záležitost lineární, stejně jako vyhledání jednoho incidujícího tetrahedronu k bodu, a má složitost $O(T_1)$. Poměr počtu tetrahedronů a velikosti vstupní množiny je průměrně 7.

Výpočet pólů se provádí pro každý bod, přičemž se u každého bodu musí prohledat množina tetrahedronů s ním incidující. Algoritmická složitost je tedy $O(N T_s)$. Hodnota T_s je závislá na distribuci bodů, málo členité modely s hodně body na konvexní obálce budou mít tuto hodnotu menší než modely více členité. Průměrně vychází tato hodnota kolem 26 tetrahedronů na jeden bod.

Pro **generování primárního povrchu** je nutné projít celou množinu tetrahedronů T_1 nebo T_2 a pro každý tetrahedron pak provést test jeho stěn. Proto složitost bude opět $O(T_1)$, nebo $O(T_2)$ v závislosti na druhu algoritmu.

Extrakce manifoldu se skládá z několika částí. Inicializace datových struktur (viz kapitola 5.5.1) se dá provést v $O(R_I)$ pro vytvoření incidence bodu s trojúhelníky a vytvoření trojúhelníkové sítě s vícenásobnou sousedností je opět lineární $O(N)$. Nalezení startovacího bodu spočívá v prostém procházení množiny S a vyhledání takového bodu, který je na konvexní obálce a má splněnou podmínku startovacího bodu, což má opět složitost $O(N)$ (jelikož každý bod má nastaven příznak, zda je na konvexní obálce už z kroku výpočtu pólů). Vlastní extrakce pak probíhá díky existující sousednosti v lineárním čase závislém na R , tedy v $O(R_I)$. Obecně platí, že dvouprůchodový algoritmus generuje větší počet trojúhelníků než jednopřechodový (pro testovací data to bylo přibližně 1.2x).

Rychlost **Detekce děr** je lineární a závislá na množství trojúhelníků v extrahovaném manifoldu, pro které bude platit $R_2 \leq R_I$. Pak složitost bude $O(R_2)$.

Celková algoritmická složitost je závislá na maximální složitosti jednoho z kroků. Tím je Delaunayova tetrahedronizace, protože ostatní kroky jsou lineární. Proto je složitost v $O(N \log N)$ v očekávaném případě a $O(N^2)$ v nejhorším případě, který se však během experimentů neobjevil.

6.2. Numerická robustnost

Nejvíce numericky nestabilní je krok výpočtu Delaunayovy tetrahedronizace. První experimenty s algoritmy proběhly s užitím tetrahedronizace, která používala standardní výpočetní prostředky pro výpočty. V průběhu vývoje jsem pak přešel na knihovnu, která používá přesnou aritmetiku za použití speciální knihovny [42] [43]. Výsledky rekonstrukce se sice moc nelišily, ale jelikož počet generovaných tetrahedronů je obou verzích odlišný, dá se předpokládat, že na určitých typech dat se mohou rozdíly vyskytnout.

Další numericky nestabilní krok je výpočet středů koulí opsaných tetrahedronům. Naštěstí přesná pozice není důležitá, protože díky výběru pólů (jako nejvzdálenější vrcholy Voronoiovy buňky) je tato chyba vzhledem ke vzdálenosti k bodu zanedbatelná.

6.3. Experimenty

6.3.1. Dvouprůchodový algoritmus

Experimenty Niny Amenty byly provedeny na SGI Onyx s 512 MB paměti. Její výsledky prezentuje následující tabulka (převzaty z [32]):

Model	čas (min)	Počet bodů
Femur	2	939
Golf Club	12	16864
Foot	15	20021
Bunny	23	35947

Tab. 6-1 : Výsledky Niny Amenty s dvouprůchodovým algoritmem.

Pro vlastní testy jsem měl k dispozici dvě datové množiny jako měla autorka, a to králíčka „Bunny“ a golfovou hůl „Golf club“. Dále jsem použil tři množiny, na kterých prezentovala své výstupy („Hotdog“, „Hypersheet“ a „Manneguin“) a zbylých pět jsem doplnil tak, aby škála velikostí byla co možná největší. Algoritmus byl nastaven tak,

aby počítal všechny kroky, stejně jako originální algoritmus Niny Amenty, přestože výsledky rekonstrukce nebyly vždy uspokojivé.

hotdog	1	knot	3	manneguain	5	teeth	7	mummy	9
hypersheet	2	engine	4	club	6	bunny	8	bone	10

Tab. 6-2 : Použité datové množiny a jejich index v dalších tabulkách.

Tabulka 6-3 obsahuje základní údaje o rekonstrukci. Poměr R_2/R_1 udává poměr trojúhelníků z manifoldu (R_2) a trojúhelníků z primárního povrchu (R_1). Je zde vidět poměrně velký rozdíl, průměrně (pouze z těchto datových množin) téměř jedna třetina trojúhelníků z primárního povrchu je nadbytečná.

	1	2	3	4	5	6	7	8	9	10
N	1748	6752	10000	11444	12772	16864	29166	35967	46969	68537
P	2656	12762	18839	18980	23740	31494	56696	70321	92186	134082
R1	5163	17038	28598	41378	29866	42583	74533	112723	122362	183584
R2	3467	11454	17846	14906	23044	32515	54510	70535	90611	135145
R2/R1	0,672	0,6723	0,624	0,3602	0,7716	0,7636	0,7314	0,62574	0,74052	0,73615
průměr R2/R1										0,66969

Tab. 6-3 : Základní údaje o datových množinách a jejich rekonstrukci (N = počet bodů, P = počet pólů, R₁ = počet trojúhelníků v primárním povrchu, R₂ = počet trojúhelníků v manifoldu)

U „engine“ (sloupec 4) je však patrný značný rozdíl od ostatních dat. Je to pravděpodobně díky tomu, že se skládá ze tří částí, které jsou úzké (obrázek viz přílohy). Proto vznikají trojúhelníky v primárním povrchu uvnitř těchto částí. Naštěstí při extrakci manifoldu jsou správně ohodnoceny jako špatné a ve finálním povrchu se již nevyskytují. Následující tabulka 6-4 pak obsahuje údaje o rychlostech výpočtu.

	1	2	3	4	5	6	7	8	9	10
Del	1,313	5,17	8,118	7,626	8,905	12,179	21,685	26,098	35,821	51,87
Prp	0,173	0,587	1,051	0,748	0,96	1,278	2,305	2,847	3,789	5,354
Pol	0,134	0,48	0,911	0,592	0,733	0,946	1,763	2,238	2,765	3,961
Del2	2,714	12,597	19,569	19,855	24,295	31,593	58,414	73,364	95,427	141,937
Njb	0,056	0,209	0,39	0,263	0,332	0,438	0,803	1,01	1,111	1,587
Ppo	0,038	0,135	0,252	0,299	0,245	0,332	0,633	0,833	0,922	1,343
Sou	0,094	0,242	0,429	0,739	0,52	0,771	1,121	1,747	1,734	2,464
Ema	0,025	0,091	0,166	0,167	0,15	0,214	0,403	0,654	0,62	0,97
D+R	0,003	0,011	0,052	0,011	0,032	0,033	0,093	0,059	0,087	0,098
OUT	0,048	0,162	0,27	0,132	0,321	0,483	0,907	1,088	1,384	1,848

Tab. 6-4 : Naměřené časy jednotlivých kroků v sekundách (Del = první DT, Prp = preprocessing, Pol = výpočet pólů, Del2 = druhá DT, Njb = výpočet nejbližšího souseda, Ppo = primární povrch, Sou = konstrukce sousednosti, Ema = extrakce manifoldu, D+R = detekce a triangulace děr, OUT = generování finálního povrchu)

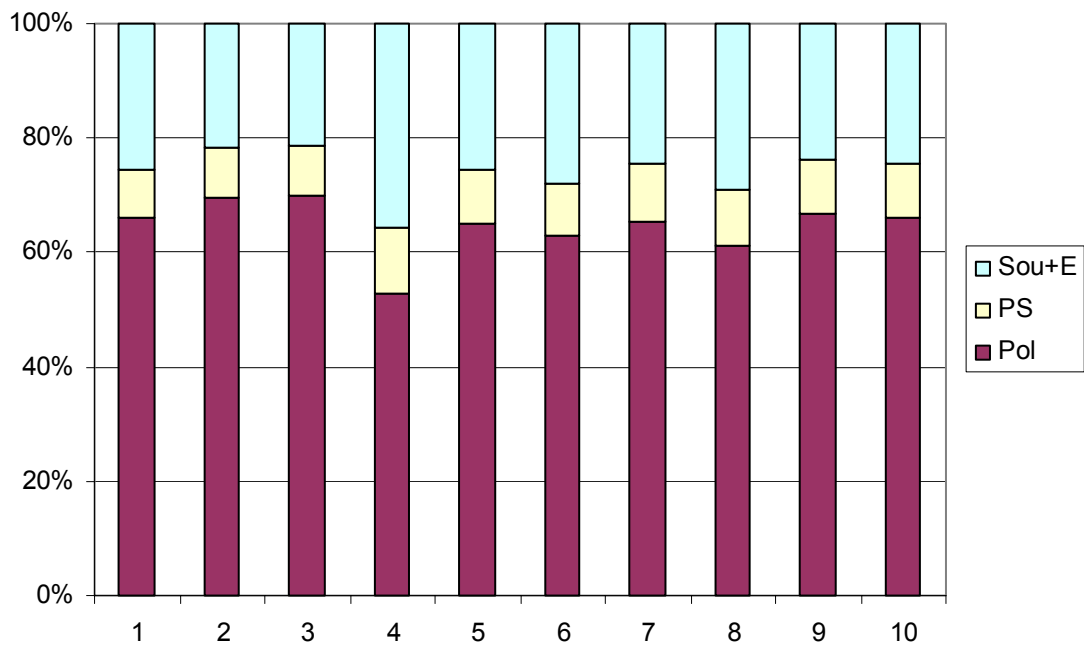
Z výsledků je patrné, že největší čas (přibližně 90%) se stráví na Delaunayově tetrahedronizaci. Ostatní kroky mají velice malou časovou spotřebu a na celkové časové spotřebě se příliš nepodílí.

