

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

## **Diplomová práce**

# **Deformace svalových vláken systémem vázaných částic**

Plzeň 2019

Bc. Martin Červenka

## ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Martin ČERVENKA**

Osobní číslo: **A17N0029P**

Studijní program: **N3902 Inženýrská informatika**

Studijní obor: **Medicínská informatika**

Název tématu: **Deformace svalových vláken systémem vázaných částic**

Zadávací katedra: **Katedra informatiky a výpočetní techniky**

### Z á s a d y p r o v y p r a c o v á n í :

1. Seznamte se s problémem modelování dynamiky svalů a jeho řešením navrženým v rámci evropského projektu VPHOP (FP7-ICT-223865).
2. Seznamte se s technikou 'position based dynamics' (PBD) určenou pro simulaci mechanických efektů. Prozkoumejte dostupné simulační metody založené na této technice relevantní pro modelování dynamiky svalů.
3. Navrhněte přístup založený na PBD pro modelování dynamiky svalů v reálném čase a proveďte jeho implementaci s využitím dostupných knihoven.
4. Nalezněte optimální parametry vytvořeného přístupu pro svaly v oblasti kyčelního kloubu.
5. Dosažené výsledky důkladně zhodnoťte.

Rozsah grafických prací: **dle potřeby**  
Rozsah kvalifikační práce: **doporuč. 50 s. původního textu**  
Forma zpracování diplomové práce: **tištěná**  
Seznam odborné literatury:  
**dodá vedoucí diplomové práce**

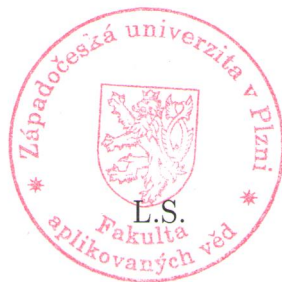
Vedoucí diplomové práce: **Doc. Ing. Josef Kohout, Ph.D.**  
Nové technologie pro informační společnost

Datum zadání diplomové práce: **10. září 2018**

Termín odevzdání diplomové práce: **16. května 2019**

*Radová*

Doc. Dr. Ing. Vlasta Radová  
děkanka



*Brada*

Doc. Ing. Přemysl Brada, MSc., Ph.D.  
vedoucí katedry

V Plzni dne 17. září 2018

# Prohlášení

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

V Plzni dne 15. května 2019

Bc. Martin Červenka

# Poděkování

Tímto bych chtěl poděkovat panu Doc. Ing. Josefovi Kohoutovi, Ph.D. za konstruktivní nápady, užitečné rady a věcné připomínky, které významnou měrou pozitivně přispěly k této práci. Dále bych také rád poděkoval své rodině za podporu během vytváření této práce.

# Abstract

This work explores state-of-the-art methods of muscle model deformation using particle system. The presented PBD-based (position based dynamics) method is enhanced by respecting muscle anisotropy. The problem is solved by deforming muscle surface model, which is transferable to muscle fibre model. The method is competitive with other similar methods as far as speed and precision are concerned. Proposed method results in fast and up to a few cases (typically when muscle is too close to the joint) realistic deformation. Proposed technique is parametric, which means it can deform different muscles depending on parameters. Moreover, this work can go beyond other scientific disciplines, where anisotropy modelling is required.

# Abstrakt

Tato práce prozkoumává existující metody deformace svalu za použití systému vázaných částic. Byla navržena metoda postavena na základě PBD (position based dynamics), zde navržená metoda je ještě obohacena o respektování anizotropie svalů. Metoda deformuje primárně povrchový model svalu, který je však v jakémkoliv okamžiku možné převést na model svalových vláken. Tato metoda je konkurenceschopná s již existujícími metodami, co se týče rychlosti a přesnosti. Výsledné deformace jsou rychlé a až na několik singulárních případů realistické, typicky když je sval poblíž kloubu. Deformace je parametrizována, což umožňuje měnit vlastnosti deformace pro různé svaly. Navržený postup má navíc přesah pro modelování i jiných (nejen anizotropních) materiálů, použití tedy může najít v mnoha různých vědních oborech.

