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

# Bakalářská práce

# Konstrukce objemové sítě uvnitř domény reprezentované povrchovou sítí

Plzeň 2014 Tomáš Chmelík

# Prohlášení

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

V Plzni dne

Tomáš Chmelík

# Abstract

Cílem této práce je navrhnout metodu převedení povrchové trojúhelníkové sítě na objemovou čtyřstěnovou síť se stejným objemem. Tato metoda používá znalost kostry pro urychlení a zlepšení kvality sítě.

This bachelor's work is concentrated on converting surface mesh into volume mesh of equal volume. Skeleton of surface mesh is used to speeding up the process and improving the quality of output mesh.

# Obsah

# 1 Úvod

Úkolem této práce je vytvořit a implementovat algoritmus, který převede povrchovou trojúhelníkovou síť do objemové čtyřstěnové sítě. Jinými slovy se má z informace o povrchu vytvořit objekt, který má stejný objem. Zachování objemu je důležité pro případné použití tohoto algoritmu. Pro reprezentaci objektu je možno použít různé objemové objekty (čtyřstěny, šestistěny, osmistěny, atd.). Čtyřstěny budou nejlepší, protože je možné nimi vyplnit libovolný prostor bez sebe protínání a přebytků.

Tato práce je zamýšlená na používání v medicínské oblasti. Různé povrchové reprezentace z počítačové tomografie a jiných vyšetřovacích metod můžou být převedeny do objemových reprezentací. Medicínská oblast není jediná oblast, kde se podobný výstup dá použít. Různé vymodelované mechanické součástky takto můžou být převedeny a následně zátěžově testovány.

Úkol převedení povrchové sítě na objemovou není příliš prozkoumán a jeho využitelnost stoupá. Výstup takového algoritmu se dá použít při deformování a animování objektů. Síla aplikovaná na povrch takového objektu, skrze vnitřní strukturu, způsobí deformaci na druhé straně tělesa. Pro účely animování jde použít stejný argument. Při různém ohýbání a otáčení si objekt zachová svůj objem a bude vypadat více přirozeně, než kdyby se deformovala pouze povrchová síť.

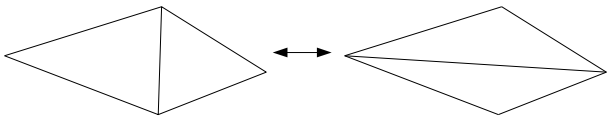
# 2 Stávající metody řešení

## 2.1 Delaunayova tetrahedronizace

K vysvětlení Delaunayovy tetrahedronizace je nutné nejdříve vysvětlit Delaunayovu triangulaci. A pro vysvětlení Delaunayovy triangulace je nejdříve nutné vysvětlit pojem triangulace.

Triangulace je proces vytvoření trojúhelníků z množiny bodů. Výstupem takového procesu je konvexní množina trojúhelníků, do které nejde přidat další hrana, která by neprotínala už existující hranu. Pro jednu skupiny vrcholů existuje mnoho různých triangulací.

Delaunayova triangulace je speciálním typem triangulace, kde každý trojúhelník splňuje Delaunayovo kritérium. Jako Delaunayovo kritérium se většinou používá podmínka prázdné opsané kružnice, ale dají se použít jiné podmínky, pokud to je žádané. Podmínka prázdné opsané kružnice říká, že v opsané kružnici daného trojúhelníku neleží žádný jiný bod vstupní sítě. Pokud tato podmínka není splněná, musí se upravit trojúhelníková síť jedním nebo více přehozením (viz obrázek 2.1).

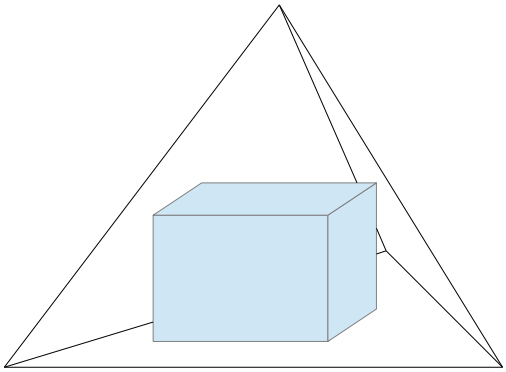


Obrázek 2.1: Přepojení dvou trojúhelníků ve 2D.

Delaunayova tetrahedronizace (dále jen DT) je varianta Delaunayovy triangulace v $E^3$. Namísto trojúhelníků se vytvářejí čtyřstěny. Delaunayo kritérium je v tomto případě obvykle podmínka prázdné opsané koule. Ta říká, že v opsané kouli každého čtyřstěnu nesmí ležet jiný bod vstupní množiny bodů. Nesplněním této podmínky se znovu musí upravovat čtyřstěnová síť, dokud není podmínka splněna (podrobněji popsáno níže).

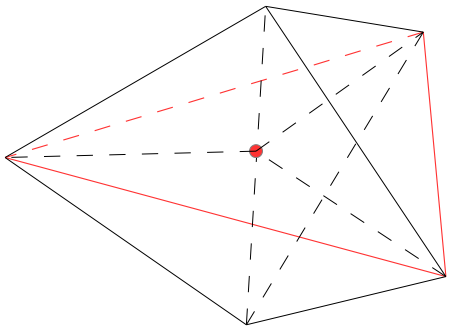
Jedním ze způsobů vytváření DT je metoda inkrementálního vkládání. Delaunayovo kritérium je pro tuto tetrahedronizaci podmínka prázdné opsané koule. Před celým procesem se vytvoří dostatečně veliký čtyřstěn, který zaručeně obsahuje všechny body vstupní množiny (viz obrázek 2.2). Poté se postupně přidávají body. Každý přidaný bod může ležet buď uvnitř už exis-

tujícího čtyřstěnu, na společné stěně dvou čtyřstěnů nebo na společné hraně několika čtyřstěnů. Podle toho se provede jedna ze 3 akcí.

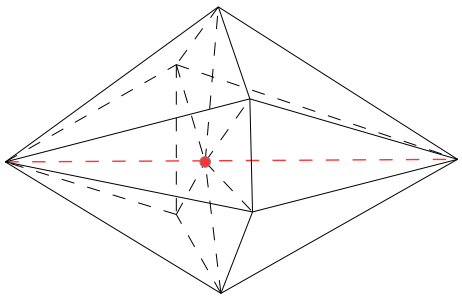


Obrázek 2.2: Čtyřstěn obsahující všechny body vstupní sítě (modrý kvádr je bounding box vstupní sítě).

Pokud je přidaný vrchol uvnitř už existujícího čtyřstěnu, daný čtyřstěn se rozdělí na 4 nové. Okolní čtyřstěny nebudou ovlivněny tímto rozdělením. Když přidávaný bod leží na stěně dvou čtyřstěnů, rozdělí se tyto čtyřstěny na 6 nových čtyřstěnů (viz obrázek 2.3). A pokud daný bod leží na hraně čtyřstěnu, je nutné rozdělit všechny čtyřstěny, které obsahují tuto hranu. Každý jednotlivý čtyřstěn se rozdělí na 2 (viz obrázek 2.4).

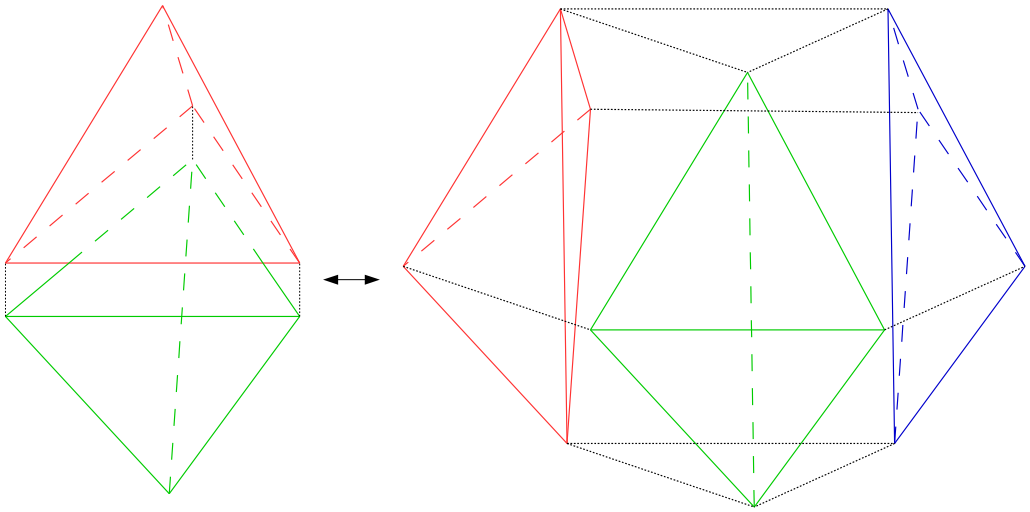


Obrázek 2.3: Rozdělení čtyřstěnů přidáním bodu na stěnu. Bod leží na červeném trojúhelníku.



Obrázek 2.4: Rozdělení čtyřstěnů přidáním bodu na hranu. Dělí se červená hrana.

Po vložení bodu a rozdělení stávajících čtyřstěnů se musí otestovat Delaunayovo kritérium. Pokud někde není splněno, je potřeba upravit čtyřstěnovou síť podle umístění okolních bodů. Provede se buď přepojení 2-3 (případně opačné 3-2), 2-2 nebo 4-4. Přepojení 2-2 a 4-4 vyžadují, aby dvě (případně

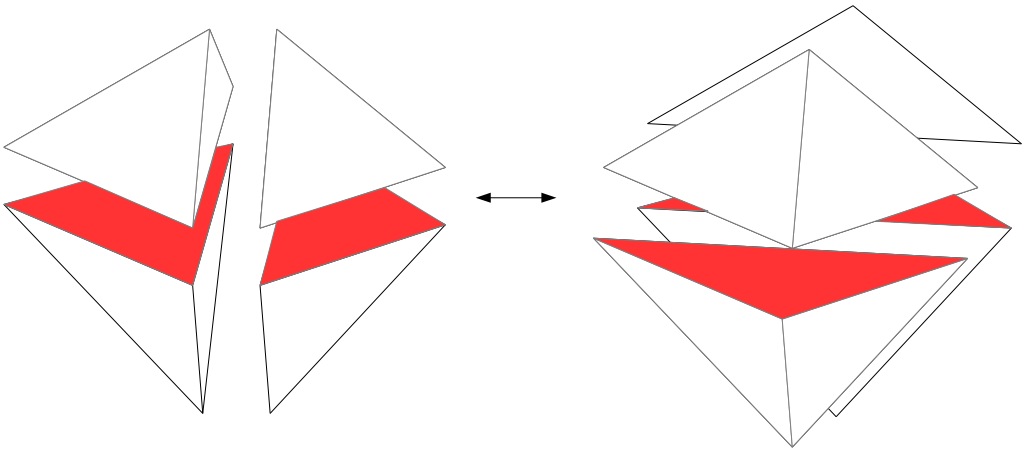


Obrázek 2.5: Přepojení 2-3 a 3-2. Jemně tečkovaná čára ukazuje jak jsou čtyřstěny rozděleny.

čtyři) stěny čtyřstěnů ležely na rovině. Někdy taková přepojení nepůjde provést. Tyto čtyřstěny se dají do fronty čtyřstěnů, které potřebují přepojit. Tato fronta se projde po přidání každého dalšího vrcholu, jestli už není možné přepojit. Ukázka přepojení 2-3 na obrázku 2.5 a 4-4 na obrázku 2.6.

Po přidání všech vrcholů a splnění Delaunayova kritéria je potřeba odstranit čtyřstěn, který byl na začátku vytvořen, aby obsahoval všechny vstupní body. Všechny čtyřstěny, které obsahují alespoň jeden ze 4 mazaných vrcholů prvního čtyřstěnu, musí být také odstraněny. Výstupem tohoto algoritmu je konvexní obálka vstupní množiny bodů. Konvexní obálka ale není žádaný výstup, takže se bude muset síť po DT upravit a všechny čtyřstěny, které jsou vně vstupní trojúhelníkové sítě, také odstranit. To už ale není spojeno s Delaunayovou tetrahedronizací.

Pro podrobnosti a další možnosti konstrukce Delaunayovy triangulace a tetrahedronzace odkazuji na články [Fuksa(2006)] a [Šmolík(2013)].



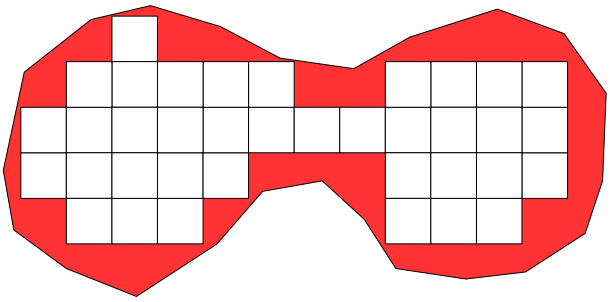
Obrázek 2.6: Přepojení 4-4. Červené plochy leží na jedné ploše.

## 2.2 Voxelizace

Další možností vytvoření objemové reprezentace povrchové sítě je voxelizace. Pojem voxel (volume pixel) je chápán jako malý kvádr, který obsahuje nějakou informaci. Voxelizace může být proces rovnoměrného rozdělování vstupní množiny do voxelů, kde tyto voxely obsahují nějakou informaci o místě, kde se nacházejí. Příklad může být voxelizace objektu zadaného implicitní funkcí (například koule). V tomto případě jsou voxely rovnoměrné rozloženy a můžou obsahovat informaci, jestli je daný voxel uvnitř nebo vně objektu. Když je voxelizace dostatečně jemná, zobrazením voxelů, které jsou uvnitř, se ukáže reprezentace vstupního objektu.

Další možnost je provádět voxelizaci adaptivně. Při takovém procesu se voxely nerozmist'ují rovnoměrně a nejsou stejně veliké. Podle rychlosti změny sledované veličiny (například znovu zda je voxel uvnitř nebo vně) se voxely dělí na menší. Tato místa s velikou změnou obsahují více malých voxelů a naopak místa s malou změnou obsahují málo, velikých voxelů. Toto dělení se provádí dokud není voxel dostatečně malý.

Pro potřeby mého problému stačí rovnoměrné rozdělení voxelů. Zjistí se, které voxely jsou uvnitř povrchu a ostatní se odstraní. Zbylé voxely se převedou na vrcholy, které se mezi sebou napojí do kvádrů, nebo rovnou na čtyřstěny. Problém nastává při napojení těchto bodů na samotnou trojúhelníkovou síť. Vrcholy vytvořené z voxelů představují přibližný tvar objektu, které se zpracovává. Tento vnitřní objekt má svojí povrchovou síť. Vytvoření



Obrázek 2.7: Síť po voxelizaci. Červená plocha bude vyplňena tetrahedronizací.

čtyřstěnů mezi vnitřním objektem a vnější trojúhelníkovou sítí je v podstatě vyplnění prostoru mezi dvěma neprotínající se povrchovými sítěmi. To jde vyřešit upravenou Delunayovou tetrahedronizací, která bude mít za úkol vytvořit čtyřstěny pouze v prostoru mezi sítěmi. Příklad takové sítě na obrázku 2.7. Toto řešení je inspirováno článkem [Kun Zhou(2005)].

## 2.3 Znalost kostry

Třetí možností vyřešení zadaného problému je použít kostru vstupní trojúhelníkové sítě. Vrcholy kostry budou zaručeně uvnitř povrchu. Vytvoření čtyřstěnů pak spočívá ve vybrání nejlepšího vrcholu kostry, se kterým by se spojil určitý trojúhelník sítě. Získání kostry trojúhelníkové sítě není triviální. Problematikou vytvoření kostry z 3D objektu se zabývá článek [Au Oscar(2008)]. Tento algoritmus už je implementován autory článku. Proces vytváření kostry se skládá ze 3 částí.

- Kontrakce sítě
- Odstranění povrchů sítě
- Posunutí kostry

### 2.3.1 Kontrakce sítě

V prvním kroku vytváření kostry se iteračně zmenšuje objem tělesa, dokud není skoro nulový. To se dělá Laplacovým vyhlazováním. Laplacovo vyhlazo-

vání přesouvá vrcholy určené v Laplacových souřadnicích směrem k počátku souřadnic. Laplacův souřadný systém je systém, kde pro každý bod systému je počátek souřadného systému v místě určeným okolními body. Body jsou posouvány, po přibližné normále, dovnitř objektu. Při tomto procesu se posouvají vrcholy do pozic, kde je objekt vyhlazenější. Kdyby se takový proces nijak neomezil, časem by se celý objekt převedl do jediného bodu. Proto se podle tloušťky (objemu) tělesa v okolí každého bodu definují síly, které nedovolí vrcholu příliš se posunout z místa, kde objekt má malý objem. Objekt se tedy nepřevede do bodu, ale do tenké reprezentace originálního objektu.

