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

DIPLOMOVÁ PRÁCE

Plzeň, 2007 Michal Zemek

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

Diplomová práce

Dělení prostoru pro rozsáhlá a měnící se data

Plzeň, 2007 Michal Zemek

Poděkování

Rád bych poděkoval svým rodičům za jejich podporu po celou dobu mého studia. Poděkování patří také vedoucí této diplomové práce Doc. Dr. Ing. Ivaně Kolingerové za užitečné rady, připomínky a poskytnutí studijních materiálů.

Prohlášení

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

V Plzni dne ……..

……

Space Division for Large and Variable Data

The Delaunay triangulation is one of the fundamental data structures of Computational Geometry. The regular triangulation is its generalization, in which weights of the input points are reflected. The first part of this thesis studies methods, which can be used to the construction of a partially dynamic regular triangulation. In the second part, tunnels in protein molecules and the application of regular triangulation for their searching are described. The tunnel's analysis is used in protein engineering for the research and modifications of protein molecule behavior.

Obsah

1. Úvod

Výpočetní geometrie a počítačová grafika jsou dva široce provázané obory. Výpočetní geometrie se zabývá problémy, jejichž podstatu lze geometricky interpretovat. V některých případech je tato interpretace jasná, například při plánování cesty robota či hledání vrstevnic v mapě, v jiných může být použití výpočetní geometrie překvapivé – například při rozpoznávání obličejů. Výpočetní geometrie se snaží určitý problém geometricky modelovat a následně hledat jeho řešení, úlohou počítačové grafiky je pak (mimo jiné) zobrazit nalezené výsledky formou, která bude srozumitelná uživateli.

Typickými vstupními daty, se kterými výpočetní geometrie pracuje, jsou souřadnice bodů a určité hodnoty v těchto bodech naměřené. Vstupní souřadnice bývají zpravidla jednorozměrné (body na přímce), dvourozměrné (body v rovině), nebo trojrozměrné (body v prostoru). Se samotnou množinou bodů se toho však příliš dělat nedá, většinou je nutné nad vstupními body vytvořit nějakou datovou strukturu, která určitým způsobem dělí prostor, a tak umožňuje rychlé nalezení blízkých bodů a/nebo výpočet sledované hodnoty v libovolném bodě. Takových struktur existuje celá řada – patří mezi ně například pravidelná a nepravidelná mřížka, různé stromové struktury (KD stromy, BSP stromy, octree atd.), široká rodina Voronoiových diagramů a mnoho druhů triangulací. Triangulace množiny bodů ve *2D*, resp. *3D*, je pospojování dvojic bodů úsečkami tak, aby tyto úsečky tvořily vzájemně se neprotínající trojúhelníky, resp. tetrahedrony (čtyřstěny), které zcela pokrývají část roviny, resp. část prostoru, ve které body leží. V praxi se používá několik různých druhů triangulací, tato práce je po dohodě s vedoucím diplomové práce zaměřena na regulární a částečně také na Delaunayovu triangulaci ve *3D*.

1.1 Hlavní cíle

Rychlost počítačů a velikost pamětí neustále stoupá, stejně tak ale rostou i objemy dat, které je nutno zpracovávat – například při digitalizaci soch obsahují výsledná data milióny bodů. Proto jsou důležité algoritmy, které nepotřebují velké množství paměti pro pomocné datové struktury a dokáží takto rozsáhlá data zpracovat.

Zajímavým problémem je dynamizace triangulace. Jestliže chceme po vytvoření triangulace množiny bodů některé body odebrat nebo naopak nové přidat, je opravdu nutné celou triangulaci počítat znovu? Odpověď je ne – jsou známy algoritmy, které umožňují v triangulaci body přidávat i odebírat. Právě na tyto algoritmy se zaměřuje

část této práce. Prvním jejím cílem je navrhnout a implementovat řešení umožňující triangulovat rozsáhlá a měnící se data.

Výsledné řešení by mělo být použito pro hledání tunelů v proteinech[1]. Analýzu tunelů využívá proteinové inženýrství – existence tunelu vedoucího do určitého místa proteinu ovlivňuje chemické vlastnosti proteinu. Cílem modifikace proteinu může být rozšíření nebo naopak uzavření tunelu tak, aby se usnadnila nebo znemožnila určitá chemická reakce. Znalost geometrické aproximace tunelu může odborníkům pomoci soustředit se na patřičnou část proteinu, kterou je nutné pro změnu určité vlastnosti proteinu upravit.

Druhým cílem této práce je navrhnout metodu využití regulární triangulace pro hledání tunelů v proteinech, otestovat tuto metodu a porovnat dosažené výsledky se stávající technikou hledání tunelů v Delaunayově triangulaci. Tato část práce je řešena ve spolupráci s týmem doc. Ing. Jiřího Sochora, CSc. z Fakulty informatiky Masarykovy univerzity v Brně, který poskytl některá testovací data a know-how hledání tunelů v Delaunayově triangulaci.

1.2 Organizace textu

Text práce je rozdělen do devíti kapitol. V druhé kapitole je definována regulární a Delaunayova triangulace, dále jsou v ní popsány některé algoritmy konstrukce regulární triangulace. Třetí kapitola se zabývá problematikou přidávání a ubírání bodů v triangulacích. Čtvrtá kapitola obsahuje definici tří typů Voronoiova diagramu, v páté jsou popsány proteiny, tunely v proteinech a použití Voronoiových diagramů při hledání tunelů. V šesté kapitole jsou na základě duality mezi Voronoiovými diagramy a Delaunayovou triangulací odvozeny metody hledání tunelů v triangulacích. Sedmá kapitola se zabývá některými detaily implementace vybraných algoritmů. V osmé kapitole jsou popsány výsledky rozsáhlých testů implementovaných algoritmů a srovnání regulární a Delaunayovy triangulace. Značná pozornost je věnována porovnání tunelů v regulární a Delaunayově triangulaci. V deváté kapitole jsou shrnuty cíle a výsledky práce. Přílohy obsahují obrázky tunelů a uživatelský manuál.

[1] Tunel v proteinu zatím chápejme intuitivně jako volný prostor vedoucí z určitého místa uvnitř proteinu ven na jeho povrch.

2. Triangulace

V úvodu byla zmíněna regulární a Delaunayova triangulace. Ty jsou základem celé této práce, proto jsou v této kapitole podrobně definovány, dále jsou popsány jejich vlastnosti a některé metody konstrukce. Hlavní pozornost je věnována regulární triangulaci – návrh jejího použití pro hledání tunelů v proteinech je jedním z hlavních cílů této práce.

2.1 Delaunayova triangulace

Triangulace. Je dána konečná množina bodů $S \subset R^3, |S| = n$. Triangulace T množiny S (dále $T(S)$) je rozdělení prostoru ohraničeného konvexní obálkou S na množinu nepřekrývajících se tetrahedronů, jejichž vrcholy jsou body z S. Mají-li dva tetrahedrony společný průnik, je jím vrchol, hrana, nebo trojúhelník triangulace T.

Delaunayova triangulace (*DT*). Triangulace $T(S)$ je Delaunayova, jestliže koule opsaná libovolnému tetrahedronu neobsahuje ve svém vnitřku žádný další bod z množiny S.

Vlastnosti *DT*. Zatímco pro Delaunayovu triangulaci ve *2D* je známo množství pěkných vlastností, ve *3D* je pro *DT* zatím dokázán pouze fakt, že minimalizuje maximální poloměr tzv. minimum containment sphere (tj. nejmenší koule, která obsahuje tetrahedron). Časová i paměťová složitost Delaunayovy triangulace ve *3D* je $O(n^2)$ v nejhorším případě.

2.2 Regulární triangulace

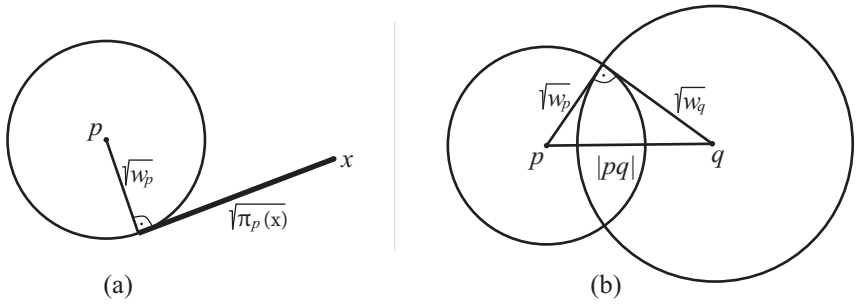
Regulární triangulace (*RT*) je zobecněním Delaunayovy triangulace. Každému bodu je v ní přiřazena váha a tyto váhy ovlivňují výslednou triangulaci (viz obr. 2.2). Regulární triangulaci zavedeme podle [F93].

Vážené body. Mějme konečnou množinu bodů $S \subset R^3, |S| = n$. Každému bodu $p \in S$ je přiřazena váha $w_p \in R$. Váha w_p může být i záporná, obvykle se ale v *RT* uvažují váhy nezáporné a vážený bod pak bývá interpretován jako koule se středem p a poloměrem $\sqrt{w_p}$.

Power vzdálenost. Pro vážený bod p je definována jeho power vzdálenost od neváženého bodu $x \in R^3$ jako $\pi_p(x) = |xp|^2 - w_p$. Power vzdálenost lze geometricky

interpretovat jako druhou mocninu délky tečny z bodu x na kouli se středem v bodě p a poloměrem $\sqrt{w_p}$ (pokud x leží mimo tuto kouli), viz obr. 2.1a.

Ortogonalita. Dvojice vážených bodů p, q je ortogonální, jestliže platí $|pq|^2 = w_p + w_q$, viz obr. 2.1b.



Obr. 2.1: (a) Power vzdálenost, (b) ortogonalita dvojice vážených bodů.

Ortogonální střed. Vážený bod z je ortogonálním středem tetrahedronu *abcd*, jestliže z je ortogonální na všechny vrcholy tohoto tetrahedronu (tedy na body *a, b, c, d*).

Regularita (globální). Nechť z je ortogonálním středem tetrahedronu *abcd*. Tetrahedron *abcd* je globálně regulární, jestliže pro všechny body $p \in S - \{a,b,c,d\}$ platí, že $\pi_z(p) > w_p$.

Regulární triangulace. Triangulace $T(S)$ je **regulární**, jestliže každý tetrahedron z T splňuje kritérium globální regularity.

Redundantní bod. Bod $p \in S$, pro který neexistuje globálně regulární tetrahedron *pabc* ($a,b,c \in S$), není vrcholem $RT(S)$ a nazývá se redundantní[2]. Množina vrcholů $RT(S)$ je tedy podmnožinou S.

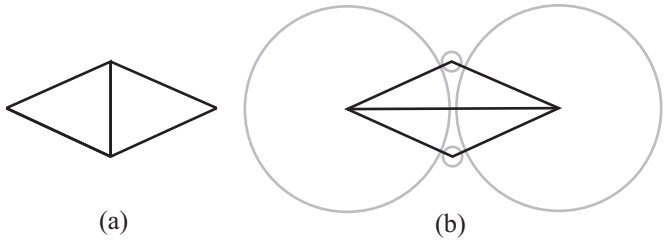
Složitost *RT*. Stejně jako u *DT* je časová i paměťová složitost regulární triangulace ve *3D* $O(n^2)$ v nejhorším případě.

Pro konstrukci triangulace budou užitečná dvě následující lemma.

Regularita (lokální). Nechť jsou dány dva tetrahedrony *abcd* a *abce* se společnou stěnou *abc* a bod z, který je ortogonálním středem tetrahedronu *abcd*. Pak stěna *abc* je lokálně regulární, jestliže $\pi_z(e) > w_e$.

[2] K redundantním bodům se ještě vrátíme v kapitole 2.3.3.

Regulární triangulace. Jestliže množina vrcholů triangulace T obsahuje všechny neredundantní body z S a všechny stěny triangulace jsou lokálně regulární, pak triangulace T je regulární triangulací množiny S.



Obr. 2.2: Srovnání minimální *DT* (a) a *RT* (b) ve *2D*. Tvar *RT* je ovlivněn váhou bodů (váhy jsou vyznačeny kružnicemi).

2.3 Metody konstrukce

2.3.1 Inkrementální vkládání

Algoritmus inkrementálního vkládání vytváří triangulaci „bod po bodu“. Při vkládání bodu nalezne tetrahedron (skupinu tetrahedronů), který s bodem inciduje, připojí bod do triangulace a sérií lokálních oprav zajistí, že je výsledná triangulace opět regulární. Algoritmus inkrementálního vkládání je použit v této práci, proto je podrobně vysvětlen v kapitole 2.4.

2.3.2 Inkrementální konstrukce

Algoritmus inkrementální konstrukce [B05] vytváří triangulaci tetrahedron po tetrahedronu, přičemž vytvořené simplexy už se dále nemění. Prvním krokem je nalezení globálně regulárního tetrahedronu. Všechny stěny tohoto tetrahedronu jsou vloženy do seznamu aktivních stěn (AFL – active face list). Pro každou stěnu z AFL je nalezen bod, který od ní má minimální znaménkovou Delaunayovu vzdálenost (viz [B05]). Připojením tohoto bodu ke stěně je stěna expandována, vznikne nový globálně regulární tetrahedron. Tím vzniknou tři nové stěny. Jestliže některá z nich už v AFL existuje, je z něj odstraněna (tak je současně nalezena sousednost mezi novým a „starším“ tetrahedronem), v opačném případě je do AFL přidána. Pokud pro stěnu neexistuje bod, který by ji mohl expandovat, pak tato stěna leží na hranici konvexní obálky, a je z AFL vyjmuta. Algoritmus končí ve chvíli, kdy je AFL prázdný.

Pro n bodů je časová složitost algoritmu v jeho základní podobě $O\left(n^3\right)$. Podle [B05] lze však použitím několika vylepšení dosáhnout takřka lineární složitosti. Vylepšení

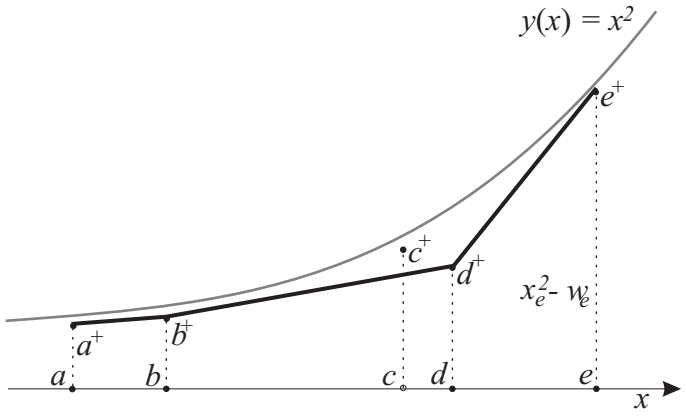
spočívají především v hashování stěn v AFL a v uložení bodů do pomocné struktury dělící prostor, například do pravidelné mřížky.

2.3.3 Převod do vyšší dimenze

Zajímavou vlastností regulární triangulace je její spojitost s konvexní obálkou ve vyšší dimenzi.

Nechť je dána množina bodů $S \subset R^3$. Definujme pro každý bod $p = \left(x_p, y_p, z_p\right) \in S$ s váhou w_p bod $p^+ = (x_p, y_p, z_p, x_p^2 + y_p^2 + z_p^2 - w_p) \in R^4$. Pokud je váha w_p nulová, je bod *p* promítnut na paraboloid[3], pokud je kladná (resp. záporná), leží bod p^+ pod (resp. nad) paraboloidem. Pro *S* definujme množinu $S^+ = \left\{p^+ \mid p \in S\right\}$. V [E92] je ukázáno, že vertikální projekcí tetrahedronů tvořících dolní konvexní obálku množiny S^+ do R^3 vznikne *RT*(*S*). Tím je konstrukce regulární triangulace ve *3D* převedena na konstrukci konvexní obálky ve *4D*, která může být vytvořena například algoritmem Quickhull[4].

Spojitost s konvexní obálkou umožňuje další pohled na redundantní body. Protože body z S^+ jsou svou váhou posunuty mimo paraboloid, nemusí být každý bod z S^+ vrcholem dolní konvexní obálky S^+. Bod $p \in S$, jemuž odpovídající bod p^+ není vrcholem dolní conv(S^+), je redundantní (viz obr. 2.3).



Obr. 2.3: Vztah *RT* v *1D* a konvexní obálky ve *2D*. Bod *c* leží nad konvexní obálkou (vyznačena silnou čarou), a tedy je redundantní.

[3] Tento paraboloid je dán rovnicí $u(x, y, z) = x^2 + y^2 + z^2$.

[4] www.qhull.org.

2.4 Inkrementální vkládání

Algoritmus inkrementálního vkládání pro *RT* je popsán například v [E92]. Algoritmus vytváří regulární triangulaci dané množiny $S = \{p_0, p_1, ..., p_n\}$ vážených bodů inkrementálně – v každém kroku vloží do *RT* jeden bod a sérií lokálních oprav převede vzniklou triangulaci znovu na regulární. Co je to lokální oprava, jak a kam se nové body vkládají a způsob vytvoření počáteční *RT* je popsán dále. Pro vložení bodu a lokální opravy je používán souhrnný název swap $X \rightarrow Y$, kde X je počet tetrahedronů vstupujících do swapu a Y počet výsledných tetrahedronů.

2.4.1 Počáteční triangulace

Je zřejmé, že výše nastíněný algoritmus musí „někde začít“. Proto se do *S* přidávají čtyři pomocné body $p_{-4}, p_{-3}, p_{-2}, p_{-1}$ takové, že jimi tvořený tetrahedron obsahuje všechny ostatní body *S*. Tím je zaručeno, že při vkládání bodu do *RT* bude bod ležet **uvnitř** triangulace (což značně zjednodušuje celý algoritmus). Po dokončení triangulace celé množiny *S* jsou z *RT* odstraněny tetrahedrony, které obsahují některý z vrcholů $p_{-4}, p_{-3}, p_{-2}, p_{-1}$.

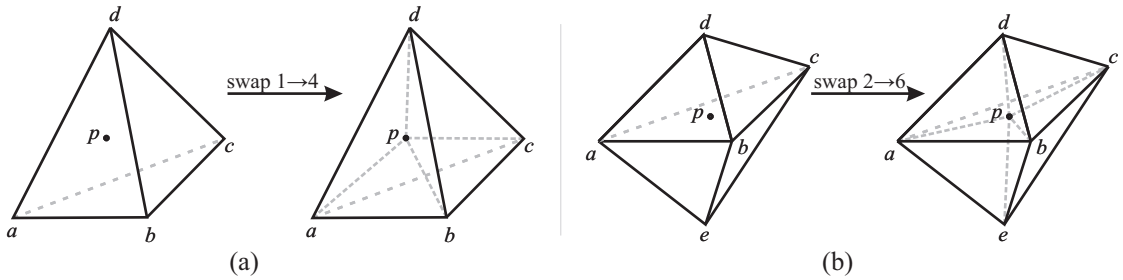
V teoretických popisech leží body $p_{-4}, p_{-3}, p_{-2}, p_{-1}$ v nekonečnech, v praxi mohou být body zvoleny například takto: $p_{-4} = (s_x + K \cdot d, s_y, s_z)$, $p_{-3} = (s_x, s_y + K \cdot d, s_z)$, $p_{-2} = (s_x, s_y, s_z + K \cdot d)$ a $p_{-1} = (s_x - K \cdot d, s_y - K \cdot d, s_z - K \cdot d)$, kde $s = (s_x, s_y, s_z)$ je střed min-max boxu *S*, *d* je jeho nejdelší hrana a *K* je konstanta. Určení *K* je delikátní záležitost – pokud bude malé, může být hranice triangulace po odebrání bodů $p_{-4}, p_{-3}, p_{-2}, p_{-1}$ (a tetrahedronů, které je obsahují) nekonvexní. Pokud bude velké, trpí numerická stabilita. V praxi používané hodnoty *K* se pohybují mezi 10 a 20.

Další možností je nalézt konvexní obálku *S*, vytvořit její *RT* (to je možné provést například technikou uřezávání uší, viz kapitola 3.4) a do takto vzniklé triangulace přidávat zbylé body. Tento způsob je méně používaný, zřejmě kvůli své větší implementační i časové náročnosti.

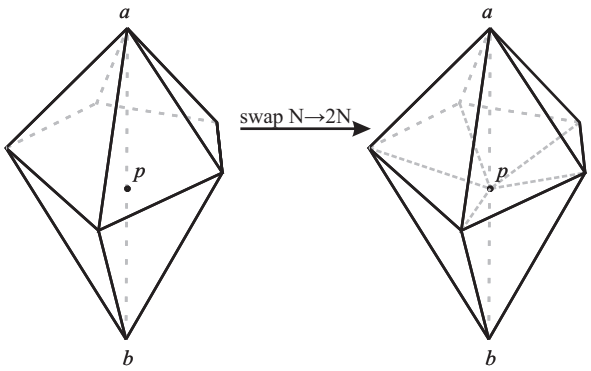
2.4.2 Vložení bodu

Při vkládání bodu *p* do triangulace je vyhledán tetrahedron, ve kterém bod *p* leží. Jestliže nepředpokládáme tzv. obecnou polohu bodů (žádné 4 body neleží v jedné rovině, žádných pět bodů neleží na jedné kulové ploše), musíme rozlišit několik případů (viz [J91, K02]):

- Bod *p* leží **uvnitř** tetrahedronu *abcd*. To je nejjednodušší (a pro reálná data také nejčastější) varianta. Bod *p* je spojen s vrcholy *a*, *b*, *c*, *d,* z jednoho tetrahedronu tak vzniknou čtyři nové (swap 1→4, viz obr. 2.4).
- Bod *p* leží **ve stěně** *abc* tetrahedronu *abcd*. V tomto případě je nalezen sousední tetrahedron *abce* (pokud existuje) a tyto dva tetrahedrony jsou rozděleny na šest nových tetrahedronů (swap 2→6, viz obr. 2.4). Pokud by byl použit postup z předchozího bodu, bude vzniklý tetrahedron *pabc* rovinný. Takový tetrahedron je z pohledu dělení prostoru zbytečný a z pohledu stability algoritmu nežádoucí.
- Bod *p* leží **na hraně** *ab* tetrahedronu *abcd*. To je nejsložitější možnost. Je nutné nalézt všechny tetrahedrony, které hranu *ab* obsahují, a každý z nich rozdělit na dva. Z původních *N* tetrahedronů tak vznikne 2*N* nových (swap *N*→2*N*, viz obr. 2.5).
- Bod *p* splývá s některým vrcholem triangulace. V tomto případě není bod *p* do triangulace vložen.