Pro lepší srovnání uvádím ještě tabulku 6-5 s přepočítanými časy tak, aby korespondovali přesně s kroky Niny Amenty. Nejsou tam uvedeny všechny kroky jako v předchozí tabulce, protože výpočet nejbližšího souseda, detekce a triangulace děr a generování finálního povrchu se v původním algoritmu nevyskytují.

	1	2	3	4	5	6	7	8	9	10
Del12	4,027	17,767	27,687	27,481	33,2	43,772	80,099	99,462	131,248	193,807
Pol	0,307	1,067	1,962	1,34	1,693	2,224	4,068	5,085	6,554	9,315
PS	0,038	0,135	0,252	0,299	0,245	0,332	0,633	0,833	0,922	1,343
Sou+E	0,119	0,333	0,595	0,906	0,67	0,985	1,524	2,401	2,354	3,434
Σ-DT	0,464	1,535	2,809	2,545	2,608	3,541	6,225	8,319	9,83	14,092
Σ	4,491	19,302	30,496	30,026	35,808	47,313	86,324	107,781	141,078	207,899

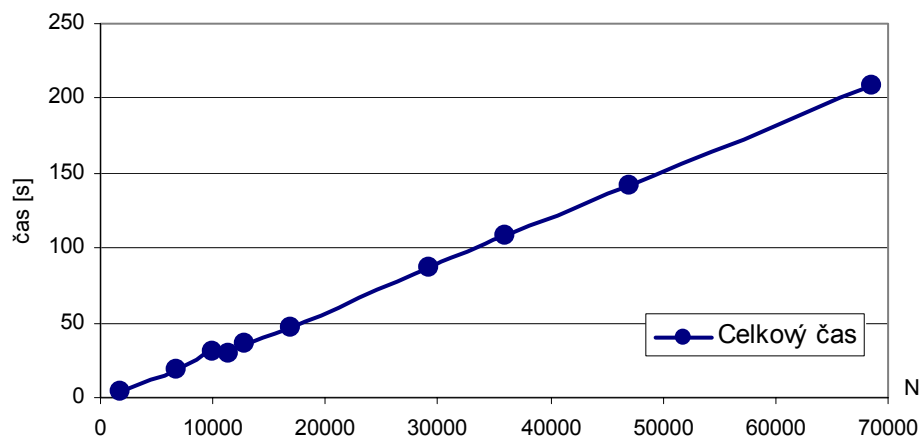
Tab. 6-5 : Přepočet časů jako u Niny Amenty (Del12 = obě DT, Pol = výpočet pólů + preprocessing, PS = primární povrch, Sou+E = konstrukce sousednosti + extrakce manifoldu, Σ -DT = součet kroků bez DT, Σ = celkový čas)

Pro srovnání časů jednotlivých kroků z tabulky 6-5 bez Delaunayovy tetrahedronizace uvádím následující graf 6-1.



Graf 6-1 : Podíl jednotlivých kroků na celkovém čase (bez DT)

A nakonec celkový graf 6-2 rychlosti dvouprůchodového algoritmu.



Graf 6-2 : Celkový čas rekonstrukce v závislosti na velikosti datové množiny

6.3.2. Jednoprůchodový algoritmus

Nina Amenta použila pro své testování dvě datové množiny „Foot“ a „Golf Club“ (k dispozici máme pouze „Golf Club“). Bohužel testy provedla opět na stroji (300 Mhz SUN s 256 MB paměti), se kterým nemůžeme udělat přímé srovnání.

Model	čas (s)	Počet bodů
Foot	153	20021
Golf Club	122	16864

Tab. 6-6 : Výsledky Niny Amenty s dvouprůchodovým algoritmem.

Přestože výsledky nejsou plně srovnatelné ani s jejím prvním algoritmem, můžeme obecně říci, že je mnohem rychlejší. Pro testování jsem použil stejné datové množiny jako u dvouprůchodové verze. Následující tabulka 6-7 uvádí základní údaje o rekonstrukci.

	1	2	3	4	5	6	7	8	9	10
N	1748	6752	10000	11444	12772	16864	29166	35967	46969	68537
R1	3590	13027	21386	31737	26218	34642	62586	83095	98334	145611
R2	3488	12516	19975	22441	25420	33698	58220	71815	93929	137213
R2/R1	0,972	0,9608	0,934	0,7071	0,9696	0,9727	0,9302	0,86425	0,9552	0,94233
průměr R2/R1										0,92078

Tab. 6-7 : Základní údaje o datových množinách a jejich rekonstrukci u jednoprůchodové verze (N = počet bodů, R₁ = počet trojúhelníků v primárním povrchu, R₂ = počet trojúhelníků v manifoldu)

Poměr R_2/R_1 hovoří jasně o tom, že počet vybraných trojúhelníků do primárního povrchu je mnohem nižší než u předchozího algoritmu a úbytek po extrakci manifoldu je přibližně osm procent. Rozdíl je samozřejmě způsoben odlišným kritériem výběru, navíc však je tento algoritmus více náchylný k díram na hranicích a v oblastech podzvorkování.

Hodně odlišný výsledek poměru má opět datová množina „engine“ (sloupec 4) kvůli stejnému problému jako dříve. Pro srovnání uvádím tabulku 6-8 rychlosti jednotlivých kroků.

	1	2	3	4	5	6	7	8	9	10
Del	1,257	4,978	7,802	7,409	8,413	11,538	20,561	24,775	34,024	48,942
Prp	0,166	0,599	1,088	0,77	0,974	1,283	2,32	2,891	3,841	5,422
Pol	0,131	0,439	0,876	0,592	0,679	0,868	1,62	2,067	2,672	3,843
Njb	0,059	0,219	0,42	0,278	0,29	0,459	0,839	1,065	1,378	1,974
Ppo	0,238	0,638	1,337	0,897	1,171	1,575	2,915	3,652	4,563	7,02
Sou	0,036	0,149	0,265	0,559	0,346	0,463	0,773	1,072	1,156	1,587
Ema	0,014	0,053	0,09	0,171	0,106	0,145	0,287	0,397	0,42	0,649
D+R	0,001	0,013	0,016	0,013	0,012	0,013	0,027	0,032	0,04	0,057
OUT	0,041	0,159	0,267	0,327	0,351	0,499	0,783	0,967	1,186	1,585

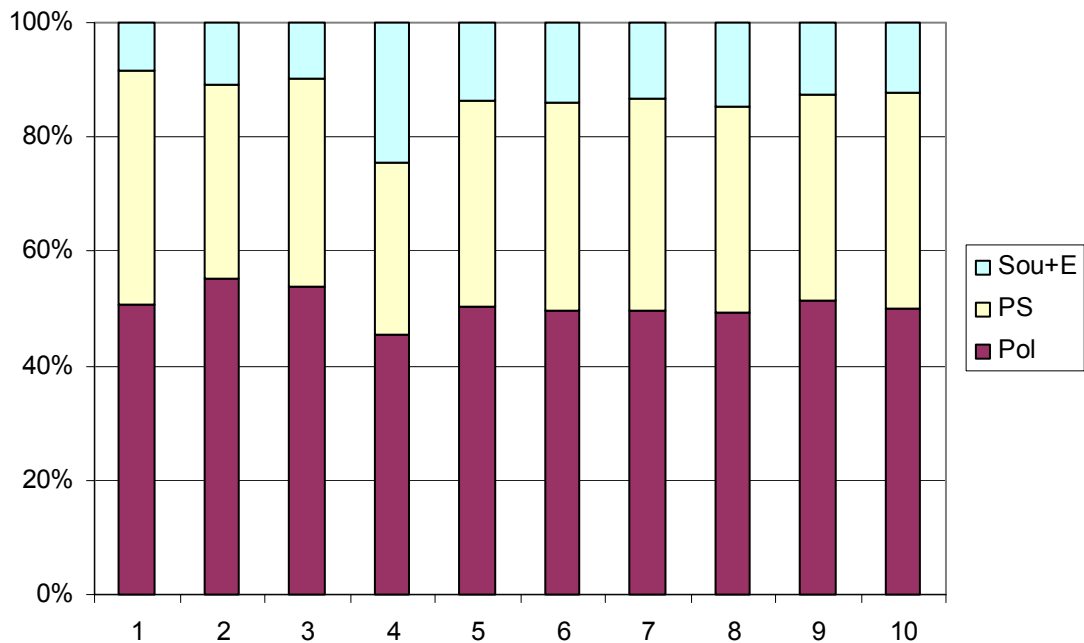
Tab. 6-8 : Naměřené časy jednotlivých časů v sekundách sekundách (Del = první DT, Prp = preprocessing, Pol = výpočet pólů, Njb = výpočet nejbližšího souseda, Ppo = primární povrch, Sou = konstrukce sousednosti, Ema = extrakce manifoldu, D+R = detekce a triangulace děr, OUT = generování finálního povrchu)

Výsledky ukazují jednoznačně na citelné urychlení, hlavně díky odstranění druhé tetrahedronizace. Po přepočtu výsledků na fáze jako používá Amenta pak bude tabulka (tab. 6-9) vypadat následovně :

	1	2	3	4	5	6	7	8	9	10
Del	1,257	4,978	7,802	7,409	8,413	11,538	20,561	24,775	34,024	48,942
Pol	0,297	1,038	1,964	1,362	1,653	2,151	3,94	4,958	6,513	9,265
PS	0,238	0,638	1,337	0,897	1,171	1,575	2,915	3,652	4,563	7,02
Sou+E	0,05	0,202	0,355	0,73	0,452	0,608	1,06	1,469	1,576	2,236
Σ-DT	0,585	1,878	3,656	2,989	3,276	4,334	7,915	10,079	12,652	18,521
Σ	1,842	6,856	11,458	10,398	11,689	15,872	28,476	34,854	46,676	67,463