Každou iteraci se počítá rovnice (2.1)

$$\begin{bmatrix} W_L L \\ W_H \end{bmatrix} V' = \begin{bmatrix} 0 \\ W_H V \end{bmatrix} \tag{2.1}$$

kde $L$ je Laplaceův operátor, $W_L$ a $W_H$ jsou matice vah, matice $V$ obsahuje pozice vrcholů před iterací a matice $V'$ obsahuje pozice vrcholů po iteraci. Prvky matice L se vypočítají podle rovnice (2.2).

$$L_{i,j} = \begin{cases} \omega_{i,j} = cotg(\alpha_{i,j}) + cotg(\beta_{i,j}) & pokud\ (i,j)\ je\ hranou \\ \sum_{(i,k)\in E} -\omega_{i,k} & pokud\ i = j \\ 0 & jinak \end{cases} \tag{2.2}$$

Váhové matice $W_L$ a $W_H$ určují rychlost konvergence a stabilitu výpočtu. Na počátku procesu je $W_H$ jednotková matice a $W_L$ je diagonální matice obsahující přibližný průměr obsahů trojúhelníků sítě.

Po každé iteraci se vypočítá nová matice $L$ a váhové matice se vynásobí příslušnými koeficienty. Zastavovací podmínka je buď provedení maximálního počtu iterací (zadaného uživatelem), nebo dosažení objemu zadaného násobkem původního objemu. Během procesu se body pouze přesouvají a nijak není upravováno propojení bodů, takže se zachovává konektivita sítě.

### 2.3.2 Odstranění povrchů sítě

Po kontrakci objektu je potřeba vytvořit samotnou kostru. Odstraňují se hrany, dokud existuje nějaký trojúhelník. Pokud mezi 2 vrcholy $i$ a $j$ existuje vrchol $k$, který je sousedem $i$ a $j$, a zároveň tyto tři vrcholy netvoří trojúhelník, hranu není možno odstranit.

Hrany se mažou pomocí half-edge collapse. Oba vrcholy se přesunou do středu hrany ($i$, $j$) a všechny trojúhelníky, které jsou neplatné (obsahující vrcholy $i$ a $j$), se také odstraní. Tento proces nikdy nerozpojí objekt a při mazání hran si jde pamatovat, které vrcholy vstupního objektu patří jakému vrcholu kostry. Tato informace se poté bude používat na vytvoření čtyřstěnů.

### 2.3.3 Posunutí kostry

Po odstranění všech trojúhelníků sítě je už kostra vytvořená, ale neleží ve správné pozici a může dokonce ležet mimo vstupní povrchovou síť. Během konstrukce objemu trvá silné části tělesa zkolabovat déle než tenké části. Kontrakce se ale používá vždy na všechny vrcholy, takže silnější místa k sobě přitahují tenčí místa. Proto se všechny vrcholy kostry na konci celého procesu posouvají do průměru pozic vrcholů jeho domény.

Takto vytvořenou kostru bych poté napojoval na trojúhelníky sítě a vytvářel tak čtyřstěny, které zaplní prostor v objektu.

# 3 Návrh řešení

Budu implementovat dvě metody řešení. Jako referenční metodu použiji DT vyvíjenou na KIV, kterou rozšířím o odstranění přebytečných čtyřstěnů. Druhá metoda bude používat znalost kostry, ze článku popsaného výše, k urychlení procesu vytváření čtyřstěnů uvnitř trojúhelníkové sítě.

## 3.1 Delaunayova tetrahedronizace

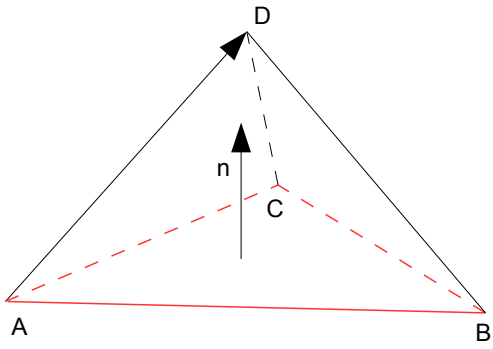
Pro vytvoření DT používám nástroj vyvíjený na KIV, který implementuje popsanou metodu inkrementálního vkládání (detaily v článku [Kohout(2002)]). Čtyřstěnová síť, kterou dostanu, bude konvexní, ale množina bodů, která se tetrahedronizovala nemusí být konvexní. To znamená, že ve čtyřstěnové síti můžou existovat čtyřstěny, které jsou vně původní povrchové sítě. Zároveň čtyřstěnová sít nemusí obsahovat všechny trojúhelníky vstupní trojúhelníkové sítě, protože by tak nesplňovala Delaunayovo kritérium. Přehození 2-3, 2-2 a 4-4, která změnily čtyřstěnovou síť tak, že poté neobsahuje hledané trojúhelníky, musí být provedeny znovu, na opačnou stranu. Poté, co se převede čtyřstěnová síť do stavu, kde její čtyřstěny obsahují všechny trojúhelníky ze vstupní trojúhelníkové sítě, se odstraní čtyřstěny vně trojúhelníkové sítě.

Všechny čtyřstěny se projdou a pokud obsahují trojúhelník sítě, vypočítá se normála tohoto trojúhelníku (podle trojúhelníku trojúhelníkové sítě). Otestuje se nerovnice 3.1 (obrázek 3.1 vysvětluje nerovnici) a odstraní se, pokud je čtyřstěn vně.

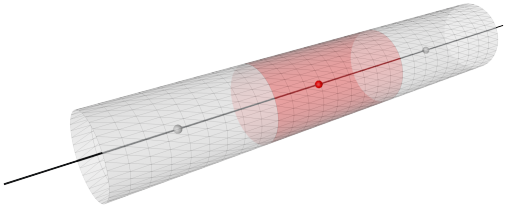
$$\vec{n} \cdot \overrightarrow{AD} \begin{cases} > 0 & \text{čtyřstěn vně} \\ \leq 0 & \text{čtyřstěn uvnitř} \end{cases} \tag{3.1}$$

## 3.2 Vyvíjený algoritmus

Druhou metodu, kterou implementuji pojmenovávám TC pro další použití v textu (zkratka z Tetrahedron Creation). Vstup mého algoritmu bude vyžadovat, aby každý vrchol kostry si s sebou nesl množinu vrcholů původní sítě,




Obrázek 3.1: Čtyřstěn vně trojúhelníkové sítě. Červeně označen trojúhelník plochy.

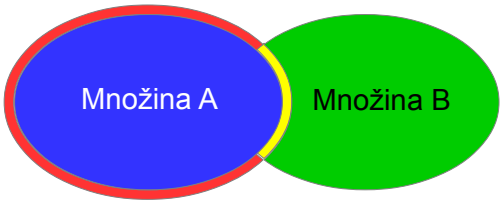


Obrázek 3.2: Červená oblast je doména červeného vrcholu kostry.

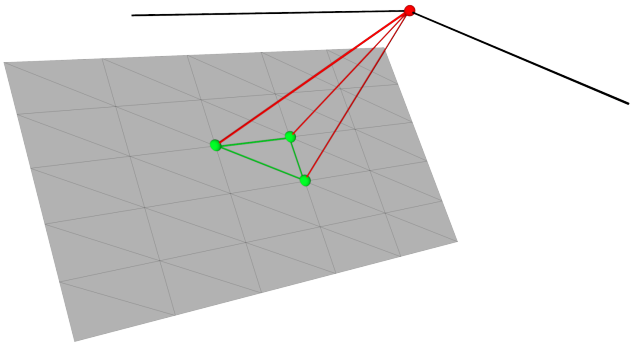
ze kterých byl daný bod kostry vytvořen (dle sekce Odstranění povrchů sítě). Tyto množiny bodů jsem nazval domény (obrázek domény 3.2). Z pohledu kostry je doména množina bodů povrchové sítě. Z pohledu bodu na povrchu sítě je doména seznam vrcholů kostry, které mají daný vrchol ve své doméně.

### 3.2.1 Rozšiřování domén

Před samotným vytvářením čtyřstěnů se nejdříve projdou všechny hrany kostry. Pro každou hranu se rozšíří jedna ze dvou domén tak, že budou mít společné vrcholy. Množina $A'$ je množina vrcholů povrchové sítě, které sousedí alespoň s jedním vrcholem množiny $A$ (doména prvního vrcholu hrany) a zároveň neleží v množině $A$. Do množiny $A$ budou přidány vrcholy z průniku množiny $A'$ a množiny $B$ (doména druhého vrcholu hrany), dle rovnice 3.2 a obrázku 3.3. Takto se budou dotýkat domény všech hran kostry.




Obrázek 3.3: Ukázka rozšiřování domén.
A' = červená oblast ∪ žlutá oblast
Žlutá oblast = A' ∩ B



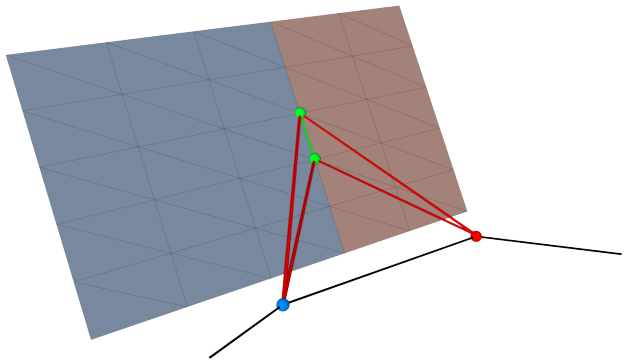
Obrázek 3.4: Doménový čtyřstěn.

$$A = A \cup (A' \cap B) \tag{3.2}$$

Takto pro každou hranu získám množinu společných vrcholů, které budou použity pro vytvoření hraničních čtyřstěnů a zacelení objemu mezi jednotlivými vrcholy kostry. Nejdříve se ale vytvoří doménové čtyřstěny.

### 3.2.2 Doménové čtyřstěny

Pro každý bod kostry se projdou vrcholy jeho domény a identifikují se všechny trojúhelníky z původní sítě, které mají všechny tři vrcholy v dané doméně. Z těchto trojúhelníků se vytvoří doménové čtyřstěny přidáním dalšího bodu: vrcholu kostry, kterému doména patří. Viz obrázek 3.4.



Obrázek 3.5: Hraniční čtyřstěn.

Toto se může zdát dostačující, protože výstup takového algoritmu poskytuje navenek správné řešení, ale uvnitř takového objektu jsou díry. Tyto díry existují pouze na rozmezí dvou domén a vyplní se hraničními čtyřstěny.

### 3.2.3 Hraniční čtyřstěny

Hraniční čtyřstěny se vytváří pro každou hranu kostry. Každá hrana kostry by měla mít množinu společných vrcholů. Tato množina společných vrcholů se projde a pro každou dvojici vrcholů, které tvoří hranu v původní síti, se vytvoří hraniční čtyřstěn. Hraniční čtyřstěn je vytvořen ze dvou bodů povrchové sítě a dvou vrcholů kostry (vrcholy hrany kostry, která se právě zpracovává). Viz obrázek 3.5.

## 3.3 Subdivision

Vytvořené čtyřstěny můžou mít nepříznivý tvar. Pokud je čtyřstěn moc vysoký a úzký, nebo nízký a široký, může nastat problém s jeho používáním. Pokud má čtyřstěn špatnou kvalitu zvyšuje se šance numerických chyb při počítání se čtyřstěnem. Problém nastává i když se takový čtyřstěn texturuje.

Špatné čtyřstěny se musí rozdělit a zmenšit, aby se odstranily tyto problémy. Rozdělovat se bude dokud se budou čtyřstěny zlepšovat (poměr ob-

jemu čtyřstěnu ku objemu koule opsané), nebo dokud čtyřstěny budou dostatečně malé, aby relativní chyba bude dostatečně malá.

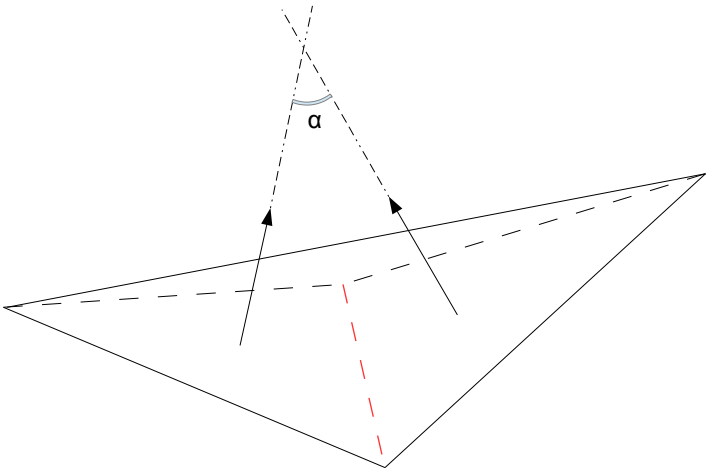
První intuitivní řešení je přidat do špatného čtyřstěnu další bod a vytvořit tak ze čtyřstěnu 4 čtyřstěny. Takto přidaný bod nezpůsobí další dělení v okolních čtyřstěnech. Toto řešení situaci jen zhoršuje, čtyřstěny se zplošťují.

Bude nutné přidávat vrcholy na kraje čtyřstěnu, tedy na hrany nebo stěny. Stěny se zdají být dobrým řešením, ale znovu nemusí vést k lepšímu výsledku. Kdyby se přidával bod na stěnu, vytvářejí se znovu více úzké čtyřstěny.

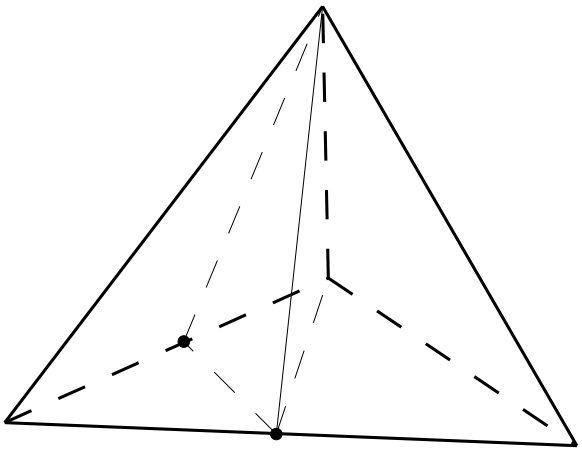
Poslední varianta je přidávat body na hrany čtyřstěnu. To vyvolá největší počet dělení sousedních čtyřstěnů a zároveň to dodá největší volnost jak nejlépe rozdělit daný čtyřstěn. Pro každý čtyřstěn platí, že se může rozdělit každá jeho hrana, takže je možné přidat až 6 vrcholů na hrany. S použití symetrií jde čtyřstěn rozdělit 11 způsoby [L. Rodriguez(2006)]. Pro každý způsob se připraví jak se budou vytvářet nové čtyřstěny. Podle kritéria dihedralového úhlu (viz obrázek 3.6) se vyberou hrany, které by se měly rozdělit. To bude odpovídat jednomu způsobu rozdělení a prostě se aplikuje šablona rozdělení a čtyřstěn bude rozdělen.

Přidáním bodů na hrany se zruší integrita sítě, to se musí opravit tak, že se rekurzivně budou rozdělovat sousední čtyřstěny. Každý sousední čtyřstěn dostane informace, kde si musí přidat na hranu vrchol (protože se sousední čtyřstěn tak už rozdělil), pak se vypočítá, kde by se tento sousední čtyřstěn měl rozdělit. Tyto dvě množiny vybraných hran se sjednotí, vybere se způsob rozdělení a aplikuje se šablona. Takto bude proces pokračovat rekurzivně. Pokud rozdělení jednoho čtyřstěnu vyvolá dělení v celé síti (všechny čtyřstěny se musí rozdělit), neznamená to, že se rozdělený čtyřstěn může znovu dělit. Během jedné iterace se každý čtyřstěn může rozdělit jen jednou (vybrání jednoho čtyřstěnu a následné rekurzivní dělení považují za jednu iteraci). Rekurze je zakončená pokud už neexistuje čtyřstěn, který nebyl rozdělen, nebo pokud čtyřstěn nevyžaduje žádné další rozdělení. Čtyřstěn, který nevyžaduje žádné vlastní rozdělování se rozdělí tak, aby uspokojil nároky svých sousedů, sám nevyvolá další rekurzi, protože nepřidal nové body na hrany (příklad zakončení ve 3D na obrázku 3.7).

Z těchto přístupů jsem si vybral poslední, čili přidávání vrcholů na hrany. Ostatní přístupy vytvářejí horší čtyřstěny než byly vstupní. Podrobnější vysvětlení, obrázky možných případů rozdělení a algoritmů ve článcích [L. Rodriguez(2006)] a [D. Ruprecht(1994)].




Obrázek 3.6: Dihedralový úhel červené hrany.
Úhel mezi dvěma stěnami čtyřstěnu, které sousedí přes vybranou hranu.
Pokud $\alpha \in (45°, 135°)$ hrana není potřeba rozdělit.