Obr. 2.4: Vlevo swap 1→4, bod *p* leží uvnitř tetrahedronu *abcd*, ten je rozdělen na čtyři tetrahedrony *abcp*, *abdp*, *acdp* a *bcdp*. Vpravo swap 2→6, bod *p* leží ve stěně *abc* , dva tetrahedrony *abcd* a *abce* jsou rozděleny na šest tetrahedronů *abdp*, *bcdp*, *acdp*, *abep*, *bcep* a *acep*.



Obr. 2.5: Swap *N*→2*N*, bod *p* leží na hraně *ab*, z každého tetrahedronu *abxy* vzniknou dva tetrahedrony *axyp* a *bxyp*.

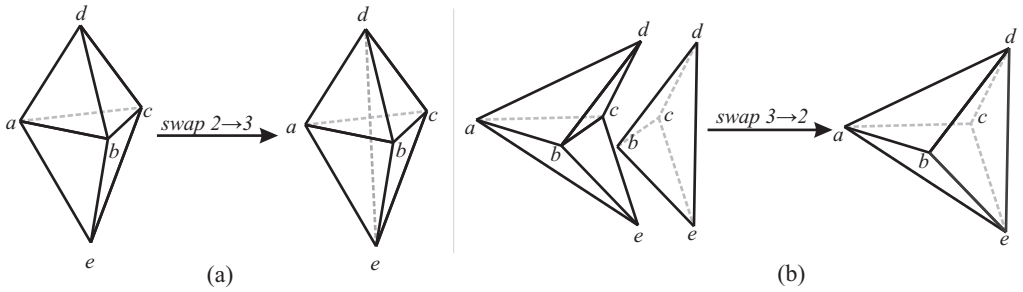
2.4.3 Lokální opravy

Vložením bodu *p* do triangulace může samozřejmě dojít k porušení její regularity. Proto musí být otestovány všechny vnější stěny nově vzniklých tetrahedronů (tzn. stěny, které neobsahují nově vložený bod *p*). Jestliže stěna porušuje kritérium lokální regularity, je na ni aplikována jedna z dále popsaných lokálních oprav. Tím v triangulaci vzniknou nové dvojice, trojice či čtveřice tetrahedronů, jejichž vnější stěny musí být opět zkontrolovány a případně opraveny (v nejhorším případě mlže dojít ke změně celé triangulace).

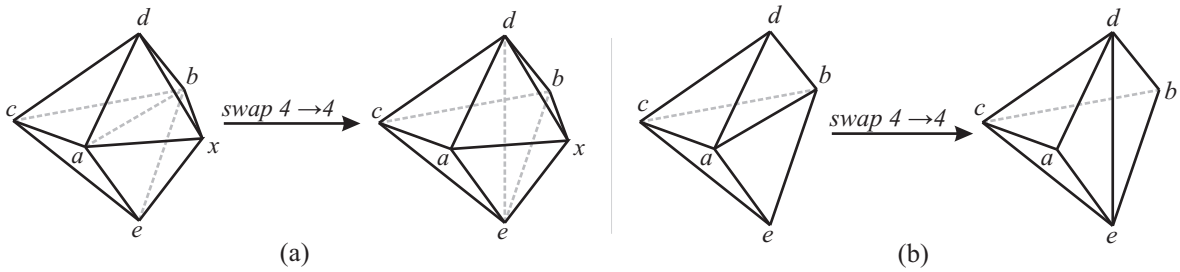
Mějme neregulární stěnu *abc*, která je sdílena dvěma tetrahedrony *abcd*, *abce*. Oprava stěny *abc* spočívá v náhradě těchto dvou tetrahedronů (a v některých případech i jednoho či dvou sousedních tetrahedronů) dvěma až čtyřmi novými tetrahedrony, které vyplňují stejný prostor jako tetrahedrony původní a neobsahují neregulární stěnu *abc*. Musíme přitom v závislosti na geometrii rozlišit tři různé případy:

- hrana *de* prochází **vnitřkem** stěny *abc*. Pak jsou dva tetrahedrony *abcd, abce* nahrazeny trojicí tetrahedronů *abde, acde, bcde* (swap 2→3, viz obr. 2.6a).
- hrana *de* prochází **mimo** stěnu *abc*. Pokud v triangulaci existuje tetrahedron, který spolu s tetrahedrony *abcd* a *abce* zcela vyplňuje prostor daný body *a*, *b*, *c*, *d*, *e*, jsou tyto tři tetrahedrony nahrazeny dvojicí tetrahedronů *abde* a *acde* (swap 3→2, viz obr. 2.6b). Tato oprava je inverzní k opravě popsané v bodě 1.
- hrana *de* prochází některou z hran stěny *abc*. Nechť protíná např. hranu *ab*. Jestliže jsou stěny *abd*, *abe* vnějšími stěnami triangulace, nebo existují dva tetrahedrony *abdx*, *abex* (tetrahedrony, které s dvojicí neregulárních tetrahedronů sdílejí stěny *abd*, *abe* a zároveň spolu sousedí, viz obr. 2.7), je stěna *abc* nahrazena stěnou *cde*, tetrahedrony *abcd* a *abce* jsou nahrazeny tetrahedrony *acde* a *bcde*. Obdobně jestliže existují sousedé *abdx*, *abex*, jsou nahrazeni tetrahedrony *adex*, *bdex* (swap 4→4, viz obr. 2.7).

Jestliže není splněna některá z popsaných podmínek, není stěna dočasně opravena. Jsou provedeny opravy dalších neregulárních stěn a v jejich důsledku dojde i k opravě dříve „neopravitelné“ stěny.



Obr. 2.6: (a) Swap 2→3, dva tetrahedrony *abcd* a *abce* jsou nahrazeny trojicí tetrahedronů *abde*, *bcde*, *acde*. (b) Swap 3→2, tři tetrahedrony *abcd*, *abce*, *bcde* jsou nahrazeny dvojicí tetrahedronů *abde*, *acde*.



Obr. 2.7: Swap 4→4. (a) V tetrahedronech *abcd* a *abce* je neregulární stěna *abc* nahrazena stěnou *cde*. Obdobně musí být prohozena i stěna *abx* v tetrahedronech *abdx* a *abex* (pokud existují). V (b) je pro větší přehlednost znázorněn swap 4→4 bez sousedních tetrahedronů.

2.4.4 Pořadí lokálních oprav

Lze říci, že na pořadí lokálních oprav nezáleží. Vnější stěny skupiny nově vzniklých tetrahedronů jsou vloženy do zásobníku[5]. Při odebrání stěny ze zásobníku je nutné pouze zkontrolovat, zda tato stěna v triangulaci stále existuje (je možné, že byla zrušena nějakou předchozí opravou). Pokud stěna existuje a je neregulární a může na ni být použita jedna z výše popsaných oprav, je opravena a do zásobníku jsou vloženy vnější stěny nově vzniklých tetrahedronů. V opačném případě je stěna zapomenuta a pokračuje se odebráním další stěny ze zásobníku. Po vyprázdnění zásobníku je výsledná triangulace opět regulární.

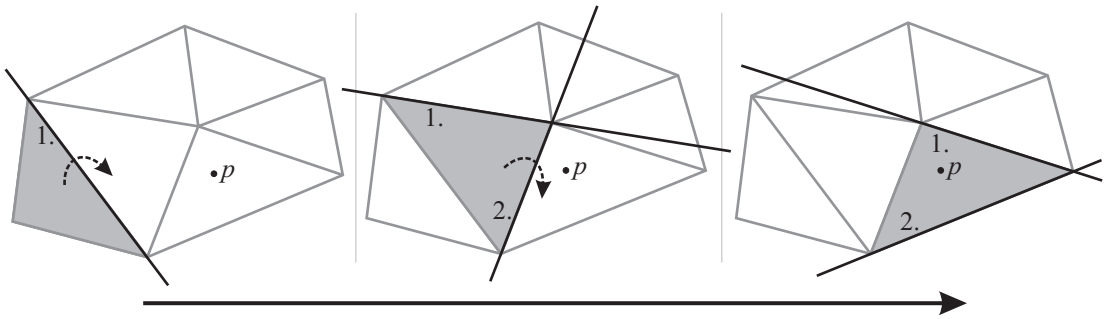
2.4.5 Vyhledání tetrahedronu

Při vkládání bodu je nutné nejprve najít tetrahedron, ve kterém vkládaný bod leží. Jednoduchou a oblíbenou skupinou algoritmů, které tento problém řeší, jsou algoritmy procházek v triangulaci. Jejich detailní popis a porovnání lze nalézt v [D01]. Jako nejlepší volba se jeví stochastická procházka, kterou zde stručně popíšeme.

Algoritmus začne v libovolném tetrahedronu t a otestuje náhodně zvolenou stěnu. Pokud rovina ρ, ve které tato stěna leží, odděluje tetrahedron t a hledaný bod p, přejde hledání do sousedního tetrahedronu, který s t sdílí testovanou stěnu, a postup se

[5] Stěnu je možné uložit jako dvojici indexů tetrahedronů, které ji obsahují.

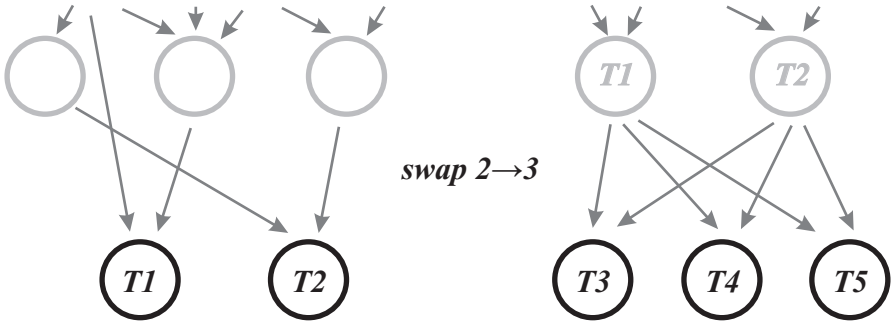
opakuje. Jestliže rovina ρ neodděluje t a p, algoritmus otestuje stejným způsobem další stěnu. To opakuje, dokud nenalezne tetrahedron, který od bodu p neodděluje žádná z jeho stěn (resp. rovin, které jsou stěnami určeny) – v tomto tetrahedronu leží bod p. Postup hledání je naznačen na obrázku 2.8 – je hledán trojúhelník obsahující bod p, trojúhelníky testované v jednotlivých krocích hledání jsou tmavě zvýrazněny, čísla označují pořadí testování hran.



Obr. 2.8: Postup algoritmu procházky ve *2D*. Čísla označují pořadí testování hran.

Očekávaná složitost algoritmů procházek ve *3D* je $O\left(n^{1/4}\right)$ na jeden bod, tedy o něco horší, než optimální $O(\log n)$. Jejich výhodou však je, že nepotřebují další pomocné struktury.

Příkladem datové struktury, která umožňuje vyhledání tetrahedronu incidujícího s vkládaným bodem v optimálním čase $O(\log n)$ (byť jen v očekávaném případě, v nejhorším případě je časová složitost $O(n)$), je acyklický orientovaný graf (DAG – Directed Acyclic Graph), v kterém je uložena historie provedených swapů. DAG má jediný kořen – počáteční pomocný tetrahedron. Vnitřní uzly jsou tvořeny tetrahedrony, které v průběhu konstrukce triangulace zanikly (při vložení bodu nebo lokální úpravě), koncové uzly jsou tetrahedrony aktuální triangulace. Orientované hrany mezi uzly popisují jednotlivé swapy. Uveďme jednoduchý příklad: dva tetrahedrony *T1* a *T2* jsou swapem 2→3 nahrazeny trojicí tetrahedronů *T3*, *T4*, *T5*. V důsledku toho jsou v DAGu pod uzly *T1* a *T2* připojeny nové uzly *T3*, *T4* a *T5*, orientované hrany směřují z uzlů *T1* a *T2* do *T3*, *T4* a *T5* (viz obr. 2.9).



Obr. 2.9: Vlevo fragment DAGu a odpovídající dvojice tetrahedronů *T1* a *T2*. Vpravo DAG po nahrazení tetrahedronů *T1* a *T2* novými tetrahedrony *T3*, *T4*, *T5*.

Nalezení tetrahedronu obsahujícího vkládaný bod *p* je jednoduché – algoritmus začne v kořenu a postupuje po tetrahedronech, které bod *p* obsahují. Hledání končí v okamžiku, kdy algoritmus sestoupí do některého koncového tetrahedronu. Před přechodem z jednoho tetrahedronu do dalšího je možné testovat, jestli bod *p* není redundantní (další variantou je testovat redundanci pouze pro koncový nebo pro každý *x*-tý simplex). Jestliže je bod *p* redundantní, hledání může být ukončeno, bod nebude vložen. Test redundance je popsán v kapitole 2.4.7.

2.4.6 Test orientace a regularity

Pro algoritmus inkrementálního vkládání jsou základní dva typy testů [F93] – test orientace a test regularity.

Test orientace čtveřice bodů *a, b, c, d* zjišťuje, na které straně roviny ρ, procházející body *a*, *b*, *c*, leží bod *d*. Jestliže je výsledek testu záporný (resp. kladný), leží bod *d* nad[6] (resp. pod) rovinou ρ. Pokud je test rovný nule, leží bod *d* na rovině ρ. Několika testy orientace lze rozhodnout, zda je možné na stěnu triangulace použít lokální opravu (a který typ), test orientace je také základem algoritmů procházek. Test lze převést na výpočet determinantu (viz výraz 2.1),

$$orient(a,b,c,d) = \begin{vmatrix} x_a & y_a & z_a & 1 \\ x_b & y_b & z_b & 1 \\ x_c & y_c & z_c & 1 \\ x_d & y_d & z_d & 1 \end{vmatrix} \tag{2.1}$$

kde x_a, y_a, z_a jsou souřadnice bodu *a*.

Důležitý je také test regularity stěny. Ze vztahu *RT* ve *3D* a konvexní obálky ve *4D* vyplývá, že stěna *abc*, sdílená tetrahedrony *abcd* a *abce*, je lokálně regulární, jestliže

[6] Pravidlo pravé ruky – prsty pravé ruky ukazují ve směru polohových vektorů bodů *a*, *b*, *c*, palec ukazuje „nad rovinu“ danou těmito body.

bod[7] e^+ (bod ve *4D*) leží nad nadrovinou určenou ve *4D* body a^+, b^+, c^+, d^+. Test regularity ve *3D* lze tedy převést na test orientace ve *4D* a opět je možné použít k jeho výpočtu determinant. Protože pořadí bodů *a*, *b*, *c*, *d* by nemělo mít vliv na výsledek testu, je nutné znaménko determinantu „přenásobit" znaménkem testu orientace bodů *a*, *b*, *c*, *d*. Shrnuto – stěna *abc* je lokálně regulární, pokud je výraz (2.2) záporný. Test regularity lze jednoduše upravit na in-sphere test pro Delaunayovu triangulaci – stačí neodčítat váhy bodů *w*.

$$regularita(a,b,c,d,e) = orient(a,b,c,d) \cdot \begin{vmatrix} x_a & y_a & z_a & x_a^2+y_a^2+z_a^2-w_a & 1 \\ x_b & y_b & z_b & x_b^2+y_b^2+z_b^2-w_b & 1 \\ x_c & y_c & z_c & x_c^2+y_c^2+z_c^2-w_c & 1 \\ x_d & y_d & z_d & x_d^2+y_d^2+z_d^2-w_d & 1 \\ x_e & y_e & z_e & x_e^2+y_e^2+z_e^2-w_e & 1 \end{vmatrix} \quad (2.2)$$

2.4.7 Redundantní body

Při vkládání bodu do triangulace je možné jednoduše otestovat, zda je vkládaný bod redundantní. Jestliže má být bod *q* vložen do tetrahedronu s vrcholy *a*, *b*, *c*, *d* a test *regularita*(*a*, *b*, *c*, *d*, *q*) je záporný, je bod *q* redundantní a neměl by být do triangulace vložen. Tomu ve vztahu s konvexní obálkou ve *4D* odpovídá situace, kdy bod q^+ leží nad nadrovinou určenou body a^+, b^+, c^+, d^+, a tedy nemůže být součástí dolní konvexní obálky ve *4D* (viz kapitola 2.3.3 a obr. 2.3).

Složitější je situace, kdy se vložením bodu *q* do triangulace stane některý její „starší" vrchol *p* redundantním. Tento problém má dvě části – nejprve musí být bod *p* rozpoznán jako redundantní a poté musí být z triangulace odstraněn. Bohužel ani [F93] ani [E92] se touto problematikou nezabývá.

2.4.8 Časová složitost algoritmu

Časová složitost algoritmu inkrementálního vkládání je v nejhorším případě $O\left(n^2\right)$, složitost v očekávaném případě záleží na použité metodě hledání tetrahedronu incidujícího s vkládaným bodem. Při použití DAGu je očekávaná složitost $O(n \log n)$, při použití algoritmu procházek $O\left(n\sqrt[4]{n}\right)$.

[7] Připomeňme, že bod e^+ vznikne zobrazením bodu *e* ze *3D* do *4D*. Čtvrtá souřadnice bodu e^+ je dána jako $x_e^2 + y_e^2 + z_e^2 - w_e$.

3. Přidání a odebrání bodu v triangulaci

Důležitým krokem při dynamizaci triangulace je schopnost přidávat a odebírat body do a z triangulace. Přidání bodu do triangulace není při použití vhodného algoritmu obtížné. Problém odebrání bodu je složitější a v odborné literatuře je řešen jen zřídka. V této kapitole zmíníme komplikace spojené s přidáním bodu a popíšeme některé algoritmy, které umožňují odebrání bodu z triangulace.

3.1 Přidání bodu

Při použití algoritmu inkrementálního vkládání je přidání bodu do triangulace triviální, neboť tento algoritmus vytváří triangulaci postupným vkládáním bodů (viz kap. 2.4). Problém může nastat, pokud má být do triangulace přidán bod *p*, který nebyl známý při vytvoření počátečního pomocného tetrahedronu (viz kap. 2.4.1). Jestliže bod *p* leží vně min-max boxu, podle kterého byl pomocný tetrahedron vytvořen, může být hranice triangulace po odebrání tetrahedronů obsahujících pomocné body (pomocných tetrahedronů) nekonvexní. Pokud by bod *p* ležel zcela mimo triangulaci, nemohl by do ní být vůbec vložen. Proto je vhodné při vytváření počátečního tetrahedronu zvětšit min-max box známých bodů o určitou „rezervu“ a vytvořit tak počáteční tetrahedron „dostatečně velký“. Pokud přesto leží vkládaný bod *p* vně min-max boxu, je možné čtyři přidané pomocné body (vrcholy počátečního tetrahedronu) posunout dále od středu min-max boxu, a tím min-max box zvětšit. V důsledku tohoto posunutí se ovšem mohou některé stěny pomocných tetrahedronů stát neregulárními. Proto je nutné stěny všech pomocných tetrahedronů zkontrolovat a neregulární opravit použitím lokálních oprav[8].

3.2 Odebrání bodu ve *3D RT*

Zde nastíníme algoritmus popsaný v [V01]. Nechť má být z regulární triangulace odebrán bod *p*. Nejprve jsou do prioritní fronty vloženy všechny stěny triangulace, které bod *p* obsahují. V dalším kroku je z fronty odebrána stěna s maximální prioritou a je prohozena (viz kapitola Lokální opravy). Jestliže některé nové stěny obsahují bod *p*, jsou vloženy do fronty. Tento krok se opakuje, dokud ve frontě nezůstane pouze 6 stěn. Těchto 6 stěn náleží čtveřici tetrahedronů, které spolu navzájem sousedí a mají společný vrchol *p*. Posledním krokem je nahrazení těchto čtyř tetrahedronů jedním.

[8] Metodu posunutí pomocných bodů nepopisuje žádná autorovi známá odborná literatura a autor nemůže zaručit, že opravy neregulárních stěn budou vždy konvergovat.

Problematickým bodem je výpočet priority stěny – priorita je určena ze složité pomocné datové struktury podobné DAGu, která je vytvářena průběžně při konstrukci a úpravách *RT*. Popis této struktury je jádrem celého článku [V01] a nebylo by rozumné snažit se ji zde vysvětlit.

3.3 Odebrání bodu ve *2D DT* podle Devillerse

Jednodušší algoritmus pro odebrání bodu z *2D DT* je popsán v [D99]. Na rozdíl od předchozího algoritmu nepotřebuje znát celou historii swapů, pracuje pouze s výslednou triangulací. Nevýhodou je, že autor popisuje pouze jeho *2D* verzi pro *DT*.

Nechť je dána *2D* Delaunayova triangulace množiny bodů S a bod $p \in S$. Odebráním bodu p a všech k trojúhelníků, které s bodem p incidují, vznikne v triangulaci „díra“, přesněji star-shaped polygon $H = \{q_0, q_1, ..., q_k = q_0\}$. Naším cílem je tento polygon triangulovat tak, aby výsledná triangulace byla opět Delaunayova.

Algoritmus funguje na principu ořezávání uší polygonu. Uchem polygonu nazveme trojice po sobě jdoucích bodů $q_i, q_{i+1}, q_{i+2} \in H$. Pokud q_i, q_{i+1}, q_{i+2} svírají nekonvexní úhel, je jim přiřazena priorita $+\infty$. Pokud svírají konvexní úhel, je jejich priorita (závislá na odebíraném bodu p) definována takto[9]:

$$priorita(q_0, q_1, q_2, p) = \frac{\begin{vmatrix} x_{q0} & y_{q0} & x_{q0}^2 + y_{q0}^2 & 1 \\ x_{q1} & y_{q1} & x_{q1}^2 + y_{q1}^2 & 1 \\ x_{q2} & y_{q2} & x_{q2}^2 + y_{q2}^2 & 1 \\ x_p & y_p & x_p^2 + y_p^2 & 1 \end{vmatrix}}{\begin{vmatrix} x_{q0} & y_{q0} & 1 \\ x_{q1} & y_{q1} & 1 \\ x_{q2} & y_{q2} & 1 \end{vmatrix}} \qquad (3.1)$$

V každém kroku je vybráno ucho q_i, q_{i+1}, q_{i+2} s **nejmenší** prioritou[10] a je „odříznuto“ – přidáním hrany q_i, q_{i+2} je vytvořen trojúhelník, který je připojen k triangulaci. Podle [D99] bude takto získaný trojúhelník vždy součástí Delaunayovy triangulace množiny bodů $S - \{p\}$. Odříznutím ucha q_i, q_{i+1}, q_{i+2} je hranice polygonu H zkrácena o bod q_{i+1}.