Tab. 6-9 : Výsledky po přepočtu na kroky Amenty (Del = DT, Pol = výpočet pólů + preprocessing, PS = primární povrch, Sou+E = konstrukce sousednosti + extrakce manifoldu, Σ -DT = součet kroků bez DT, Σ = celkový čas)

Poměr jednotlivých kroků na celkové době výpočtu vypadá takto (graf 6-3)

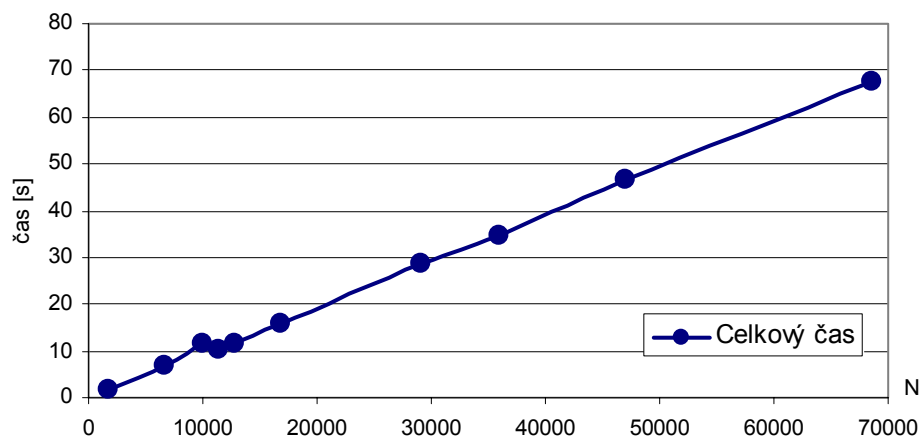


Graf 6-3 : Podíl jednotlivých kroků algoritmu na celkovém čase.

Na rozdíl od grafu 6-1 je zde patrný nárůst času u kroku výpočtu primárního povrchu. Je to z toho důvodu, že kritérium výběru trojúhelníka patřící do primárního povrchu je mnohem náročnější a tvoří v podstatě hlavní krok (zatímco u dvouprůchodového algoritmu to byla druhá tetrahedronizace, díky níž jsme měli trojúhelníky hrubě určené).

Je vidět také rozdíl datové množiny „engine“ (sloupec 4, u obou grafů 6-3 a 6-1), pro kterou značně narostl čas vytvoření sousednosti a extrakce manifoldu, protože poměr $R2/R1$ je menší než u ostatních množin.

Co se týče porovnání rychlosti ostatních kroků, jsou srovnatelné, protože hlavní rozdíly jsou právě v konstrukci primárního povrchu. Nakonec je ještě nutné uvést poslední graf 6-4, a to graf celkového času rekonstrukce.



Graf 6-4 : Celkový čas rekonstrukce v závislosti na velikosti datové množiny.

7. Závěr

Velké množství existujících algoritmů pro rekonstrukci napovídá, že se jedná o poměrně aktuální téma. Otázkou je, zda existuje nějaký univerzální algoritmus alespoň pro určité typy vstupních dat. Nové algoritmy většinou využívají již existujících a snaží se je různými způsoby vylepšit ať odstraněním jejich nedostatků či kombinací s jiným algoritmem.

Tato diplomová práce přinesla alespoň částečný přehled již existujících algoritmů a bližší pohled na dva z nich. Implementace bohužel nebyla bez problémů, převážně díky složité formulaci v původních materiálech, ale naštěstí po větších či menších obtížích se problémy podařilo překonat.

Obecně dává dvouprůchodový algoritmus lepší výsledky než jednaprůchodový, i když to není vždy pravidlem. Projevují se zde více chyby v podobě překrývajících se trojúhelníků, protože extrakce manifoldu pracuje s větším počtem trojúhelníků než jsou ve finálním povrchu.

Hlavní výhodou jednaprůchodového algoritmu je jeho rychlost. Rekonstrukce je správná v případech, které splňují teoretické požadavky metody. V případě, že tyto požadavky nejsou splněny (na rozdíl od dvouprůchodové metody, která je v těchto případech více robustní) se na výsledném povrchu objevují nedostatky v podobě děr, částečně také v podobě překrývajících se trojúhelníků. Ty zde vznikají většinou ale z jiného důvodu než u dvouprůchodové metody a podílejí se na nich právě díry v povrchu. Pak v kroku extrakce je vybrán špatný trojúhelník (protože kvůli díře tam žádný jiný není), který většinou prochází vnitřkem objektu. Druhotným efektem je pak možnost prohození orientace trojúhelníků, které incidují s tímto trojúhelníkem.

Další práce na algoritmu by proto měly směřovat nejprve k vylepšení stávající extrakce manifoldu, aby byl výsledný povrch bez chyb (překrývajících se trojúhelníků). Dobrou myšlenkou je ale vylepšení už vstupu tohoto kroku. Krok extrakce manifoldu dvouprůchodového algoritmu generuje mnohem větší počet trojúhelníků, které ale mohou být vadné, než jednaprůchodový, kde zase na druhou stranu některé trojúhelníky chybí. Ideální by proto bylo spojení obou algoritmů. V první fázi by se dal povrch spočítat jednaprůchodovým algoritmem a vygenerovaný povrch by se zkontroloval na existenci výše uvedených chyb (stačilo by např. zjistit, zda úhly trojúhelníků u vrcholu incidujícího s bodem tvoří plný úhel). V případě chybných míst pak pro tyto místa spočítat lokálně povrch jako u dvouprůchodového algoritmu a výsledky zkombinovat. Zajímavé by také bylo vymyslet způsob automatického určení vhodných parametrů metod.

Do rekonstrukce by bylo také vhodné zahrnout detekci hranic, popřípadě i oblastí podvzorkování. Otázkou je, jestli tyto oblasti retriangulizovat, protože se nedá jednoznačně říci, že oblast je podvzorkovaná, nebo se jedná o skutečnou díru v povrchu.

Co se týče srovnání s ostatními metodami, tak se nedá jednoznačně říci, která je lepší a horší. Každá má své pro a proti, jedna je rychlá, další má lepší výsledky ve vlastní rekonstrukci. Navíc autoři často nedávají své zdrojové kódy k dispozici, proto nelze porovnat ani jejich výstupy.

7.1. Seznam literatury

- [1] M.-E. Algorri and F. Schmitt. Surface reconstruction from unstructured 3D data. *Computer Graphic Forum*, 15(1):47-60, 1996
- [2] H.Hoppe, T.DeRose, T.Duchamp, M.Halstead, H.Jin, J.McDonald J.Schweitzer and W.Stuetzle, Piecewise smooth surface reconstruction, *Comput. Graphics Proceedings*, 1994, pp. 295-302 [annual conference series (SIGGRAPH '94)]
- [3] H.Hoppe, T.DeRose, T.Duchamp, J.McDonald and W.Stuetzle, Surface reconstruction from unorganized points, *Comput. Graphics* 26 (2) , 1992, 71-78
- [4] H.Hoppe, T.DeRose, T.Duchamp, J.McDonald and W.Stuetzle, Mesh optimization, *Comput. Graphics Proceedings*, 1993, pp. 16-26 [annual conference series (SIGGRAPH '93)]
- [5] W. E. Lorensen and H. E. Cline. Marching Cubes : A high resolution 3d surface reconstruction algorithm. *Computer Graphics*, 21(4):163-169, July 1987
- [6] F.Bernardini, C.Bajaj. A triangulation based. Sampling and reconstruction manifolds using α -shapes, 9th canadian conference on Computational Geometry, 1997, 193-198
- [7] H.Edelsbrunner . Weighted alpha shapes, 1992. Technical report UIUCDCS-R92-1760, Department of computer science, University of Illinois at Urbana-Champaign, Urbana, Illinois.
- [8] H.Edelsbrunner and E.P.Mücke, Three-dimensional alpha shapes, *ACM Trans. Graphics* 13, (1994), 43-72
- [9] E. P. Mücke.Shapes and implementations in three-dimensional geometry. PhD.thesis, Department of computer science, University of Illinois at Urbana-Champaign, Urbana, Illinois. September 1993.
- [10] D. Attali. R-regular shape reconstruction from unorganized points. In *ACM Symposium on Computational Geometry*, pages 248-253, 1997, Nice, France.
- [12] J.D.Boissonat, Geometric structures for three-dimensional shape representation, *ACM Trans.Graphics* 3, 1984, 266-286
- [13] F. Isselhard, G. Brunett and T. Schreiber. Polyhedral reconstruction of 3 D objects by tetrahedra removal. Technical report, Fachbereich Informatik, University of Kaiserslautern, Germany, February 1997, Internal report No. 288/97.
- [14] R. C. Veltkamp. Boundaries through scattered points of unknown density. *Graphics Models and Image Processing*, 57 (6):441-452,November 1995.
- [15] R. C. Veltkamp. Closed object boundaries from scattered points. In *Lecture Notes in Computer Science* 885. Springer Verlag, 1994.
- [16] T. Schreiber and G. Brunett. Approximating 3d objects from measured points. In *proceedings 30th ISATA*, Florence, Italy, 1997.
- [17] G. Roth and E. Wibowoo. An efficient volumetric metod for building closed triangular meshes from 3d images and point data. In *Graphios Interface '97*, pages 173-180, 1997.