Obrázek 3.7: Zakončení rekurze rozdělování čtyřstěnů.
Tučně původní čtyřstěn, tence nově vytvořené.
Tečky jsou body vytvořené kvůli sousedním čtyřstěnům.

# 4 Implementační detaily

Práci jsem naprogramoval jako filtry (viz níže) pro VTK (také viz níže). Rozhodl jsem se použít VTK kvůli prostředkům, které nabízí pro načítání, zpracování a zobrazování grafických dat. Dále jsem se rozhodl pro jazyk C++, kvůli možnosti používat VTK bez jazykových obalů.

## 4.1 VTK

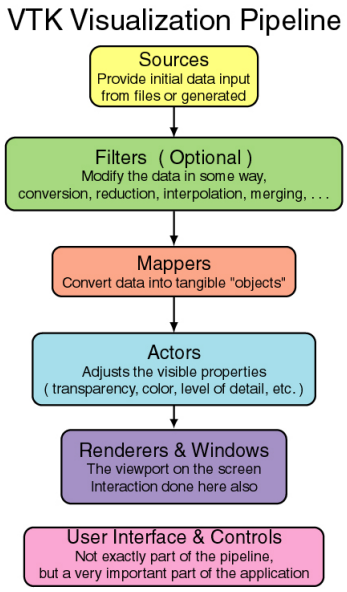
VTK neboli Visualization Toolkit je volně dostupný software pro 3D počítačovou grafiku, zpracování obrazu a vizualizaci dat. V základu obsahuje mnoho způsobů načtení vstupu, zpracování dat (vyhlazování, ořezávaní, prohledávání grafů a další) a zobrazení dat. Jádro programu je psáno v jazyce C++, ale obsahuje jazykové obaly, které umožňují programování v jiných programovacích jazyků (podporuje Python, Java, TLC, C# a C).

VTK pracuje v tzv. *pipeline* režimu zpracování dat. Tedy data z jednoho modulu (části programu) se posílají do dalšího modulu. Takto za sebou může být poskládáno mnoho modulů a program jde napsat jednoduše správným pospojováním předem připravených modulů. Směr proudění dat je obecně od zdrojů dat, přes filtry, které data upravují a zpracovávají, až k rendererům, které zpracovaná data zobrazí (viz obrázek 4.1).

Objekty ve VTK pipelině jde rozdělit do 5 kategorií:

- Source - moduly pro načtení dat ze souboru, jiné pipeline, nebo pro vygenerovaní nových dat.
- Filter - moduly pro upravování dat (transformace, ořezávání, ...).
- Mapper - moduly pro převedení dat z formy, které snadno zpracují *Filtery* na data, které snadno zpracují *Actori*.
- Actor - moduly pro upravení vizuálních vlastností (průhlednost, ...).
- Renderer - moduly pro vytvoření okna a samotné zobrazení dat.

VTK obsahuje ještě jeden typ modulů, které nejsou v samotné pipelině, a to ovládací prvky. Jsou to hlavně moduly pro interakci s uživatelem a




Obrázek 4.1: VTK pipelina. Převzato z [Bell(2004)]

upravování parametrů modulů v pipelině. S těmito moduly souvisí i způsob, jak se vyžádá přepočítání pipeliny.

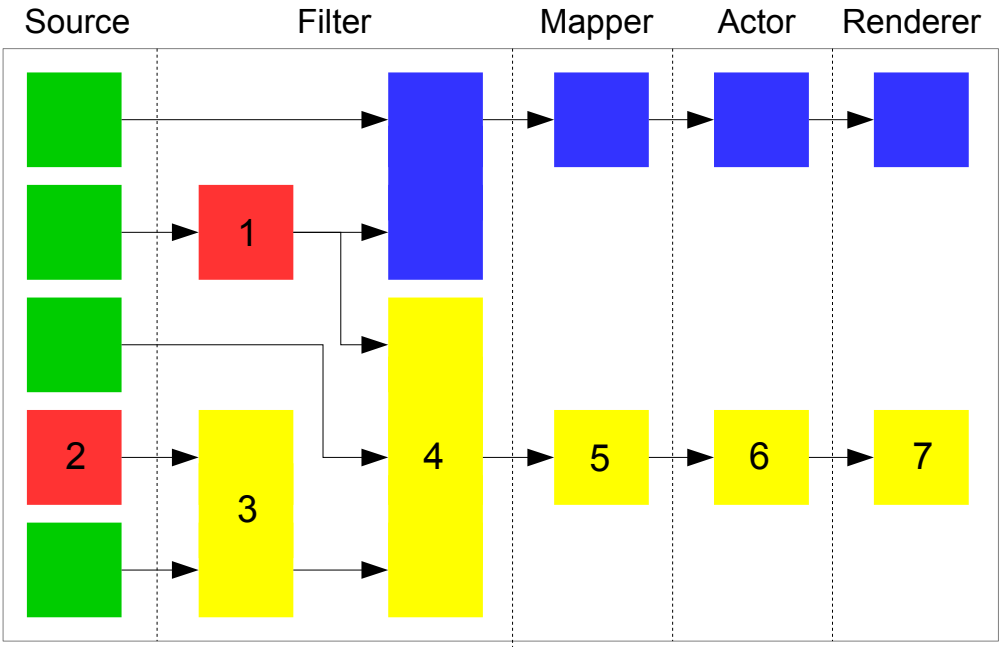
- Po změně - parametry jakéhokoliv modulu se změnily.
- Na vyžádání - uživatel nebo program si sám vyžádal přepočítání. Nějaký z výstupů pipeliny potřebuje data, všechny moduly před ním se začnou přepočítávat.

### 4.1.1 Přepočítání změnou

Program si sleduje, kdy se nějaký parametr změní, a po změně přepočítává modul, který se změnil, a všechny moduly, které na něm jsou závislé. Problém s tímto přístupem může být v momentě, když je přepočítání časově náročné.

### 4.1.2 Přepočítání vyžádáním

Potom, co si uživatel upraví parametry, ručně podá požadavek na překreslení, který je interpretován jako žádost konce pipeliny o data. Tuto žádost podá



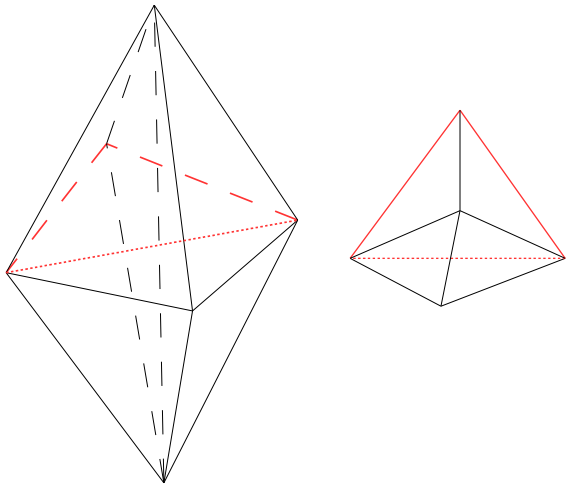

Obrázek 4.2: Příklad pipeliny při požadavku na přepočítání. Čísla na modulech ukazují v jakém pořadí budou přepočítány.

pipelina buď na požádání uživatele nebo sama, když potřebuje překreslit. Zpětnou propagací zprávy o přepočítání se zjistí moduly, které jsou potřebné pro daný výstup pipeliny a musí se přepočítat, a následně se přepočítají.

Příklad pipeliny na obrázku 4.2. Čtverce a obdélníky značí moduly zapojené v pipelině. V pipelině požádal modul 7 o aktuální data. Zelené moduly neměly přenastaveny parametry a nemusejí se přepočítávat. Červené moduly měly přenastaveny parametry a tím pádem se musí přepočítat. Žluté moduly neměly přenastavené parametry, ale musí se přepočítat, protože závisí na modulu, který se musí přepočítat. Modré moduly závisí na modulu, který se musí přepočítat, ale ještě neproběhl požadavek na přepočítají těchto modulů.

## 4.2 Delaunayova tetrahedronizace

Při implementaci DT se odhalilo několik problémů. Předpoklad, že odstraněním čtyřstěnů, které nesplňovali podmínku normály se odstraní všechny

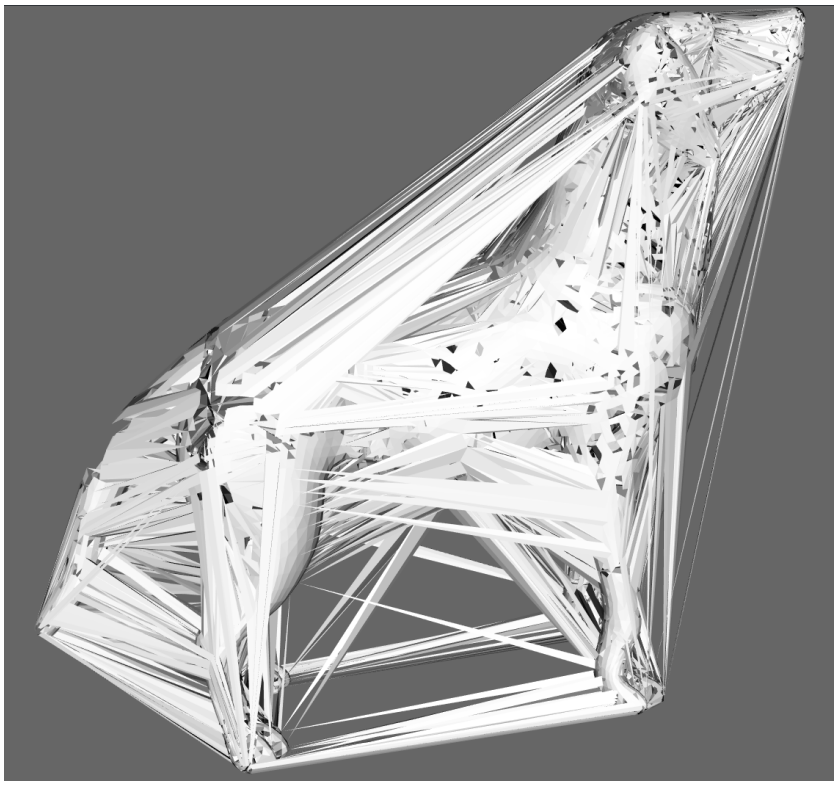


Obrázek 4.3: Špatný typ sítě pro přepojení 2-3, vpravo půdorys.
Červená hrana označuje žádaný trojúhelník.
Jemně čárkovaná hrana neexistuje ve čtyřstěnové síti.

vnější čtyřstěny, byl špatný. Po takovém procesu stále zůstanou čtyřstěny, které neobsahují trojúhelník plochy, ale pouze dvě hrany plochy. Tyto čtyřstěny nejdou rozlišit od čtyřstěnů uvnitř. Tento problém šel vyřešit prohledáváním čtyřstěnové sítě do šířky, přes sousedy, z náhodně vybraného vrcholu. Toto prohledávání označuje čtyřstěny na které narazí. Když se při prohledávání narazí na trojúhelník plochy, tímto směrem se prohledávání ukončí. Takto se rozdělí čtyřstěny do dvou skupiny. Jedna z těchto skupin bude uvnitř a druhá vně. Poté se střídavě prohledávájí obě skupiny a testují se nalezené čtyřstěny. Pokud vybraný čtyřstěn obsahuje trojúhelník plochy a nesplňuje podmínku normály (viz rovnice 3.1), celá tato skupina je vně a musí se odstranit. Zbylé čtyřstěny by měly být správné.

Bohužel druhý předpoklad, že vrácením převrácení 2-3, 2-2 a 4-4 se vytvoří sít, která obsahuje všechny trojúhelníky vstupní trojúhelníkové sítě, je také špatný. DT nemusí vytvořit čtyřstěnovou síť, kde je každá hrana ze vstupní sítě zastoupena. Problém předvedu na převrácení 2-3. Očekávám 3 čtyřstěny, které se po převrácení převedou do dvou čtyřstěnů, které obsahují žádaný trojúhelník. Tyto čtyřstěny v síti nemusejí být, síť může vypadat podle obrázku 4.3 (objem je stále vyplněn). Tento problém z hlediska prohledávání vytvoří díru v trojúhelníkové síti, prohledávání proteče na druhou stranu a přiřadí všechny čtyřstěny do jedné skupiny.

Obrázek 4.4 ukazuje tento problém na reálné síti. Na obrázku je zobrazen



Obrázek 4.4: Díry v povrchu trojúhelníkové sítě po odstranění čtyřstěnů bez trojúhelníku sítě.

pes. Všechny čtyřstěny, které neobsahují trojúhelník plochy, byly odstraněny a jde vidět díry v povrchu.

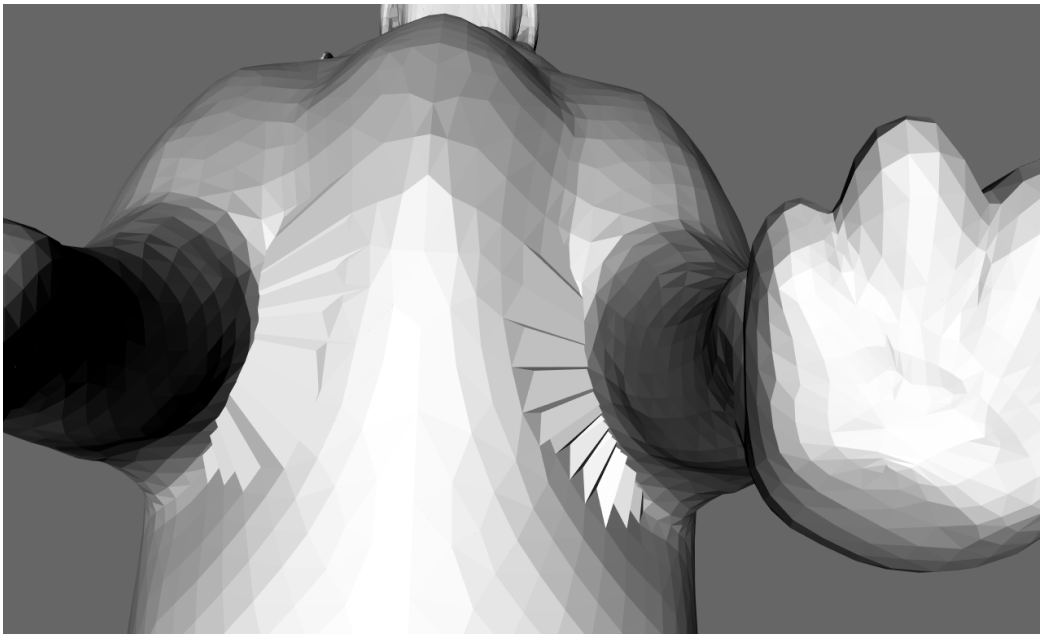
Tento problém vyžaduje náročnější opravování. Potřebuji převést výstup z DT tak, aby obsahoval všechny hrany původní sítě. Jedná se tedy o Delaunayovu tetrahedronizaci s omezením (dále jen CDT). CDT je tetrahedronizace, které se určí, jaké hrany musí obsahovat. Většinou se tento proces provádí použitím standardní DT a poté se teprve opravují všechny vynucené hrany. Pro každou vynucenou hranu se najdou čtyřstěny, které tuto hranu protínají a odstraní se. Tímto se vytvoří díra uvnitř objemu. Tato díra se poté vyplňuje další DT, která z hraničních bodů této oblasti vytvoří čtyřstěny, které lokálně splňují Delaunayovo kritérium. CDT je vždy možná vytvořit ve 2D, ale ve 3D jí nemusí být možné provést bez přidání nových (Steinerových) bodů. Tyto body můžou být přídány na povrch nebo dovnitř objektu. Jsou následně použity k vytvoření CDT, kde by bez nich nebylo možné CDT vytvořit.

Takový proces je náročný na implementaci a byl to důvod pro změnu použitého programu. Odstoupil jsem od DT vyvíjené na KIV a na doporučení jsem zkusil TetGen, který je možno více nastavit.

### 4.2.1 TetGen

TetGen je volně stažitelný software pro vytváření čtyřstěnových sítí ([Si(2013)]. Je psán v jazyce C++, takže není potřeba používat jazykové obaly. Implementuje veliké množství algoritmů pro různé operace se čtyřstěnovými sítěmi, podporuje veliký počet možných typů vstupů a zároveň obsahuje i nástroj na zobrazení výstupních čtyřstěnových sítí.

Pro mé potřeby obsahuje algoritmus CDT, který budu používat. Jako vstup používá po částech lineární povrch (PLC - Piecewise Linear Complexes), což je v případě mého vstupu zaručeno. Další podmínka na vstup je, že vstupní síť nesmí sama sebe protínat, což je znovu přijatelná podmínka. Při správném nastavení program vytvoří CDT a hned při tom odstraní čtyřstěny, které by byly vně povrchové sítě a musel jsem je poté ručně odstraňovat. Program detekuje sebe protínající se sítě, duplicitní vrcholy a neplatné trojúhelníky. Na tyto případy zareaguje buď vypnutím programu s chybovou hláškou, nebo opravením sítě, když to je možné.