# Obsah

<b>1</b>	<b>Úvod</b>	<b>2</b>
<b>2</b>	<b>Základní fyziologie svalu</b>	<b>4</b>
<b>3</b>	<b>Data a jejich měření</b>	<b>6</b>
3.1	Neinvazivní metody . . . . .	6
3.2	Invazivní metody . . . . .	7
3.3	Dekompozice modelu na svalová vlákna . . . . .	8
<b>4</b>	<b>Existující postupy deformace</b>	<b>9</b>
4.1	Deformační metody . . . . .	9
4.1.1	Metoda konečných prvků . . . . .	9
4.1.2	Aproximace křivkami . . . . .	10
4.1.3	Mass-spring systém . . . . .	11
4.1.4	PBD . . . . .	13
4.2	Detekce kolizí . . . . .	13
4.2.1	Kolize elementárních prvků . . . . .	13
4.3	Reakce na kolizi . . . . .	16
4.3.1	Okamžitá oprava . . . . .	17
4.3.2	Eventuální oprava . . . . .	19
4.3.3	Pohyb kostí . . . . .	20
4.4	Dělení prostoru . . . . .	21
4.4.1	Brutální síla . . . . .	21
4.4.2	Voxelizace . . . . .	22
4.4.3	Rekurzivní metody . . . . .	22
<b>5</b>	<b>Zvolené řešení</b>	<b>24</b>
5.1	Dostupné implementace . . . . .	24
5.1.1	PositionBasedDynamics . . . . .	24
5.1.2	NVFlex . . . . .	25
5.1.3	Vlastní implementace . . . . .	26
5.2	PBD algoritmus . . . . .	26
5.2.1	Pořadí optimalizace omezení . . . . .	26
5.2.2	Matematické pozadí . . . . .	28
5.2.3	Zachování vzdálenosti bodů . . . . .	28
5.2.4	Zachování objemu . . . . .	29
5.2.5	Zachování tvaru . . . . .	31
5.3	Detekce kolizí . . . . .	34

---

5.3.1	Směr detekce . . . . .	35
5.3.2	Další zjednodušení . . . . .	37
5.3.3	Dělení prostoru . . . . .	37
5.4	Anizotropie . . . . .	40
<b>6</b>	<b>Implementace</b>	<b>42</b>
6.1	Třídy a moduly . . . . .	42
6.1.1	Správa paměti . . . . .	43
6.1.2	Formát a načítání dat . . . . .	43
6.2	Základní testy funkčnosti . . . . .	44
6.2.1	Test zachování vzdáleností . . . . .	45
6.2.2	Test zachování tvaru . . . . .	45
6.2.3	Test zachování objemu . . . . .	45
<b>7</b>	<b>Dosažené výsledky</b>	<b>47</b>
7.1	Volba parametrů . . . . .	48
7.1.1	Počet iterací . . . . .	48
7.1.2	Zachování tvaru . . . . .	48
7.1.3	Setrvačnost . . . . .	49
7.1.4	Tlak . . . . .	49
7.1.5	Gravitace . . . . .	50
7.2	Reálná data . . . . .	51
7.2.1	Gluteus maximus . . . . .	51
7.2.2	Gluteus medius . . . . .	54
7.2.3	Iliacus . . . . .	55
7.3	Výsledek dekompozice . . . . .	57
<b>8</b>	<b>Závěr</b>	<b>58</b>
<b>A</b>	<b>Uživatelská příručka</b>	<b>61</b>



# Přehled zkratek

- **BSP** – **B**inary **S**pace **P**artitioning – způsob dělení prostoru
- **CPU** – **C**entral **P**rocessing **U**nit – procesor
- **CUDA** – **C**ompute **U**nified **D**evice **A**rchitecture – platforma pro vývoj programů pro GPU
- **FEM** – **F**inite **E**lement **M**ethod – numerický způsob řešení diferenciálních rovnic, typicky ve fyzice
- **FPS** – **F**rames **P**er **S**econd – počet snímků za sekundu
- **GUI** – **G**raphical **U**ser **I**nterface – grafické uživatelské rozhraní
- **GPU** – **G**raphics **P**rocessing **U**nit – grafická karta
- **HW** – **H**ard**W**are – fyzické komponenty počítače
- **LHDL** – **T**he **L**iving **H**uman **D**igital **L**ibrary – projekt s daty a algoritmy pro modelování muskuloskeletálního systému
- **MIT** – **M**assachusetts **I**nstitute of **T**echnology – v kontextu práce MIT Licence - svobodná licence, možno použít i v proprietárním software
- **MRI** – **M**agnetic **R**esonance **I**maging – magnetická rezonance
- **MSS** – **M**ass **S**pring **S**ystem – způsob simulace systému vázaných částic
- **PBD** – **P**osition **B**ased **D**ynamics – simulace systémem vázaných částic, typicky v počítačové grafice
- **RAM** – **R**andom-**A**ccess **M**emory – paměť s náhodným přístupem, používá se vesměs jako operační paměť
- **SW** – **S**oft**W**are – programové vybavení počítače.
- **VTK** – **V**isualisation **T**ool**K**it – knihovna pro vizualizaci 3D dat.
- **VPHOP** – **V**irtual **P**hysiological **H**uman (**O**steo**P**orotic) – evropský projekt pro výzkum osteoporózy.