- [18] E. Bittar, N. Tsingos and M.-P. Gascuel. Automatic reconstruction of unstructured data : Combining medial axis and implicit surfaces. *Computer Graphics forum*, 14(3):457-46, 1995. Proceedings of EUROGRAPHICS 95.
- [19] S. Muraki. Volumetric shape description of range data using “Blobby model”. *Computer Graphics*, pages 217-226, July 1991. Proceedings of SIGGRAPH 91.
- [20] D. Terzopoulos and D. Metaxas. Dynamic 3d models with local and global deformations : Deformable superquadrics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(7):703-714, July 1991.
- [21] D. Terzopoulos, A. Witkin and M. Kass. Constraints on deformable models. Recovering 3d shape and nongrid morión. *Artificial Inteligence* , 36:91-123, 1988.
- [22] J. V. Miller, D. E. Breen, W. E. Lorenzem, R. M O’Bara and M. J. Wozny. Geometrically deformed models : A Metod for extracting closed geometric models from volume data. *Computer Graphics*, pages 217-226, July 1991. Proceedings of SIGGRAPH 91.
- [23] R. Szeliski and D. Tonnesen. Surface modeling with oriented particle systems. *Computer Graphics* 26:185-194, July 1992, SIGGRAPH 92 Proceedings.
- [24] A. Baader and G. Hirzinger. A self organizing algorithm for multisensory surface reconstruction. In *International Conf. on Robotics and Intelligent systems IROS ’94*, September 1994, Munich Germany
- [25] A. Baader. Ein Umweltverfassungssystem für multisensorielle Montageroboter. PhD thesis, Universitat der Bundeswehr, Munich , Germany, 1995, Fortschrittberichte, VDI Reihe 8 Nr. 486, ISBN 3-18-3 48608-3, ISSN 0178-9546 (germany)
- [26] A. Baader and G. Hirzinger. Three dimensional surface reconstruction based on a self-organizing Kohonen map. In *Proc. 6th Int. Conf. Advan. Robotics*, pages 273-278, 1993 Tokyo.
- [27] R. Mencl. A graph based approach to surface reconstruction. *Computer Graphics forum*, 14(3): 445-456, 1995, Proceedings of EUROGRAPHICS 95, Maastricht , The Netherlands , August 28-September 1, 1995
- [28] R. Mencl and H. Müller. Graph based surface reconstruction using structures in scattered point sets. In *Proceedings of CGI 98 (Computer Graphics International)*, Hannover, Germany, June 22 – 26th 1998
- [29] P. Fua and P. T. Sander. From points to surfaces. In *Baba C. Vermuri* , editor, *Geometric Methods in Computer Vision*, pages 286-296. *Proc. SPIE Vol. 1970*, 1991.
- [30] P. Fua and P. T. Sander. Reconstructing surfaces from unstructured 3D points. In *Proc. Image Understanding Workshop*, pages 615-625, San Diego, CA, 1992
- [31] P. Fua and P. T. Sander. Segmenting unstructured 3d points into surfaces. In *G. Sandini*, editor, *Computer vision : ECCV 92*, *Proc. Second European Conference on Computer Vision*, pages 676 - 680, Santa Margherita Ligure, Italy, May 1992, Springer
- [32] N.Amenta, M.Bern, M.Kamvysselis. A new Voronoi based surface reconstruction algorithm. *SIGGRAPH 1998*, 415-421
- [33] N. Amenta and M.Bern. Surface reconstruction by Voronoi filtering. *Discr. Comput. Geom.* 22 (1999), 481-504
- [34] N. Amenta, S. Choi, T.K.Dey and N.Leekha. A simple algorithms for homeomorphic surface reconstruction. 16th. *Sympos. Comput. Geom.*, 2000
- [35] N. Amenta, M.Bern and D.Eppstein. The crust and β -skeleton : combinatorial curve reconstruction. *Graphical Models and Image Processing*, 60 (1998), 125-135
- [36] T.K.Dey and P.Kumar. A simple provable algorithm for curve reconstruction. *Proc.ACM-SIAM Sympo. Discr. Algorithms* (1999), 893-894

- [37] T.K.Dey, K. Mehlhorn and E. A. Ramos. Curve reconstruction: connecting dots with good reason. *Comput. Geom. Theory Appl.*, 15 (2000), 222-244.
- [38] I. Kolingerová: Rovinné triangulace, habilitační práce, Západočeská univerzita, Plzeň, 1999
- [39] J. Kohout. Paralelní Delaunayova triangulace ve 2D a 3D. Diplomová práce, Západočeská univerzita. Plzeň 2002.
- [40] R. Mencl and H. Müller. Interpolation and approximation of surfaces from three-dimensional scattered data points. *Eurographics 98*.
- [41] F. Bernardini and H. Rushmeier. The 3d model acquisition pipeline. *Eurographics 2000*.
- [42] J. R. Shewchuck. Robust adaptive floating-point geometric predicates. *Proceedings of the twelfth Annual Symposium on Computational Geometry, ACM, 1996*.
- [43] P. Maur. Delaunay triangulation in 3D. Technical report NO. DCSE/TR-2002-02, University of West Bohemia, Pilsen, January, 2002,.

Odkazy

- [URL1] <http://www.cs.utexas.edu/users/amenta>
- [URL2] <http://www.cis.ohio-state.edu/~tamaldey>
- [URL3] <http://herakles.zcu.cz/research.php>
- [URL4] <http://herakles.zcu.cz/people.php>
- [URL5] <http://www.nist.gov/dads/HTML/complexity.html>

8. Přílohy

8.1. Uživatelská dokumentace samostatného programu

Pro vyzkoušení jednorůchodového i dvouprůchodového algoritmu byl vytvořen samostatný program, který umožňuje nastavit základní parametry algoritmů a ve vlastním renderovacím okně vidět výstup algoritmů. Minimální systémové požadavky programu jsou :

- Systém Microsoft Windows NT, 2000, XP.
- Paměť 128 MB.
- CPU Pentium.
- Grafická karta podporující OpenGL.

Tyto požadavky jsou skutečně minimální a je jasné, že čím lepší počítač, tím lépe. Zvláště na paměť je program velice citlivý, protože Delaunayova tetrahedronizace má na ni velké nároky. Pro počítač s 1GB paměti je možné zpracovat kolem 230 000 bodů, pro počítač s 220MB kolem 50 000 bodů bez swapování na disk (počítáno i s pamětí pro systém). Pro vlastní rendering je také vhodné mít dobrou grafickou kartu (od Voodoo3 výše), protože pro větší modely vzniká poměrně velké množství trojúhelníků.

Pro svoji práci potřebuje kromě standardních systémových knihoven (DLL) i knihovnu pro přesnou aritmetiku („*exact.dll*“) a knihovnu vizuálních komponent („*vcl50.bpl*“).

Program, je řešen jako aplikace **SDI**¹⁶, což znamená, že má jedno hlavní okno (formulář), jehož zavřením se uzavře celá aplikace. Další okna jsou tvořena také jako samostatná, ale v případě jejich uzavření dojde pouze k jejich skrytí a mohou se znova objevit. V průběhu práce je možné pracovat s 7 okny (v závorce je jejich název).

- Hlavní okno („*surface reconstruction by Voronoi filtering*“).
- Okno pro nastavení parametrů algoritmů („*Algorithms options*“).
- Okno pro nastavení parametrů vizualizace („*Visualization options*“).
- Okno pro povolení zobrazování různých dat („*SH*“).
- Okno s informacemi („*Information window*“).
- Okno pro zobrazení průběhu výpočtu („*Computing process window*“).
- Okno pro zobrazení výpočtu („*Renderer*“).

¹⁶ SDI - single document interface (x MDI - multiple document interface)

8.1.1. Hlavní okno



Po startu aplikace se zobrazí v levém horním rohu obrazovky hlavní okno, z kterého je možné ovládat celý program. Aplikaci je možné ukončit stisknutím křížku v levém horním rohu. Přesunout se dá okno standardním způsobem myši chytutím názvu aplikace a tažením na určené místo.



Soubor je možné nahrát pomocí tohoto tlačítka. Formát souboru musí být textový, přičemž první řádek obsahuje celé číslo určující počet bodů v souboru. Další řádky pak obsahují vždy tři čísla (na každém řádku) určující souřadnice x y z bodu v prostoru. Pokud dojde v průběhu čtení souboru k chybě, pak v případě, že se načetly více než 4 body, se pokračuje dál ve výpočtu. V opačném případě je zhlášena chyba.



Po výpočtu povrchu je možné uložit výslednou trojúhelníkovou síť do souboru ve formátu **TRI**. Tento formát se používá hlavně v systému (**MVE**), ale díky své jednoduchosti jde snadno konvertovat do jiných formátů..



Pokud uživatel změní pouze parametry výpočtu a chce použít stejná data, stačí stisknout toto tlačítko a výpočet se provede s použitím dat původní tetrahedronizace (nepočítá se tedy znova a výpočet je mnohem rychlejší). Toto neplatí v případě, že uživatel vybere druhý typ algoritmu (dvou x jednorůchodový). Pak pro výpočet s použitím nově vybraného algoritmu musí znova data nahrát pomocí tlačítka „load“. Je to z důvodu, že oba algoritmy jsou v různých třídách. Nikdy ovšem nejsou vytvořeny obě najednou, vždy pouze jedna podle aktuálního nastavení metody, proto není možné mezi nimi nějakým způsobem sdílet data. Uživatel však toto omezení zaplatí pouze o málo delším časem výpočtu, nutným k nahrání nových bodů



Toto tlačítko slouží pro nastavení parametrů výpočtu. Po otevření nového okna („*Algorithms options*“) je možné nastavit typ algoritmu a hlavní parametry obou metod (zapnutí/vypnutí filtrování, nastavení úhlů...).



Po výpočtu se zobrazují výsledky v zvláštním OpenGL okně („*Renderer*“). Pro nastavení, jaká primitiva zobrazovat, se používá zvláštní okno („*SH*“, zkratka od „*show*“), které se zobrazí po stisknutí tohoto tlačítka.