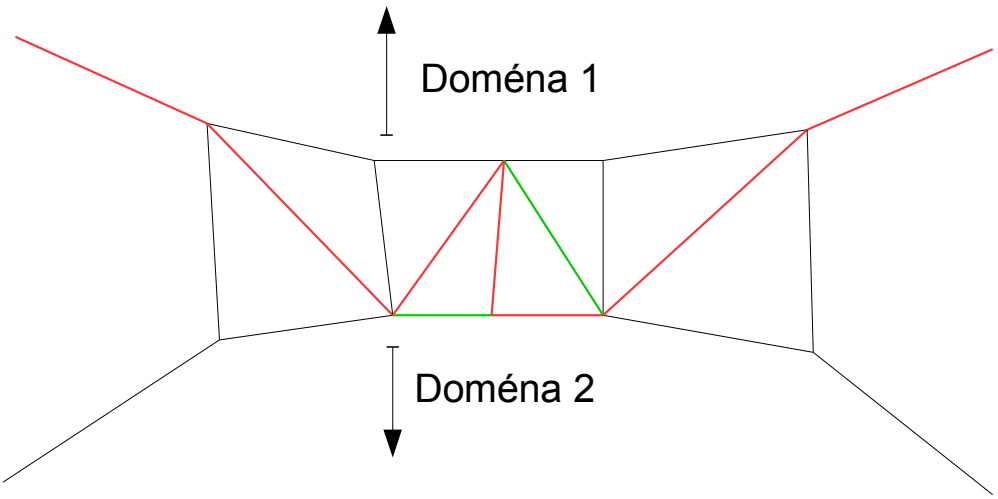
Obrázek 4.5: Příklad čtyřstěnů vně povrchové sítě.

## 4.3 Vytváření čtyřstěnů

Při implementaci doménových čtyřstěnů se ukázal první problém. Aby algoritmus fungoval správně, musela by kostra mít veliké množství vrcholů. Když množství vrcholů bylo příliš malé, některé doménové čtyřstěny byly alespoň částečně vně (příklad na obrázku 4.5). Řešení je před vytvořením doménových čtyřstěnů najít tyto špatně napojené trojúhelníky a přepojit je na lepší vrchol kostry. Pro kostru s málo vrcholy nemusí takový vrchol vůbec existovat, takže se musí přepojit na vrchol kostry, kde bude chyba nejmenší. Toto se musí provést před rozšířením domén. Takto se ale vytvoří ostrůvky v doménách a rozšiřování domén se ztěžuje.

Další problém je s hraničními čtyřstěny. Pokud jsou společné vrcholy dvou domén ve speciálním postavení, program vytvoří hraniční čtyřstěny, které nemá (viz obrázek 4.6). Tento problém jsem zkoušel vyřešit prohledáváním nejbližšího okolí hran, jestli neexistuje cyklus. Pokud existuje, vybere se špatná hrana a odstraní se.

Toto nefungovalo ve všech případech. Druhé řešení je nepracovat s vrcholy, ale s celými trojúhelníky. Změnil jsem význam domény na seznam trojúhelníků (ze seznamu vrcholů). Vrchol kostry tedy stále může mít v doméně více




Obrázek 4.6: Příklad špatných hraničních čtyřstěnů. Pohled na povrch trojúhelníkové sítě.
Červené hrany: správné hrany pro hraniční čtyřstěny.
Zelené hrany: špatné hrany, ze kterých se vytvoří špatný hraniční čtyřstěn.

trojúhelníků, ale každý trojúhelník je pouze v jedné doméně. Takže před celým procesem se musí nejdříve přepracovat domény z vrcholů na trojúhelníky. Toto není triviální, ale ukázalo se, že nepotřebuji od vstupu kostry, aby s sebou nesla informaci o doménách. Dokonce je náročnější převádět doménu vrcholů na doménu trojúhelníků, než řešení vysvětlené v dalším odstavci.

Pro vytvoření trojúhelníkových domén jednoduše projdu všechny trojúhelníky sítě. Pro každý trojúhelník sítě projdu všechny body kostry. Hledám nejbližší vrchol kostry, pro který platí, že by vytvořený doménový čtyřstěn byl uvnitř povrchu. Podmínka je akorát trochu upravená (rovnice 4.1), aby se nepřidávaly trojúhelníky, které mají normálu skoro kolmou s vektorem k bodu kostře. Také se používá těžiště trojúhelníku namísto bodu trojúhelníku.

$$\vec{n} \cdot \overrightarrow{TD} > 0.1 \tag{4.1}$$

Tento algoritmus znovu nenajde pro každý trojúhelník vrchol kostry, který by splnil podmínku normály. Tyto nepřiřazené trojúhelníky vkládám do fronty. Tuto frontu poté procházím. Pro každý trojúhelník se podívám na domény jeho sousedů. Doménu prvního trojúhelníku z fronty nastavuji

podle většinové domény sousedních trojúhelníků (beru v potaz pouze platné domény). Pokud se pro vybraný trojúhelník nenajde v daný okamžik sousední trojúhelník s platnou doménou, znovu se přidá na konec fronty. Takto se postupně přiřadí každý trojúhelník do nějaké domény.

Když jsou domény vytvořeny z trojúhelníků, vytváření čtyřstěnů se velice zjednodušuje. Nemusí se vůbec rozšiřovat domény, protože se už dotýkají. Doménové čtyřstěny se vytvoří prostým průchodem domén všech vrcholů kostry. Hraniční čtyřstěny nejsou tak jednoduché.

Pro vytvoření hraničních čtyřstěnů se projdou všechny hrany kostry. Pro každou hranu kostry se projde doména prvního bodu hrany. Každý trojúhelník domény prvního vrcholu kostry prohledá své sousedy. Pokud sousedí s trojúhelníkem z domény druhého vrcholu hrany, je potřeba vytvořit hraniční čtyřstěn mezi těmito trojúhelníky. To znovu ale není všechno.

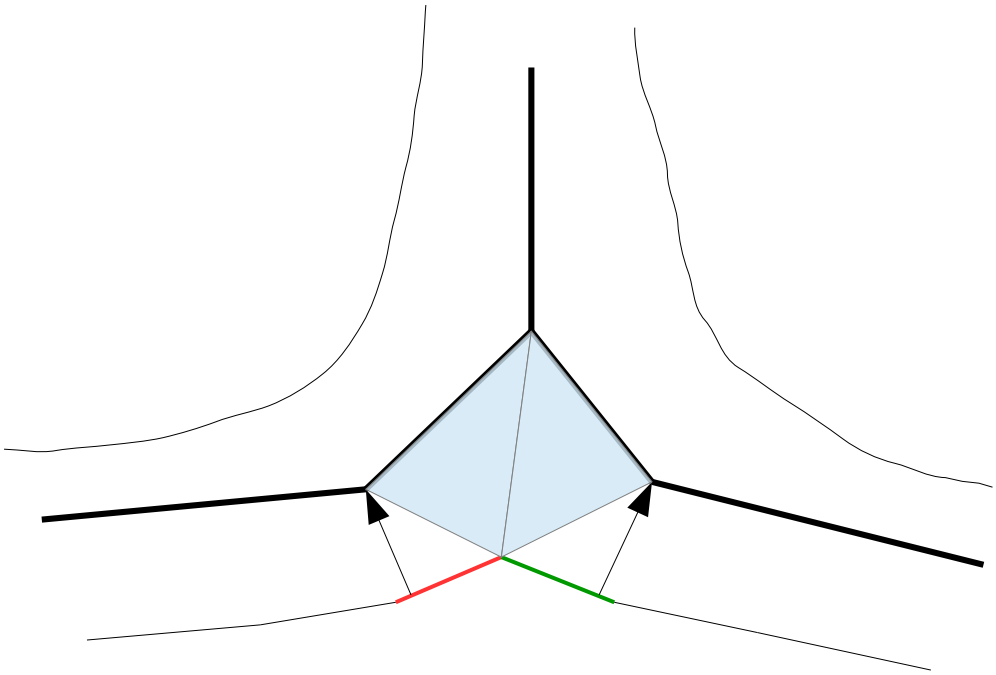
Pokud libovolný trojúhelník sousedí s trojúhelníkem, který je v jiné doméně, je potřeba vytvořit hraniční čtyřstěn mezi těmito trojúhelníky. Pokud první trojúhelník je v doméně, která v kostře nesousedí s doménou sousedícího trojúhelníku, musí se najít cesta v kostře, která spojuje tyto dvě domény. Prohledávání do šířky z jednoho bodu do druhého s pamatováním zpětné cesty to vyřeší. Poté je potřeba vytvořit hraniční čtyřstěny pro každou hranu kostry nalezené cesty (příklad ve 2D na obrázku 4.7).

## 4.4 Subdivision

Subdivision prodělalo jenom jednu změnu oproti návrhu. Místo přidání až šesti vrcholů je povoleno přidat pouze jeden. Tímto odpadá složitá implementace všech možností dělení a zároveň nebude potřeba provádět rekurzi.

Vyzkoušel jsem dělení podle několika kriterií. Všechna dělení jsou implementována prioritní frontou a zastavovací podmínkou. Zkoušel jsem následující typy front a zastavovacích podmínek:

- fronta - poměr objemu čtyřstěnu ku objemu koule opsané; zastavovací podmínka - objem čtyřstěnu
- fronta - součet dihedral úhlů ve čtyřstěnu; podmínka - objem čtyřstěnu
- fronta - velikost dihedral úhlů kolem hrany; podmínka - délka hrany



Obrázek 4.7: Nalezení a vytvoření hraničních trojúhelníků (čtyřstěnů). Vrcholy kostry (kostra tučně) nejsou sousední, nalezla se cesta mezi vrcholy. Pro každou hranu po cestě se vytvoří hraniční trojúhelník.

Všechny tyto přístupy nebraly v potaz, zda se situace po dělení zlepší a měly téměř stejné výsledky. Tyto výsledky nebyly příznivé a převážně zhoršovaly situaci. Nakonec jsem implementoval algoritmus, který naplní minimální prioritní frontu a každé hraně přiřadí prioritu podle okolních čtyřstěnů. Každá hrana má prioritu spočítanou podle rovnice 4.2. Vedle fronty se uchovávají i délky všech hran. Ty budou použity jako zastavovací podmínka dělení.

$$priorita_{(i,j)} = \sum_{\check{c}ty\check{r}st\check{e}ny\ C\ s\ hranou\ (i,j)} \frac{objemCtyrstenu(C)}{objemOpsaneKoule(C)} \tag{4.2}$$

Algoritmus prochází prioritní frontu, vybírá hranu s nejhoršími čtyřstěny kolem ní. Pokud je délka vybrané hrany kratší než uživatelem určený násobek průměrné délky hrany před začátkem dělení, tato hrana se odstraní z fronty a nebude se dělit. Vypočítají se priority dvou hran, kdyby se vybraná hrana rozdělila. Pokud jsou tyto dvě priority v průměru vyšší (zlepšila se situace), provede se dělení této hrany. Všechny čtyřstěny kolem vybrané hrany se rozdělí na dva menší a zaznamenají se hrany, které se budou muset přepočítat. Po rozdělení čtyřstěnů se přepočítají priority všech ovlivněných hran a začíná se vybírat další hrana na dělení.

Díky zjišťování zda dělení určité hrany vylepší situaci, jsou výsledky tohoto přístupu lepší něž všech ostatních.

# 5 Testování a výsledky

Všechna testování jsem prováděl na své domácí počítačové sestavě:

- CPU: Intel® Core™ i5-2500K (4 × 3.3 GHz)
- GPU: AMD Radeon HD 6950 (800MHz), paměť 2GB (1250MHz)
- Paměť: 20GB
- OS: Microsoft Windows 7 64bit

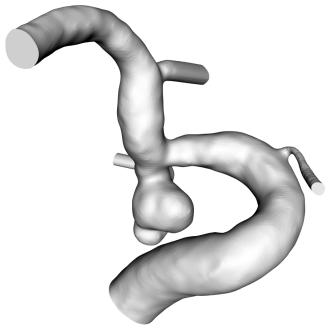
## 5.1 Testovací data

Program jsem testoval na umělých datech (krychle, válec, koule, torus) a na reálných datech (viz obrázek 5.1). Kouli jsem vygeneroval ve šesti různých stupních kvality (1x, 2x, 4x, 8x, 16x a 32x) na testování rychlosti algoritmu v závislosti na velikosti vstupu.

## 5.2 Způsob měření

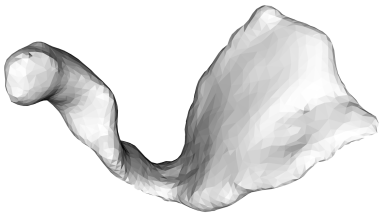
Pro změření určitých charakteristik čtyřstěnové sítě jsem napsal filtr, který neupravuje data, pouze spočítá určité charakteristiky. Takto můžu porovnávat vstup a výstup subdivision a zjišťovat kvalitu sítí na výstupu z DT a TC. Většina následujících grafů neobsahuje úplná data (všechny grafy s výjimkou grafů dihedralových úhlů). Vždy byl vybrán interval, kde se nachází převážné množství dat. Implementoval jsem 3 kritéria, která filtr dokáže sledovat.

- Poměr objem čtyřstěnu ku objemu opsané koule.
- Objem čtyřstěnů.
- Dihedralové úhly.

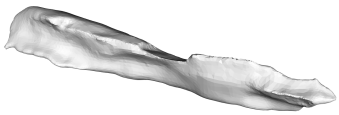


(a) Aneurysma

(b) Stehenní kost



(c) Sval 1



(d) Sval 2

Obrázek 5.1: Reálná data.

Všem těmto kritériím se nastavují intervaly, do kterých mají rozdělit vstupní čtyřstěnovou síť. Všem kritériím jde nastavit, do jakých bucketů se mají čtyřstěny vstupní sítě rozdělit. Nastavují body v intervalu (0, 1), které se seřadí a vytvoří se z nich buckety, do kterých se budou čtyřstěny dělit. Pro každé kritérium se tyto buckety transformují jinak.

- Poměr - neupravuje se.
- Objem čtyřstěnů - základní interval (0, 1) se transformuje na (0, objem největšího čtyřstěnu).
- Dihedralové úhly - interval (0, 1) se transformuje na (0°, 180°).

Toto se provádí, aby uživatel nemusel vědět žádné informace o vstupní čtyřstěnové síti a dostal smysluplná data. Filtr má zároveň možnost uložit data do souboru typu CSV. Zapisuje hranice bucketů a procentuální zastoupení všech bucketů.

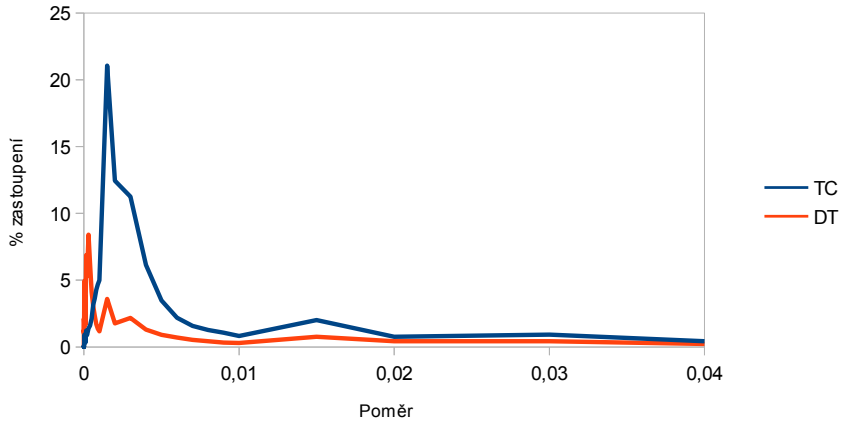
Čím je poměr objemu čtyřstěnu ku objemu jeho opsané koule vetší, tím je daný čtyřstěn lepší. Absolutně nejlepší poměr má rovnostranný čtyřstěn a

to přibližně 0.123. S tímto na mysli považuji za uspokojivý poměr 0.01, protože v reálné síti se nikdy nepřiblížím k rovnostrannému čtyřstěnu. Objem čtyřstěnů je znovu lepší, když je větší. Tento parametr přímo závisí na velikosti vstupní sítě a nejde na něj položit žádný pevný limit. Dihedralové úhly jsou nejlepší znovu v rovnostranném čtyřstěnu. V rovnostranném čtyřstěnu jsou všechny dihedralové úhly 60°. Pro účely tohoto programu považuji za postačující dihedralové úhly v intervalu $(45°, 135°)$.