[9] Nechť C je kružnice procházející body q_0, q_1, q_2. Pak $priorita(q_0, q_1, q_2, p)$ je kladná pro bod p ležící uvnitř kružnice C, nulová pro p ležící na C a záporná pro p ležící vně C. Pokud bod p leží vně C, je absolutní hodnota priority rovna druhé mocnině délky tečny z bodu p na kružnici C. Také je možné prioritu interpretovat jako vertikální vzdálenost bodu p^+ od nadroviny procházející body q_0^+, q_1^+, q_2^+ (kde p^+ značí bod vzniklý zobrazením bodu p do vyšší dimenze, viz kapitola 2.3.3).

[10] Tedy takové ucho, jehož kružnice opsaná má svůj střed v největší vzdálenosti od bodu p.

Po odříznutí k-2 uší bude polygon H tvořen pouze třemi body, tedy jedním trojúhelníkem. Přidáním tohoto trojúhelníku do triangulace je „díra“ zcela zaplněna a výsledná triangulace je opět Delaunayova.

Složitost algoritmu je $O(k \log k)$, kde k je počet trojúhelníků obsahujících odebraný bod. Tato složitost je dána nutností udržovat prioritní frontu. Nicméně operace s frontou jsou „levné“, časově náročný je výpočet priority ucha. Na začátku je vypočtena priorita k uší, při každém odříznutí ucha je nutné přepočítat prioritu pouze dvou sousedních uší[11], dohromady je tedy priorita počítána přibližně $3k$–krát.

3.4 Rozšíření Devillersova algoritmu pro *3D RT*

Dále vysvětlený algoritmus odebírání bodů ve *3D* regulární triangulaci je mým rozšířením Devillersova algoritmu pro odebírání bodů ve *2D DT* a pokud vím, nebylo dosud podobné rozšíření publikováno. Proto tento algoritmus popíšu detailně.

3.4.1 Popis algoritmu

Základní postup zůstává stejný jako u *2D* algoritmu – z triangulace jsou odebrány tetrahedrony obsahující odstraňovaný bod p a vzniklý star-shaped polyhedron H je retriangulován postupným ořezáváním uší. Podrobněji je algoritmus popsán následujícím pseudokódem, jednotlivé body jsou vysvětleny dále.

Algoritmus odebrání bodu z *RT*:

1. Najdi všechny tetrahedrony obsahující bod p.
2. Odeber tyto tetrahedrony z triangulace a vytvoř z nich polyhedron H.
3. Najdi všechny uši polyhedronu H.
4. Pokud má polyhedron H více než 4 stěny, pokračuj bodem 5, jinak bodem 8.
5. Odřízni ucho s nejmenší prioritou.
6. Aktualizuj seznam uší.
7. Jdi zpět na bod 4.
8. Ze zbylých 4 stěn vytvoř tetrahedron, zapoj ho do triangulace a skonči.

[11] Odříznutím ucha q_i, q_{i+1}, q_{i+2} zaniknou uši q_{i-1}, q_i, q_{i+1} a $q_{i+1}, q_{i+2}, q_{i+3}$ a vzniknou uši q_{i-1}, q_i, q_{i+2} a q_i, q_{i+2}, q_{i+3}.

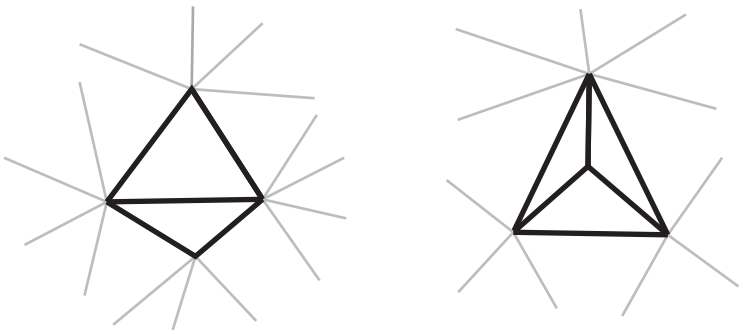
Polyhedron

Z triangulace jsou odstraněny všechny tetrahedrony obsahující bod *p*. Stěny, které v těchto tetrahedronech leží proti bodu *p*, tvoří dohromady hranici star-shaped polyhedronu *H* (tj. díry v triangulaci). Kromě samotných stěn polyhedronu *H* je nutné znát jejich vzájemné sousednosti (ty jsou použity při hledání uší polyhedronu). Polyhedron je tak vlastně uložen jako uzavřená trojúhelníková síť.

Dále pro každou stěnu potřebujeme znát tetrahedron „z druhé strany“, tedy tetrahedron který tuto stěnu obsahuje a leží mimo polyhedron *H*. Tato informace je použita při odříznutí ucha k jeho zapojení do triangulace. Každá stěna má tedy čtyři sousedy a k její reprezentaci lze použít odstraněný tetrahedron, jehož byla součástí.

Uši polyhedronu

Zde se dostáváme k hlavnímu rozdílu mezi *3D* a *2D* verzí algoritmu. Zatímco ve *2D* bylo ucho vždy tvořeno dvojicí sousedních hran polygonu, ve *3D* je situace složitější. Ucho totiž může být tvořeno dvojicí nebo trojicí sousedních stěn. Proto budeme dále rozlišovat 2-ucho a 3-ucho (zřejmě 2-ucho je tvořeno dvojicí a 3-ucho trojicí sousedících stěn polyhedronu *H*, viz obr. 3.1).



Obr. 3.1: Fragmenty trojúhelníkové sítě tvořící polyhedron *H*. (a) 2-ucho, (b) 3-ucho.

Uši jsou nalezeny procházením trojúhelníkové sítě reprezentující polyhedron *H*. Pro každé ucho je také výhodné znát jeho sousedy (tedy uši, se kterými sdílí nějakou stěnu polyhedronu *H*). Tato znalost je užitečná při aktualizaci uší.

Důležité je důsledně rozlišovat 2-uši a 3-uši. Zde platí pravidlo „větší bere“ – pokud některé tři stěny tvoří 3-ucho, nesmí být vytvořeny 2-uši, které by se skládaly pouze z těchto stěn. Vytvoření takového „duplicitního“ 2-ucha by způsobilo závažné problémy při retriangulaci polyhedronu *H*.

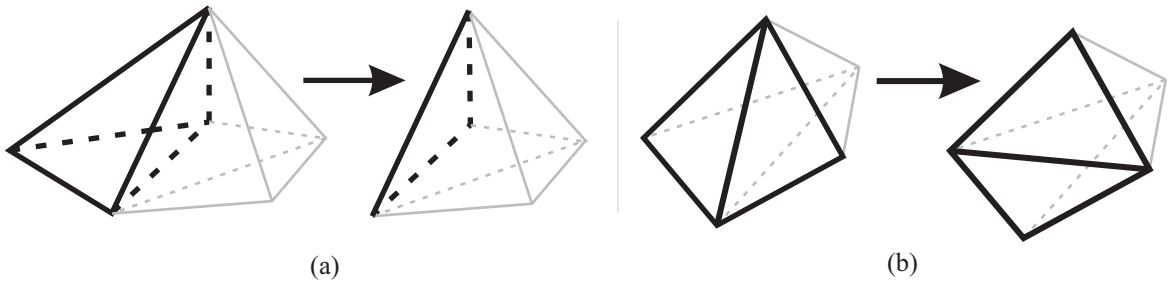
Priorita ucha

Priorita ucha je definována obdobně jako ve *2D* verzi algoritmu. Pokud stěny tvořící ucho svírají nekonvexní úhel, ucho nesmí být odříznuto a jeho priorita je nastavena na $+\infty$. V opačném případě je priorita ucha (tvořeného body *a*, *b*, *c*, *d*) vhledem k bodu *p* dána takto[12]:

$$priorita(a,b,c,d,p) = \frac{\begin{vmatrix} x_a & y_a & z_a & x_a^2+y_a^2+z_a^2-w_a & 1 \\ x_b & y_b & z_b & x_b^2+y_b^2+z_b^2-w_b & 1 \\ x_c & y_c & z_c & x_c^2+y_c^2+z_c^2-w_c & 1 \\ x_d & y_d & z_d & x_d^2+y_d^2+z_d^2-w_d & 1 \\ x_p & y_p & z_p & x_p^2+y_p^2+z_p^2-w_p & 1 \end{vmatrix}}{\begin{vmatrix} x_a & y_a & z_a & 1 \\ x_b & y_b & z_b & 1 \\ x_c & y_c & z_c & 1 \\ x_d & y_d & z_d & 1 \end{vmatrix}} \tag{3.2}$$

Odříznutí ucha

Vždy je odřezáváno ucho s nejmenší prioritou. Ucho je při odříznutí doplněno na tetrahedron a ten je připojen do triangulace. Při zpracování 3-ucha stačí doplnit ucho o jedinou stěnu. Tato stěna nahradí v polyhedronu *H* původní tři stěny ucha, polyhedron se tak zmenší o dvě stěny a jeden vrchol. Při odříznutí 2-ucha je nutné do ucha doplnit dvě stěny. Tyto nové dvě stěny nahradí v polyhedronu *H* původní dvě stěny ucha, polyhedron *H* tak zmenší svůj objem, ale počet jeho stěn i vrcholů zůstane nezměněn (není tedy možné odřezávat pouze 2-uši). Změnu polyhedronu *H* při odříznutí ucha ukazuje obr. 3.2.



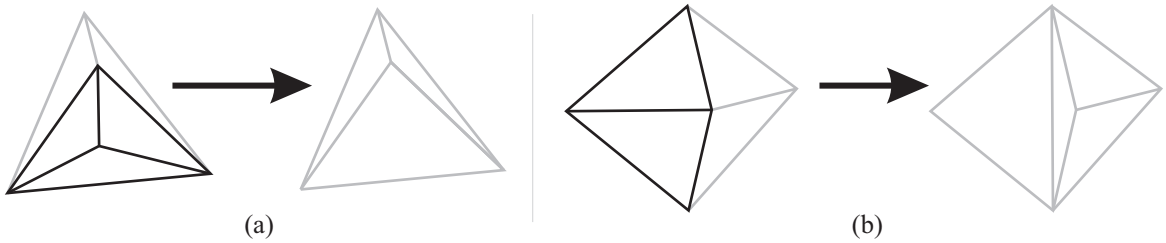
Obr. 3.2: Odříznutí 3-ucha a 2-ucha (odřezávané uši jsou vyznačeny silnou čarou). (a) Odříznutí 3-ucha: tři stěny 3-ucha jsou nahrazeny jednou. (b) Odříznutí 2-ucha: dvě stěny ucha jsou nahrazeny novou dvojicí stěn.

[12] Geometricky jde o vertikální vzdálenost bodu p^+ od nadroviny procházející body. a^+, b^+, c^+, d^+. p^+ značí transformaci bodu p do vyšší dimenze, viz kapitola 2.3.3.

Aktualizace uší

Po odříznutí ucha musí být seznam uší aktualizován. Nejjednodušším řešením by bylo znovu nalézt všechny uši upraveného polyhedronu H, tím by však výrazně vzrostla časová složitost algoritmu. Je zřejmé, že odříznutím ucha se trojúhelníková síť, reprezentující polyhedron H, změní pouze lokálně. Aktualizovány musí být uši, které se s odebraným uchem u překrývají, a také některé uši, které leží „vedle“ ucha u. Protože nejnáročnější operací při vytváření ucha je výpočet jeho priority, mělo by být při aktualizaci zachováno co nejvíce existujících uší. Dále jsou popsány změny v seznamu uší po odříznutí ucha u.

- Musí být odstraněno odříznuté ucho u.
- Musí být odstraněny všechny 2-uši, které s uchem u sdílely některou stěnu.
- Všechny 3-uši, které s uchem u sdílely některou stěnu, musí být upraveny na 2-uši (odebráním sdílené stěny).
- Všechny 2-uši, jejichž obě stěny sousedily s některým stěnami ucha u, musí být upraveny na 3-uši (přidáním některé nové stěny), viz obr. 3.3.
- Musí být přidány všechny nově vzniklé uši, které obsahují jednu nebo dvě nově přidané stěny a přitom se neshodují s uchem z předchozího bodu.



Obr. 3.3: Odříznutím černě vyznačeného 3-ucha (a) resp. 2-ucha (b) se vedlejší 2-ucho (vyznačeno šedě) změní na 3-ucho.

3.4.2 Složitost algoritmu

Paměťová a časová složitost algoritmu je závislá na počtu tetrahedronů, které obsahují odstraňovaný bod. Odstraněním k tetrahedronů z triangulace vznikne polyhedron H s k stěnami. Z těchto k stěn může být vytvořeno nejvýše $3k/2$ uší (jedna stěna může být součástí maximálně tří uší, jedno ucho se skládá minimálně ze dvou stěn). Při retriangulaci polyhedronu s k stěnami je počet odříznutých 3-uší přesně $(k-4)/2$ (odříznutím 3-ucha se počet stěn polyhedronu H zmenší o dvě, odřezávání uší končí ve chvíli, kdy má polyhedron pouze čtyři stěny). Počet odříznutých 2-uší nelze přesně

stanovit (nemusí být odříznuto žádné), nicméně nemůže být větší než $3k/2$ (tj. maximální počet uší na k stěnách)[13].

Shrnuto – celkový počet odříznutých uší je $O(k)$, na odříznutí jednoho ucha a následnou aktualizaci seznamu uší je zapotřebí $O(1)$ čas (počet uší, které musí být upraveny, se sice může mírně lišit, ale vždy jde pouze o lokální opravu nezávislou na k). Pokud jsou uši uložené v prioritní frontě, je čas potřebný k nalezení ucha s nejmenší prioritou a úpravě fronty po aktualizaci uší $O(\log k)$. Výsledná složitost algoritmu je $O(k \log k)$.

Protože v průměrném případě není k velké (pro triangulace proteinů se průměrné k pohybuje kolem 26) a protože údržba prioritní fronty je náročná (po aktualizaci uší je nutné z fronty odebrat zhruba 3 nebo 4 zaniklé uši a podobný počet nových uší vložit), nepřinese zřejmě použití fronty výrazné urychlení. Vyhledání ucha s minimální prioritou v nesetříděném seznamu má složitost $O(k)$, celková složitost algoritmu bez použití prioritní fronty je $O\left(k^2\right)$. Nicméně časově nejnáročnější operací je výpočet priority ucha, jejich počet je pouze $O(k)$.

3.4.3 Stabilita algoritmu

Algoritmus byl rozsáhle testován pro různé typy dat. Při odebírání bodů z triangulací proteinů a dalších objemových dat pracuje algoritmus bezproblémově. Potíže nastávají při odebírání z triangulací povrchových modelů. Ty obsahují úzké tetrahedrony, jejichž orientace je blízká nule, v důsledku čehož je výpočet priority ucha nestabilní[14]. Může tak nastat situace, kdy je odříznuto nesprávné ucho. To má v některých případech za následek neregularitu výsledné triangulace, jindy dochází k degeneraci polyhedronu H, například k jeho „zaškrcení“ a rozdělení na dvě části, které jsou spojené pouze hranou. V době odevzdání této práce se popsané problémy nepodařilo plně vyřešit. Nicméně chybné stavy je možné detekovat v seznamu uší a při nalezení chyby vrátit triangulaci do stavu před odebráním bodu.

[13] Ucho, které obsahuje obě nové stěny, vzniklé při odříznutí 2-ucha (jakýsi nástupce odříznutého 2-ucha), je nutně nekonvexní, a tedy nemůže být znovu odříznuto.

[14] Použití exaktní aritmetiky sice počet problematických bodů silně zredukuje, ale neodstraní problém zcela.

4. Voronoiovy diagramy

Pojem „Voronoiův diagram“ (*VD*) označuje širokou skupinu geometrických konstrukcí. Jejich základní vlastnost lze popsat takto: Pro danou množinu generátorů rozdělí *VD* prostor tak, že každý bod prostoru náleží tomu generátoru, ke kterému je „nejblíže“. Generátorem přitom může být libovolný geometricky popsatelný objekt, typicky se používá bod, koule či přímka. Dále popíšeme tři typy Voronoiových diagramů, které jsou použity v teoretické části této práce. Detailní popis těchto i dalších typů Voronoiových diagramů lze nalézt v [A91].

4.1 Obecná definice

Nechť je dána množina generátorů $S = \{p_0, p_1, ..., p_n\}$ v R^3 a funkce $D : R^3 \times R^3 \to R$.

Voronoiův region $VR(p_i)$ generátoru $p_i \in S$ je definován takto:

$$VR(p_i) = \left\{ x \middle| x \in R^3, D(p_i, x) \leq D(p_j, x), p_i \neq p_j, p_j \in S \right\} \tag{4.1}$$

Voronoiův diagram $VD(S)$ je sjednocení všech Voronoiových regionů $VR(p_i)$ pro $\forall p_i \in S$.

Průnik dvou Voronoiových regionů budeme nazývat **Voronoiova stěna**, průnik Voronoiových stěn **Voronoiova hrana** a průnik Voronoiových hran **Voronoiův vrchol**. Plochu, jejíž všechny body x splňují rovnost $D(p, x) = D(q, x)$, nazveme **bisektor** generátorů p, q. Zásadní význam má předpis funkce $D(p, x)$ a typ generátorů – spolu určují tvar Voronoiových regionů.

4.2 Voronoiův diagram pro body

Voronoiův diagram pro body (*VDB*) je nejjednodušším a zároveň nejpoužívanějším členem rodiny Voronoiových diagramů. Jak je z názvu patrné, jako generátory používá body v R^3. Funkce $D(p, x)$ je definována jako $D(p,x) = |px|$, kde $|px|$ označuje Euklidovskou vzdálenost bodů x a p.

U tohoto typu diagramu jsou Voronoiovy stěny rovinné konvexní polygony a Voronoiovy hrany jsou úsečky. Bisektor bodů p, q je rovina, která protíná spojnici bodů p, q v polovině a je na tuto spojnici kolmá. Voronoiův vrchol leží ve středu koule opsané čtveřici nejbližších generátorů (bodů).

4.3 Power diagram

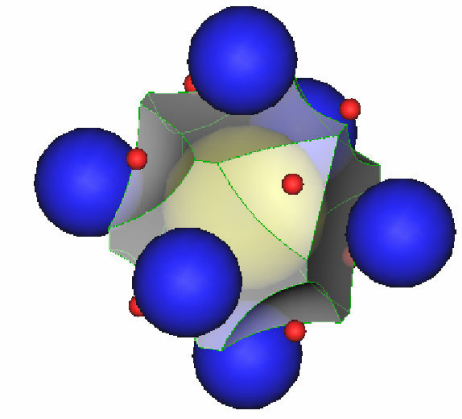
Power diagram (*PD*), někdy také nazýván vážený Voronoiův diagram, používá jako generátory vážené body. Funkce $D(p, x)$ je definována jako $D(p,x) = |xp|^2 - w_p$, kde w_p je váha bodu p. Geometricky lze interpretovat funkci $D(p, x)$ jako druhou mocninu délky tečny z bodu x na kouli se středem v p a poloměrem $\sqrt{w_p}$ (viz obr. 2.2).

Stejně jako u Voronoiova diagramu pro body, i u power diagramu jsou Voronoiovy stěny rovinné konvexní polygony a Voronoiovy hrany úsečky. Bisektor bodů p, q je rovina kolmá na spojnici bodů p, q, ale na rozdíl od *VDB* nemusí tuto spojnici protínat v polovině – dokonce ji nemusí protínat vůbec. Poloha bisektoru je ovlivněna váhami bodů p, q, zhruba řečeno je bisektor blíže k bodu s nižší vahou. Voronoiův vrchol leží ve středu koule, která je ortogonální na generátory (vážené body), v průniku jejichž Voronoiových regionů vrchol leží.

Voronoiovy regiony některých generátorů mohou být v *PD* prázdné – tyto generátory nijak výsledný diagram neovlivní a nazývají se redundantní.

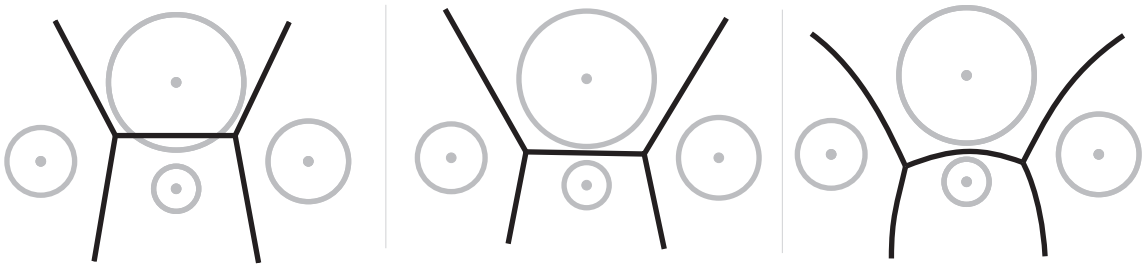
4.4 Euklidovský Voronoiův diagram

V Euklidovském Voronoiovu diagramu (*EVD*) ve *3D*, nazývaném také Voronoiův diagram koulí, jsou jako generátory použity koule. Funkce $D(p, x)$ pak udává vzdálenost bodu x od povrchu koule p, tedy $D(p,x) = |xp| - r_p$, kde r_p je poloměr koule p. Voronoiovy stěny jsou hyperboloidy, Voronoiovy hrany jsou planární kuželosečky a jsou popsatelné racionální kvadratickou Beziérovou křivkou [Y04]. Voronoiův vrchol leží ve středu koule, která je tečná na generátory (koule), v průniku jejichž Voronoiových regionů vrchol leží. Voronoiův region v *EVD* ukazuje obrázek 4.1.



Obr. 4.1: Voronoiův region žluté koule, kterou obklopuje dalších čtrnáct koulí. Převzato z [Y04].

Na obrázku 4.2 jsou porovnány tři popsané typy Voronoiových diagramů ve *2D*. Voronoiův diagram pro body (4.2a) zcela ignoruje váhy bodů (poloměry kružnic). Power diagram (4.2b) váhy bodů zohledňuje, jeho hrany neprotínají kružnice (pokud tyto nemají společný průnik). V *EVD* (4.2c) jsou hrany kuželosečky a vzdálenosti libovolného bodu hrany od kružnic, které určují hranu, jsou shodné.