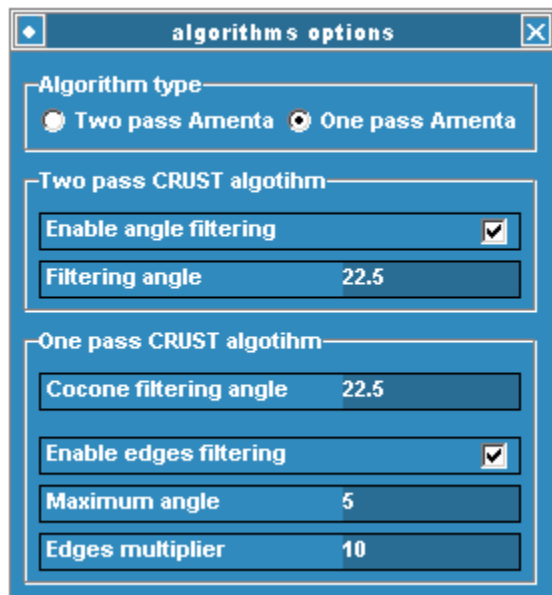


Při zobrazení se renderují různá grafická primitiva, které mají z počátku nastavené určité vizuální vlastnosti (barva, šířka...). Uživatel si však může ve zvláštním okně změnit nastavení těchto primitiv.






Stisknutím tohoto tlačítka je možné prohlédnout si ve zvláštním okně informace o průběhu výpočtu. Informace o rychlosti výpočtu je nutné však brát s rezervou, protože se do nich započítává i čas strávený vizualizací průběhu výpočtu. Pro měření v kapitole 6 bylo toto zobrazování interně vypnuto, proto nemá toto zkrácení vliv na prezentované výsledky.

8.1.2. Nastavení parametrů výpočtu



Pokud chce uživatel změnit parametry výpočtu, je nutné použít toto okno. Nejprve ovšem uvedu, jak toto okno používat, proto že to je společná pro všechna okna takto vypadající..

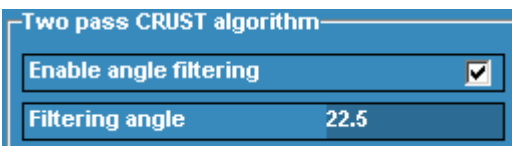
Všechna okna, která slouží pro nastavení, se dají přesouvat pomocí myši chytnutím a tažením za svůj název. Kliknutím na tlačítko  se dá okno uzavřít (protože se nejedná o hlavní okno aplikace, pouze zmizí a nechá se z hlavního okna znova otevřít). Pokud uživatel potřebuje, dá se okno přepínat režim okna pomocí stisknutí tlačítka  nebo . Pokud je tlačítko červené, značí, že je ve speciálním režimu,

kdy se nachází nade všemi okny dalších aplikací, i když s ním uživatel nepracuje. Využití je jednoduché, pokud je zobrazen renderer na celou obrazovku, pak by uživatel neměl přístup k ovládacím prvkům, avšak ve zvláštním režimu bude okno pro nastavení viditelné i v tomto případě.



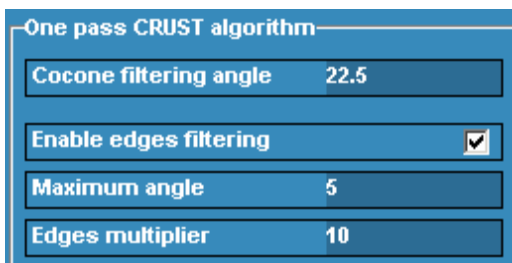
Rekonstrukce povrchu je možná dvěma metodami, a to jednorůchodovým CRUST algoritmem („*One pass Amenta*“), nebo

dvourůchodovým („*Two pass Amenta*“). Pokud uživatel zde vybere druhý algoritmus, než který je aktuální, pak pro provedení výpočtů s tímto algoritmem musí použít tlačítko „*load*“ a nahrát znova data.



Dvourůchodový algoritmus funguje automaticky, nicméně je někdy třeba použít i filtrování trojúhelníků pomocí jejich normál a úhlů („*Enable angle filtering*“). Zde je možné

filtrování zapnout a nastavit, jaký úhel („*Filtering angle*“) bude pro filtrování kritický („*být či nebýt*“).

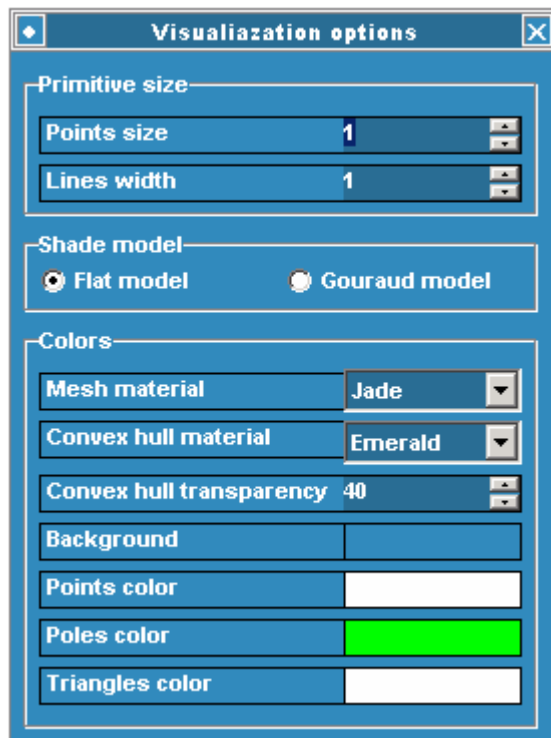


Jednorůchodový algoritmus potřebuje pro svůj vstup (kromě bodů) i jeden parametr. Tím je velikost úhlu (odchylka), jakou mohou mít body trojúhelníka od své duální Voronoiovy hrany (viz rozbor algoritmu). Tento úhel se dá nastavit v editovacím okně „*Cocone filtering angle*“.

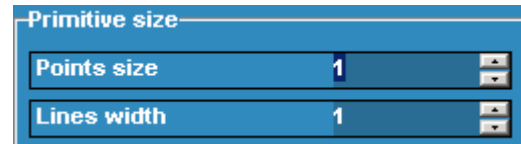
Dále se dá zapnout i další úhlo-hranové filtrování („*Enable edge filtering*“) a nastavit jeho dva parametry, maximální úhel pro filtrování („*Maximum angle*“) a hranový multiplikátor („*Edges multiplier*“).

Všechny parametry, které se zde dají nastavit, musí být v intervalu $\langle 0.0, 90.0 \rangle$.

8.1.3. Nastavení parametrů vizualizace



Stejně jako nastavení parametrů výpočtu se dají nastavit i možnosti zobrazování ve renderovacím okně.

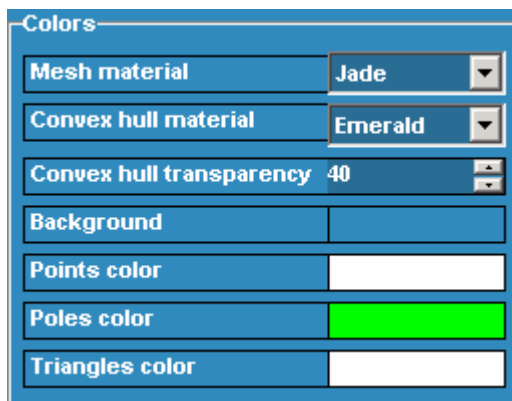


Při zobrazení bodů a pólů se dá nastavit jejich velikost pomocí „*Point size*“. Při zobrazení bodů větší velikosti než jedna se ale poměrně rychle ztrácí rychlost zobrazování, proto je velikost omezena na max. 20. Zobrazení větších bodů pak závisí na grafické kartě, některé (Voodoo3) dokáží zobrazit body jako kolečka, jiné (Riva TNT2) pouze jako čtverečky. Stejně jako nastavení bodů se dá dále nastavit šířka čar (při kreslení hran trojúhelníků) pomocí „*Lines width*“.



Gouraudova stínování („*Gouraud model*“), kdy každý trojúhelník má spočteny normály ve všech třech vrcholech. V případě nastavení Gouraudova stínování je možné vidět, kde se případně mění orientace trojúhelníků, protože na tomto místě budou mít normály jiný průběh (vůči svým sousedním bodům) a objeví se zde místa o skokové intenzitě. Stejně tak dopadnou i místa, kde se budou trojúhelníky překrývat.

Model osvětlení je možné volit jako klasický „*Flat model*“, kdy pro každý trojúhelník je spočtená jedna normála nebo pomocí



Poslední část slouží k nastavení barev a materiálu primitiv. Materiál se dá nastavit pro vykreslení samotného povrchu („*Mesh material*“) a konvexní obálky („*Convex hull material*“). Pro ni je možné ještě nastavit průhlednost („*Convex hull transparency*“) v rozmezí 0 (nejméně, obálka není vidět) až 100 (nejvíce, je vidět pouze obálka). Další čtyři tlačítka („*Background*“, „*Points color*“, „*Poles color*“, „*Triangles color*“) slouží pro nastavení barvy dalších primitiv. Po poklepnání

myši na barevný obdélníček se objeví standardní dialogové okno pro výběr barvy. Zde je možné vybrat si rovnou barvu z nabídky nebo si namíchat svou vlastní po stisknutí tlačítka „*Namíchat vlastní barvy*“ (platí pro české Windows, pro jinou jazykovou verzi bude mít tlačítko jazykově odlišný text).

8.1.4. Povolení zobrazování různých dat



V renderovacím okně je možné zobrazit až šest primitiv. Jaké zobrazit se nastavuje zde přepnutím tlačítek¹⁷.



Body (vstupní množina bodů) se zobrazí stisknutím tohoto tlačítka. Pokud je tlačítko (platí pro všechny tlačítka) více modré, pak se body již zobrazují a po stisknutí se zobrazovat přestanou.



Toto tlačítko slouží pro přepnutí zobrazování pólů. Pokud byl výpočet proveden pomocí dvouprůchodové metody, pak bude pólů přibližně dvojnásobek původního počtu bodů, jinak jich bude stejně.