## 5.3 Testování DT a TC

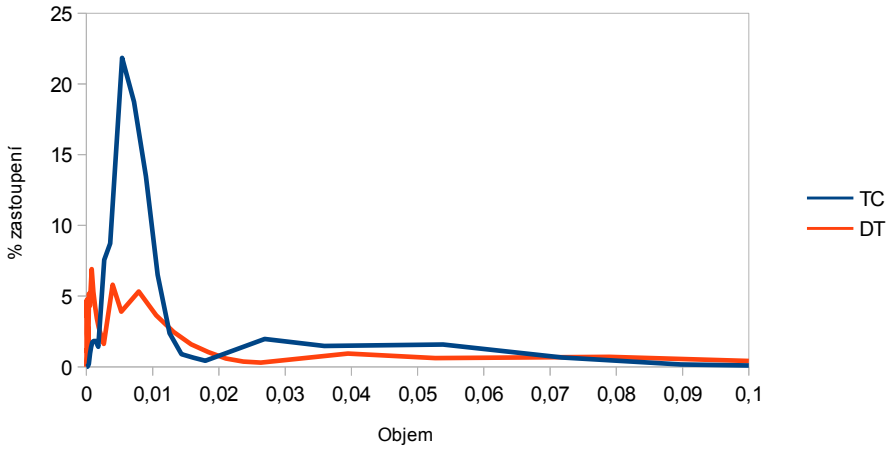
Testuji najednou obě metody řešení a zároveň testuji všechny možné testovací vstupy, aby se data dala lépe porovnat. Z DT očekávám obecně horší čtyřstěnovou síť než z TC. To je způsobeno tím, jak se DT vytváří. Pro všechna měření bylo nastaveno TetGenu, aby zachovával povrchovou síť a přidával nutné vrcholy pouze dovnitř sítě. Takto se musí vytvářet vysoké a úzké čtyřstěny, aby se vyplnil celý prostor.

Graf 5.2 porovnává poměry objemů čtyřstěnů ku objemu jejich opsaných koulí pro oba přístupy řešení na vstupu Aneurysma. Je vidět, že oba přístupy generují čtyřstěny nad žádanou hranicí poměrů 0.01. Většina čtyřstěnů je ale rozložena pod touto hranicí. Právě tyto čtyřstěny se budou dělit. Rozhodně je vidět očekávaný výsledek. Čtyřstěny generované DT jsou blíže nuly, zatímco TC se drží poměrně blízko žádaného poměru.




Obrázek 5.2: Poměry objemů ku objemů koulí pro Aneurysma.

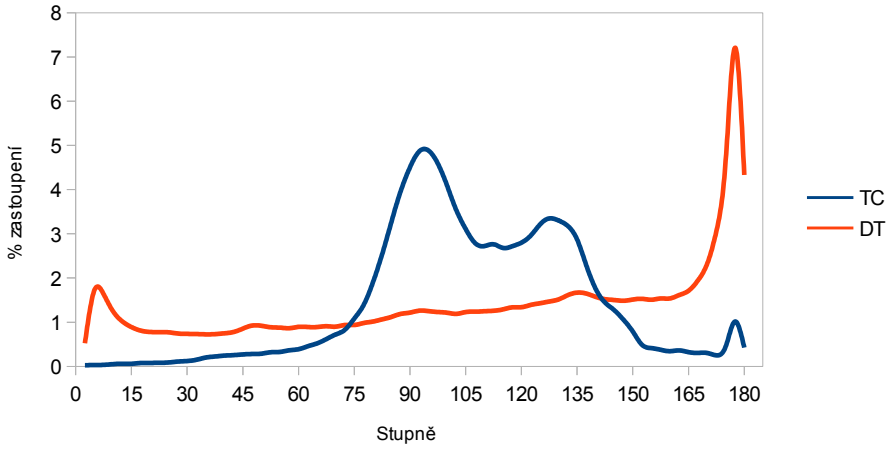
Graf 5.3 obsahuje porovnání objemů čtyřstěnů obou přístupů řešení, znovu pro Aneurysma. Na grafu je vidět, že TC generuje čtyřstěny, jejíchž objemy jsou rozpoloženy hlavně v intervalu $(0.001, 0.015)$, zatímco DT generuje čtyřstěny, jejíchž objemy jsou rozprostřeny více rovnoměrně. V extrémech ale DT vytváří více čtyřstěnů, které se zařadí velice blízko nuly.




Obrázek 5.3: Objemy čtyřstěnů pro Aneurysma.

Graf 5.4 ukazuje rozložení dihedralových úhlů pro obě metody na testovacím vstupu Aneurysma. Zde je přesvědčivě vidět, že TC generuje čtyřstěny, které mají dihedralové úhly v žádaném intervalu $(45°, 135°)$. TC vykazuje jenom velice malé zastoupení v extrémních úhlech okolo $0°$ a $180°$. To se nedá říct o DT, která má rozložené dihedralové úhly téměř rovnoměrně. Jediná výjimka je v extrémních úhlech, kde je zastoupení mnohokrát větší než jinde. Toto naznačuje že DT vytváří veliké množství čtyřsřtěnů, které jsou vysoké a úzké, nebo nízké a široké. Takovéto čtyřstěny nejsou žádané a bez subdivision takové výsledky nedoporučuji používat.

Ve všech kritériích vykazuje TC lepší výsledky něž DT pro testovací vstup Aneurysma. Testoval jsem i ostatní reálná data a všechna uměle vytvořená data. Grafy ostatních testovacích dat jsou v příloze B.




Obrázek 5.4: Dihedralové úhly pro Aneurysma.

## 5.4 Testování subdivision

Subdivision jsem testoval pouze na reálných datech. Znovu jsem testoval obě metody řešení. Přenastavováním subdivision filtru jsem se snažil, aby počty čtyřstěnů po subdivision byly, pro daný vstup a pro obě metody, srovnatelné.

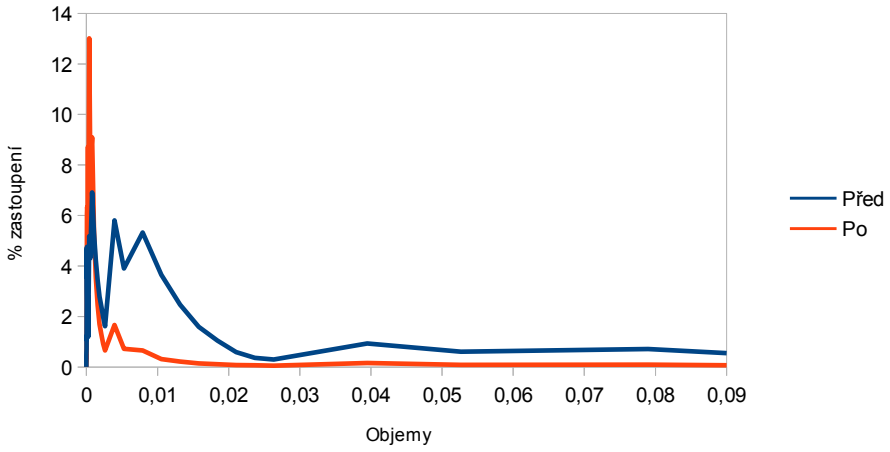
### 5.4.1 Delaunayova tetrahedronizace

V grafu 5.6 je zobrazen výsledek subdivision po DT pro Aneurysma. Je vidět očekávaný nárůst vyšších poměrů na úkor nízkých poměrů. Rozdíl ale není tak výrazný. Tento výsledek napovídá o tom, že čtyřstěnová síť z DT nemá dobrou kvalitu ani po subdivision. Pro rozdělování čtyřstěnů po DT je možná potřeba použít jiný způsob subdivision, který síť vylepší více.

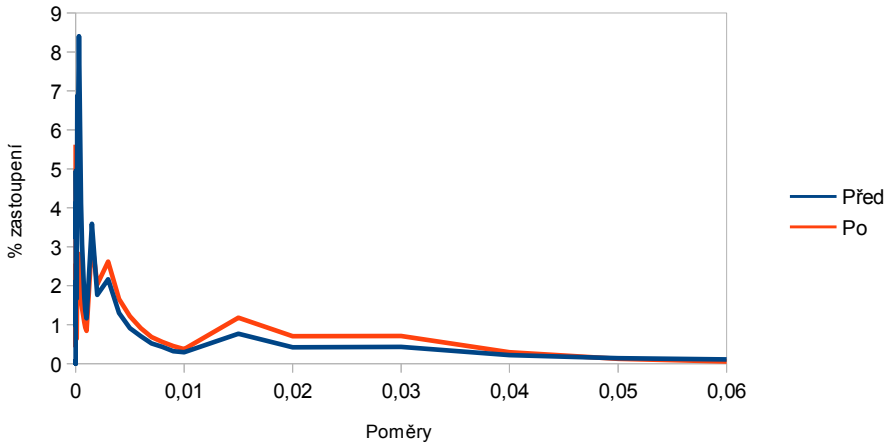
Graf 5.5 ukazuje, že se po subdivision čtyřstěny obecně zmenšily. To je žádaný výsledek a vysvětluje to, proč se příliš nevylepšily poměry čtyřstěnu (viz odstavec nahoře). Subdivision narazil na zastavovací podmínku příliš malého čtyřstěnu dříve, než stihl vylepšit čtyřstěnovou síť.

Z grafu 5.7 je vidět, že subdivision vylepšil situaci s extrémními úhly okolo 0° a 180°, ale ne dostatečně. Dále mírně snížil zastoupení okolo 90° a tím navýšil zastoupení okolo 45° a 135°. Rozložení ale stále není dobré a není

vidět značné navýšení v intervalu $(45°, 135°)$. Zlepšení dihedralového úhlu ale není cíl daného subdivision filtru.

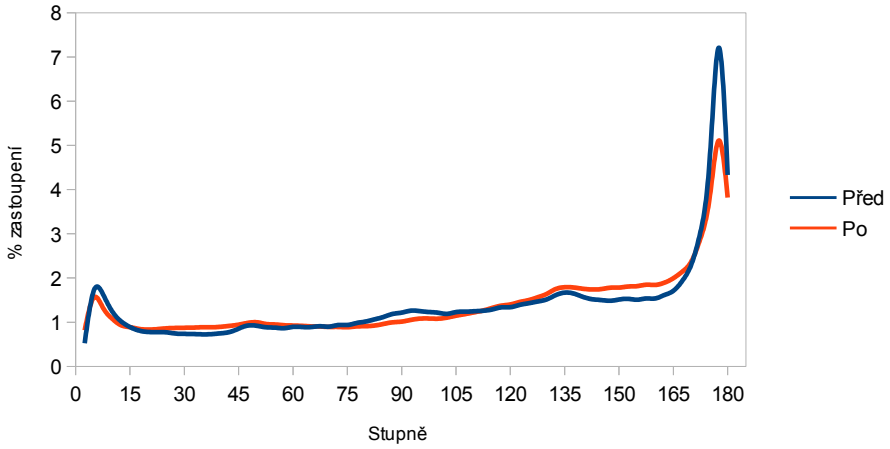



Obrázek 5.5: Subdivision objemů čtyřstěnů po DT pro Aneurysma.




Obrázek 5.6: Subdivision poměrů po DT pro Aneurysma.

Subdivision výsledek z DT příliš nezlepšil. Pro znatelnější výsledek by se muselo prozkoumat přesně jak DT funguje a implementovat nový subdivision filtr. Bez subdivision nedoporučuji DT používat. Subdivision kvalitu čtyřstěnové sítě vylepší, ale změna je příliš málo znatelná, abych doporučil použití i po subdivision. Časová náročnost subdivision podle mého názoru nepřináší dostatečné zlepšení, aby se jí vyplatilo použít.




Obrázek 5.7: Subdivision dihedralových úhlů po DT pro Aneurysma.

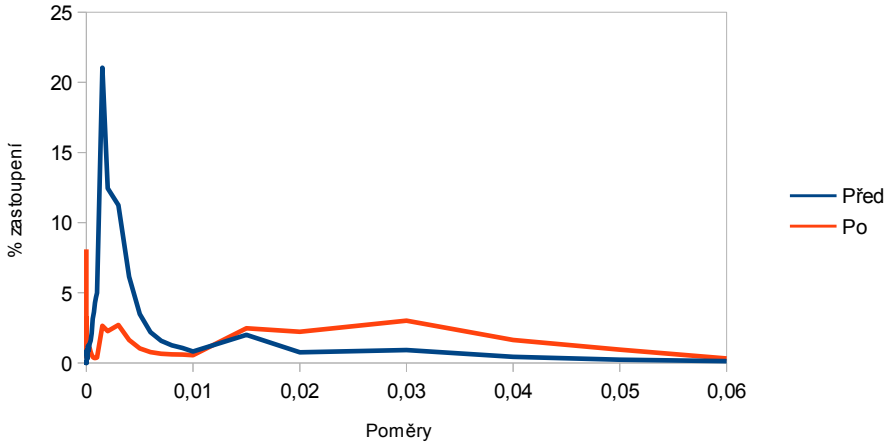
### 5.4.2 Vytváření čtyřstěnů

Na grafu 5.8 jde vidět znatelnější změnu poměrů než u DT. Poměry blízko nuly se také zmenšily, zatímco se poměry okolo 0.03 zvětšily. Tato změna je ale znatelně větší než u DT. Veliká skupina čtyřstěnů byla převedena nad požadovanou hranici 0.01.

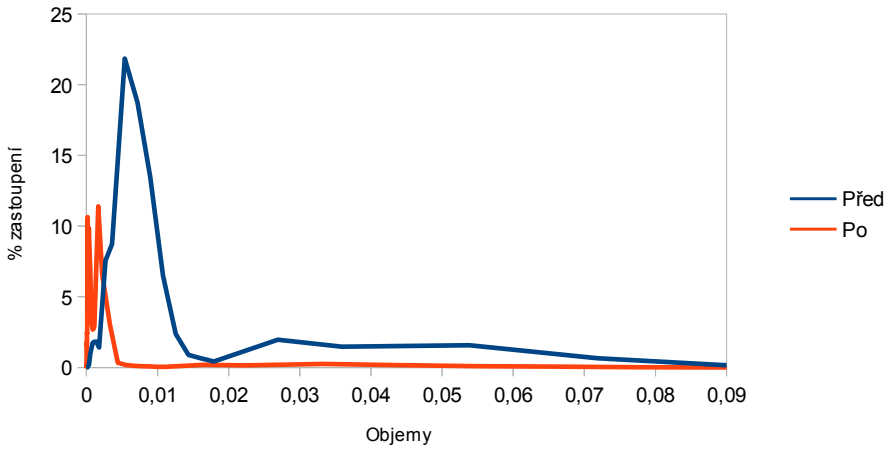
Graf 5.9 je velice podobný grafu objemů pro DT po subdivision. V tomto případě je to doprovázeno lepšími výsledky v poměrech čtyřstěnů.

Graf dihedralových úhlů (5.10) bohužel není příliš dobrý. Je vidět značné navýšení v oblasti extrémů okolo 0° a 180°. Graf vykazuje podobný účinek na zastoupení okolo 90°, které se značně snížilo, zatímco zastoupení okolo 45° a 135° zaznamenalo nárůst. Vypadá to, že úhly okolo 90° jsou s tímto způsobem subdivision nestabilní. V porovnání s DT je zastoupení v extrémech podobné, nejspíše další vlastnost implementovaného subdivision.

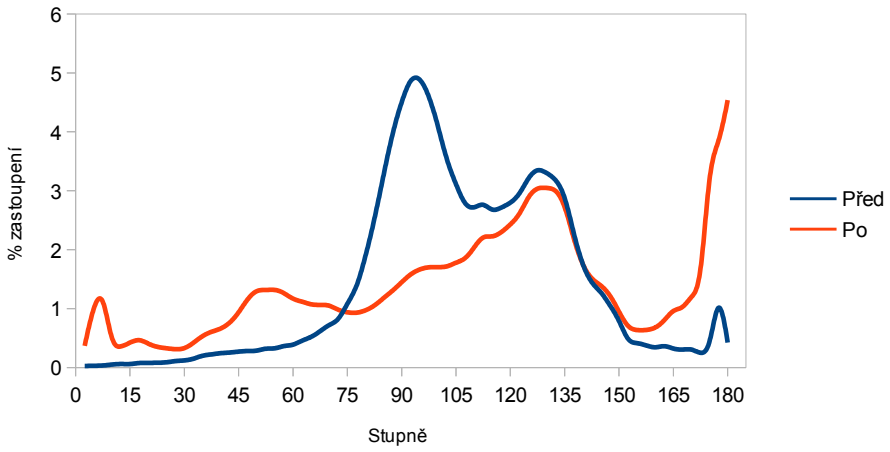
Výsledek subdivision po TC je mnohem příznivější než výsledek DT. Hlavní je, že veliká část poměrů čtyřstěnů se dostala nad žádanou hranici 0.01. Toto se stalo na úkor nyní horších dihedralových úhlů.




Obrázek 5.8: Subdivision poměrů po TC pro Aneurysma.




Obrázek 5.9: Subdivision objemů čtyřstěnů po TC pro Aneurysma.




Obrázek 5.10: Subdivision dihedralových úhlů po TC pro Aneurysma.