(a) (b) (c)

Obr. 4.2: Voronoiovy diagramy ve *2D*. (a) Voronoiův diagram pro body, (b) power diagram, (c) Euklidovský Voronoiův diagram.

4.5 Dualita

Pojem dualita má v matematice velké množství významů, obecně řečeno dualita převádí větu, koncept nebo strukturu na jinou větu, koncept nebo strukturu. Přitom platí, že A je duální k B právě tehdy, když je také B duální k A. Například tvrzení „Přímka v rovině je dána dvěma body“ je duální k tvrzení „Bod v rovině je dán dvěma přímkami“.

Dále se zaměříme na dualitu mezi Voronoiovým diagramem bodů a Delaunayovou triangulací ve *3D*. Stejný princip platí pro *VD* a *DT* v obecné dimenzi a velmi podobný pro power diagram a regulární triangulaci[15].

Pro $i = 0,\ldots,3$ existuje pro každý i-dimenzionální simplex Voronoiova diagramu odpovídající duální $(3-i)$-dimenzionální simplex v Delaunayově triangulaci.

- Tetrahedronu *t* Delaunayovy triangulace odpovídá vrchol *v* Voronoiova diagramu, přičemž vrchol *v* leží ve středu koule opsané tetrahedronu *t*.
- Stěně *s* Delaunayovy triangulace odpovídá Voronoiova hrana *h*. Přímka, na které hrana *h* leží, je kolmá na stěnu *s* a prochází středem kružnice opsané této stěně.
- Hraně *h* Delaunayovy triangulace odpovídá Voronoiova stěna *s*, hrana *h* spojuje dva generátory, které určují stěnu *s*.

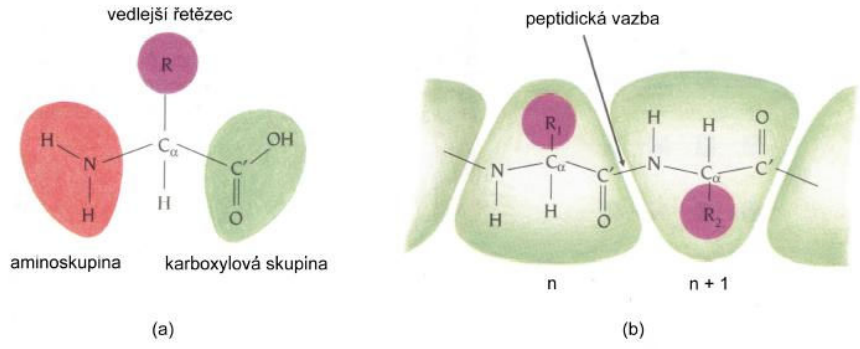
[15] Pokud v dále popsaných vztazích mezi *DT* a *VD* nahradíme pojmy „opsaná koule“ resp. „opsaná kružnice“ pojmy „ortogonální koule“ resp. „ortogonální kružnice“, získáme popis duality mezi power diagramem a regulární triangulací.

- Vrcholu *v* Delaunayovy triangulace odpovídá Voronoiova buňka, vrchol *v* je generátorem této buňky.

Díky dualitě lze výpočty prováděné ve Voronoiovu digramu převést na výpočty v Delaunayově triangulaci a naopak. Také je možné z jedné struktury vygenerovat její duál v lineárním čase. Dualitu Voronoiova diagramu a Delaunayovy triangulace využívá výpočet tunelů v proteinech, který je detailně popsán v následujících kapitolách.

5. Proteiny

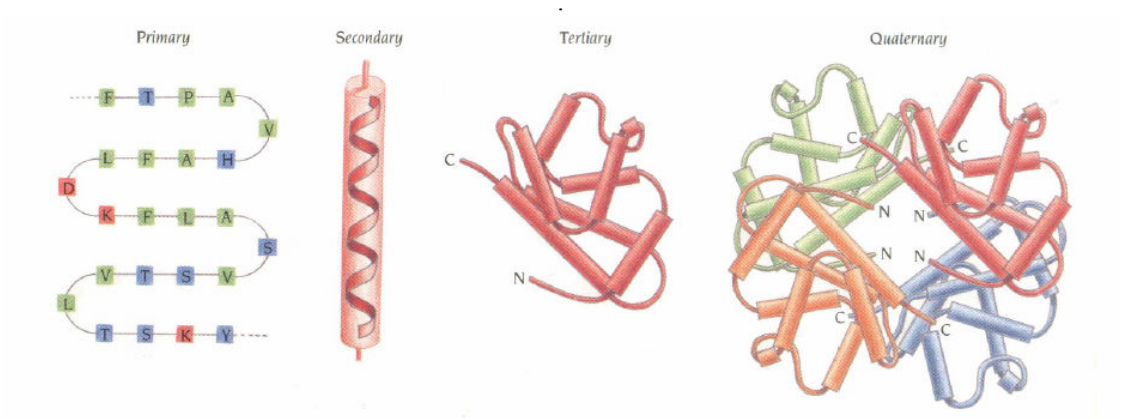
Proteiny (bílkoviny) jsou tvořené několika desítkami až tisíci aminokyselin, spojených peptidickou vazbou. Jsou obsaženy ve všech buňkách živých organismů, kde plní různé funkce – stavební, transportní, obranné, řídící atd. Obrázek 5.1a zachycuje strukturní vzorec aminokyseliny, obrázek 5.2b fragment polypeptidového řetězce (proteinu).



Obr. 5.1: (a) Aminokyselina, (b) polypeptidový řetězec. Převzato z [B99].

5.1 Struktura proteinu

Pořadí aminokyselinových zbytků v polypeptidovém řetězci udává jeho primární strukturu a při syntéze bílkovin v organismu je určeno genetickou informací. Uspořádání polypeptidového řetězce do tvaru šroubovice či do neuspořádaných úseků tvoří sekundární strukturu. Terciární struktura je dána interakcemi mezi vzdálenějšími částmi polypeptidového řetězce. Pokud se protein skládá z více hlavních řetězců (ale vždy stejných), nazývá se jejich vzájemná poloha kvartérní struktura (viz obr. 5.2).



Obr. 5.2: Hierarchie struktur proteinů. Převzato z [B99].

5.2 Geometrický model proteinu

Pro počítačové zpracování proteinu je nutné vytvořit jeho zjednodušený model. V této práci je použit následující:

- Jednotlivé atomy proteinu jsou aproximovány koulí. Atom je pak popsán souřadnicemi středu koule a její velikostí, která je dána Van der Waalsovým poloměrem odpovídajícího typu atomu (ten je udáván v jednotách angstrom [A], $1A = 1.0 \times 10^{-10}$ metru). Tento popis atomů jednotlivých proteinů lze získat na serveru www.pdb.org v podobě *pdb* souborů[16].
- Znalost chemických vazeb mezi atomy není užita, v modelu se projeví pouze částečným překrytím koulí.

5.3 Modifikace proteinů a aktivní místa

Drobné změny primární struktury mohou vést k podstatné změně vlastností dané bílkoviny. Této skutečnosti využívá proteinové inženýrství, které se cílenými mutacemi snaží změnit (vylepšit) vlastnosti jednotlivých proteinů.

Bod, ve kterém má být protein upraven, budeme nazývat aktivním místem. Při úpravě proteinu je důležité k aktivnímu místu (nebo od něj) dopravit určitou molekulu, aniž by byla narušena struktura proteinu (i sekundární, terciární a kvartérní struktura bílkoviny ovlivňují její funkci). Proto je důležitá otázka existence a velikosti „tunelu“ vedoucího z aktivního místa proteinu na jeho povrch. Samotná existence tunelu samozřejmě není postačující (ani nutnou) podmínkou toho, zda je určitá modifikace proteinu možná. Nicméně znalost geometrické aproximace tunelů v molekule může pomoci odborníkům zaměřit pozornost na její patřičné části.

5.4 Tunely

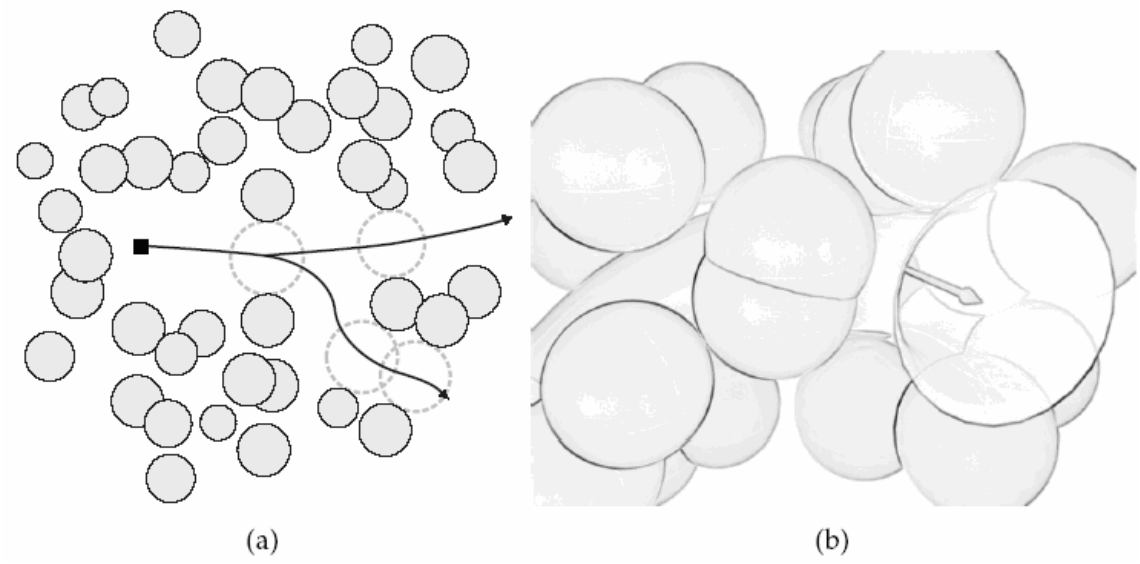
V této kapitole upřesníme pojem tunel a odvodíme způsob hledání tunelu v Euklidovském Voronoiově diagramu. V praxi (například v této práci) je často místo Voronoiova diagramu používána jeho duální struktura – Delaunayova triangulace[17]. Proto v 6. kapitole popíšeme i způsob hledání tunelu v triangulacích. Celá tato kapitola čerpá ze [M07, B06].

[16] Různými metodami (X-Ray diffraction, NMR, ER) bylo analyzováno již několik desítek tisíc proteinů. Výsledkem jsou mimo jiné *pdb* (Protein Data Bank) soubory.

[17] Pro uložení *DT* v paměti lze použít jednodušší a efektivnější struktury než pro *VD*. Dále při výpočtu vrcholů *VD* dochází k numerickým nepřesnostem, naproti tomu *DT* používá jako vrcholy přímo vstupní body.

5.4.1 Tunel ve Voronoiovu diagramu

Látku, která má proniknout do proteinu, budeme aproximovat koulí, která látku plně obklopuje. Při hledání tunelu se tak můžeme omezit na tunely s kruhovým příčným průřezem (viz obr. 5.5), což výrazně zjednoduší výpočet. Tunel bude tvořen koulemi, které neprotínají žádný atom proteinu. Středy těchto koulí budou určovat osu tunelu. Dále formalizujeme pojem tunel a popíšeme princip výpočtu tunelu.



Obr. 5.3: (a) Tunel ve *2D*, (b) tunel ve *3D*. Převzato z [B06].

Minimální vzdálenost bodu x od nejbližší koule (atomu) je dána funkcí $r(x) = \min\{D(p,x) \mid p \in S\}$, kde $D(p,x) = |xp| - r_p$. Tunel T vedoucí z bodu u (aktivního místa) do bodu v (bod na povrchu proteinu) je dán svou osou a_T a svým objemem. Osa je spojitá křivka, objem je dán sjednocením všech koulí se středem v $x \in a_T$ a poloměrem $r(x)$.

Střed nejmenší koule tunelu nazveme **nejužší bod** tunelu[18]. Poloměr tunelu v jeho nejužším bodě budeme stručně nazývat **šířka tunelu**. Podle šířky tunelu budeme různé tunely vedoucí do stejného aktivního místa hodnotit – naším cílem je nalézt tunel s maximální šířkou[19]. Takový tunel nazveme **ideální tunel**.

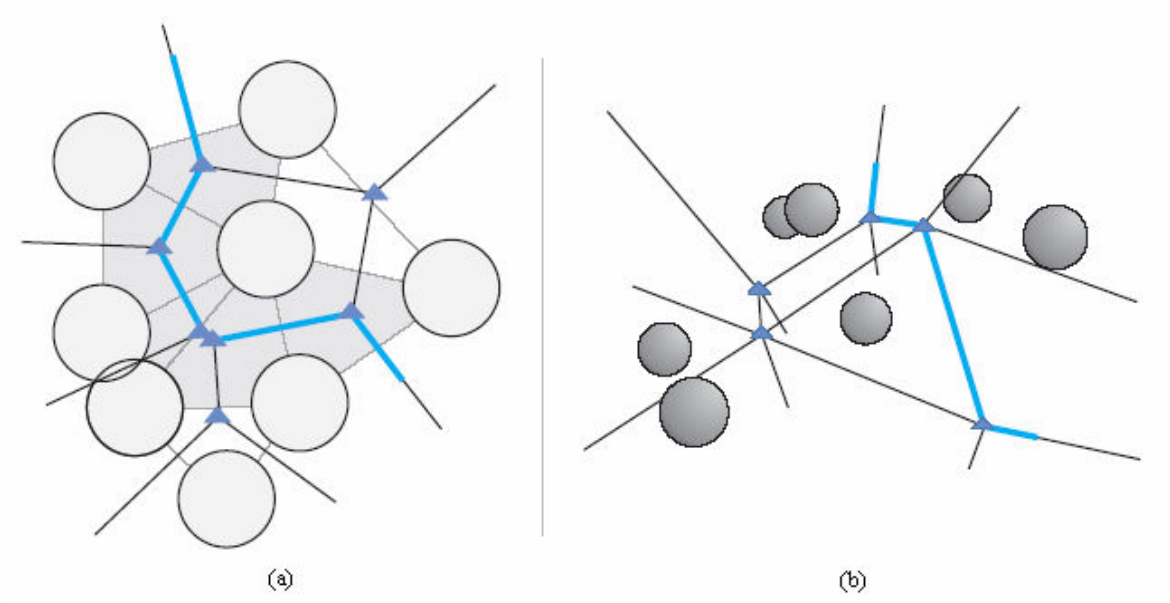
[18] Tunel může mít více nejužších bodů.

[19] Další možná kritéria by mohla být například délka, objem, nebo křivost tunelu. Nicméně v této práci se jimi nebudeme zabývat.

Postup hledání tunelů vychází z následujícího lemma ([M07], důkaz tamtéž):

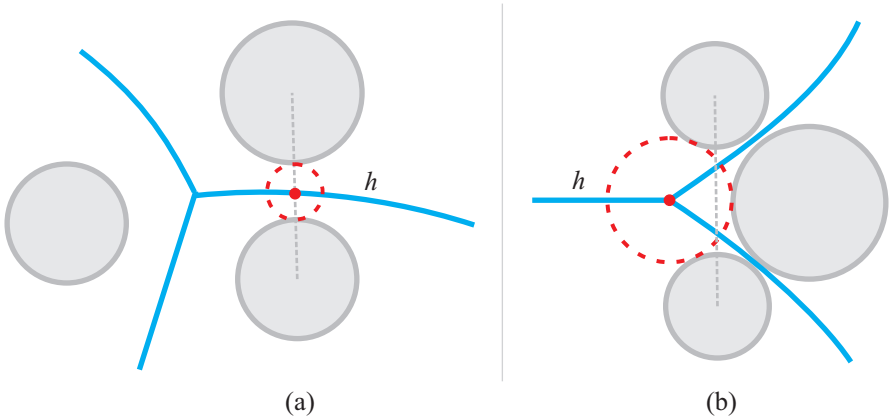
Lemma: Nechť S je množina koulí ve $3D$, $|S| > 1$. Uvažujme ideální tunel T s osou a_T vedoucí z bodu u do bodu v. Jestliže u ani v není nejužším bodem tunelu, pak alespoň jeden nejužší bod tunelu leží na některé Voronoiově stěně Euklidovského Voronoiova diagramu množiny S.

V důsledku tohoto lemma je možné se při hledání osy tunelu v molekule omezit na stěny Euklidovského Voronoiova diagramu této molekuly. Dokonce je podle [M07, B06] možné uvažovat pouze hrany diagramu. Osa tunelu tedy bude tvořena posloupností hran *EVD* (viz obr. 5.4).



Obr. 5.4: Osa tunelu je tvořena Voronoiovými hranami. (a) Tunel ve *2D*, (b) ve *3D*. Převzato z [B06].

Při hledání ideálního tunelu jsou hrany diagramu ohodnoceny podle šířky svého nejužšího bodu. Ten leží v průsečíku hrany a roviny procházející středy koulí, které hranu určují. Pokud takový průsečík neexistuje, je nejužším bodem hrany jeden její koncový bod – Voronoiův vrchol. Význam nejužšího bodu hrany je následující: jestliže po hraně „projede“ koule, jejíž poloměr je menší nebo roven šířce nejužšího bodu hrany, neprotne žádnou kouli triangulace (= atom proteinu). Obě možné polohy nejužšího bodu hrany ukazuje obr. 5.5.



Obr. 5.5: Nejužší bod Voronoiovy hrany ve *2D*. Střed přerušované kružnice vyznačuje nejužší bod hrany *h*, kružnice samotná poloměr tunelu v tomto bodě. (a) Nejužší bod hrany *h* leží na jejím průsečíku se spojnicí středů atomů, (b) nejužší bod hrany *h* leží ve Voronoiově vrcholu na konci hrany *h*.

Protože výpočet Euklidovského Voronoiova diagramu (*EVD*) je náročný na čas i paměť, je možné místo něj použít Voronoiův diagram pro body (*VDB*) nebo power diagram (*PD*). Voronoiův diagram pro body zcela ignoruje poloměry atomů, power diagram je jimi ovlivněn, ale stále se od *EVD* liší. Proto lze očekávat, že tunely nalezené v různých typech diagramů budou rozdílné. Avšak protože rozdíl poloměrů atomů je v proteinech malý (nejmenší vodík má poloměr 1,2A, největší síra 1,85A), budeme považovat Voronoiův diagram pro body i power diagram za „dostatečně dobrou“ aproximaci Euklidovského Voronoiova diagramu. Aby se celá věc ještě trochu zkomplikovala, použijeme pro hledání tunelů místo *VDB* a *PD* jejich duální struktury – Delaunayovu a regulární triangulaci. Tím už není do výpočtu zanášena další nepřesnost, pouze si ulehčujeme práci (s triangulacemi se snadněji zachází). Popis hledání tunelů v triangulacích je vysvětlen v následující kapitole.

6. Užití triangulací pro výpočet tunelu

Nyní vysvětlíme výpočet tunelů v triangulacích. Popis v kapitole 6.1 předpokládá, že máme triangulaci, která je duální k Euklidovskému Voronoiovu diagramu. Tuto triangulaci nazveme *EDT*.

V kapitolách 6.2 popíšeme, jak modifikovat výpočet tunelu, pokud je místo *EDT* použita „obyčejná“ Delaunayova nebo regulární triangulace[20]. Připomeňme, že jedním z cílů této práce je prozkoumat výhody a nevýhody použití regulární triangulace pro hledání tunelů v proteinech. Pro srovnání použijeme tunely v Delaunayově triangulaci.

6.1 Tunel v *EDT*

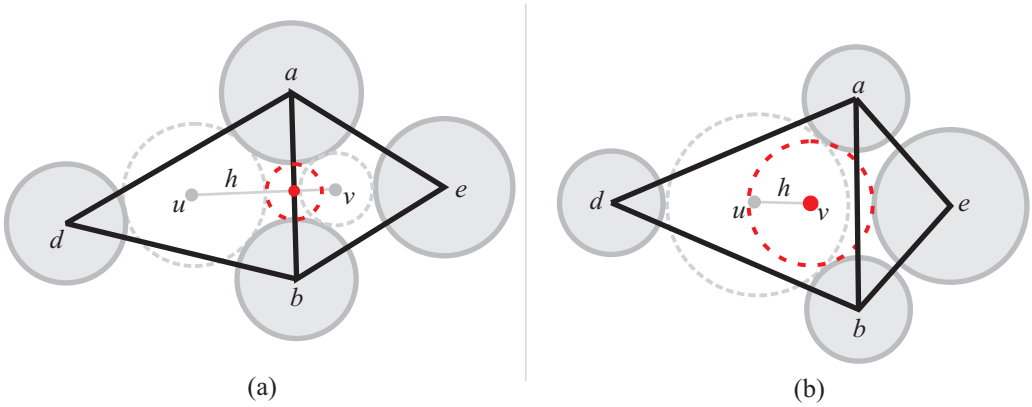
Předpokládejme, že máme triangulaci *EDT*, která je duální k Euklidovskému Voronoiovu diagramu. Důležitou vlastností *EDT* vyplývající z její duality s *EVD* je, že každý tetrahedron triangulace splňuje kritérium prázdné tečné koule – tzn. pro každý tetrahedron existuje koule, která se dotýká atomů (koulí) ve vrcholech tetrahedronu a žádný jiný neprotíná ani neobsahuje.

EDT je možné použít k výpočtu tunelu. Hrana v Euklidovském Voronoiově diagramu je duální ke stěně v *EDT*, přechod po hraně diagramu tak odpovídá přechodu mezi sousedními tetrahedrony v *EDT*. Triangulaci lze chápat jako graf G – tetrahedrony odpovídají uzlům v G, stěna sdílená dvěma sousedními tetrahedrony odpovídá hraně v G. Každá hrana v G je popsána trojicí koulí – dvě určují její koncové uzly a třetí její nejužší bod. Poloměr této třetí koule je použit jako ohodnocení hrany. Tyto tři koule jsou použity také při vizualizaci tunelu, jejich výpočet je popsán dále:

Mějme dva tetrahedrony *abcd* a *abce* se společnou stěnou *abc*. V grafu G odpovídá tetrahedronu *abcd* uzel u a tetrahedronu *abce* uzel v, společné stěně odpovídá hrana $h = (u, v)$. Souřadnice uzlu u jsou určeny jako střed prázdné koule tečné na atomy (koule) *a, b, c, d*. Obdobně jsou určeny souřadnice uzlu v.