Standardně se zobrazuje pouze trojúhelníková síť. Pro zobrazení i hran trojúhelníků se musí použít toto tlačítko. Je to výhodné např. v případě, kdy se zobrazuje pomocí Gouraudova stínování, protože hranice mezi trojúhelníky nejsou zřetelné.



Posunutí modelu od počátku je možné hrubě zobrazit pomocí tohoto tlačítka. Pak budou (nebo nebudou) zobrazeny základní osy (v počátku souřadného systému) a minimax box modelu.



Toto tlačítko povoluje či zakazuje zobrazení konvexní obálky. Přestože se dá nastavit její transparentnost na 0, i pak se zobrazuje (účastní se procesu renderování, i když není vidět), proto je možné ji zde úplně vypnout.



Poslední tlačítko pak slouží pro zapnutí nebo vypnutí zobrazení celého spočteného povrchu. Jedná se o nejnáročnější (na zobrazení) operaci celého rendereru.

8.1.5. Informační okno

Zde je možné získat informace o průběhu celého výpočtu. Sekce „*Source file*“ obsahuje základní údaje o souboru. Sekce „*Computing process info*“ pak údaje o vlastním průběhu, např. kolik trojúhelníků bylo vytvořeno v primárním povrchu a kolik z nich prošlo do výstupu. V poslední sekci „*Computing times (sec)*“ jsou časy jednotlivých kroků algoritmu.

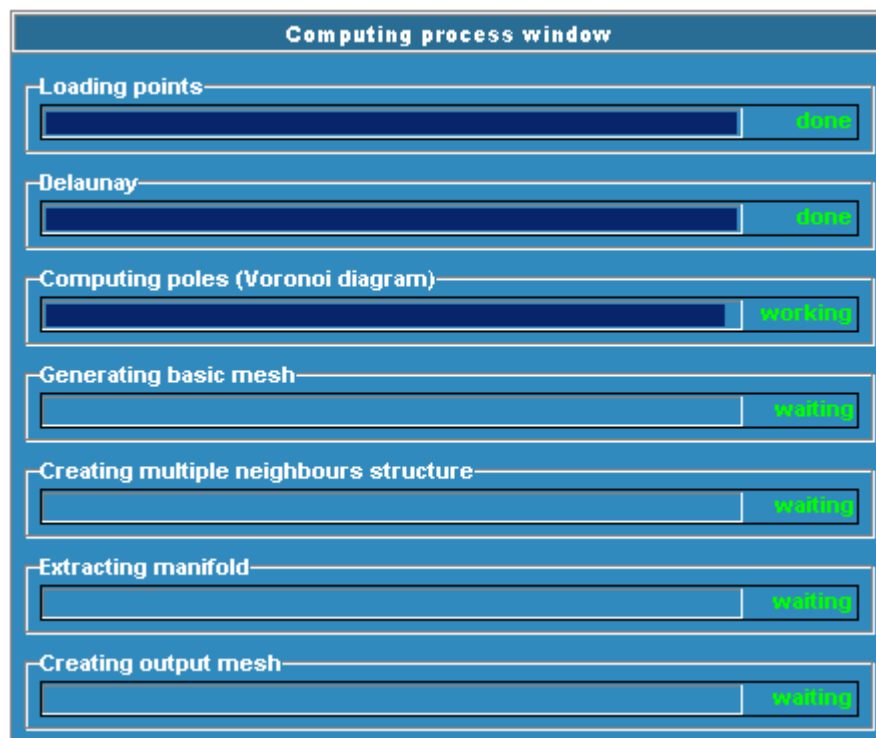
Tyto časy nejsou přesné z hlediska časování algoritmu, protože obsahují i čas potřebný k zobrazení okna, které informuje o průběhu výpočtu. Z hlediska uživatele naopak přesné jsou, protože přesně tak dlouho si uživatel počká, než se mu data zobrazí.

¹⁷ Název tohoto okna je od „show“.



8.1.6. Okno o průběhu výpočtu

Toto okno se objevuje pouze v průběhu výpočtu a není možné s ním nic dělat. Slouží pouze pro informaci uživateli, že se něco děje.

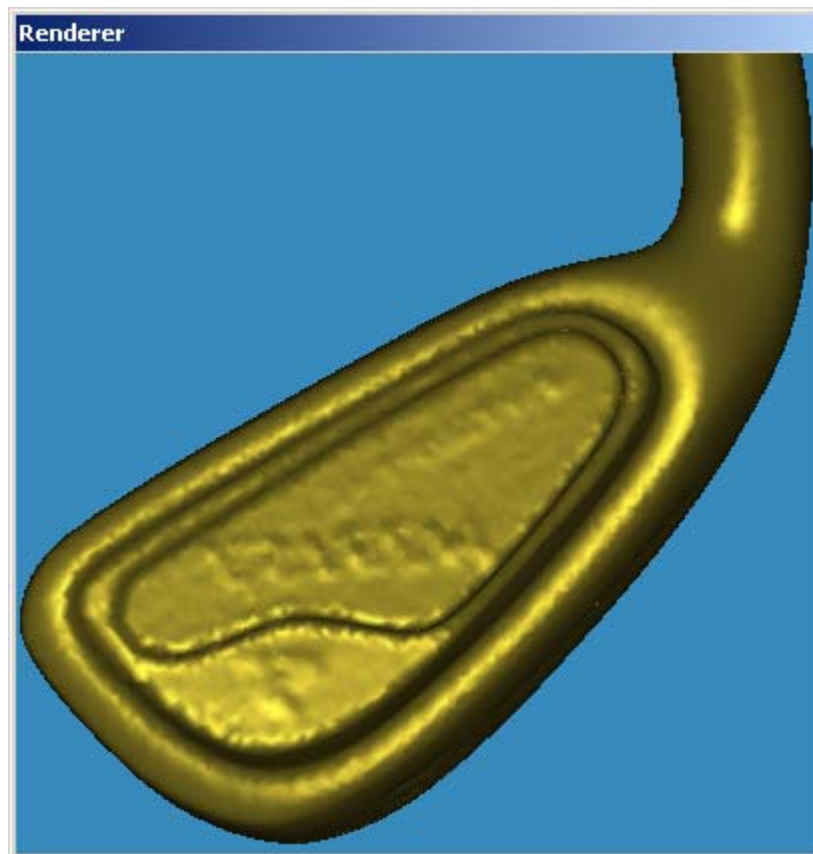


8.1.7. Renderer

Toto okno slouží pro prezentaci výstupu. Na rozdíl od ostatních oken je možné ho podle potřeby zvětšovat nebo zmenšovat. Nemá sice žádná další tlačítka na ovládání, ale je možné ho maximalizovat či vrátit do původní velikosti dvojitým poklepáním na jeho název (ale i pokud je maximalizováno, dá se přesunout a získat tak přístup k prvkům v oknech pod).

Model se dá ovládat pouze pomocí myši. Levé tlačítko slouží pro rotaci, pravé pro přesun a prostřední pro přibližování a oddalování. Oddálení je omezeno na maximálně desetinásobek původní velikosti modelu (který zpočátku vyplňuje okno ideálně) a přiblížení na tisícinásobek.

Renderování je řešeno pomocí ortogonální projekce, proto nedochází k žádnému perspektivnímu zkreslení. Při zobrazování je navíc použito jedno světlo bílé barvy, jehož zdroj se nachází v nekonečnu směrem za kamerou a svítí souběžně s kamerou.



Na závěr je ještě nutné říci, že všechny ovládací prvky v celé aplikaci mají svou kontextovou nápovědu, která je zobrazena, pokud uživatel najede myší na určitý prvek a počká tam určitý okamžik.

Shows window for enabling and disabling various types of rendering.

surface reconstruction by Voronoi filtering

load save run options show visual info

points poles edges axis c.hull mesh

Runs the actual type (that was set before last "Load" command) of the algorithm without computing time expensive Delaunay tetrahedrization.

Visualization options

- Primitive size: [Slider]
- Points size: [Slider: 1]
- Lines width: [Slider: 1]
- Shade model:
 - Flat model
 - Gouraud model

algorithms options

- Algorithm type:
 - Two pass Amenta
 - One pass Amenta
- Two pass CRUST algorithm
 - Enable angle filtering:
 - Filtering angle: 22.5
- One pass CRUST algorithm
 - Curvature intensity angle: 22.5
 - Enable edges filtering:
 - Maximum angle: 5
 - Edges multiplier: 10

Information window

Source file: **011410A7A0001.gifs**

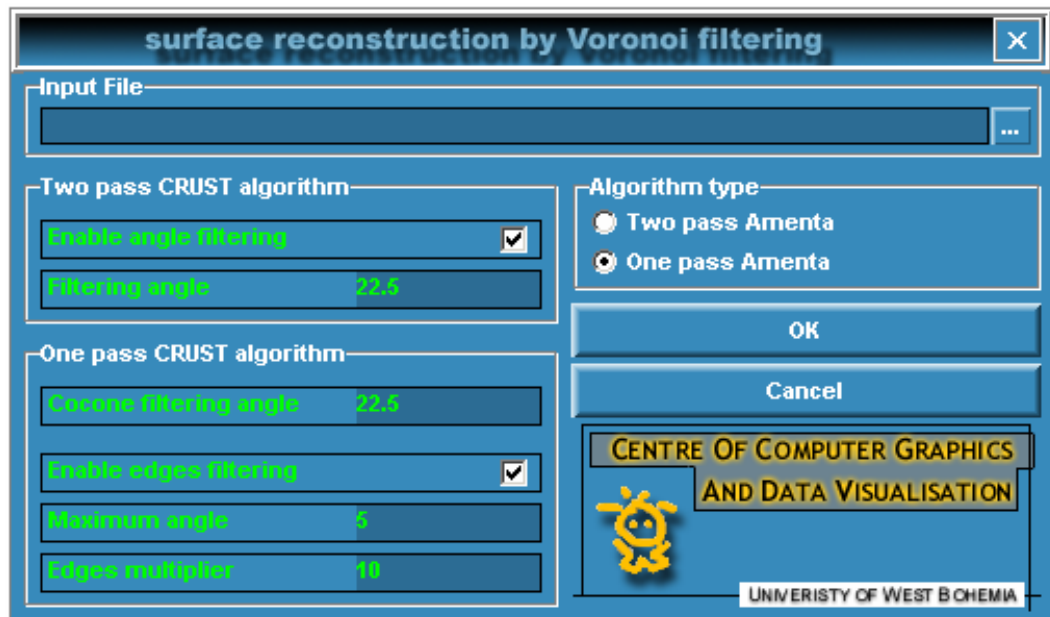
Source points: **10000**

Computing process info

Computing times (sec)	
Delaunay	2.535
Preprocessing	0.345
Poles computing	0.874
Delaunay 2	0.000
Closest points	0.335
Basic mesh	4.358
Multiple neighbours	0.248
Manifold extraction	0.003
Holes search	0.033
Output mesh	0.254
All (without delaunay)	4.108
All	11.243