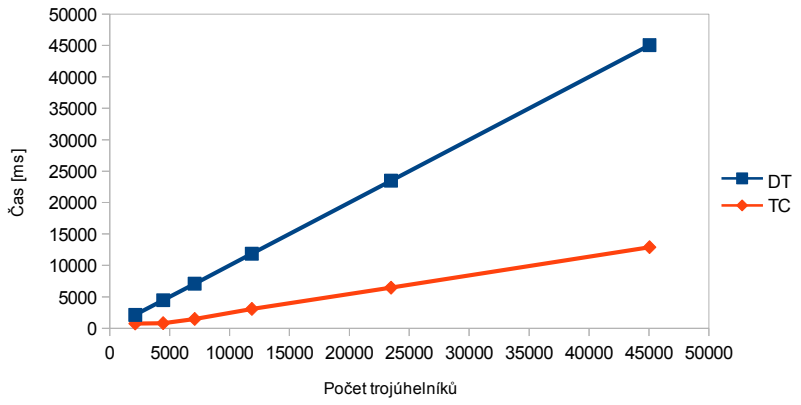
## 5.5 Testování časové náročnosti

Pro měření časové náročnosti budu používat uměle vygenerované vstupy. Přesněji kouli a to v 6 úrovních detailů. Budu měřit časovou náročnost DT nebo TC a následné subdivision. Aby výsledky měření byly porovnatelné, upravoval jsem nastavení subdivision filtru, aby pro každý vstup byl podobný poměr počtu čtyřstěnů před a po subdivision. Tento poměr jsem si vybral v intervalu (3.5, 3.6), takže výstupní čtyřstěnová síť má přibližně 3.5 krát víc čtyřstěnů než vstupní čtyřstěnová síť do subdivision filtru.

### 5.5.1 Testování DT a TC

Tabulka 5.1: Tabulka závislosti času na velikosti vstupní sítě obou metod řešení.

| Trojúhelníků vstupní sítě | Čas DT [ms] | Čas TC [ms] |
|---:|---:|---:|
| 6960 | 2130 | 750 |
| 14110 | 4460 | 810 |
| 28320 | 7090 | 1480 |
| 57120 | 11860 | 3080 |
| 114240 | 23480 | 6460 |
| 228486 | 45050 | 12900 |

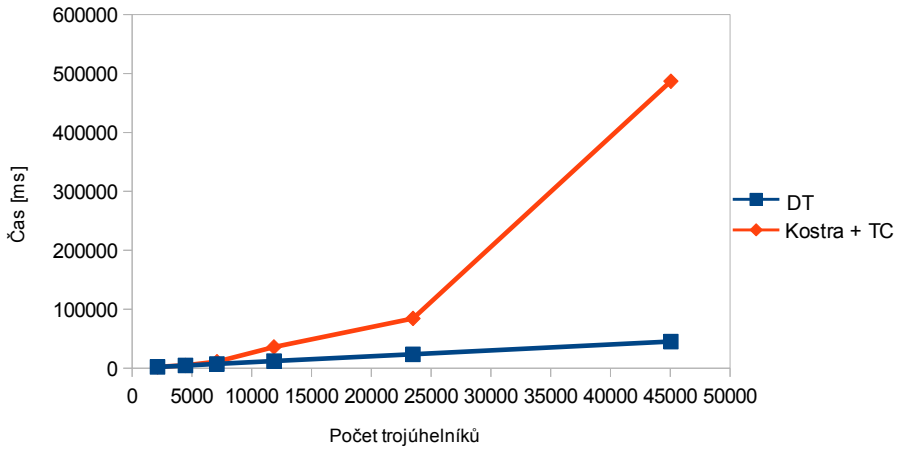



Obrázek 5.11: Graf závislosti času trvání obou metod na velikosti vstupní sítě.

Z grafu 5.11, který zobrazuje data z tabulky 5.1, je vidět, že algoritmus TC je mnohem rychlejší ve vytváření čtyřstěnové sítě než DT. Oba přístupy mají téměř lineární složitost. Je nutné ale podotknout, že TC potřebuje kostru vstupního objektu. Pro vytváření kostry jsem použil implementaci dodanou autory článku [Au Oscar(2008)]. Vytváření kostry značně zpomalí TC.

Tabulka 5.2: Tabulka časové závislosti vytváření kostry na velikosti vstupní sítě.

| Trojúhelníků | Čas kostry [ms] |
|---|---|
| 6960 | 1482 |
| 14110 | 3829 |
| 28320 | 9413 |
| 57120 | 32718 |
| 114240 | 77912 |
| 228486 | 474121 |




Obrázek 5.12: Graf závislosti času trvání obou metod na velikosti vstupní sítě, když se bere v potaz časová náročnost vytváření kostry.

Časová náročnost vytváření kostry je nepřekvapivě vysoká. Během algoritmu vytváření kostry se několikrát počítá soustava lineárních rovnic (pro každý vrchol jedna rovnice). Řešení soustavy lineárních rovnic je náročnosti $O(n^3)$. Takto časová náročnost samotného vytváření kostry přeroste celý algoritmus DT už u třetího testoavcího vstupu (graf 5.12). Pokud tedy není dostupná kostra, TC bude pro veliké vstupní sítě vždy pomalejší než DT. Je-

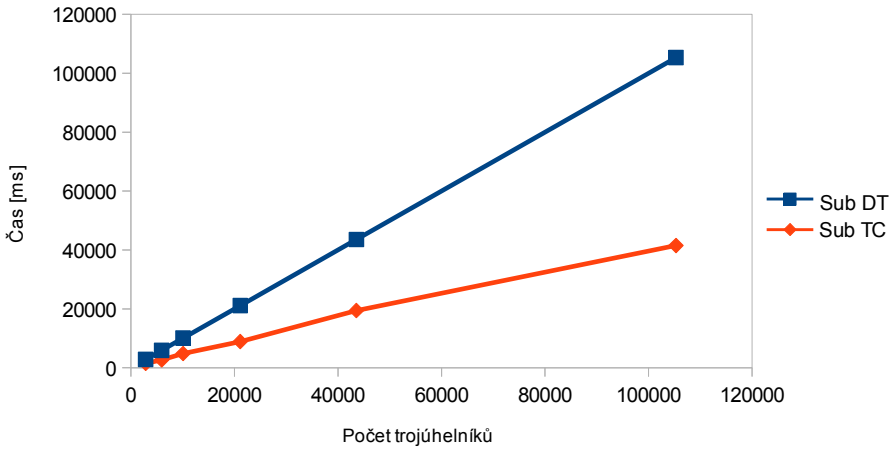
diná možnost je použít jinou implementaci vytváření kostry, která je řádově rychlejší (alespoň $O(n^2)$).

### 5.5.2 Testování subdivision

Časová náročnost subdivision (tabulka 5.3 a její znázornění na grafu 5.13) znovu ukazuje, že subdivision TC bylo rychlejší. Při tomto měření oba subdivision vytvářely stejný procentuální nárůst počtu čtyřstěnů. Počty čtyřstěnů ale nebyly stejné, protože nejdou ovládat.

Tabulka 5.3: Tabulka časové závislosti subdivision obou přístupů na velikosti vstupní sítě.

| Trojúhelníků | Čas sub DT [ms] | Čas sub TC [ms] |
|---:|---:|---:|
| 6960 | 2840 | 1500 |
| 14110 | 5930 | 3829 |
| 28320 | 10070 | 9413 |
| 57120 | 21110 | 8890 |
| 114240 | 43540 | 19450 |
| 228486 | 105300 | 41500 |




Obrázek 5.13: Graf závislosti času trvání subdivision obou přístupů na velikosti vstupní sítě.

Tabulka 5.4 ukazuje samotné počty čtyřstěnů, které vygenerovaly jednotlivé metody a jejich následné subdivision. Je vidět, že TC obecně generuje

méně čtyřstěnů. I kdyby se čas DT vynásobil těmito poměry, stále vychází, že TC má rychleji dokončenou subdivision.

Tabulka 5.4: Tabulka počtu čtyřstěnů před a po subdivision pro oba přístupy řešení.

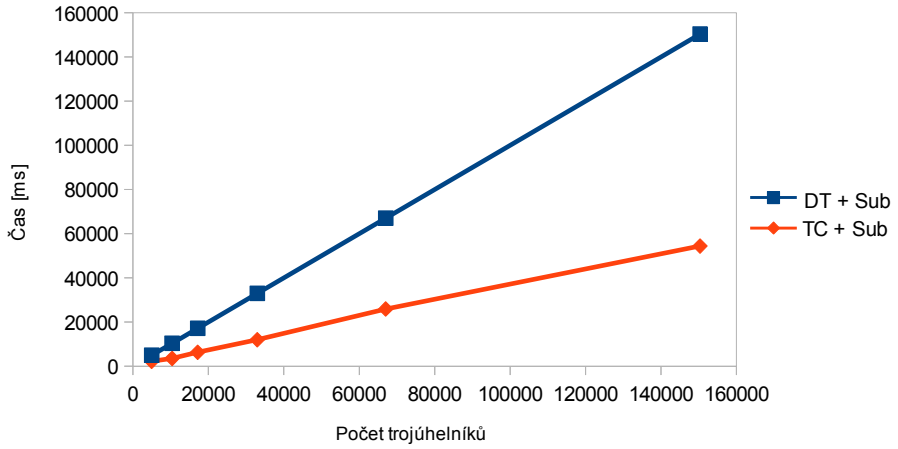
| DT | | TC | | Poměr TC ku DT | |
|---|---|---|---|---|---|
| Před | Po | Před | Po | Před | Po |
| 11685 | 41335 | 7680 | 27354 | 0.657 | 0.662 |
| 23532 | 84108 | 15470 | 54476 | 0.657 | 0.648 |
| 46239 | 164269 | 29760 | 108846 | 0.644 | 0.663 |
| 91325 | 326450 | 59840 | 212768 | 0.655 | 0.652 |
| 179583 | 638727 | 118560 | 423648 | 0.660 | 0.663 |
| 354430 | 1272530 | 233232 | 825054 | 0.658 | 0.648 |

### 5.5.3 Celkové výsledky

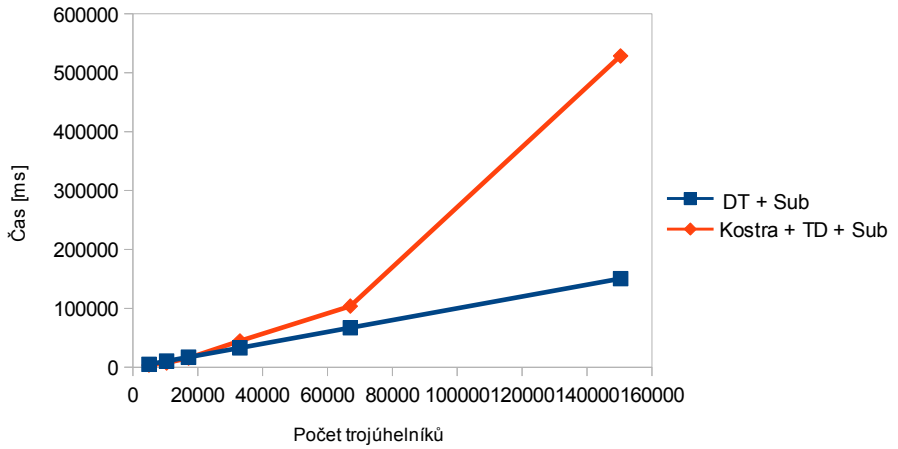
Tabulka 5.5 a následné grafy 5.14 a 5.15 ukazují celkové výsledky obou metod řešení. Zaznamenávají celkovou časovou náročnost obou metod. První graf (5.14) zobrazuje časovou náročnost obou přístupů, pokud je kostra dostupná. Pokud je kostra dostupná, TC vykazuje mnohem lepší výsledky. Pokud je ale potřeba nejdříve kostru extrahovat, časová náročnost TC se razantně zhoršuje. Pokud kostra není dostupná, nedoporučoval bych použití TC, nebo bych naopak doporučoval použít jiný algoritmu na výpočet kostry.

Tabulka 5.5: Tabulka časové závislosti obou přístupů na velikosti vstupní sítě. Všechny časy v milisekundách.

| Trojúhelníků | DT + Sub | TC + Sub | Kostra + TC + Sub |
|---|---|---|---|
| 6960 | 4970 | 2250 | 3732 |
| 14110 | 10390 | 3510 | 7339 |
| 28320 | 17160 | 6320 | 15733 |
| 57120 | 32970 | 11970 | 44688 |
| 114240 | 67020 | 25910 | 103822 |
| 228486 | 150350 | 54400 | 528521 |




Obrázek 5.14: Graf časové náročnosti obou algoritmů a jejich následného subdivision.




Obrázek 5.15: Graf časové náročnosti obou algoritmů a jejich následného subdivision. Pro TC se ještě počítá kostra.

## 5.6 Problém s výstupem

Během testování se ukázal problém s navrženým algoritmem. Objem vstupní trojúhelníkové sítě neodpovídal součtu objemů všech vytvořených čtyřstěnů z TC. Tyto odchylky objemu generoval samotný TC. Objem následného subdivision souhlasil s objemem z TC. Objem čtyřstěnové sítě byl přibližně o 0.1% větší než měl být (pro Aneurysma). Naznačuje to nedokonalosti vybírání správných domén všem trojúhelníkům. Malá odchylka objemu bude nejspíše způsobená protínáním čtyřstěnů, které jsou vytvořeny v místě, kde je náročné rozhodnout, které doméně má být daný trojúhelník přidán (například místa, kde z tlusté cévy vychází tenká céva). Tato chyba ale nebyla tak zanedbatelná pro ostatní reálné testovací vstupy.

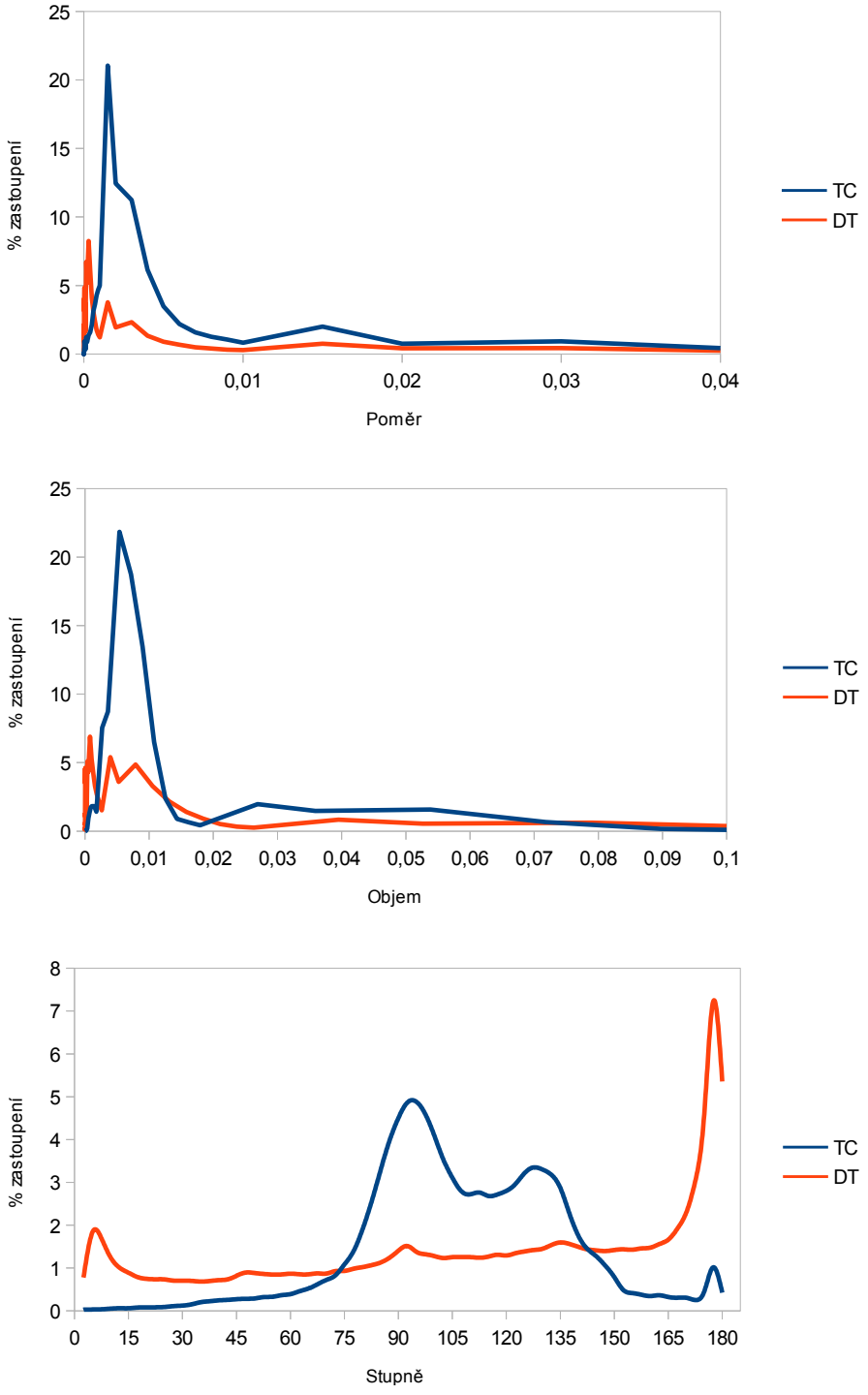
Tabulka 5.6: Tabulka odchylek skutečných objemů a součtů objemů čtyřstěnů po TC.

| Vstup | Skutečný objem | Změřený objem | Chyba |
|---|---|---|---|
| Stehenní kost | 530205 | 568077 | +7.1% |
| Aneurysma | 719.358 | 720.119 | +0.1% |
| Sval 1 | 43987.1 | 47814.7 | +8.7% |
| Sval 2 | 266375 | 302891 | +13.3% |

Je vidět (tabulka 5.6), že chyba objemu může být mnohonásobně větší než u Aneurysma. Upravováním (zjednodušováním) kostry se ale tato chyba zmenšuje. Pro dobré výsledky by to chtělo napsat program, který upraví vstupní kostru, aby se vygenerovala co nejmenší chyba. Nebo danou chybu nalézt a odstranit, ale pouhé nalezení chyby je poměrně náročné a můžu se pouze domnívat, čím je chyba způsobená. Dle mého názoru dochází k problému na rozhraní dvou domén, kde rozhodovací kritérium může dva sousední trojúhelníky rozdělit do domén tak, že se vytvořené čtyřstěny budou protínat. Subdivision pro každý vstup měnilo objem v rozsahu numerické chyby sčítání objemů, takže problém v subdivision nebude.