Pokud existuje průsečík hrany h a stěny *abc*, je nejužším bodem hrany tento průsečík. Váha hrany je dána jako poloměr koule tečné na atomy a, b, c se středem v průsečíku hrany h a stěny *abc*. Pokud průsečík neexistuje, je nejužším bodem hrany h uzel u nebo v a váha hrany je dána poloměrem tečné koule se středem v tomto uzlu (viz obr. 6.1).

[20] Tato kapitola částečně čerpá z [M07, B06], poznatky týkající se regulární triangulace jsou autorovy.



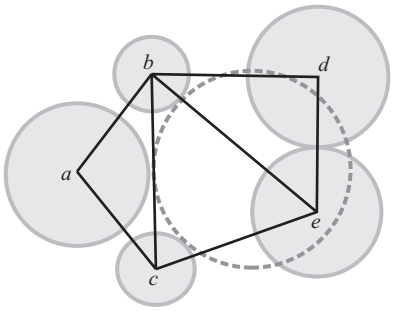
Obr. 6.1: Nejužší bod hrany ve *2D*. Nejužší bod hrany *h*, která spojuje uzly *u* a *v*, leží ve středu přerušované kružnice. (a) Nejužší bod leží na průsečíku hrany *h* a hrany *ab*, (b) nejužší bod hrany splývá s uzlem *v*.

Graf *G* je prohledán hladovou verzí Dijkstrova algoritmu – do množiny známých uzlů je v každém kroku přidán neznámý uzel, do nějž vede z některého známého uzlu hrana s nejvyšším ohodnocením. Důležité je, že pro nalezení tunelu není nutné znát ohodnocení a přesné souřadnice vrcholů všech hran grafu *G*, stačí je vypočítat pro ty hrany, které jsou v průběhu hledání opravdu navštíveny. Tento algoritmus je detailně popsán v kapitole 6.3.

6.2 Tunel v regulární a Delaunayově triangulaci

Algoritmus konstrukce *EDT* je implementačně i časově náročný, například algoritmus konstrukce Euklidovského Voronoiova diagramu (duálu *EDT*), který je popsán v [Y04], má pro *n* koulí časovou složitost $O(n^3)$ v nejhorším případě. Proto je v této práci použita pro výpočet tunelů místo *EDT* regulární, resp. Delaunayova triangulace. Jejich užití je ekvivalentem aproximace Euklidovského Voronoiova diagramu power diagramem, resp. Voronoiovým diagramem pro body.

Při použití *RT* nebo *DT* namísto *EDT* se postup hledání tunelů popsaný v předchozí kapitole změní jen velmi málo. Triangulaci můžeme stále chápat jako graf *G* s oceněnými hranami, v němž hledáme cestu maximalizující minimální váhu hrany (cestu bez úzkých úseků). Změní se pouze výpočet hran grafu *G*, přesněji souřadnic jejích koncových bodů a jejího nejužšího bodu. Důvod je zřejmý – tetrahedrony v *RT* ani *DT* nesplňují kritérium prázdné tečné koule, které jsme předpokládali u *EDT*. Koule, která je tečná na atomy ve vrcholech tetrahedronu v *RT* či *DT*, může protínat i obsahovat mnoho dalších atomů (viz obr. 6.2), což v *EDT* nastat nemohlo. Proto musíme výpočet hran grafu *G* přizpůsobit vlastnostem regulární a Delaunayovy triangulace.



Obr. 6.2: Kružnice vyznačená přerušovanou čarou, která je v regulární triangulaci tečná na atomy *a, b, c* (vrcholy trojúhelníku triangulace), protíná atomy *d* a *e*.

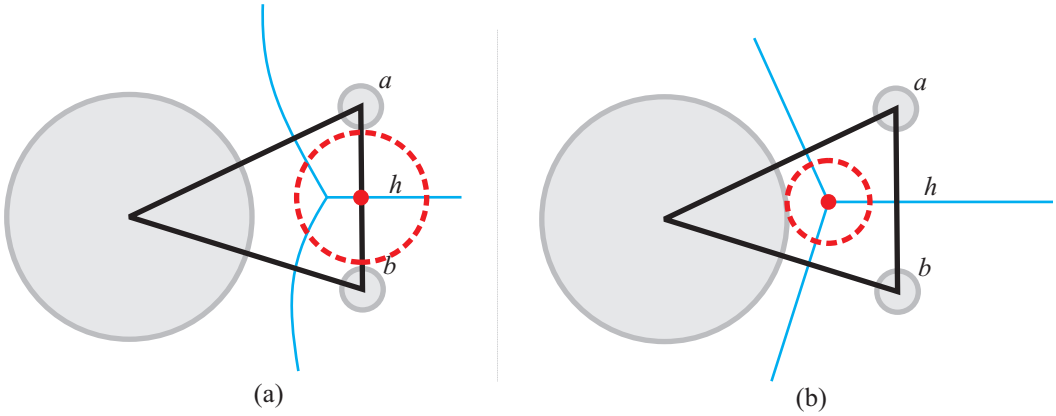
Tetrahedrony v Delaunayově triangulaci splňují známé kritérium prázdné opsané koule: koule procházející vrcholy tetrahedronu neobsahuje ve svém vnitřku žádný jiný vrchol. Tetrahedrony regulární triangulace splňují kritérium prázdné ortogonální koule. To lze zhruba vyjádřit takto: Mějme kouli *K*, která je ortogonální na koule (vážené body) ve vrcholech tetrahedronu. Pak všechny ostatní koule ve vrcholech triangulace jsou „příliš daleko na to, aby byly na kouli *K* také ortogonální“ (přesné vyjádření viz kapitola 2.2, definice globální regularity).

Vzhledem k těmto kritériím budeme v *DT*, resp. v *RT*, hledat souřadnice konců hran grafu *G* jako středy koulí opsaných tetrahedronům, resp. koulí ortogonálních na atomy ve vrcholech tetrahedronů[21]. Tyto koule však stále protínají některé atomy. Je tedy nutné nějakým způsobem zmenšit jejich poloměr. Podle typu této úpravy rozlišíme **pesimistický** a **optimistický** tunel. Pesimistický tunel nesmí obsahovat chyby, tzn. nesmí do něj zasahovat žádný atom proteinu. Cenou za tuto bezchybnost bude jeho menší průměr. U optimistického tunelu připustíme, že do něj atomy proteinu zasahovat mohou. Horní odhad velikosti chyby však bude známý. Oba typy tunelů popíšeme v následujících kapitolách.

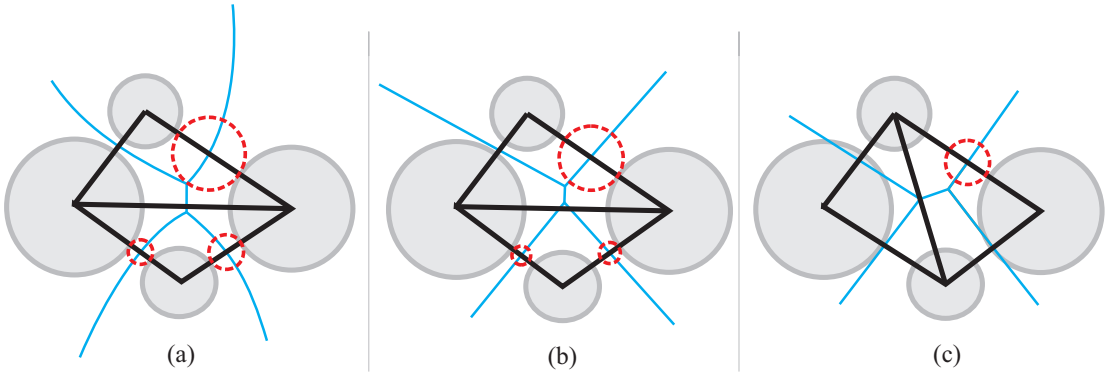
Změní se i výpočet nejužšího bodu hrany – v *RT* a *DT* může být nejužším bodem hrany jeden z jejích konců i v případě, že hrana protíná společnou stěnu tetrahedronů, které hranu určují. Na obrázku 6.3 je srovnání pozice nejužšího bodu hrany (střed přerušované kružnice) v *EDT* a *RT*. V *EDT* leží nejužší bod hrany *h* v průsečíku této hrany a spojnice atomů *a*, *b*, zatímco v *RT* leží v koncovém bodě hrany *h*.

Obrázek 6.4 ukazuje rozdíly v šířkách tunelů (vyznačeny přerušovanou kružnicí) v nejužších bodech hrany pro různé typy triangulací. Z obrázku je patrné, že tunely v *RT* a *DT* jsou užší než tunely v *EDT*, některé tunely v *DT* vůbec neexistují.

[21] Jinými slovy je z *DT* počítán Voronoiův diagram a z *RT* power diagram.



Obr. 6.3: Rozdíly v poloze nejužšího bodu hrany (vyznačen jako střed přerušované kružnice): (a) *EDT* a duální Euklidovský Voronoiův diagram, (b) *RT* a duální power diagram. V *RT* leží nejužší bod hrany *h* v jejím koncovém bodě, přestože hrana *h* protíná spojnici atomů *a*, *b* (hranu regulární triangulace).



Obr. 6.4: Rozdíly v šířkách tunelů v nejužších bodech hran (šířky tunelů jsou vyznačeny jako přerušované kružnice): (a) *EDT* a duální Euklidovský Voronoiův diagram, (b) *RT* a duální power diagram, (c) *DT* a duální Voronoiův diagram pro body.

6.2.1 Optimistický tunel

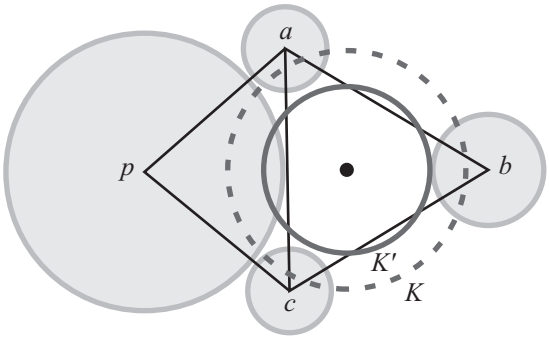
Při výpočtu optimistického tunelu v *DT*, resp. *RT*, je poloměr koule *K* vypočtené na tetrahedronu *abcd* zmenšen tak, aby výsledná koule *K'* neprotínala atomy (koule) *a*, *b*, *c*, *d* [22]. Obdobně pokud je při výpočtu nejužšího bodu hrany počítána koule ve stěně *abc*, je její poloměr zmenšen tak, aby neprotínala atomy *a*, *b*, *c*.

Do zmenšené koule *K'* však stále může v *DT* i *RT* zasahovat nějaký jiný atom – takový stav nazveme **chyba v tunelu**. Příklad takové chyby ve *2D* ukazuje obrázek 6.5. Kružnice *K'*, vypočítaná na trojúhelníku *abc*, protíná atom *p*, přestože nezasahuje do atomů *a*, *b*, *c*.

V *DT* je maximální možná velikost takovéto chyby rovna rozdílu poloměrů největšího a nejmenšího atomu v proteinu (typicky síra s poloměrem 1,85A a vodík s poloměrem 1,2A). V *RT* lze maximální chybu odhadnout přesněji v závislosti na

[22] Přesněji: koule *K'* na tetrahedronu abcd má poloměr $r_{K'} = \min\{|aK| - r_a, |bK| - r_b, |cK| - r_c, |dK| - r_d\}$.

lokální geometrii (viz následující kapitola 6.2.2), nikdy však nebude větší než maximální možná chyba v *DT*.



Obr. 6.5: Chyba v optimistickém tunelu v regulární triangulaci. Kružnice *K* vypočtená na trojúhelníku *abc* je na atomy *a, b, c* ortogonální. Jejím zmenšením vznikne kružnice *K*', která už do atomů *a*, *b*, *c* nezasahuje. Přesto protíná atom *p*.

6.2.2 Pesimistický tunel v *DT*

Pro výpočet pesimistického tunelu v *DT* jsou poloměry všech atomů v proteinu nastaveny na poloměr největšího atomu v proteinu. V důsledku toho je Euklidovský Voronoiův diagram totožný s Voronoiovým diagramem bodů, a tedy jsou totožné i duální struktury *EDT* a *DT*. Tunel pak lze hledat postupem popsaným v kapitole 6.1. Výsledný tunel nemůže obsahovat chyby – neprotíná ho žádný ze zvětšených atomů, a tedy ho nemůžou protínat ani původní, nezvětšené atomy. Jak už bylo zmíněno, cenou za jistotu bezchybnosti je užší tunel. Největší možné zmenšení poloměru tunelu oproti skutečnému stavu je dáno rozdílem poloměrů největšího a nejmenšího atomu v molekule.

6.2.3 Pesimistický tunel v *RT*

Shodný přístup by bylo možné použít i pro pesimistický tunel v regulární triangulaci, ta však tím, že zohledňuje váhy vrcholů (poloměry atomů), poskytuje lepší možnost. Při výpočtu koule v tetrahedronu či ve stěně je možné shora odhadnout velikost chyby (= jak hluboko do koule tunelu zasahuje nějaký atom). Po zmenšení poloměru koule o tento odhad chyby už nemůže koule žádný atom protínat. Horní odhad chyby je závislý na lokální geometrii, je tedy počítán pro každou kouli tunelu zvlášť. Složitost jeho výpočtu pro jednu kouli je O(1). Odvoďme nyní vzorec odhadu chyby .

Nechť *S* je množina vážených bodů, *PD*(*S*) je power diagram této množiny, *RT*(*S*) její regulární triangulace. Zdůrazněme, že vážený bod *p* s váhou w_p zde budeme chápat jako kouli se středem v *p* a poloměrem $r_p = \sqrt{w_p}$.

Osa tunelu v regulární triangulaci je tvořena posloupností hran power diagramu. Hrana v *PD*(*S*) leží v průsečíku bisektorů, tedy pro každý bod *z* hrany *h* existují alespoň tři vážené body $a,b,c \in S$ (generátory *PD* ~ vrcholy *RT* ~ atomy proteinu), pro které platí, že jejich power vzdálenost od bodu *z* je stejná[23]:

$$\pi_a(z) = \pi_b(z) = \pi_c(z) = w \tag{6.1}$$

Dále platí, že power vzdálenost ostatních vážených bodů $p \in S - \{a,b,c\}$ od bodu *z* je větší:

$$\pi_p(z) \geq w = \pi_a(z) = \pi_b(z) = \pi_c(z) \tag{6.2}$$

Nechť *a* má z koulí *a*, *b*, *c* největší poloměr ($r_a \geq r_b, r_a \geq r_c$). Potom poloměr koule *z* určíme jako:

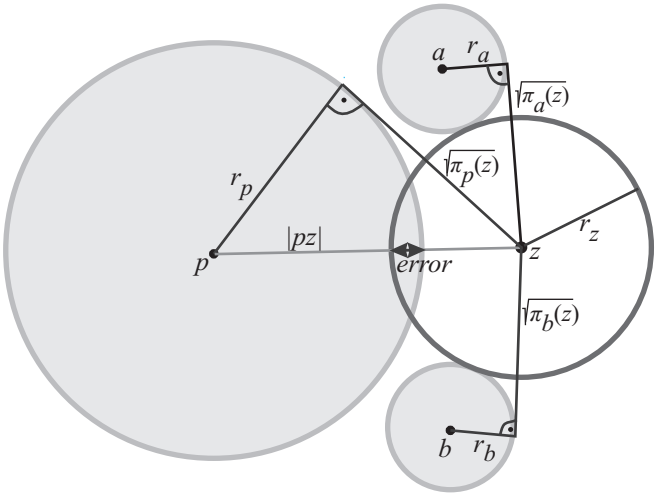
$$r_z = |az| - r_a \tag{6.3}$$

Při tomto poloměru se koule *z* dotýká koule *a* a neprotíná koule *b*, *c*.

Nyní nás zajímá maximální velikost chyby, tedy jak hluboko může libovolné $p \in S - \{a,b,c\}$ zasahovat do koule *z*, přesněji:

$$error = r_p + r_z - |pz| \tag{6.4}$$

Power vzdálenosti $\pi_a(z), \pi_b(z), \pi_p(z)$ a chybu *error* ukazuje obrázek 6.6. Velikost této chyby se budeme snažit shora odhadnout, aniž bychom při tom znali kouli *p* (známe pouze její maximální možný poloměr).



Obr. 6.6: Power vzdálenosti a chyba. Power vzdálenosti vážených bodů *a* a *b* od bodu *z* si jsou rovné a jsou menší než power vzdálenost váženého bodu *p* od bodu *z*. Kružnice *z* má takový poloměr, aby neprotínala kružnice *a* a *b*. Přesto protíná kružnici *p*.

[23] Připomeňme geometrický význam této rovnice pro kladné *w*: koule se středem v bodě *z* a poloměrem $r_z = \sqrt{w}$ je ortogonální na koule se středy v bodech *a*, *b*, *c* a poloměry r_a, r_b, r_c.

Pokud je výraz (6.4) kladný, zasahuje koule *p* do koule *z*. Pokud by byl výraz (6.4) vyhodnocován při výpočtu každé koule tunelu pro všechna $p \in S$, značně by tím vzrostla časová náročnost algoritmu hledání tunelu. Proto je výhodnější vzdálenost $|pz|$ zdola odhadnout (tím získáme horní odhad chyby).

Ze vztahu (6.2) plyne:

$$\pi_p(z) = |pz|^2 - r_p^2 \geq |az|^2 - r_a^2 = \pi_a(z)$$

$$|pz|^2 \geq |az|^2 - r_a^2 + r_p^2$$

$$|pz| \geq \sqrt{|az|^2 - r_a^2 + r_p^2} \tag{6.5}$$

Pokud za $|az|$ dosadíme podle vztahu (6.3), získáme:

$$|pz| \geq \sqrt{|az|^2 - r_a^2 + r_p^2} = \sqrt{(r_z + r_a)^2 - r_a^2 + r_p^2} = \sqrt{r_z^2 + 2r_a r_z + r_p^2} \tag{6.6}$$

Dosazením (6.6) do (6.4) získáme horní odhad chyby:

$$error = r_p + r_z - |pz| \leq r_p + r_z - \sqrt{r_z^2 + 2r_a r_z + r_p^2} \tag{6.7}$$

Ve výrazu (6.7) stále zůstává neznámá r_p. Pokud do levé strany nerovnosti dosadíme za r_p poloměr r_{max} největšího atomu v proteinu, nerovnost zůstane zachována[24]:

$$error = r_p + r_z - |pz| \leq r_{\max} + r_z - \sqrt{r_z^2 + 2r_a r_z + r_{\max}^2} \tag{6.8}$$

Určení poloměru koule v pesimistickém tunelu v *RT* pak vypadá takto:

1. Urči střed koule *z*.
2. Urči poloměr r_z koule *z* podle (6.3).
3. Vypočítej odhad chyby *error* podle (6.8) a odečti ho od r_z.

Do takto upravené koule nemůže zasahovat žádný atom a tunel popsaný těmito koulemi nemůže obsahovat chyby. Zbývá ukázat, že tento způsob výpočtu pesimistického tunelu je lepší, než prosté zvětšení všech atomů proteinu na velikost největšího atomu. Výraz (6.8) je možné chápat jako funkci proměnné r_z, tedy $y(r_z) = r_{\max} + r_z - \sqrt{r_z^2 + 2r_a r_z + r_{\max}^2}$. Analýzou průběhu funkce *y* zjistíme, že funkce *y* je

[24] O tom se lze snadno přesvědčit zderivováním funkce $y = r_{\max} + r_z - \sqrt{|az|^2 - r_a^2 + r_{\max}^2}$ podle r_{max}. Pro kladné r_{max} je funkce *y* rostoucí.

spojitá, **ostře rostoucí** (pro $r_a \neq r_{\max}$), $y(0) = 0$ a $\lim_{r_z \to \infty} y(r_z) = r_{\max} - r_a$. Nejdůležitější je monotónnost a limita. Při výpočtu pesimistického tunelu v *DT* je poloměr koule *z* zmenšen vždy přesně o $r_{\max} - r_a$. V *RT* je poloměr koule *z* zmenšen o $r_{\max} - r_a$ až při jejím nekonečném poloměru. A vzhledem k monotónnosti funkce *y* je zřejmé, že čím je poloměr r_z menší, tím méně bude zmenšen. Předpoklad, že pesimistické tunely nalezené v *RT* jsou širší než pesimistické tunely v *DT*, se potvrdil i při testech popsaných v kapitole 8.3.

Další výhodou *RT* oproti *DT* je fakt, že pesimistické tunely v *RT* zachovávají existenci tunelu – pokud v *RT* existuje k určitému místu optimistický tunel, vede k němu i pesimistický tunel (což u *DT* neplatí). To je dáno tím, že pro libovolné kladné r_z platí nerovnost $y(r_z) < r_z$. Tedy hodní odhad chyby, o který je tunel zúžen, nemůže být větší než šířka tunelu.

6.3 Algoritmus hledání tunelů

Naší snahou je k danému aktivnímu místu nalézt tunel s maximálním šířkou, tedy s největším poloměrem v nejužším bodě. K tomu lze použít hladovou verzi Dijkstrova algoritmu, která je popsána dále.

Triangulace je při hledání tunelů chápána jako graf *G* (viz kap. 6.1), tetrahedrony odpovídají uzlům a stěny hranám grafu *G*. Váha hrany je dána jejím nejužším bodem. Algoritmus hledání tunelů rozlišuje dva druhy uzlů: neznámé a expandované uzly. Neznámý uzel je uzel, do kterého ještě není známa optimální cesta. Expandovaný uzel je takový uzel, do kterého již byla optimální cesta nalezena. Algoritmus je popsán následujícím pseudokódem:

1. Nalezni uzel *u*, jemuž odpovídající tetrahedron obsahuje aktivní místo.
2. Vypočti ohodnocení hran, které z uzlu *u* vycházejí, a hrany spolu s jejich ohodnocením vlož do haldy (expanze uzlu). Uzel *u* označ jako expandovaný.
3. Vyjmi z haldy hranu *h* s maximálním ohodnocením.
4. Pokud koncový uzel *v* hrany *h* dosud nebyl expandovaný, pokračuj bodem 5, jinak bodem 3.
5. Označ uzel *v* jako expandovaný, zapamatuj si hranu, přes kterou jsi do *v* přišel.