# Kapitola 1

## Úvod

Osteoporóza je v dnešní době nemoc s vysokou prevalencí [1]. Jedná se o onemocnění, které způsobuje řídnutí kostní tkáně. Oslabené kosti pak dokáží absorbovat mnohem méně síly, než dojde k fraktuře. To je dobrým důvodem, proč se zabývat hodnotami působících sil na kosti. Dle Bergmann et. al. na kosti kyčelního kloubu může během obvyčejné pomalé chůze působit síla o maximální velikosti přesahující dvojnásobek hmotnosti daného jedince [2]. V krajním případě pokročilé osteoporózy se může tedy stát, že kost sílu o takové velikosti nedokáže rozložit a zlomí se.

Dalším dnes častým onemocněním muskuloskeletálního systému je osteoartritida. V jejím pokročilém stádiu kdy je nutné provést kloubní náhradu je důležité zvolit vhodnou náhradu vzhledem k individuálnímu pacientovi [3]. Při výběru je mimo jiné také nutné zvážit, jaké v dané oblasti působí síly. Špatně zvolená kloubní náhrada pak může působit pacientovi zbytečné obtíže, kterým by bylo možné se vyhnout.

Modelování deformací svalových vláken a svalů samotných je nejen z výše uvedených důvodů v dnešní době stále běžnější. Vytvoření kvalitního modelu umožňuje zjistit fyzikální vlastnosti (sílu, moment síly atd.) v jednotlivých částech svalu, úponů apod.

V případě uvedené osteoporózy je důležité hlavně dostatečně přesně odhadnout působící síly na přilehlé kosti, v případě osteoartritidy působící síly v oblasti konkrétního kloubu.

Uvedené problémy s postupně se vyvíjející dnešní technologií vedly k tomu, že se kosti, svaly a jejich vzájemný pohyb modelují virtuálně. Dnes existující modely s více či méně kvalitními výsledky (například [4], [5]); většina z dnes existujících modelů je však obecného charakteru – modely nejsou personalizovány. Ačkoliv by se mohlo jevit jako dobrý nápad použít jeden kvalitní univerzální model lidského těla, není to vhodné. Takové modely zanedbávají individuální rysy jedince, které mohou být signifikantní. Příkladem signifikantního rysu může být například úhel, který svírá krček s tělem stehenní kosti. Z tohoto důvodu se vytváří i statistické modely, například již zmíněný Zhang et. al. [4] ukazují statistický model založený na PCA.

Protože požadované poměry (pozice, síly, momenty apod.) uvnitř ani na povrchu svalu během různých pohybů zatím dostatečně neznáme (zvláště u konkrétních pacientů), je modelování prováděno typicky opačně, než je tomu reálně. Nejprve se pohne kostí (tento pohyb musí být znám), a poté se teprve

modeluje nějaká vlastnost svalu. To je hlavním důvodem, proč se typicky modeluje pohyb svalů na místo pohybu kostí.

Na Katedře informatiky a výpočetní techniky Fakulty aplikovaných věd Západočeské univerzity v Plzni se v současné době vyvíjejí modely a metody deformací svalových vláken, které si kladou za cíl používání personalizovaných modelů. Zde používané modely se jeví jako dostatečně dobré, bohužel zatím nebyl nalezen dostatečně přesný postup, jak modelovat pohyb svalů (tzn. jak ho v čase deformovat). Právě tato práce by měla deformační metody prozkoumat a najít mezi nimi metodu vhodnou pro modelování svalových vláken, konkrétně se tato práce zaměří na podmnožinu těchto metod – na systémy vázaných částic.