## 5.7 Testování navíc

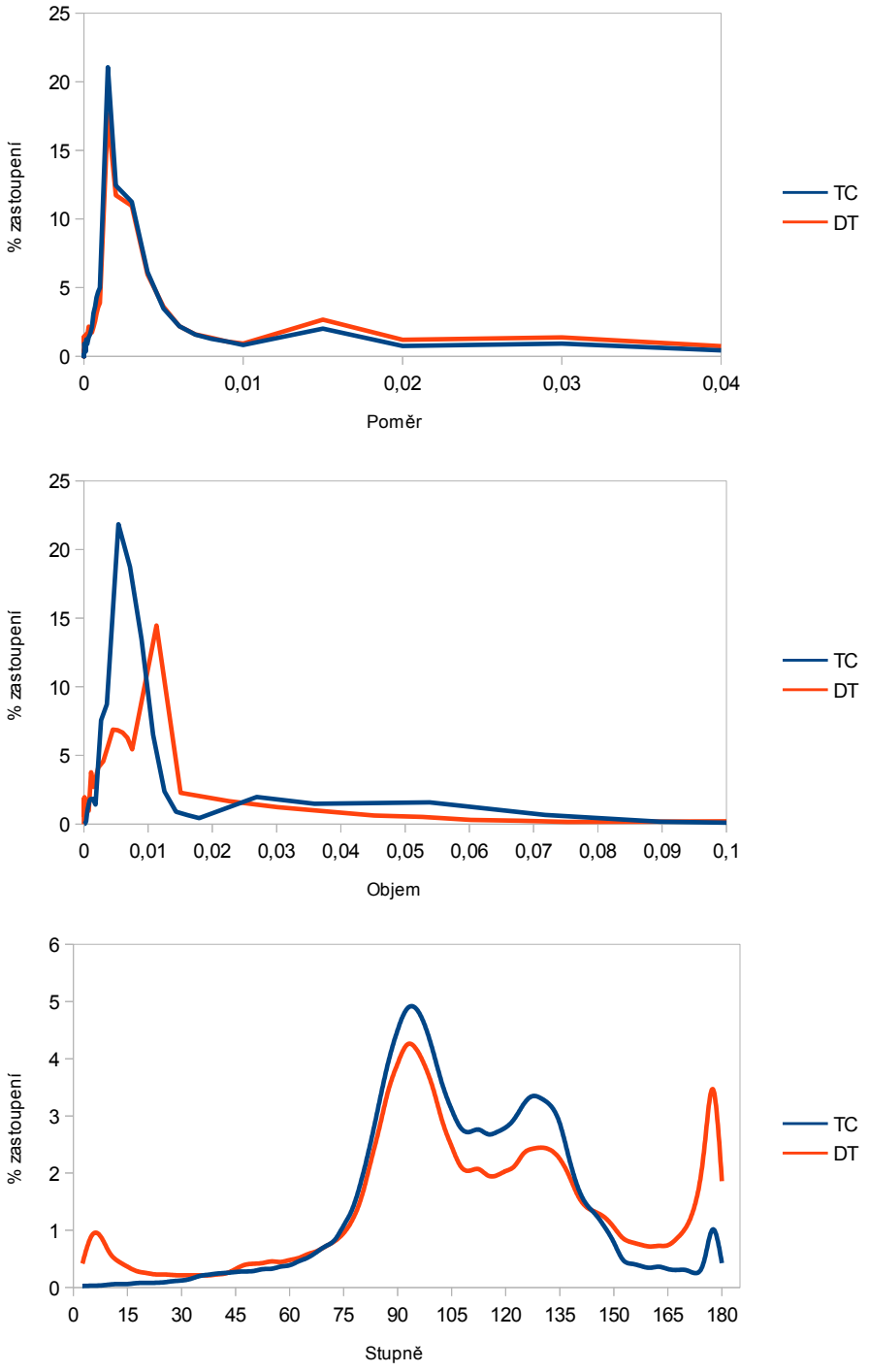
Pro jeden test jsem odstranil z DT podmínku, že musí být zachována vstupní trojúhelníková síť a dovolil přidávání bodů na povrch. To by mělo vytvářet lepší výsledky. Stále testované na Aneurysma.




Obrázek 5.16: Graf porovnání DT, které má povolené přidávat nové vrcholy na povrch, s TC.

Výsledky (graf 5.16) jsou o trochu lepší, než když se nepovolí přidávat vrcholy na povrch, ale rozdíl je minimální.

Zkoušel jsem jestě srovnat TC a DT, když DT dodám i kostru. Toto by mělo vylepšit výsledky DT. Z grafu 5.17 je vidět, že výsledky obou metod jsou velice podobné. Poměry DT jsou mírně lepší než poměry TC. DT si stále zachovává vysoké zastoupení extrémních úhlu okolo 0° a 180°. Graf dihedral úhlů je stále lepší než normálního DT. Z pohledu časové náročnosti je stále lepší použít TC, když je dopstupná kostra.




Obrázek 5.17: Graf porovnání DT, které dostalo i kostru, s TC.

# 6 Závěr

V rámci bakalářské práce jsem prozkoumal různé způsoby tvoření čtyřstěnových sítí. Jednu metodu řešení jsem implementoval. Na porovnání implementované metody s jiným přístupem jsem převzal metodu Delaunayovy tetrahedronizace ([Si(2013)]. Obě metody jsem implementoval jako filtry do VTK. Navíc k dvou implementovaným metodám jsem vytvořil filtr na rozdělování čtyřstěnů na menší čtyřstěny s cílem zlepšit kvalitu čtyřstěné sítě. Na testování jsem vytvořil diagnostický filtr, který mi umožnil vypracované metody důkladně otestovat.

Řešení jsem pečlivě otestoval na uměle vygenerovaných datech (koule, krychle, torus a válec) i na reálných datech (stehenní kost, mozkové aneurysma a dva různé svaly). Na všechny výstupy obou metod jsem použil subdivision čtyřstěnů, který rozdělil čtyřstěny na menší. Výsledná data jsem znázornil na grafech a porovnal mezi sebou. Dále jsem otestoval časovou náročnost obou algoritmů, včetně jejich subdivision, na umělých testovacích datech v několika stupních kvality.

Výsledky ukázaly, že implementovaná metoda je rychlejší a vytváří lepší čtyřstěny i bez subdivision. Časová náročnost implementované metody je ale zkreslená faktem, že je jí dodána kostra, která byla vytvořena v preprocessingu programem podle článku [Au Oscar(2008)], jeho autory. Generování kostry trvá dlouho pro veliké vstupní trojúhelníkové sítě a zpomaluje tak razantně celý algoritmus. Vytváření kostry zpomaluje algoritmus natolik, že je pomalejší než referenční metoda Delaunayovy tetrahedronizace. Z pohledu kvality sítě ale TC generuje lepší výsledky než DT.

Výsledek implementované metody trpí problémem. Vytváří čtyřstěnovou síť, která nemá stejný objem jako vstupní trojúhelníková síť. Tato chyba je závislá na kostře, která se dodá algoritmu. Nejspíše se vytvářejí protínající se čtyřstěny na místech, kde algoritmus může vytvořit dva čtyřstěny dvěma způsoby a rozhodne se špatně. Na reálných testovacích datech se chyba pohybovala od 0.1% do 13.3% skutečného objemu.

Zadání bakalářské práce bylo splněno, i když implementovaná metoda vykazuje chyby. V práci by šlo pokračovat odstraněním stávajících chyb, nebo implementováním lepšího algoritmu na výpočet kostry.

# Literatura


[Au Oscar(2008)] AU OSCAR, C. C.-O. L. T. *Skeleton Extraction by Mesh Contraction* [online]. 2008. [cit. 31.10.2013]. Dostupné z: http://dl.acm.org/citation.cfm?id=1360643&dl=ACM&coll=DL&CFID=283330537&CFTOKEN=27083038.

[Bell(2004)] BELL, J. T. *Vizualization Toolkit (VTK) Tutorial*, 2004. Dostupné z: http://www.cs.uic.edu/~jbell/CS526/Tutorial/Tutorial.html.

[D. Ruprecht(1994)] D. RUPRECHT, H. M. *A Scheme for Edge-based Adaptive Tetrahedron Subdivision* [online]. 1994. [cit. 23.4.2014]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.42.9474&rep=rep1&type=pdf.

[Fuksa(2006)] FUKSA, M. *Diplomová práce - Delaunayova triangulace s omezením (CDT) v $E^2$ a $E^3$* [online]. 2006. [cit. 23.4.2014]. Dostupné z: http://graphics.zcu.cz/files/DP_2006_Fuksa_Miroslav.pdf.

[Kohout(2002)] KOHOUT, J. *Diplomová práce - Paralelní Dolaunayova triangulace ve 2D a 3D* [online]. 2002. [cit. 23.4.2014]. Dostupné z: http://graphics.zcu.cz/files/DP_2002_Kohout_Josef.pdf.

[Kun Zhou(2005)] KUN ZHOU, J. S. X. L. H. B. B. G. H.-Y. S. J. H. *Large Mesh Deformation Using the Volumetric Graph Laplacian* [online]. 2005. [cit. 25.4.2014]. Dostupné z: http://research.microsoft.com/pubs/69206/vgl.pdf.

[L. Rodriguez(2006)] L. RODRIGUEZ, . V. I. N. *Data-driven Tetrahedral Mesh Subdivision* [online]. 2006. [cit. 23.4.2014]. Dostupné z: http://moving.lsi.upc.edu/papers/Data.pdf.

[Šmolík(2013)] ŠMOLíK, M. *Diplomová práce - Metody triangulace v paralelním prostředí* [online]. 2013. [cit. 23.4.2014]. Dostupné z: `https://otik.uk.zcu.cz/bitstream/handle/11025/7594/Diplomova%20prace.pdf?sequence=1`.

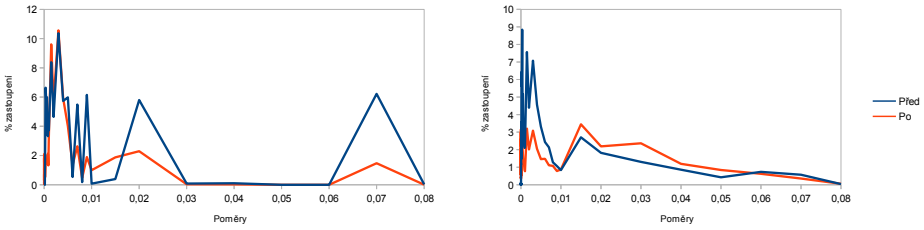
[Si(2013)] SI, H. *A Quality Tetrahedral Mesh Generator and a 3D Delaunay Triangulator*. Weierstrass Institute for Applied Analysis and Stochastics, 2013. Dostupné z: `http://wias-berlin.de/software/tetgen/`.

# A Seznam zkratek

- DT - Delaunayova tetrahedronizace nebo Delaunayova triangulace
- CDT - Delaunayova tetrahedronizace s omezením (Constrained Delaunay Tetrahedronization)
- TC - implementovaná metoda (Tetrahedron Creation)
- Sub - subdivision

# B Další výsledky testování

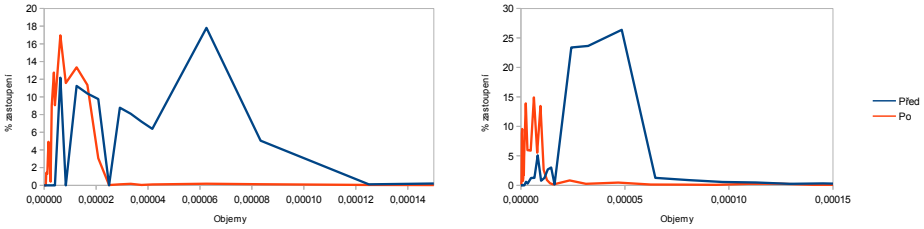
Následují grafy všech ostatních testovacích dat. Všechny ukazují výsledky dané metody před a po subdivision.