6. Pokud z uzlu *v* vychází méně než čtyři hrany[25], pokračuj bodem 8, jinak bodem 7.
7. Vypočítej ohodnocení hran vedoucích z uzlu *v* do dosud neexpandovaných uzlů. Hrany a jejich ohodnocení vlož do haldy. Pokračuj bodem 3.
8. Zpětným průchodem od posledního expandovaného uzlu rekonstruuj tunel.

Pokud má algoritmus nalézt více tunelů k jednomu aktivnímu místu, nepřejde na bod 8 při nalezení prvního tetrahedronu ležícího na povrchu triangulace. Pouze si odpovídající „cílový“ uzel zapamatuje a pokračuje, dokud nenalezne potřebný počet cílových uzlů. Pro každý cílový uzel rekonstruuje tunel zvlášť.

Pokud nás zajímají pouze tunely, jejichž minimální poloměr je větší než určitá mez $r_{\min}$, stačí algoritmus upravit tak, aby do haldy vkládal pouze hrany s ohodnocením větším než $r_{\min}$.

Složitost algoritmu je závislá na počtu expandovaných uzlů, tento počet označíme *k*. Odebrání hrany z haldy má složitost $O(1)$, složitost vložení hran do haldy při expanzi uzlu je $O(\log k)$. Při *k* expandovaných uzlech je celková složitost $O(k \log k)$. Počet expandovaných uzlů se teoreticky může blížit počtu všech tetrahedronů v triangulaci, prakticky je mnohem nižší a doba hledání tunelů není z hlediska časové náročnosti kritickým místem programu.

[25] Tzn. některá stěna tetrahedronu, který odpovídá uzlu *v*, leží na konvexní obálce triangulace. Byl tedy nalezen tunel na povrch molekuly.

7. Návrh programu

7.1 Výběr algoritmů

Podle zadání má navržený program zvládat práci s rozsáhlými dynamickými daty. To značně ovlivnilo volbu algoritmů, které jsou v programu implementovány.

Pro konstrukci triangulace byl vybrán algoritmus inkrementálního vkládání, který je poměrně stabilní a umožňuje jednoduché vkládání nových bodů do triangulace. V jeho popisu v [F93, E92] bohužel není zahrnuta práce s redundantními body, implementovaný program proto předpokládá, že vstupní data redundantní body neobsahují[26] (což proteiny splňují).

K vyhledání tetrahedronu incidujícího s vkládaným bodem je použit algoritmus procházek. Další možností bylo užití rychlejšího DAGu, ten má ovšem oproti algoritmům procházek dvě nevýhody. Zaprvé potřebuje při k tetrahedronech výsledné triangulace další $O(k)$ množství paměti, což může způsobit při zpracování velkých dat potíže. Druhým problémem je, že odebrání bodu z triangulace způsobí v DAGu netriviální změny, které by algoritmus zkomplikovaly a zpomalily.

Odebírání bodů z triangulace má řešit rozšíření Devillersova algoritmu. Devillersův algoritmus pro odebírání bodů ve *2D DT* se zdál poměrně jednoduchý a rychlý. Bohužel jeho rozšíření do *3D* se ukázalo být pro některé typy dat málo stabilní (viz kap. 3.4.3).

7.2 Detaily implementace

V rámci této práce byl implementován algoritmus inkrementálního vkládání vytvářející regulární a Delaunayovu triangulaci (viz kap. 2.4), algoritmus odebírání bodů z triangulací (viz kap. 3.4) a algoritmus hledání tunelů (viz kap. 6.3). Program je doplněn o uživatelské rozhraní a jednoduchou vizualizaci proteinů, tunelů a triangulací. Vedlejším produktem této práce jsou dll knihovny pro konstrukci Delaunayovy a regulární triangulace, které by měly být použity v dalších projektech.

Pro implementaci byl použit jazyk C# a vývojové prostředí Microsoft Visual Studio 2005. Jazyk C# umožňuje efektivní tvorbu uživatelských rozhraní, má jednoduchou syntaxi a jakožto člen rodiny C jazyků je i dostatečně rychlý. Naopak jeho nevýhodou je obtížná práce s pointry, značně problematický by zřejmě byl i přenos aplikace

[26] Při pokusech s daty obsahujícími redundantní body program konvergoval, ovšem některé tetrahedrony nebyly regulární.

z prostředí Microsoft Windows pod jiný operační systém. Dále jsou popsány některé detaily implementace, které nejsou z popisu algoritmů zřejmé.

7.2.1 Použité datové struktury

Základními objekty celého programu jsou bezesporu tetrahedrony. Datová struktura pro uložení tetrahedronu je tvořena čtyřmi indexy na jeho vrcholy a čtyřmi indexy na jeho sousedy (dva tetrahedrony jsou sousední, pokud mají společné tři vrcholy). Existují i další způsoby uložení tetrahedronu (například stěnová reprezentace), nicméně v použité struktuře se autorovi „nejlépe uvažuje“.

Vstupní body i vygenerované tetrahedrony jsou uloženy v polích. S tím jsou spojeny určité problémy – u tetrahedronů není dopředu znám jejich počet, takže velikost pole pro jejich uložení může být pouze odhadnuta z počtu bodů. Maximální počet tetrahedronů na n bodech je sice $O(n^2)$, nicméně pro reálná data je jejich počet podstatně nižší, v závislosti na n pouze lineární. Proto je počáteční velikost pole tetrahedronů stanovena na $k^* n$ (konstanta k byla testováním stanovena na 15). Pokud je pole průběhem výpočtu zaplněno, je zvětšeno metodou Array.Resize(). To sice může pro extrémní vstupní data program zpomalit, ale pro většinu dat ke zvětšení pole vůbec nedojde.

Další drobnou komplikací jsou tetrahedrony zaniklé při swapech. Pokud by byly ponechávány v poli, výrazně by vzrostla paměťová náročnost programu. Proto jsou indexy zaniklých tetrahedronů uloženy do pomocného zásobníku a nově vzniklé tetrahedrony jsou ukládány na pozici z vrcholu zásobníku.

7.2.2 Implementace testů orientace a regularity

Testy orientace, regularity a in-sphere test lze převést na výpočet determinantu. Tento způsob výpočtu je poměrně robustní, přesto pokud je determinant implementován přímočaře jako vynásobení double-čísel, dochází v důsledku omezené přesnosti datového typu double k numerickým chybám. Tento problém z velké části řeší použití knihovny[27] profesora J. Shewchuka, která poskytuje metody pro exaktní výpočet orientačního a in-sphere[28] testu. Bohužel neobsahuje metodu pro výpočet testu regularity – pro ten je použita metoda z knihovny MathNet.Numerics[29]. Ta sice není

[27] http://www.cs.cmu.edu/~quake/robust.html

[28] Popravdě, pokusy s Shewchukovou knihovnou ukazují, že její in-sphere test také není zcela stabilní. Občas se výsledky testů stejných bodů (pouze zadaných v odlišném pořadí) liší. Ovšem k této chybě dochází v naprosto mizivém počtu případů.

[29] http://mathnet.opensourcedotnet.info/Default.aspx

exaktní, přesto jsou její výsledky přesnější než výsledky mnou implementovaného výpočtu determinantu (v testech v kapitole 8 bude přesto formálně označována jako exaktní, ovšem pouze pro jednoduché rozlišení použité aritmetiky).

Použití exaktních metod samozřejmě není zadarmo, jejich výpočet je časově náročný, což výrazně ovlivňuje dobu běhu programu (viz graf 8.1). Numerické problémy vznikají především při zpracování umělých dat (body ležící na pravidelné mřížce apod.) a povrchových dat. V mnoha případech jsou klasické testy dostačující. Proto je uživateli ponechána možnost vybrat si, který typ testů chce použít. Aplikace nabízí možnost zkontrolovat výslednou triangulaci. Pokud obsahuje chyby, může uživatel „zaškrtnout“ použití přesnější aritmetiky a spustit výpočet triangulace znovu.

7.2.3 Vizualizace

Aplikace obsahuje i jednoduchý prohlížeč, jehož implementace využívá grafické rozhraní Direct3D. Prohlížeč umožňuje zobrazit vygenerovanou triangulaci, model proteinu a samozřejmě nalezené tunely. Atomy proteinů jsou zobrazovány jako koule, druhy atomů jsou rozlišeny barvou. Pro vizualizaci tunelu je použita jeho osa (lomená čára) a množina koulí, které byly nalezeny při výpočtu tunelu. Pro kompaktnější vzhled tunelu by bylo možné dopočítat další koule se středy na ose tunelu, nicméně použitý způsob zobrazení tunelu se zdá dostačující. Prohlížeč umožňuje i „průlet“ po ose tunelu, při kterém jsou zobrazovány textové informace o šířce tunelu.

8. Testy a výsledky

V této kapitole jsou popsány výsledky porovnání regulární a Delaunayovy triangulace, tunelů v nich nalezených a testy algoritmů konstrukce triangulace a odebírání bodů z triangulace. Veškeré testy proběhly na stařičkém počítači Duron 1,4GHz s 512MB operační paměti a operačním systémem Windows XP.

8.1 Porovnání regulární a Delaunayovy triangulace

Jestliže porovnáváme použitelnost regulární a Delaunayovy triangulace pro hledání tunelů, je rozumná otázka, jak moc se výsledné triangulace stejné množiny vstupních bodů liší. Pokud by byly obě triangulace totožné, nemělo by použití regulární triangulace příliš smysl. Naopak pokud by byly obě triangulace (a tunely v nich nalezené) zcela odlišné, mohlo by to svědčit o špatné podmíněnosti úlohy a zpochybnit použití triangulací pro hledání tunelů. V této kapitole porovnáme podobnost triangulací, podobností tunelů se zabývá kapitola 8.3.6.

Regulární a Delaunayova triangulace se samozřejmě mohou pro obecná data totálně lišit, například jediný bod s velkou váhou může v *RT* „vytěsnit“ všechny body, které neleží na konvexní obálce. Nicméně z hlediska této práce je zajímavé především porovnání triangulací molekul, proto se při testech zaměříme pouze na ně.

Pro porovnání triangulací jsou použity tyto ukazatele:

- Počet tetrahedronů.
- Procento tetrahedronů obsažených v *RT* i *DT* (z celkového počtu tetrahedronů v *RT*).
- Procento stěn, které obsahuje *RT* i *DT* (z celkového počtu stěn v *RT*).

Výsledky testů čtyř různých molekul ukazuje tab. 8.1.

Soubor	# atomů	# atomů podle typu H, N, S, C	# tetra v *DT*	# tetra v *RT*	% stejných tetra	% stejných stěn
awj	1250	600, 132, 106, 1, 407	8474	8900	52	74
dbja	2332	0, 418, 412, 7, 1491	15646	15663	74	88
dhaa	4710	2319, 426, 401, 9, 1551	32489	35063	52	75
2f61	7864	0, 1422, 1342, 32, 5064	52364	53073	73	87

Tab. 8.1: Porovnání regulární a Delaunayovy triangulace proteinů. Ve třetím sloupci je udán počet atomů vodíku, dusíku, síry a uhlíku (v tomto pořadí) v proteinu.

Počet tetrahedronů vzniklých v *RT* a *DT* je prakticky totožný. Ve zbylých ukazatelích už se *RT* a *DT* liší. Triangulace molekul awj a dhaa, které obsahují atomy vodíku, obsahují zhruba 50% stejných tetrahedronů a 75% stejných stěn: Triangulace molekul

dbja a 2f61, které atomy vodíku neobsahují[30], si jsou podobnější – obsahují zhruba 75% stejných tetrahedronů a 88% stejných stěn. Pro další testované molekuly byly výsledky prakticky totožné. Spíše pro zajímavost byl proveden další test s modifikovanými daty. V molekule awj byl upraven poloměr atomů – části atomů byl nastaven poloměr na 1,2A, pro zbylé atomy byl v pěti testech postupně použit poloměr 1A, 0,8A, 0,6A, 0,4A a 0,2A. Výsledky testů podobnosti *RT* a *DT* (viz tab. 8.2) odpovídají očekávání – čím větší je rozsah poloměrů atomů, tím více se regulární a Delaunayova triangulace liší.

Upravené poloměry atomů [v A]		% stejných tetra	% stejných stěn
1,2	1	80	91
1,2	0,8	68	84
1,2	0,6	60	79
1,2	0,4	56	75
1,2	0,2	53	73

Tab. 8.2: Podobnost regulární a Delaunayovy triangulace pro modifikovaná data.

8.2 Časová a paměťová náročnost inkrementálního algoritmu

Pro praktickou použitelnost programu je důležitá doba vytváření triangulace a její paměťová náročnost, tj. především počet vzniklých tetrahedronů. Ten samozřejmě není závislý na použitém algoritmu, ale je dobré mít představu, jaká je závislost počtu tetrahedronů na počtu bodů pro typická vstupní data. Protože program je primárně určen pro triangulace proteinů, zaměříme se při testování především na ně, ale chování programu otestujeme i pro povrchová data.

8.2.1 Časová náročnost konstrukce *DT* a *RT*

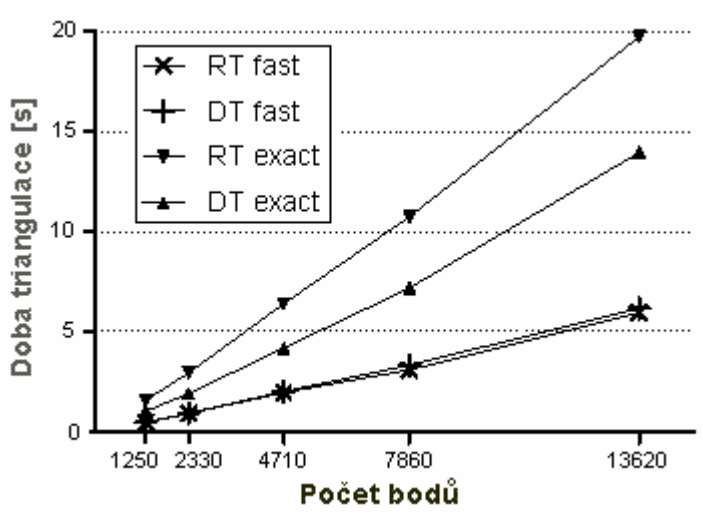
V prvním testu jsou porovnány časy konstrukce regulární a Delaunayovy triangulace při použití klasické (fast) a exaktní (exact) aritmetiky. Pro testování bylo použito pět proteinů s různým počtem atomů. Výsledné odhady časových závislostí zobrazuje graf 8.1, přesné hodnoty naměřených časů jsou v tabulce 8.3.

Z výsledku testu je zřejmé, že doba výpočtu *RT* a *DT* je při použití klasické aritmetiky totožná[31]. Použití exaktní aritmetiky způsobí nárůst doby výpočtu u *DT* na dvojnásobek u *RT* dokonce na trojnásobek[32].

[30] Protože tyto molekuly neobsahují atomy vodíku, nejedná se o proteiny. Nicméně pro testování to není na závadu.

[31] Konstrukce *RT* by se ovšem zpomalila, pokud by program zvládal práci s redundantními body.

[32] Připomeňme, že orientační test je pro *RT* i *DT* stejný, liší se pouze in-sphere test. V *DT* je pro něj použita exaktní Shewchukova knihovna, v *RT* numerická knihovna MathNet.Numerics.



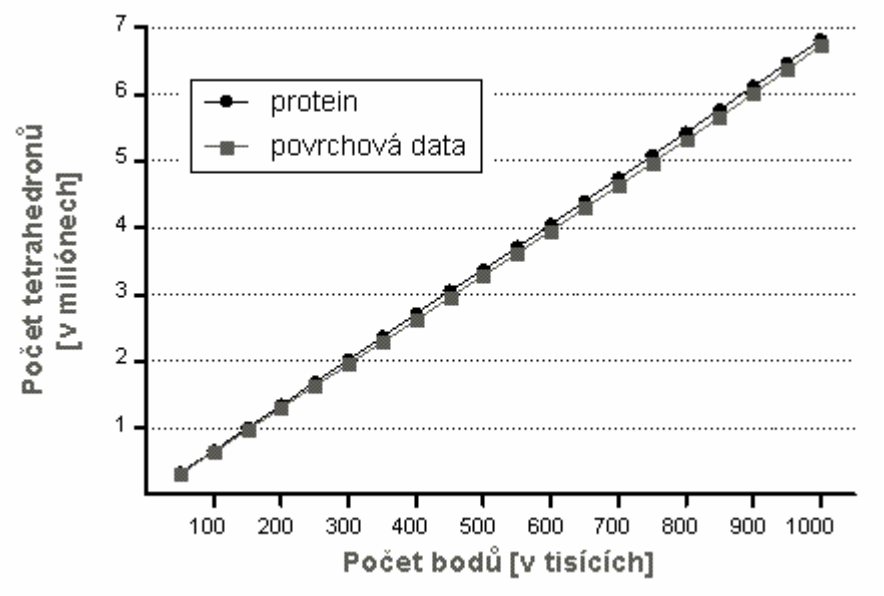
Graf. 8.1: Závislost doby konstrukce triangulace na počtu bodů, typu triangulace a použité aritmetice.

	Doba triangulace [s]			
	Klasická aritmetika		Exaktní aritmetika	
# atomů	*RT*	*DT*	*RT*	*DT*
1250	0,457	0,471	1,572	1,033
2332	0,904	0,920	2,944	1,912
4710	1,958	2,004	6,404	4,186
7864	3,092	3,326	10,757	7,200
13623	5,947	6,178	19,761	13,983

Tab. 8.3: Přesné časy konstrukce triangulací, použité v grafu 8.1.

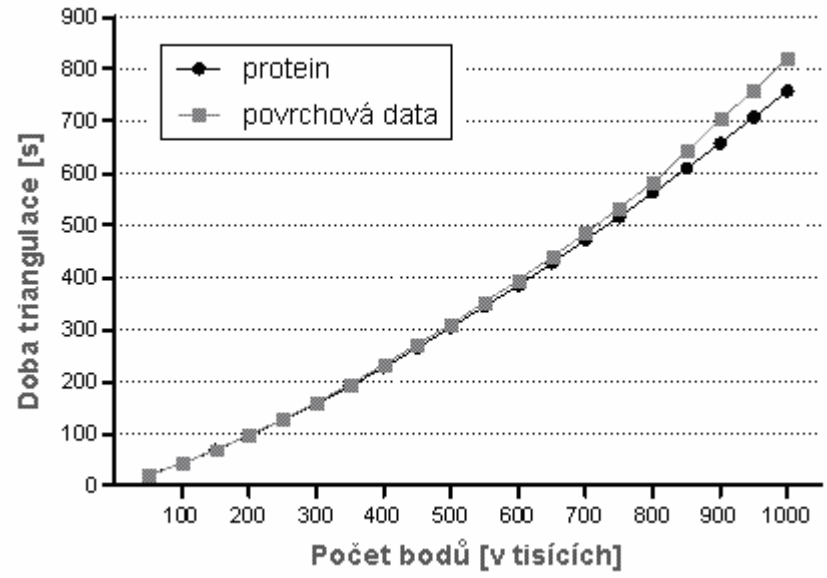
8.2.2 Zátěžový test

Další test je zaměřen na chování algoritmu při zpracování rozsáhlých dat, zajímat nás bude počet vzniklých tetrahedronů, časová náročnost a rozdělení času mezi vyhledání tetrahedronu a vložení bodu s následnou opravou triangulace. K testování byly použity dva soubory s miliónem bodů, sledované údaje byly odměřovány vždy po vložení dalších padesáti tisíců bodů. Oba soubory vznikly „naklonováním“ menších modelů, jeden z proteinu s šedesáti tisíci atomy, druhý z povrchového modelu zvonu s dvěma sty tisíci body. Při zpracování obou souborů byla použita klasická aritmetika.



Graf 8.2: Závislost počtu tetrahedronů regulární triangulace na počtu bodů.

Graf 8.2 ukazuje, že počet tetrahedronů je pro protein i pro povrchový model lineárně závislý na počtu bodů a je zhruba sedmkrát vyšší než počet bodů (obdobné jsou i výsledky pro další data). Tento údaj je důležitý při odhadu velikosti pole pro uložení tetrahedronů – v důsledku tohoto testu (a dalších podobných) je v programu počáteční velikost pole stanovena na patnáctinásobek počtu bodů. Graf 8.3 a tabulka 8.4 zobrazují závislost doby konstrukce regulární triangulace na počtu bodů. Ta roste rychleji než $O(n)$, ale pomaleji než $O\left(n^{4/3}\right)$.



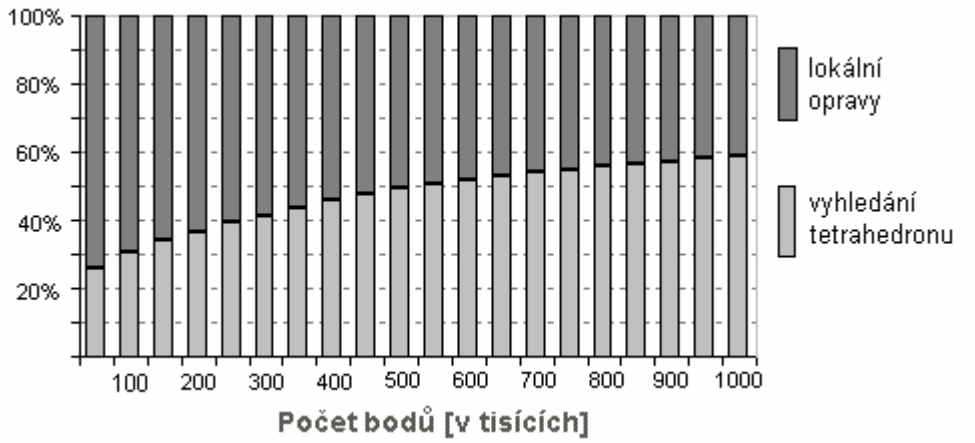
Graf 8.3: Závislost doby běhu konstrukce regulární triangulace na počtu bodů.

	Doba triangulace [s]			Doba triangulace [s]	
# bodů	Protein	Povrchový model	# bodů	Protein	Povrchový model
50000	20,4	20,3	550000	345,4	352,6
100000	43,5	43,6	600000	385,3	395,2
150000	69	69,4	650000	427,9	439,2
200000	96,8	97,5	700000	472,5	485,9
250000	126,2	127,9	750000	516,7	533,5
300000	157,4	160,1	800000	562,9	582
350000	192	195,2	850000	609,5	642,2
400000	228	232,7	900000	658	704,8
450000	265,3	271,4	950000	707,7	757,8
500000	304,6	311,1	1000000	758	820,7

Tab. 8.4: Přesné časy konstrukce regulární triangulace, použité v grafu 8.2.