V následující kapitole se nejprve zaměříme na několik základních informací o muskuloskeletálním systému, které jsou v této práci relevantní. Hlavním cílem této kapitoly je popsat fungování systému generujícího sílu v lidském těle, tedy svalů. Dále se v kapitole „Data a jejich měření“ text práce zaměří na možné způsoby měření reálného systému a vytvoření virtuálních modelů, které bude možné dále programově zpracovávat. V další kapitole se prozkoumají dnes známé systémy vázaných částic, které se používají v rámci různých vědeckých oborů od mechaniky přes fyziku až po počítačovou grafiku. Z těchto systémů je následně zvolen vhodný kandidát pro deformace svalových vláken. Prozkoumány jsou i již existující implementace zvolené metody modelování. V kapitole „Návrh implementace“ jsou uvedeny implementační detaily demonstračního programu vytvořeného v rámci této práce, jehož vhodné parametry jsou diskutovány hned v kapitole následující. Nakonec jsou uvedeny a zhodnoceny výsledky této metody v porovnání s ostatními existujícími přístupy modelování.

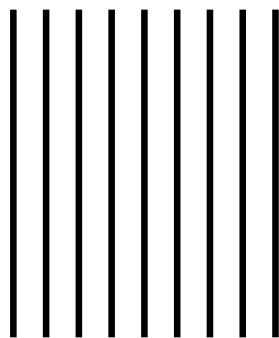
## Kapitola 2

# Základní fyziologie svalu

Svaly v lidském těle jsou velmi komplexní a rozmanité, proto je lze dělit podle zvolených kritérií do mnoha skupin.

Prvním základním kritériem určujícím poměry sil působícího svalu je jeho velikost; obecně platí, že větší sval dokáže vyvinout větší sílu. Dále také záleží na tvaru svalu. Má-li sval větší délku než šířku (délkou je myšleno směr výslednice síly svalu), pak se dokáže lépe prodlužovat a zkracovat, oproti tomu široký sval produkuje větší kontraktilní sílu. Existují i jiné faktory, které sílu svalu ovlivňují.

Protože sval je složen z jednotlivých vláken, základní a v této práci relevantní dělení je na svaly s paralelními a neparalelními (tzv. zpeřenými – *pennate*) vlákny. Paralelní svaly mají vlákna organizována jak je ukázáno na obrázku 2.1, vnitřní struktura zpeřeného svalu je pak znázorněna na obrázku 2.2.



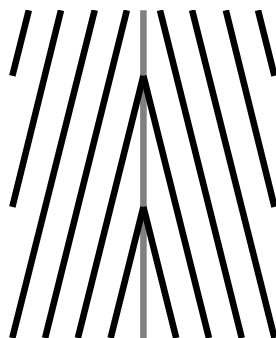
Obrázek 2.1: Struktura svalu s paralelními vlákny.

Mezi svaly s paralelními vlákny patří například dvojhlavý sval pažní (*biceps brachii*) nebo sval krejčovský (*sartorius*). Složitější svaly pak nemají vlákna paralelně. Mezi tyto svaly patří například přímý sval stehenní (*rectus femoris*) nebo podlopatkový sval (*subscapularis*).

Směr vláken má signifikantní vliv na výslednou sílu svalu. Jsou-li vlákna paralelní, pak jejich směr odpovídá směru celkové působící síly (angl. *line of action*). Sval se v tomto případě dokáže zkracovat a prodlužovat nejvíce. Nejsou-li vlákna ve svalu paralelní, pak sval nemá takovou schopnost měnit svoji délku.

Jakou úlohu pak ale mají svaly, jejichž vlákna nejsou paralelní? Trik je v tom, že zpeřená vlákna dokáží vyvinout větší sílu než je tomu v případě svalu

s paralelními vlákny [3] a to i přes to, že síla jednotlivých vláken nepůsobí rovnoběžně s celkovou výslednicí sil. K tomuto jevu dochází proto, že vlákno jednoho typu dokáže vyvinout stejnou sílu nezávisle na jeho délce, avšak vláken ve zpeřeném svalu je typicky více než vláken ve svalu s paralelními vlákny (viz uvedené obrázky 2.1 a 2.2).



Obrázek 2.2: Struktura svalu se zpeřenými vlákny.

Jednotlivá vlákna se dále sdružují do větších celků, svalových snopečků a snopců, které již jsou viditelné na makroskopické úrovni. Svalové snopce se poté pomocí šlach (přes které působí síla jednotlivých svalových vláken) upínají na jednotlivé kosti, což tvoří základ celého muskuloskeletálního aparátu.

Z výše uvedeného je zřejmé, že pro dobrý odhad působících sil uvnitř svalu se musí zjistit (nebo alespoň dostatečně dobře odhadnout) vnitřní struktura svalu. Pro dostatečně kvalitní modelování je tedy nutné znát co nejpřesněji tvar a vnitřní strukturu svalu. Dále se podíváme na to, jakými způsoby se dají reálně takové informace zjistit a také jakými způsoby lze z reálného muskuloskeletálního systému vytvořit virtuální model.