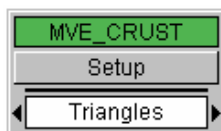
Centre of Computer Graphics and Data Visualisation
UNIVERSITY OF WEST BOHEMIA
http://www.klms.zcu.cz, (c) 2002 mmw@students.zcu.cz, kelling@kld.zcu.cz

8.2. Uživatelská dokumentace MVE modulu



Modul do MVE má stejnou grafickou úpravu jako samostatný program. Nejdůležitější položkou je položka „*Input file*“. Zde je nutné zadat název souboru, který bude obsahovat vstupní data. Tento soubor se vybere pomocí standardního dialogu, který se otevře po stisknutí tlačítka „...“ a není možné jej do editovacího okna vepsat ručně.

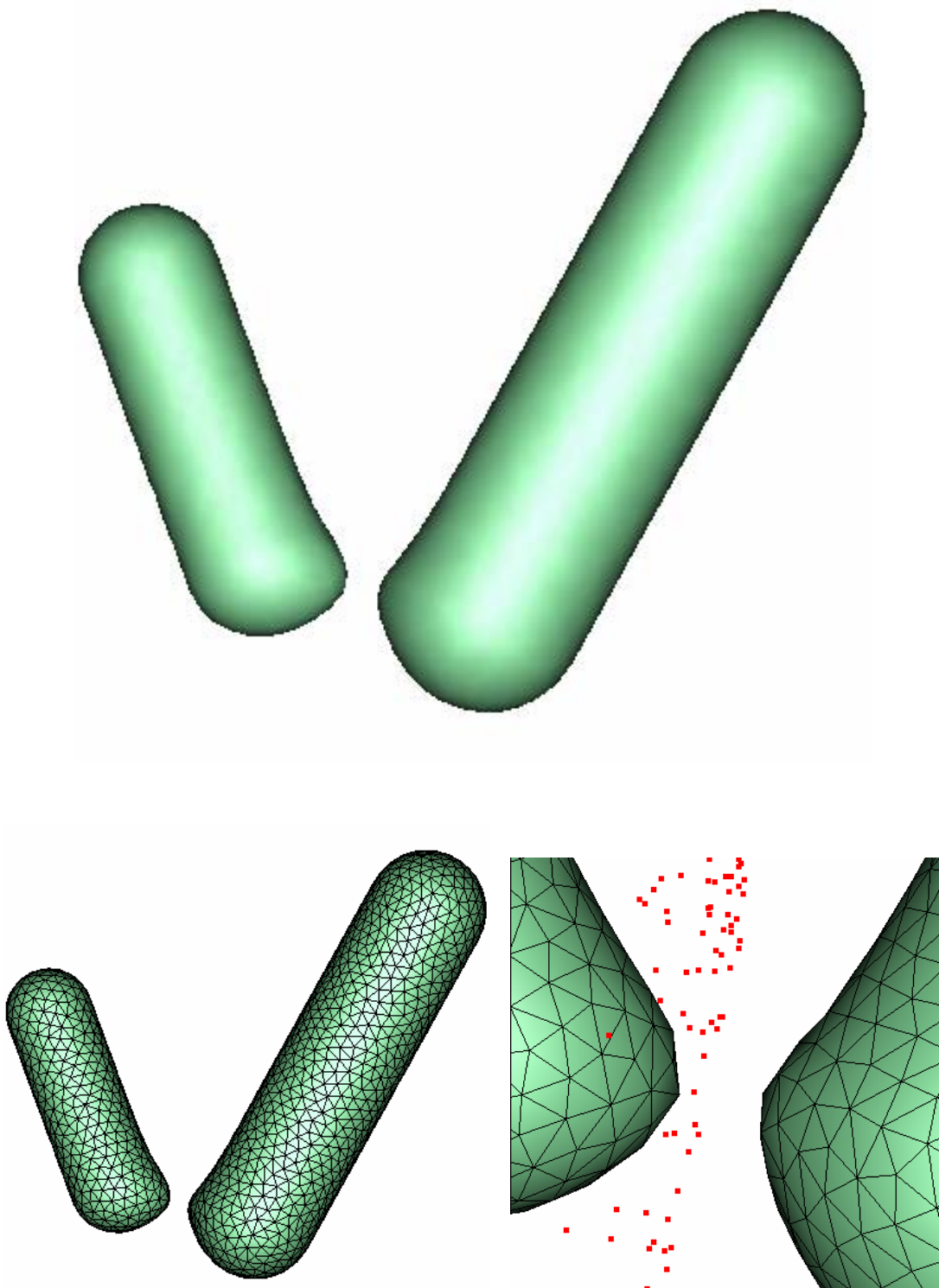
Další parametry jsou již stejné jako u samostatného programu, proto nemá smysl je dál rozebírat. Položka „*TWO Pass CRUST algorithm*“ a „*One pass CRUST algorithm*“ je již probrána v kapitole 8.1.2, stejně jako položka „*Algorithm type*“. Po nastavení parametrů se výpočet spustí tlačítkem „*OK*“, zrušit se může tlačítkem „*Cancel*“.



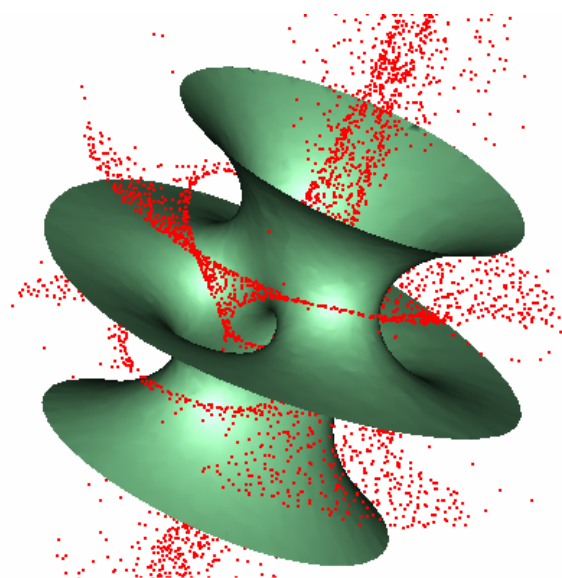
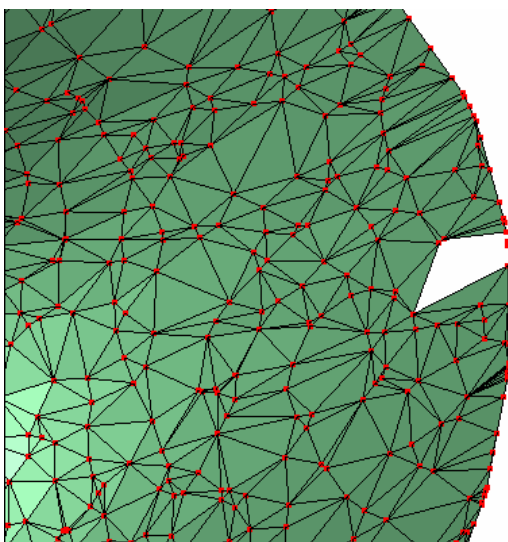
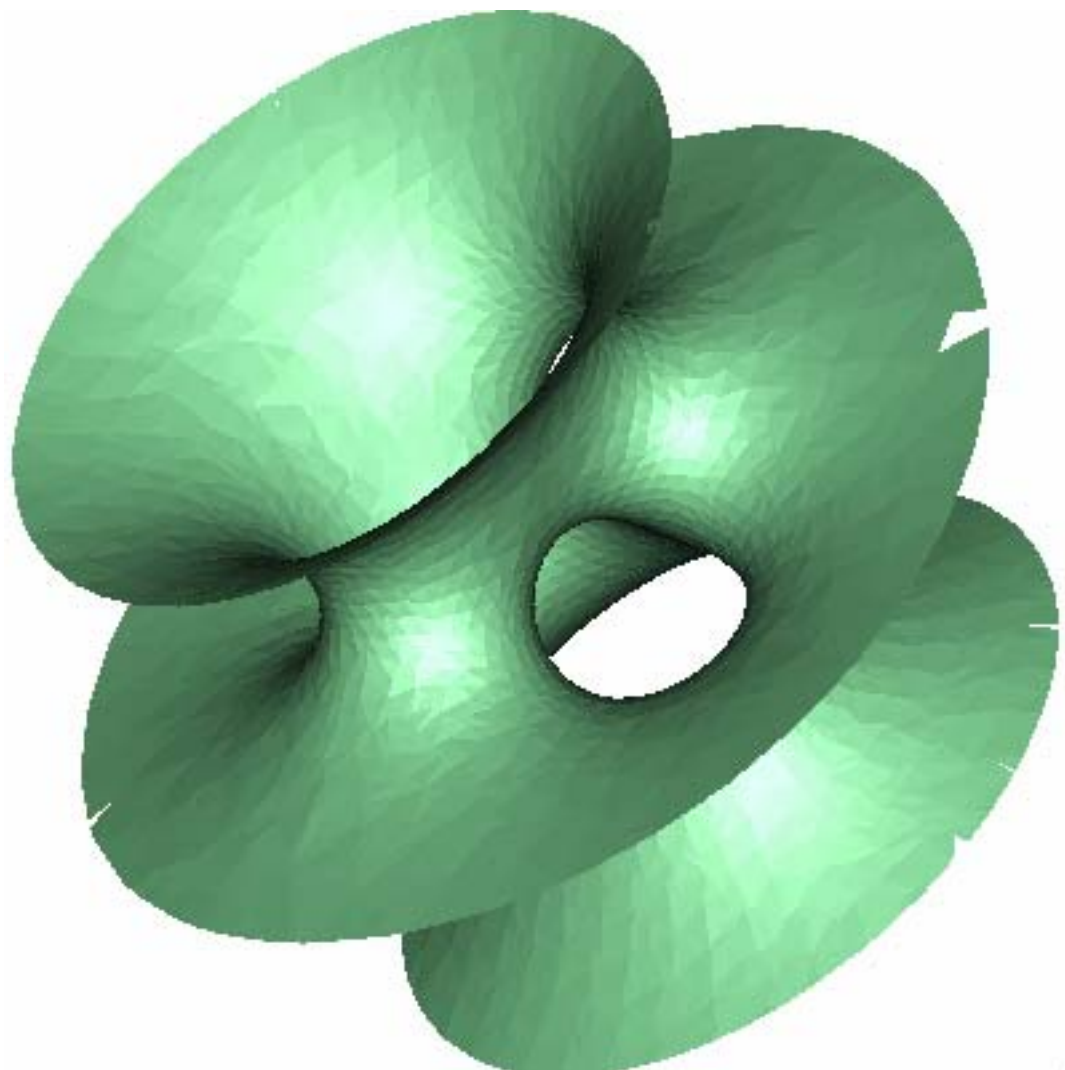
Ve systému MVE vypadá modul takto. Nemá žádný vstup, protože si sám nahrává body ze souboru. Výstupem je trojúhelníková síť, která se dá použít v dalších modulech. Přestože má modul tlačítko „*Setup*“, po jeho stisknutí se nic nestane. Je to z toho důvodu, že před modulem nemůže být žádný jiný (nemá vstup) a proto stačí, když se formulář pro nastavení se spustí až po zapnutí výpočtu.