Zajímavý je údaj, kolik času stráví program vyhledáváním tetrahedronů, do kterých má být vložen další bod, a kolik času při následném vložení bodu a opravě triangulace. Poměr těchto časů limituje další možné vylepšení programu. Z grafu 8.4 je zřejmé, že podíl času stráveného hledáním tetrahedronů se pro rostoucí data zvětšuje. Nicméně i pro milión bodů stále tvoří pouze 60% celkové doby běhu programu, pro malá data je to dokonce pouze 20%. Tedy i kdyby byla použita nějaká zázračná metoda, která by snížila dobu hledání tetrahedronů na nulu, celková doba výpočtu triangulace klesne pouze o 20 až 60 procent.

V důsledku toho je jasné, že použití DAGu (který umožňuje vyhledání tetrahedronu v logaritmickém čase v očekávaném případě) nebo jiné pomocné vyhledávací struktury bude pouze plýtváním pamětí, pokud nebude spojeno s optimalizací lokálních oprav.



Graf 8.4: Poměr času spotřebovaného na lokální opravy a vyhledání tetrahedronu při triangulaci proteinu. Pro povrchová data jsou výsledky obdobné.

8.2.3 Časová náročnost algoritmu odebírání bodů

V kapitole 3.4.2 byla stanovena asymptotická složitost odebrání bodu p z triangulace na $O(k \log k)$, kde k je počet tetrahedronů, které bod p obsahují. V nejhorším případě může být tento bod vrcholem všech tetrahedronů triangulace a jeho vyjmutí může být asymptoticky stejně náročné jako vytvoření celé triangulace znovu (ovšem bez bodu p). To je ovšem extrémní případ. Reálné časy mazání bodů v proteinech a umělých objemových datech jsou výrazně menší. Při testování byla vytvořena triangulace a následně z ní byly vymazány všechny body (kromě vrcholů pomocného tetrahedronu). Naměřené časy a průměrný počet tetrahedronů incidujících s odebíraným bodem jsou v následujících tabulkách 8.5 a 8.6.

Algoritmus vymazání vrcholu používá pro výpočet priority uší exaktní aritmetiku a jak je z tabulek 8.5 a 8.6 zřejmé, pro testovaná data trvá vymazání celé triangulace zhruba stejně dlouho jako její konstrukce při použití exaktní aritmetiky

# bodů	Doba konstrukce triangulace [s]		Doba mazání všech bodů triangulace [s]	Průměrná doba mazání jednoho vrcholu [ms]	Průměrný počet tetrahedronů obsahující odebíraný bod
	Klasická aritmetika	Exaktní aritmetika			
1250	0,43	1,53	1,59	1,28	25,40
2332	0,93	3,00	2,98	1,28	25,55
4710	1,95	6,47	6,64	1,41	26,60
7864	3,69	11,30	10,91	1,39	26,15
13623	7,12	20,67	20,49	1,50	27,07
65588	41,95	107,00	101,99	1,56	26,76

Tab. 8.5: Časy konstrukce a destrukce triangulace různých proteinů.

# bodů	Doba konstrukce triangulace [s]		Doba mazání všech bodů triangulace [s]	Průměrná doba mazání jednoho vrcholu [ms]	Průměrný počet tetrahedronů obsahující odebíraný bod
	Klasická aritmetika	Exaktní aritmetika			
1000	0,31	1,14	1,20	1,20	23,74
2000	0,75	2,50	2,62	1,31	24,65
4000	2,05	5,23	6,42	1,28	25,45
8000	4,14	10,94	11,09	1,39	25,75
16000	6,72	23,52	22,34	1,40	26,32

Tab. 8.6: Časy konstrukce a destrukce triangulace umělých objemových dat.

Doby uvedené v tabulkách 8.5 a 8.6 ovšem nezahrnují čas potřebný na nalezení tetrahedronu incidujícího s odebíraným vrcholem. Pokud by byl tento čas započten, vzrostou doby mazání v závislosti na počtu tetrahedronů triangulace. Nicméně pokud bude každý bod obsahovat index na libovolný tetrahedron, se kterým inciduje, je možné se tomuto vyhledávání zcela vyhnout.

8.3 Porovnání tunelů nalezených v *RT* a *DT*

Cílem této kapitoly je detailně porovnat jednotlivé typy tunelů. Především nás bude zajímat, zda jsou tunely v regulární triangulaci „lepší“ než tunely v triangulaci Delaunayově. Princip srovnání je jednoduchý – program vytvoří regulární a Delaunayovu triangulaci vybraného proteinu a pokusí se v těchto dvou triangulacích ke každému atomu nalézt optimistický a pesimistický tunel[33]. Na čtyřech takto získaných sadách tunelů program provede určitá měření a výsledky zpracuje jednoduchými statistickými metodami.

Při porovnávání pro nás bude hlavním kritériem poloměr tunelu v jeho nejužším bodě (ten budeme stručně nazývat **šířka tunelu**). Dále porovnáme jednotlivé typy tunelů podle počtu a velikosti chyb, kde chybou rozumíme průnik tunelu a nějakého atomu.

Pro srovnávání šířek tunelů definujme funkci $radius(A,i)$ kde A označuje typ tunelu (optimistický/pesimistický v Delaunayově/regulární triangulaci) a i je index atomu molekuly.

$$radius(A,i)=\begin{cases}\text{poloměr tunelu v jeho nejužším bodě,}\\ \text{pokud tunel typu } A \text{ k atomu } i \text{ existuje}\\ \\ 0;\ \text{pokud tunel neexistuje}\end{cases} \tag{8.1}$$

8.3.1 Průměrná šířka tunelů

Prvním sledovaným ukazatelem je průměrná šířka tunelů. Ta je počítána podle následujícího vzorce[34]:

$$\text{šířka tunelů typu } A = \frac{1}{\text{počet nalezených tunelů}} \sum_{i\in\left\{\substack{\text{atomy s tunelem}\\ \text{typu } A}\right\}} radius(A,i) \tag{8.2}$$

Výsledky pro jednotlivé typy tunelů v různých molekulách shrnuje tabulka 8.7.

[33] Výsledkem jsou tedy čtyři množiny tunelů: optimistické a pesimistické tunely v *DT* a optimistické a pesimistické tunely *RT*. Pochopitelně k některým atomům nevede žádný tunel, k jiným pouze jeden typ tunelu v jedné triangulaci (nejčastěji optimistický tunel v *RT*).

[34] Zdůrazněme, že “počet nalezených tunelů“ může být menší než počet atomů (ne ke každému atomu vede tunel). Jestliže k nějakému atomu tunel neexistuje, považujeme jeho šířku za nulovou.

	Průměrná šířka tunelů v nejužších bodech [v A]			
	Delaunayova triangulace		Regulární triangulace	
Soubor	Optim. tunely	Pesim. tunely	Optim. tunely	Pesim. tunely
1awj	1,257	1,072	1,398	1,270
1dbja	1,477	1,320	1,536	1,468
dhaa	1,038	0,776	1,100	0,986
2f61	1,771	1,619	1,829	1,761

Tab. 8.7: Průměrné šířky tunelů v nejužších bodech.

Tabulka 8.7 ukazuje, že v průměru jsou optimistické, resp. pesimistické tunely v *RT* širší než tunely stejného typu v *DT*. Dokonce se pesimistické tunely v *RT* blíží optimistickým tunelům v *DT*. Nicméně toto srovnání je značně hrubé, neporovnává odpovídající si tunely ani nezohledňuje počet existujících tunelů – pokud by v *DT* existoval tunel ke každému atomu, zatímco v *RT* by vedl tunel pouze k jednomu, těžko by bylo možné o *RT* prohlásit, že je pro analýzu tunelů vhodnější. Tyto otázky řeší následující metody srovnání.

8.3.2 Porovnání úspěšnosti

Zde se zaměříme na porovnání úspěšnosti hledání tunelů – nebude nás zajímat šířka tunelů ale pouze skutečnost, zda tunely jednotlivých typů k atomům vůbec existují.

	Úspěšnost hledání tunelů k atomům [v %]			
	Delaunayova triangulace		Regulární triangulace	
Soubor	Opt. tunely	Pes. tunely	Opt. tunely	Pes. tunely
1awj	92,1	81,9	99,7	99,7
1dbja	99,8	99,8	99,8	99,8
dhaa	43,8	38,5	50,0	50,0
2f61	50,0	50,0	50,0	50,0

Tab. 8.8: Úspěšnost hledání tunelů různých typů.

Výsledky se pro jednotlivé soubory značně liší – v souboru 1awj vedou tunely k více než 90% atomů, v souboru 1dbja pouze k 50%. Přesto jsou ve všech případech úspěšnější (nebo alespoň stejně úspěšné) tunely v regulární triangulaci. Za povšimnutí stojí fakt, že výsledky optimistických a pesimistických tunelů v *RT* jsou zcela totožné. To odpovídá tvrzení z kapitoly 6.2.3, že pokud v *RT* vede k nějakému atomu optimistický tunel, existuje k němu i tunel pesimistický. Z tabulky je rovněž patrné, že obdobné tvrzení pro tunely v *DT* neplatí.

8.3.3 Průměrný poměr a rozdíl šířek tunelů

V této metodě se zaměříme na porovnání dvou tunelů různého typu vedoucích ke stejnému atomu. Na rozdíl od předchozí metody zahrneme do porovnání pouze **úplné dvojice tunelů** (dva existující tunely různého typu ke stejnému atomu[35]). Zajímat nás bude poměr a rozdíl šířek tunelů a také procento případů, kdy je tunel v *RT* širší než odpovídající tunel v *DT*.

$$\text{poměr tunelů typů } A,B = \frac{1}{\text{počet existujících dvojic}} \sum_{i \in \left\{ \begin{array}{c}\text{atomy s tunely} \\ \text{typu } A \text{ i } B\end{array} \right\}} \frac{radius(A,i)}{radius(B,i)} \qquad (8.3)$$

$$\text{rozdíl tunelů typů } A,B = \frac{1}{\text{počet exist. dvojic}} \sum_{i \in \left\{ \begin{array}{c}\text{atomy s tunely} \\ \text{typu } A \text{ i } B\end{array} \right\}} \left(radius(A,i) - radius(B,i) \right) \qquad (8.4)$$

	Poměr tunelů *DT*/*RT* [v %]		Rozdíl tunelů *RT*–*DT* [v A]		Tunel v *RT* je širší než v *DT* v [%]	
Soubor	Opt. Tunely	Pes. tunely	Opt. tunely	Pes. Tunely	Opt. tunely	Pes. tunely
1awj	77,7	60,7	0,217	0,360	71,9	97,6
1dbja	94,9	85,9	0,067	0,156	74,4	93,6
dhaa	84,2	47,6	0,133	0,359	66,2	99,0
2f61	95,2	86,7	0,065	0,150	76,2	99,0

Tab. 8.9: Průměrné poměry a rozdíly optimistických resp. pesimistických tunelů v *DT* a *RT*.

Z tabulky 8.9 je zřejmé, že tunely v *RT* opět vycházejí širší než tunely v *DT*[36]. Například v proteinu dhaa jsou pesimistické tunely v *DT* průměrně o více než 50% užší než pesimistické tunely v *RT*, čemuž odpovídá průměrný rozdíl šířek tunelů 0,359A. Za povšimnutí snad stojí tři skutečnosti:

- Rozdíly mezi *RT* a *DT* jsou větší u pesimistických tunelů – speciální odvození pesimistických tunelů v *RT* se tedy vyplácí.
- Rozdíly mezi *RT* a *DT* jsou menší u molekul 1dbja a 2f61 (druhý a čtvrtý řádek tabulky). Tyto molekuly neobsahují atomy vodíku (což je nejmenší atom), takže rozsah velikostí atomů je menší a regulární triangulace se více podobá Delaunayově.
- Optimistický tunel v *RT* je širší než odpovídající optimistický tunel v *DT* zhruba v 75% případů, pesimistický tunel v *RT* je širší než pesimistický tunel v *DT* takřka vždy.

[35] Například pokud optimistický tunel k atomu číslo 5 v *RT* existuje a v *DT* ne, není tato dvojice úplná a nebude zahrnuta do porovnání optimistických tunelů v *RT* a *DT*.

[36] Průměrné rozdíly šířek tunelů neodpovídají průměrným šířkám popsaným v tabulce 8.8. To není chyba, pouze důsledek toho, že do srovnání v tabulce 8.9 jsou zahrnuty pouze úplné dvojice tunelů. V reálu tak z porovnání vypadnou především úzké tunely v *RT* (protože jejich protějšky v *DT* neexistují) a tím vzroste průměrná šířka tunelů v *RT*.

8.3.4 Porovnání širokých tunelů

Značná část nalezených tunelů je úzká, jejich poloměr se pohybuje v desetinách angstromu (pro srovnání – poloměr vodíku je 1,2A). Tyto tunely patrně nejsou při úpravách proteinů použitelné, a proto je budeme v následujícím testu ignorovat.

Jednotlivé typy tunelů opět porovnáme podle průměrného poměru a rozdílu šířek, ovšem do srovnání zahrneme pouze ty úplné dvojice tunelů, v nichž alespoň jeden tunel má poloměr větší než 1,85A (poloměr atomu síry).

	Tunely s poloměrem > 1,85A			
	Poměr tunelů *DT*/*RT* [v %]		Rozdíl tunelů *RT*– *DT* [v A]	
Soubor	Opt. Tunely	Pes. tunely	Opt. Tunely	Pes. tunely
1awj	0,966	0,904	0,101	0,267
1dbja	0,972	0,948	0,086	0,15
dhaa	0,96	0,903	0,125	0,283
2f61	0,973	0,952	0,081	0,138

Tab. 8.10: Porovnání tunelů s šířkou větší než 1,85A.

Z tabulky 8.10 je jasné, že poměry šířek tunelů se oproti předchozímu testu výrazně změnily – tunely v *RT* už jsou pouze o několik procent širší než tunely v *DT*, nezávisle na typu tunelu nebo proteinu. Nicméně rozdíl šířek tunelů se příliš nezměnil, tunely v *RT* zůstávají širší o jednu až tři desetiny angstromu. Obdobně dopadají i testy s jiným minimálním poloměrem – poměr šířek tunelů se s rostoucím minimálním poloměrem blíží k jedné, ale jejich rozdíl zůstává zhruba stejný. Jinými slovy – nezáleží, jaká je šířka tunelu v *DT*, odpovídající tunel v *RT* bude průměrně o několik desetin angstromu širší. Tomu odpovídá i „ruční“ porovnání šířek jednotlivých tunelů.

8.3.5 Chyby v tunelech

V posledním testu se zaměříme na chyby v tunelech. Protože pesimistické tunely v *DT* i *RT* jsou bezchybné, porovnáme pouze optimistické tunely.

Pro potřeby testování je každý nalezený tunel porovnán se všemi atomy proteinu. Zajímat nás bude procento tunelů, které obsahují chyby (první dva sloupce tab. 8.11), průměrný počet chyb v tunelu (třetí a čtvrtý sloupec) a průměrná velikost[37] maximální chyby v tunelech (poslední dva sloupce).

[37] Velikost chyby atomu *p* a koule tunelu *z* je dána takto: $error = r_p + r_z - |pz|$.

	Počet chybných tunelů [v %]		Počet chyb na tunel		Průměrná velikost maximální chyby	
Soubor	Delaunay	Regular	Delaunay	Regular	Delaunay	Regular
1awj	87,5	74,8	6,715	3,058	0,232	0,125
1dbja	25,1	14,6	0,597	0,332	0,133	0,071
Dhaa	92,1	85,4	21,095	8,133	0,284	0,145
2f61	27,8	24,0	0,746	0,681	0,088	0,074

Tab. 8.11: Chyby v optimistických tunelech.

První dva sloupce tabulky 8.11 ukazují, že procento chybných tunelů je velké a značně se liší molekulu od molekuly. Například v molekule dhaa protínají atomy zhruba 90% tunelů, zatímco v molekule 1dbja je to pouze 25% tunelů (resp. 15% v *RT*). Tomu odpovídá i průměrný počet chyb v tunelu, který se také pohybuje v širokém rozmezí. Průměrná velikost maximálních chyb v tunelech v *DT* a *RT* (poslední dva sloupce tabulky) je poměrně vysoká, blíží se rozdílu šířek optimistických a pesimistických tunelů. Nicméně chyby prakticky neovlivňují poloměr tunelů v jejich nejužším bodě – jestliže jsou při kontrole tunelu jeho chybné koule zmenšeny tak, aby do nich opravdu žádný atom nezasahoval (tzn. koule je zmenšena přesně o velikost chyby, ne o nějaký její odhad), poklesne průměrná šířka tunelů jen o setiny angstromu (viz tabulka 8.12).

	Průměrná šířka tunelů v *DT* [v A]		Průměrná šířka tunelů v *RT* [v A]	
Soubor	S chybami	Zmenšené a bez chyb	S chybami	Zmenšené a bez chyb
1awj	1,2574	1,2220	1,3976	1,3898
1dbja	1,4773	1,4760	1,5357	1,5351
dhaa	1,0377	0,9813	1,0998	1,0936
2f61	1,7711	1,7706	1,8295	1,8291

Tab. 8.12: Porovnání průměrných šířek optimistických tunelů před opravou (s chybami) a po opravě.

8.3.6 Tvar tunelů

Tunely v *RT* a *DT*, vedoucí do stejného místa molekuly, se liší nejen svým průměrem, ale občas také svou dráhou. Rozdíly v drahách jsou většinou malé, tunely často splývají, popřípadě se rozdělí a po chvíli znovu spojí. Jen výjimečně se dráhy tunelů zcela rozcházejí (ukázky tunelů viz obr. 8.1 až 8.4 a příloha A).

8.3.7 Shrnutí

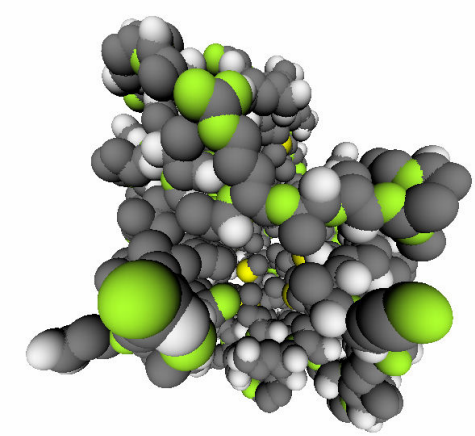
V uvedeném srovnání tunelů v regulární a Delaunayově triangulaci jasně vítězí regulární triangulace. Má větší úspěšnost při hledání tunelů (tzn. jsou v ní nalezeny tunely do míst, které se *DT* jeví jako nepřístupné) a tunely jsou průměrně o několik

desetin agstromu širší. Optimistické tunely v *RT* obsahují méně chyb a i průměrná velikost maximálních chyb je menší. Pesimistické tunely v *RT* na rozdíl od *DT* zachovávají existenci tunelu (pokud v *RT* do určitého místa vede optimistický tunel, pak existuje i odpovídající pesimistický tunel).

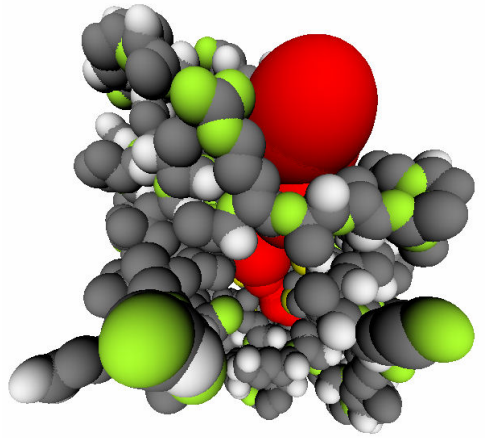
Nicméně použití *DT* pro hledání tunelů nelze zavrhnout. V průběhu minulého roku upravil tým doc. Ing. Jiřího Sochora svou metodu hledání tunelů v *DT*. Místo koulí opsaných tetrahedronům jsou při konstrukci tunelu použity koule tečné na atomy ve vrcholech tetrahedronů. Takto vypočtené koule jsou zkontrolovány vůči blízkým atomům a případně zmenšeny. Výsledné tunely se zdají být svou šířkou srovnatelné s tunely v regulární triangulaci. Bohužel pro podrobnější porovnání obou typů tunelů nebyla v době psaní této práce k dispozici potřebná data.

8.3.8 Ukázky tunelů

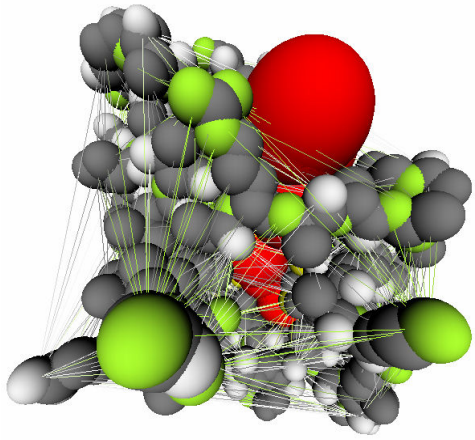
Obrázky 8.1 až 8.4 ukazují protein 1lin, jeho regulární triangulaci a v ní nalezený tunel.



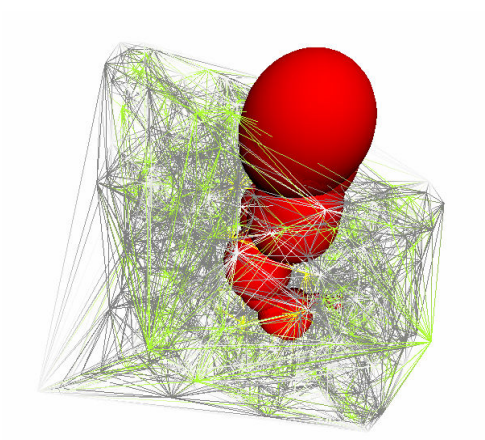
Obr. 8.1: Protein 1lin.



Obr. 8.2: Tunel (zobrazen červeně) v proteinu.



Obr. 8.3: Protein, jeho regulární triangulace a tunel (zobrazen červeně).



Obr. 8.4: Tunel (zobrazen červeně) v regulární triangulaci, atomy proteinu jsou skryté.