## Kapitola 3

# Data a jejich měření

K jakémukoliv modelu s reálnou podstatou je nejprve nutné naměřit nějaká data. Samozřejmě není tomu jinak ani v případě modelů lidských tkání – kostí a svalů.

Virtuální modely lidských tkání lze v zásadě dělit na dva druhy podle metody jejich měření. Měření se provádí buď metodami neinvazivními nebo invazivními.

### 3.1 Neinvazivní metody

Neinvazivní metody jsou jedinou možností, jak získat personalizovaný model. Mezi takové postupy měření lze zařadit snímkovací metody typicky založené na rentgenovém záření nebo MRI (Magnetic Resonance Imaging).

Zobrazovací techniky založené na rentgenovém záření se používají na zobrazení kostí, nelze s ním příliš dobře měřit měkké tkáně. Další nevýhodou tohoto přístupu je, že pacient je po dobu snímání vystaven radiaci. Výhodou je, že snímání je velmi rychlé (v rámci sekund).

Oproti tomu MRI lze použít o něco lépe na měření měkkých tkání. Pomocí této metody lze zjistit například tvar svalu, ani z těchto snímků však není patrná vnitřní struktura svalu. Také během měření pacient není vystaven radioaktivnímu záření. Problémem této metody je, že snímání trvá relativně dlouhou dobu (typicky v rámci desítek minut) a během této doby by se neměla snímaná část těla příliš pohybovat. To za prvé úplně prakticky zajistit nejde, za druhé minimalizace pohybů je nekomfortní pro pacienta.

Zásadní problém obou těchto neinvazivních snímkovacích metod však je, že za jejich pomoci nelze zjistit vnitřní struktura svalu, lze pouze extrahovat povrchový model svalu.

Konkrétní, v této práci použitá data byla naměřena pomocí dvou ortogonálních rentgenových snímků<sup>1</sup>, tato technika však nedodá dostatečná data pro vytvoření kompletního modelu, proto je potřeba provést registraci na nějaký obecný model (zde konkrétně provést registraci s [6]).

Nyní se přesuneme do konkrétní roviny, jak takový proces probíhá nad zde použitými daty dvou ortogonálních snímků a obecného modelu [6]. Vzhledem ke komplexnosti problému v tomto případě provádí registrace poloautomaticky.

---

<sup>1</sup>pomocí EOS scanneru (podrobnosti viz. EOS imaging, Inc. EOS system, <https://www.eos-imaging.com/professionals/eos/eos/>).

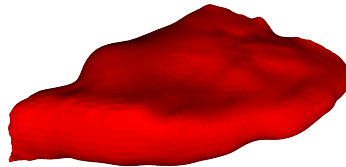
Nejprve expert zvolí landmarky, což jsou anatomicky významné body zvolené na povrchu svalů, a poté se provede nerigidní transformace (například Laplaceova deformace) pro zachování anatomických znaků daného jedince při co nejlepší možné registraci na obecný model [7].

Dále se práce bude zabývat pouze daty trojúhelníkové sítě obecného modelu svalu a odpovídajících svalových vláken. Bylo však důležité zmínit, že existují postupy, které dokáží takový model vytvořit personalizovaný.

## 3.2 Invazivní metody

Invazivní postup měření je takový, kdy je potřeba buď provést nějaký operační zákrok, ať na živé osobě nebo na mrtvém těle, nebo provést nějaký jiný destruktivní úkon. Těmito metodami lze častěji získat přesnější modely než metodami neinvazivními, avšak potřeba operace pacienta pro naměření modelu s sebou nese zbytečná rizika. Bude-li se měřit na mrtvém těle, získáme pouze obecné modely, ne modely personalizované.

Existuje mnoho modelů lidských tkání (např. [4], [5]) naměřené na mrtvých tělech, které jsou vhodné pro různá lékařská odvětví. Pro oblast deformace svalových vláken lze použít model který je naměřen pomocí disekce mrtvého těla člověka. Tato data původně vznikla v rámci projektu [6] a byla dále upravena [7] v rámci projektu VPHOP (FP7-ICT-223865). Výsledkem jsou třídimenzionální manifoldní povrchové trojúhelníkové sítě svalů a kostí. Výsledný model svalu je na obrázku 3.1



Obrázek 3.1: Trojúhelníková manifoldní síť svalu *gluteus maximus* vytvořený invazivní cestou.