(a) DT poměry



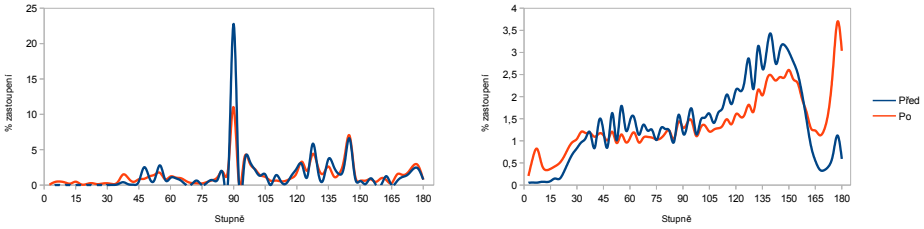

(b) TC poměry




(c) DT objemy




(d) TC objemy

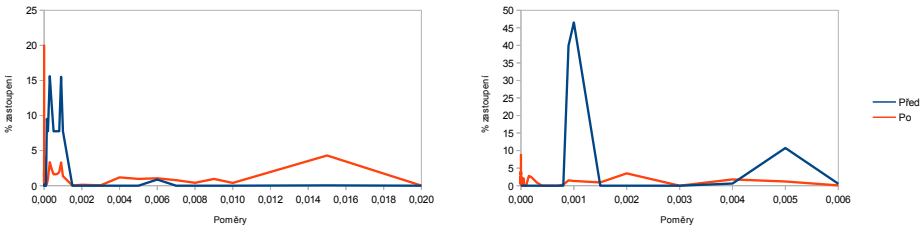



(e) DT dihedral úhly

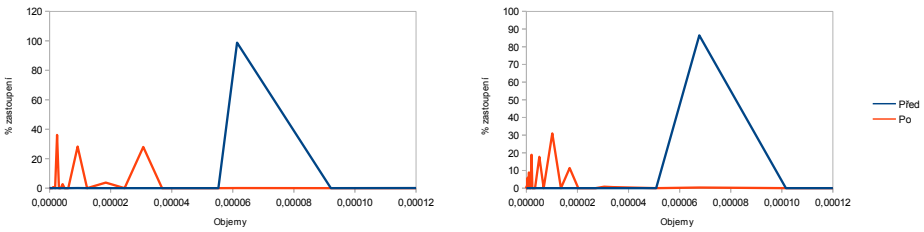



(f) TC dihedral úhly

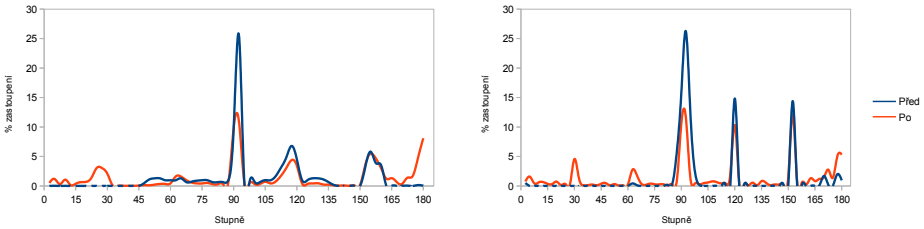
Obrázek B.1: Krychle




(a) DT poměry (b) TC poměry

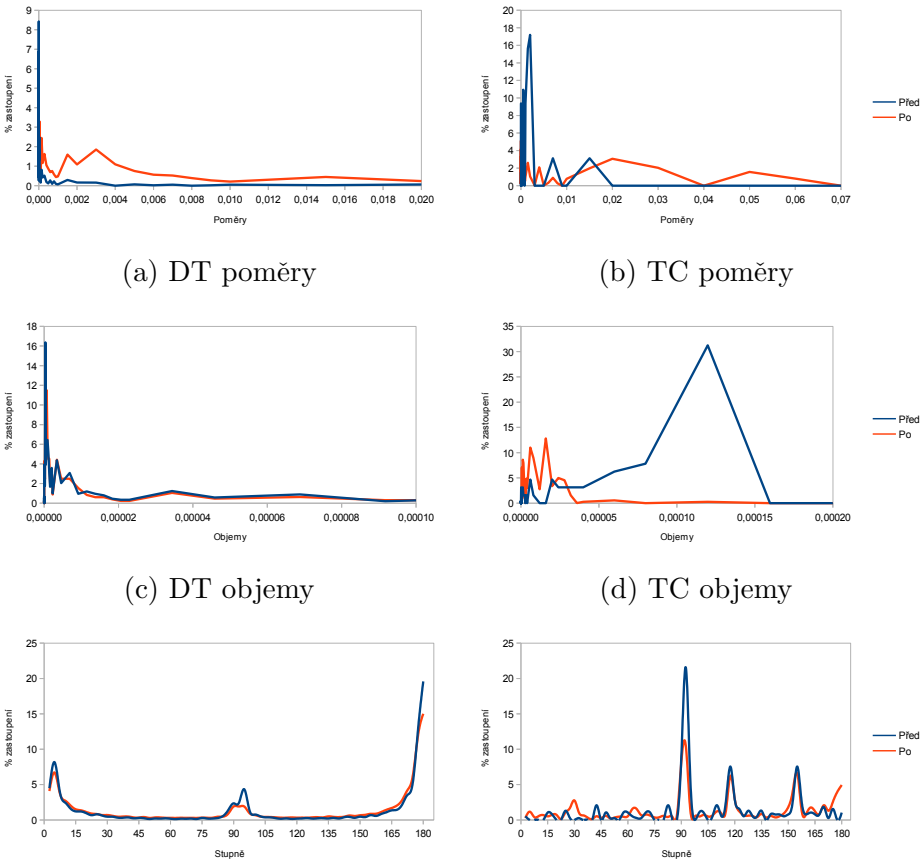



(c) DT objemy (d) TC objemy




(e) DT dihedral úhly (f) TC dihedral úhly

Obrázek B.2: Válec




(a) DT poměry

(b) TC poměry

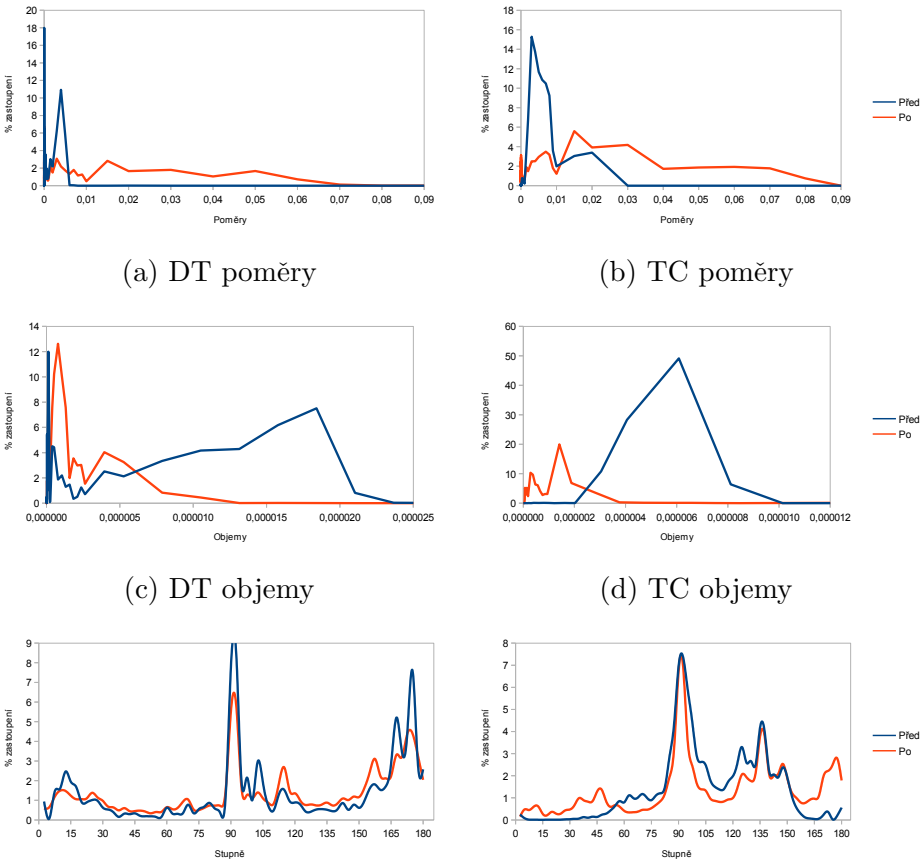
(c) DT objemy

(d) TC objemy

(e) DT dihedral úhly

(f) TC dihedral úhly

Obrázek B.3: Koule




(a) DT poměry

(b) TC poměry

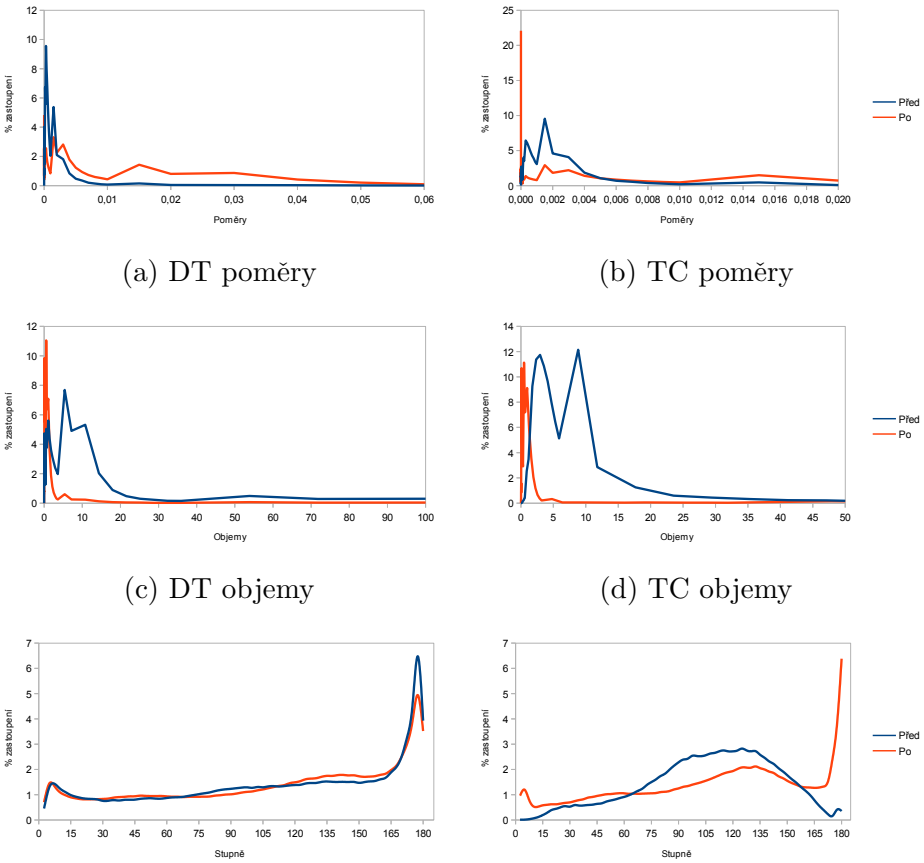
(c) DT objemy

(d) TC objemy

(e) DT dihedral úhly

(f) TC dihedral úhly

Obrázek B.4: Torus




(a) DT poměry

(b) TC poměry

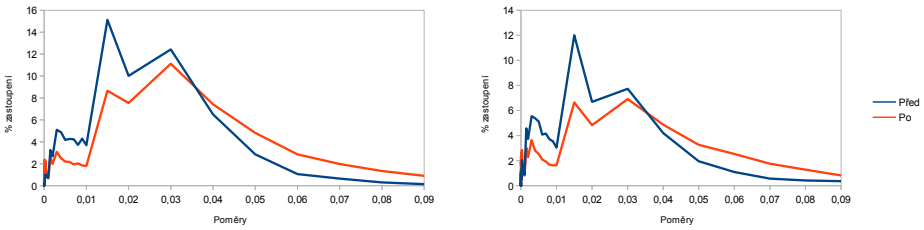
(c) DT objemy

(d) TC objemy

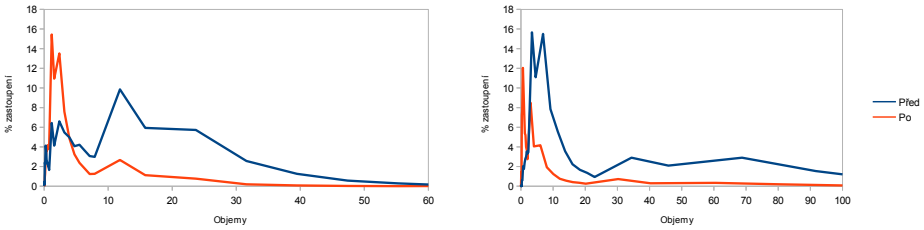
(e) DT dihedral úhly

(f) TC dihedral úhly

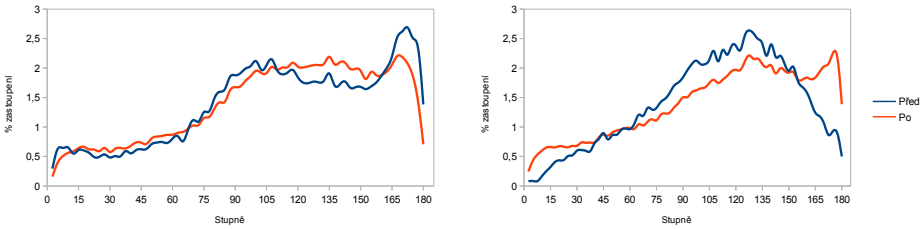
Obrázek B.5: Stehení kost




(a) DT poměry (b) TC poměry

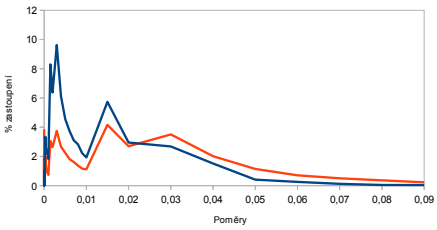



(c) DT objemy (d) TC objemy

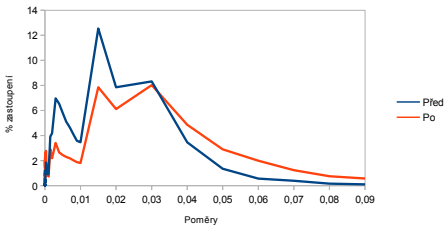



(e) DT dihedral úhly (f) TC dihedral úhly

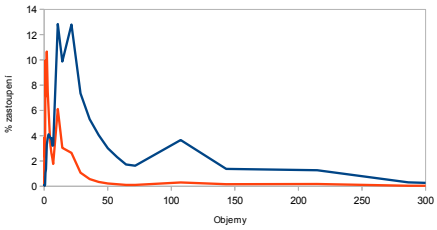
Obrázek B.6: Sval 1



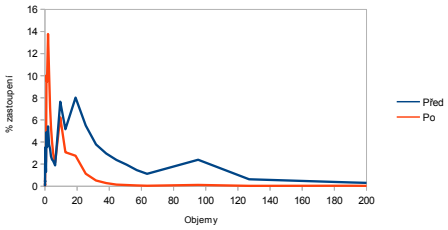

(a) DT poměry



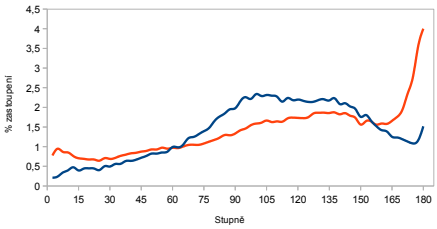

(b) TC poměry



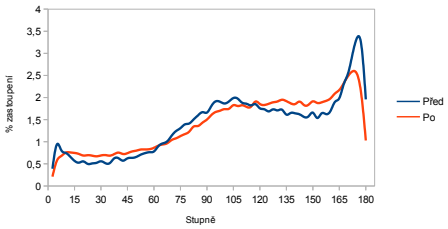

(c) DT objemy




(d) TC objemy




(e) DT dihedral úhly




(f) TC dihedral úhly

Obrázek B.7: Sval 2

# C Uživatelská příručka

## C.1 Instalace

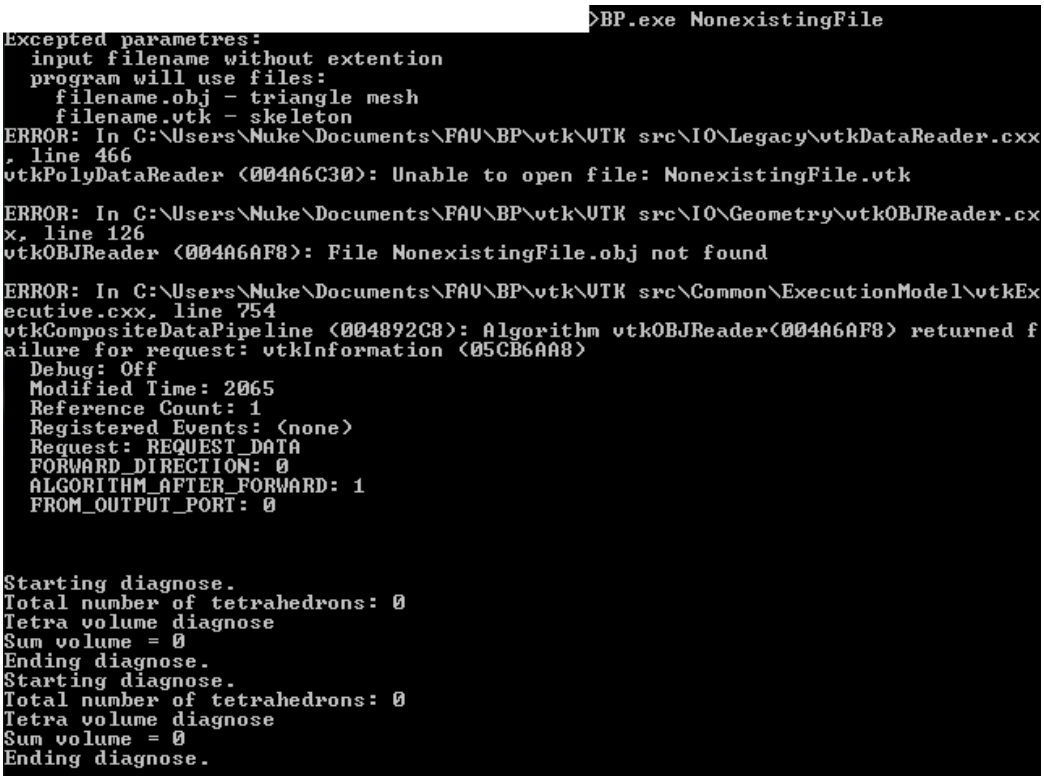
Práce byla vypracována ve Visual Studiu 2012. Je přeložena pro operační systém Microsoft Windows 7. Pro svůj běh vyžaduje všechny knihovny přiložené u přeložené verze programu. Navíc potřebuje nainstalovaný balík Visual C++ Redistributable for Visual Studio 2012 (je přiložen u programu). Samotný program nevyžaduje žádnou vlastní instalaci.

## C.2 Spouštení

Přeložený program se spouští s jedním nepovinným parametrem. Tento parametr je název souborů, které má zpracovat (pokud není zadaný žádný parametr, použije se jako vstup testovací příklad Aneurysma). Tento název musí být zadán bez přípony a program vyžaduje, aby ve stejné složce, jako je program, byly 2 soubory se zadaným jménem a příponami *.vtk* a *.obj*. Soubor s příponou *.obj* obsahuje vstupní trojúhelníkovou síť a soubor s příponou *.vtk* obsahuje kostru dané trojúhelníkové sítě.

Při nastavení vstupního parametru na neexistující soubor zahlásí program chybu a zobrazí prázdný výsledek (viz obr C.1).

Při správném nastavení vstupních souborů program provede implementovaný algoritmus, následovaný diagnostikou, kterou uloží do souboru *DiagnosePre.csv*. Po diagnostice se provede subdivision a další diagnostika, uložená do *DiagnosePost.csv*. Následně se výsledek zobrazí.




Obrázek C.1: Výpis konzole při nastavení neexistujícího vstupního souboru.