9. Závěr

Proteinové inženýrství se zabývá zkoumáním a modifikacemi struktury proteinů. K tomu využívá (mimo jiné) analýzu tunelů v proteinech. Existence tunelu vedoucího v proteinu k určité aminokyselině ovlivňuje jeho chemické vlastnosti, úpravou tunelu (rozšířením či naopak zablokováním) je možné tyto vlastnosti změnit. Výpočetní geometrie poskytuje různé nástroje pro nalezení tunelů v proteinech. Jedním z cílů této práce bylo navrhnout metodu, která by pro hledání tunelů použila regulární triangulaci, a porovnat dosažené výsledky se stávající technikou hledání tunelů v Delaunayově triangulaci. Tento cíl byl splněn, byly navrženy dva způsoby konstrukce tunelu v regulární triangulaci – optimistický, produkující širší tunely, které ovšem mohou obsahovat chyby, a pesimistický, který za cenu mírného zúžení vytváří bezchybné tunely. Podrobné porovnání těchto tunelů s tunely nalezenými obdobnou technikou v Delaunayově triangulaci ukázalo, že tunely v regulární triangulaci jsou průměrně o několik desetin angstromu širší (což odpovídá zhruba 10 – 40% jejich průměrné šířky) a obsahují méně chyb.

Proteiny mohou obsahovat až několik stovek tisíc atomů, proto je pro jejich triangulaci nutné použít metody, které umožní zpracovat i rozsáhlá data. Při malých změnách struktury proteinu (například při odstranění atomů obklopujících nejužší bod tunelu) je nevýhodné vytvářet znovu celou triangulaci upraveného proteinu, rychlejším řešením je pouze lokálně modifikovat stávající. Tyto požadavky byly zohledněny při návrhu programu, který umožňuje práci s velkými daty i přidávání a odebírání bodů v triangulaci, a splňuje tak druhý cíl této práce. Program při testech bez problémů trianguloval protein s miliónem bodů. Pro odebírání bodů byl navržen nový algoritmus, založený na Devillersově algoritmu odebírání bodů z Delaunayovy triangulace ve *2D*. Navržený algoritmus funguje dobře pro odebírání bodů z triangulací proteinů i dalších testovaných objemových dat, nicméně v některých případech selhává v triangulacích povrchových modelů (zřejmě vinou numerických nepřesností). S odstraněním tohoto problému se počítá v dalším vývoji aplikace.

Dalším možným směřováním aplikace je její plná dynamizace, tedy schopnost body nejen vkládat a odebírat, ale také měnit jejich polohu. To může být užitečné například při sledování tunelů v proteinu, jehož struktura se v čase mění (například při zahřívání). Program také bude upraven tak, aby plně zvládal i data obsahující redundantní body.

Vedlejším produktem této práce jsou dll knihovny pro konstrukci Delaunayovy a regulární triangulace, které mohou být použity v dalších projektech.

Přehled zkratek a značení

[A]		Jednotka angstrom, $1A = 1.0 \times 10^{-10}$ metru.
2D		Dvourozměrný prostor
3D		Trojrozměrný prostor
conv(*S*)		Konvexní obálka bodů z množiny *S*
d-simplex		*d*-simplex je konvexní útvar, který má *d*+1 vrcholů a nalézá se v dimenzi *d* nebo vyšší. 0-simplex je tedy bod, 1-simplex je přímka, 2-simplex trojúhelník atd.
DAG		Directed Acyclic Graph – Acyklický orientovaný graf
DT	*DT*(*S*)	Delaunayova triangulace (množiny *S*)
EDT	*EDT*(*S*)	Triangulace duální k Euklidovskému Voronoiovu diagramu množiny *S*
EVD	*EVD*(*S*)	Euklidovský Voronoiův diagram množiny *S*
p+		Bod v dimenzi d+1, který vznikne zobrazením d-dimenzionálního bodu *p* na paraboloid.
PD	*PD*(*S*)	Power diagram množiny *S*
RT	*RT*(*S*)	Regulární triangulace množiny *S*
S+		Množina bodů *p*+, které jsou zobrazením bodů $p \in S$.
VD	*VD*(*S*)	Voronoiův diagram množiny S
VDB	*VDB*(*S*)	Voronoiův diagram pro body množiny *S*
VR	*VR*(*p*)	Voronoiův region bodu *p*

Reference

[A91] F. Aurenhammer: Voronoi Diagrams – A Survey of a Fundamental Geometric Data Structure. ACM Computing Surveys 23, 345-405, 1991.

[B05] T. Beyer, G. Schaller, A. Deutsch, M. Meyer-Hermann: Parallel dynamic and kinetic regular triangulation in three dimensions. Computer Physics Communications 172, 86-108, 2005.

[B06] Petr Beneš: Voroného diagramy v molekulární chemii. Diplomová práce při Fakultě informatiky Masarykovy univerzity, 2006.

[B99] C. Branden, J. Tooze: Introduction to protein structure. Garland, 1999.

[D01] O. Devillers, S. Pion, M. Teillaud: Walking in a Triangulation. ACM Annual Symposium on Computational Geometry 17, 106-114, 2001.

[D99] O Devillers: On Deletion in Delaunay Triangulations. ACM Annual Symposium on Computational Geometry 15, 181-188, 1999.

[E92] H. Edelsbrunner, N. R. Shah: Incremental Topological Flipping Works for Regular Triangulations. ACM Annual Symposium on Computational Geometry 8, 43-52, 1992.

[F93] M. A. Facello: Implementation of a randomized algorithm for Delaunay and regular triangulations in three dimensions. Computer Aided Geometric Design 12, 349-370, 1993.

[J91] B. Joe: Construction of three-dimensional Delaunay triangulations using local transformations. Computer Aided Geometric Design 8, 123-142, 1991.

[K02] J. Kohout: Paralelní Delaunayova triangulace ve *2D* a *3D*. Diplomová práce při Fakultě aplikovaných věd Západočeské univerzity, 2002.

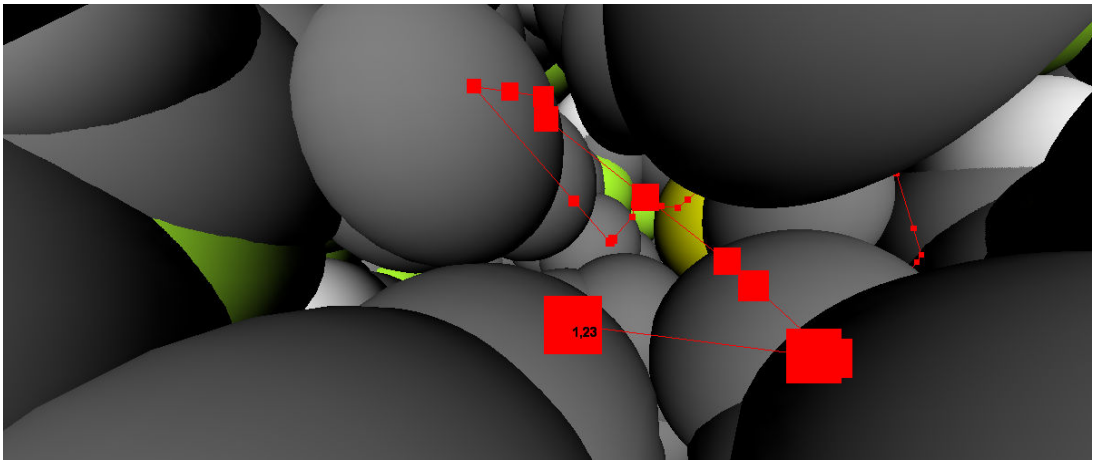
[M07] P. Medek, P. Beneš, J. Sochor: Computation of tunnels in protein molecules using Delaunay triangulation. Journal of WSCG, 2007.

[V01] M. Vigo, N.Pla, J. Cotrina: Regular Triangulations of Dynamics Sets of Points. Computer Aided Geometric Design 19, 127-149, 2002.

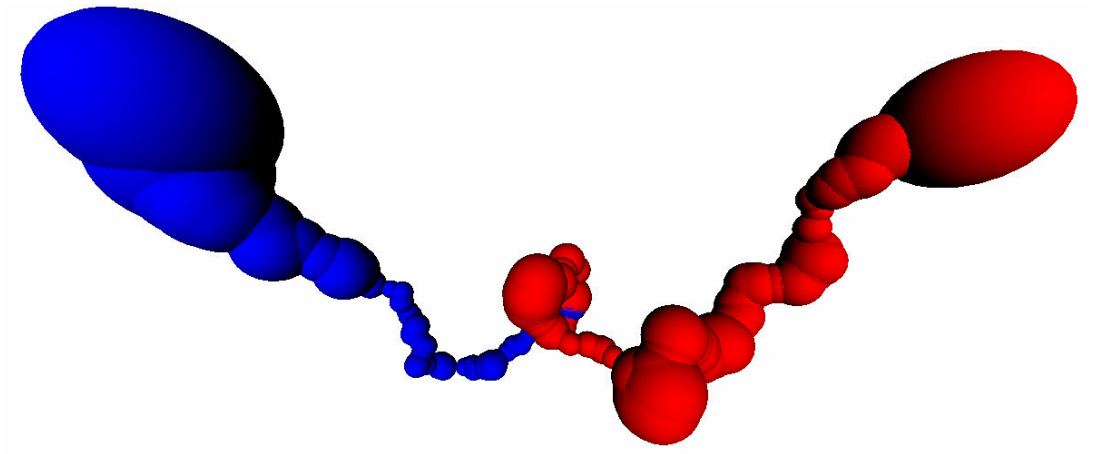
[Y04] Youngsong Cho, Deok-Soo Kim, Donguk Kim: Edge-tracing algorithm for Euclidean Voronoi diagram of *3D* spheres. 16th Canadian Conference on Computational Geometry, 2004.

Příloha A

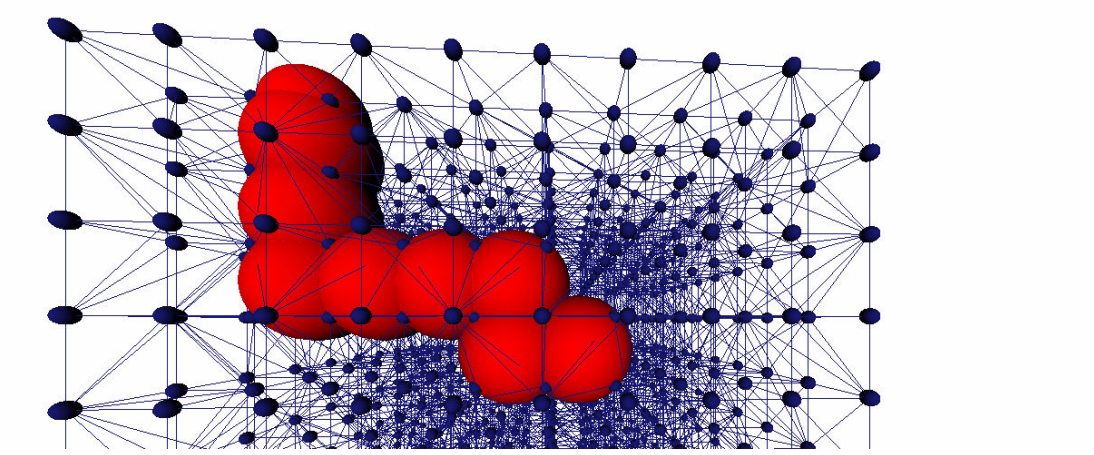
Ukázky tunelů



Obr. A.1: Průlet tunelem v proteinu. Červená lomená čára ukazuje osu tunelu, červené čtverce znázorňují konce jednotlivých hran a jejich nejužší místa. Číslo v prvním čtverci je poloměr tunelu v tomto bodě.



Obr. A.2: Porovnání tunelů nalezených v regulární (červený tunel) a Delaunayově (modrý tunel) triangulaci. Tunely vedoucí do stejného aktivního místa se mohou zcela lišit.



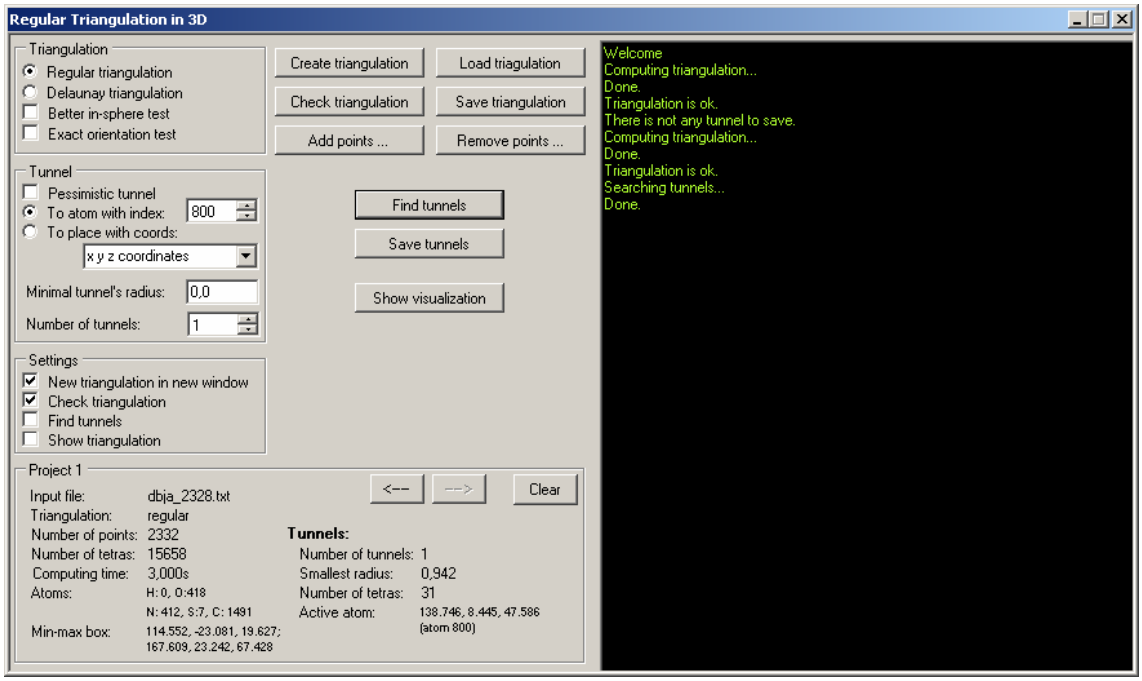
Obr. A.3: Tunel v regulární triangulaci bodů tvořících pravidelnou mřížku.

Příloha B

Uživatelský manuál

Hlavním účelem aplikace je vytvářet regulární a Delaunayovy triangulace množin bodů a hledat v nich tunely. Současně může být vytvořeno několik triangulací – každá je chápána jako samostatný projekt, mezi kterými se lze přepínat tlačítky v dolní části formuláře. Aplikace také umožňuje vizualizaci triangulace a tunelů v ní nalezených, možné je také uložení a načtení triangulace do/ze souboru, (triangulace je ukládána do textového souboru ve formátu popsaném na konci této přílohu). Ukládat lze i tunely.

Hlavní formulář aplikace (viz obr. B.1) obsahuje konzoli (do té jsou vypisovány informace o běhu programu a chybová hlášení) a řadu ovládacích prvků, rozdělených do čtyř panelů – panel Triangulation, Tunnel, Settings a Project.



Obr. B.1: Hlavní formulář aplikace.

Panel Triangulation

Tento panel obsahuje volby, které ovlivňují vytváření triangulace.

- Regular triangulation – vytvořená triangulace bude regulární.
- Delaunay triangulation – vytvořená triangulace bude Delaunayova.
- Better in-sphere test – výpočet in-sphere testu bude přesnější, ale také pomalejší.
- Exact orientation test – výpočet testu orientace bude přesnější, ale také pomalejší. Tuto volbu je vhodné zaškrtnout při zpracování povrchových modelů.

Panel Tunnel

V tomto panelu se nastavují předvolby, podle kterých budou hledány tunely.

- Pessimistic tunnel – budou hledány pesimistické tunely (ty neobsahují chyby, ale jsou užší než optimistické tunely).
- To atom with index – bude hledán tunel k atomu se zadaným indexem.
- To place with coords – bude hledán tunel k místu se zadanými souřadnicemi.
- Minimal tunnel radius – tunely, jejichž šířka je menší než zadané číslo, budou při hledání ignorovány.
- Number of tunnels – hledání skončí po nalezení zadaného počtu tunelů (pokud jich tolik existuje).

Panel Settings

Zde se nastavují akce, které mají být provedeny při vytvoření nové triangulace.

- New triangulation in new project – pro novou triangulaci je vytvořen nový projekt. Jestliže není tato volba vybrána, přepíše nová triangulace při svém vytvoření aktuální triangulaci.
- Check triangulation – po vytvoření triangulace je zkontrolována její topologie, geometrie a regularita.
- Find tunnels – po vytvoření triangulace jsou podle aktuálního nastavení vyhledány tunely.
- Show triangulation – triangulace bude zobrazena ihned po svém vytvoření.

Panel Project

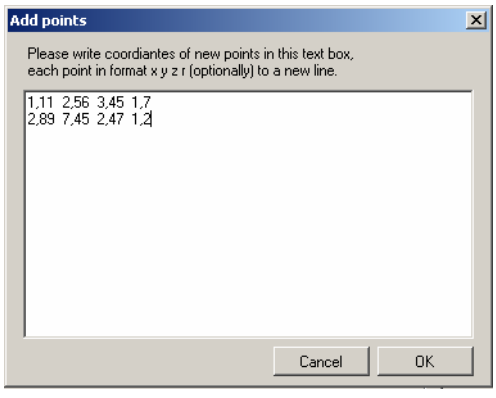
V tomto panelu jsou shrnuty různé informace o aktuálním projektu. V prvním sloupci je uvedeno jméno vstupního souboru, počet bodů a počet tetrahedronů triangulace, čistý čas výpočtu triangulace, počty atomů podle druhů a souřadnice min-max boxu vstupních bodů. V druhém sloupci jsou informace o nalezených tunelech – počet nalezených tunelů, poloměr nejširšího tunelu v jeho nejužším bodě, počet tetrahedronů, kterými nejširší tunel prochází, a souřadnice aktivního místa, do kterého tunel vede. Dále panel obsahuje tři tlačítka:

- ← – přejde na předchozí projekt.
- → – přejde na následující projekt.
- Clear – vymaže aktuální projekt.

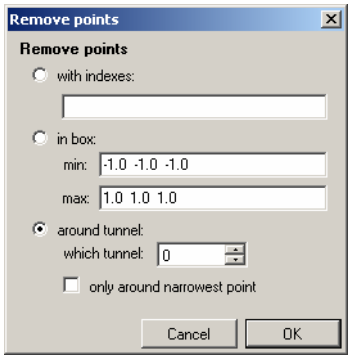
Hlavní panel

Přímo na hlavním panelu je umístěno devět tlačítek.

- Create triangulation – zeptá se na jméno vstupního souboru a poté vytvoří jeho triangulaci (*DT*/*RT* podle nastavení).
- Load triangulation – načte triangulaci ze souboru.
- Check triangulation – zkontroluje topologii, geometrii a regularitu aktuální triangulace.
- Save triangulation – uloží triangulaci do souboru.
- Add points… – otevře modální formulář (viz obr. B.2), do kterého může uživatel zadat souřadnice bodů (a volitelně jejich váhu), které chce přidat do triangulace. Po odklepnutí tlačítka OK jsou tyto body vloženy do triangulace.
- Remove points… – otevře modální formulář (viz obr. B.3), ve kterém může uživatel zadat body, které chce z triangulace odstranit. Tyto body mohou být zadány třemi způsoby: svým indexem, min-max boxem (odebrány budou všechny body ležící v zadaném min-max boxu) a tunelem (odebrány budou body obklopující vybraný tunel, popřípadě pouze body obklopující nejužší místo tunelu). Po odklepnutí tlačítka OK jsou body z triangulace odebrány.
- Find tunnels – jsou vyhledány tunely podle nastavení v panelu Tunnels.
- Save tunnels – nalezené tunely jsou uloženy do souboru.
- Show visualization – otevře se okno s vizualizací triangulace a tunelů v ní nalezených. Popis ovládání lze při otevřeném okně vizualizace získat stisknutím klávesy F1.



Obr. B.2: Formulář přidání bodů



Obr. B.3: Formulář odebrání bodů

Složitější akce

Zajímavé je vizuální porovnání tunelů v Delaunayově a regulární triangulaci. To je možné provést následujícím postupem:

1. Vytvořit *DT* proteinu a najít v ní tunel do určitého místa.
2. Vytvořit *RT* stejného proteinu a najít v ní tunel do stejného místa.
3. Zobrazit jeden z projektů. Ve vizualizaci je zobrazen kromě červeného tunelu v aktuální triangulaci také modrý tunel, který byl nalezen v druhé triangulaci.

Užitečné může být také porovnání tunelů před a po odstranění atomů obklopujících tunel (nebo jeho nejužší místo). To lze provést takto:

1. Vytvořit triangulaci proteinu a najít v ní tunel.
2. Kliknout na tlačítko „Remove points“ a ve formuláři zvolit nabídku „Remove points around tunnel“. Tím jsou z triangulace odebrány všechny vrcholy tetrahedronů, jimiž tunel procházel.
3. Najít znovu tunel do stejného místa a pozorovat změnu jeho šířky v nejužším bodě (v kombinaci s předchozím postupem je možné zobrazit oba tunely najednou).

Formáty uložení triangulací a tunelů

Formát uložení triangulace

Triangulace je ukládána do textového souboru (s příponou .tetra) v tomto formátu:

- První řádek obsahuje typ triangulace („[Delaunay triangulation]”, nebo „[Regular triangulation]”).
- Druhý řádek obsahuje počet vrcholů triangulace.
- Další řádky obsahují záznamy bodů, každý bod je uložen na jednom řádku jako čtveřice desetinných čísel oddělených tabulátorem. První tři hodnoty jsou souřadnice bodu, čtvrtá hodnota je jeho váha.
- Řádek, který následuje za seznamem bodů, obsahuje počet tetrahedronů.
- Další řádky obsahují záznamy tetrahedronů, každý tetrahedron je uložen na jednom řádku jako osm celočíselných indexů (oddělených tabulátorem), první čtveřice jsou indexy vrcholů tetrahedronu, druhá čtveřice jsou indexy sousedních tetrahedronů.

Formát uložení tunelu

Každý tunel je uložen do vlastního textového souboru (s příponou .path) ve formě lomené čáry. První řádek obsahuje počet vrcholů lomené čáry, každý vrchol je pak uložen na vlastním řádku jako čtveřice desetinných čísel (oddělených tabulátorem) – první tři jsou souřadnice vrcholu, čtvrtá hodnota je poloměr tunelu ve vrcholu.