8.3. Výstupy

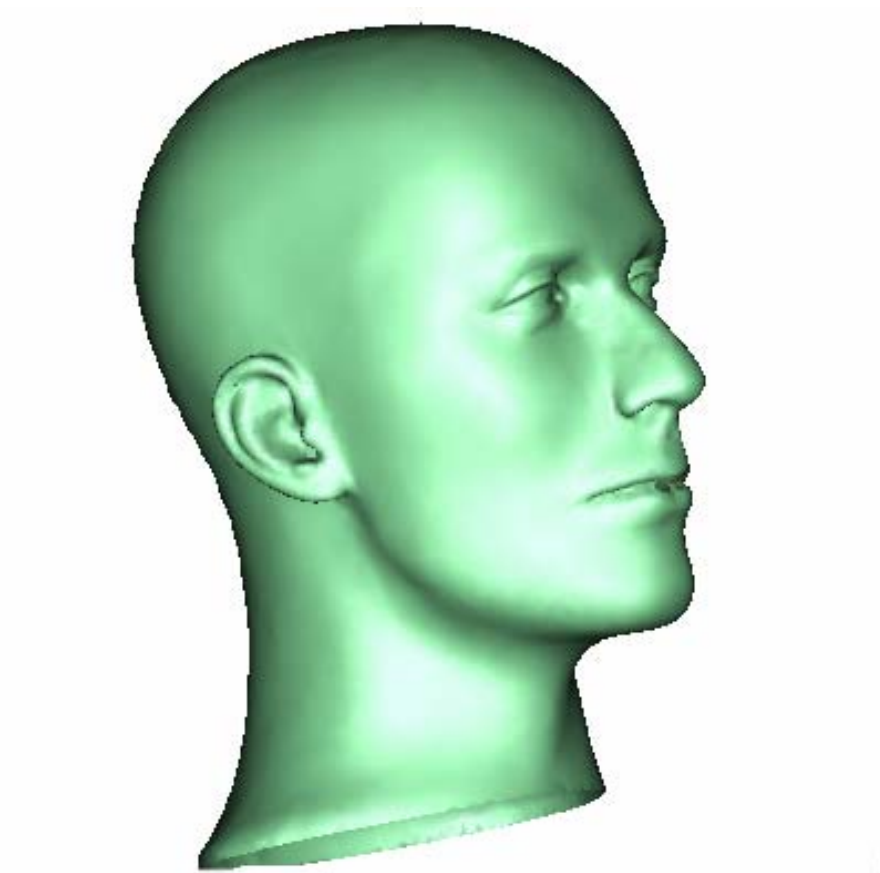
- **hotdog.pts** (jednoprůchodově: rekonstruovaný povrch (Gouraudovo stínování), povrch s vyznačenou sítí, detail povrchu se sítí a s póly).



- **hypersheet.pts** (jednoprůchodově : povrch i s chybami (flat stínování), detail chyby podzorkování, dvouprůchodový správný povrch s póly)



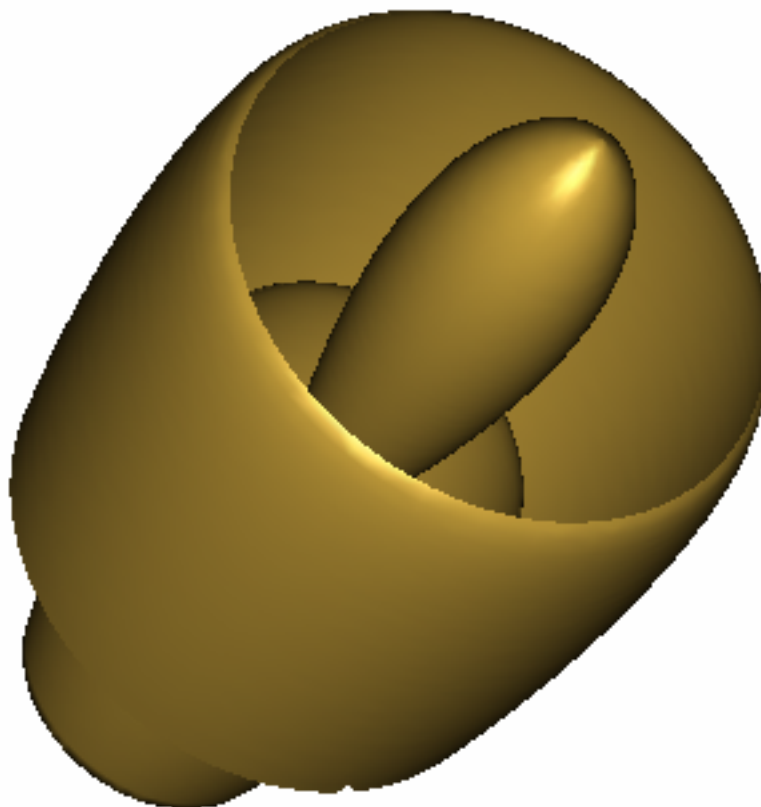
- **manneguin.pts** (jednoprůchodový, Gourardovo stínování)



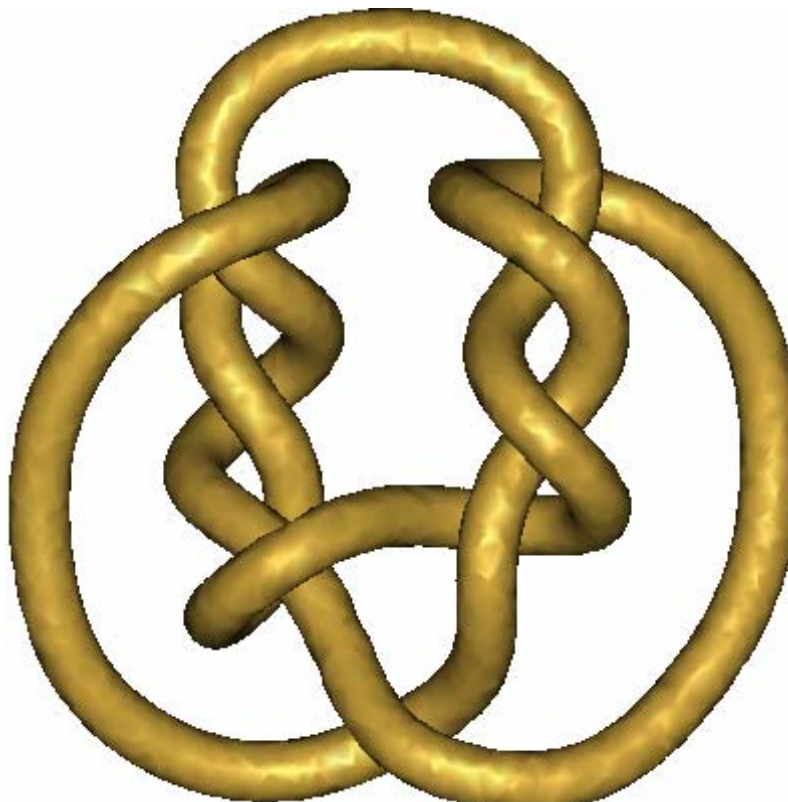
- **bunny.pts** (jednoprůchodově, Gouraudovo stínování)



- **engine.pts** (jednopřúchodově, Gouraudovo stínování)



- **knot.pts** (jednopřúchodově, Gouraudovo stínování)



- **club.pts** (dvouprůchodově, Gouraudovo stínování)