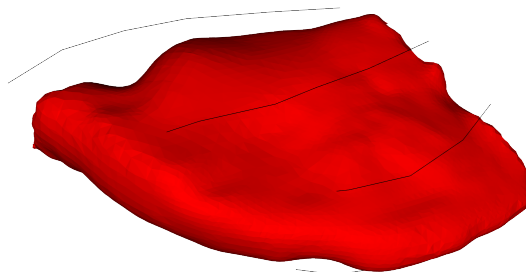
V předchozí kapitole jsme se dozvěděli, že směr svalových vláken není zanedbatelný. Chceme-li pozorovat fyzikální vlastnosti svalu, je nutné si uvědomit, že sval je anizotropní, tzn. že má různé vlastnosti v závislosti na směru působení sil. Pro tyto účely lze použít například data o směru vláken, která byla naměřena disekcí mrtvého člověka v rámci projektu LHDL [8].

Pojem svalových vláken je zde hodně zjednodušen. Samozřejmě prakticky nelze získat kompletní vnitřní strukturu svalu (všechna vlákna), modely svalových vláken jsou modely, které nějakým způsobem aproximují směr většiny vláken, svalových snopečků nebo spíše množiny snopců.

Protože jsou však data často naměřena rozdílnými postupy, je nutné provést jejich registraci. Existuje několik způsobů registrace, ruční, poloautomatické (např. [7], [9]) a plně automatické (např. [10]).

To, že je často nutné provést nějakou formu registrace je zřejmé z obrázku 3.2. Data o povrchovém modelu svalu a naměřená data o směru vláken na povrchu byla naměřena rozdílnými postupy (opět se jedná o sval *gluteus maximus*).

Z obrázku je zřejmé, že naměřená vlákna leží daleko mimo povrch svalu, některá tato vlákna mohou naopak ležet uvnitř svalu nebo ho protínat.



Obrázek 3.2: Model svalu a směr vláken. Modely nejsou registrovány.

### 3.3 Dekompozice modelu na svalová vlákna

Protože většinou dostupných (personalizovaných) metod získáme pouze povrchový model svalu na místo informace o vnitřní struktuře svalu, je nutné vnitřní strukturu odhadnout z daného povrchového modelu. K dobrému odhadu nám také můžou posloužit svalová vlákna, která byla naměřena na povrchu svalu. Poté lze provést tzv. dekompozici

Dekompozice povrchového modelu svalu je operace, kdy dojde k převodu povrchového modelu na jednotlivá svalová vlákna<sup>2</sup>. Práci s povrchovým modelem namísto svalovými vlákny se usnadní výpočet deformace. Problémem deformace povrchového modelu se bude tato práce zabývat dále, zde je však nutné uvést, jakým způsobem lze následně deformovaný povrchový model svalu převést na svalová vlákna. Existuje několik postupů, které tento problém dokáží řešit. Khouř a Kukačka například v kontextu dekompozice svalových vláken navrhli metodu [11], která iterativně mapuje vektorové pole směru svalových vláken (vektorové pole) do obecného povrchového modelu a také dokáže převod provést v reálném čase, jediný problém je v tom, že k tomu potřebuje právě tento vektor, jakým způsobem jsou vlákna ve svalu orientována. Vzor se bude také samozřejmě deformací měnit.

Další metoda [12] dekompozice svalových vláken dokáže také dekomponovat sval na svalová vlákna, je však méně spolehlivá. Pracuje na principu interpolace (a extrapolace) povrchových vláken do objemu celého svalu. Její výhodou je, že používá stejný typ dat (povrchový model svalu a svalová vlákna na povrchu) jako je použit v této práci.

Problematiku dekompozice je dnes již možné celkem efektivně řešit. Proto se tato práce zaměří hlavně na problematiku deformace povrchových (a objemových) modelů, dekompozice v rámci této práce bude popisována pouze okrajově. Převod na svalová vlákna tedy může být proveden například pomocí výše zmíněných dekompozičních metod [11] nebo [12].

<sup>2</sup>Přesněji řečeno na aproximaci směru soustavy svalových snopců, dále toto bude zjednodušeno pouze na „vlákna“



## Kapitola 4

# Existující postupy deformace

Jsou-li dobře známy dostupné možnosti, jak lze data o svalech a kostech naměřit a vytvořit z nich modely, je možné dále tyto modely využít. Existuje dnes několik známých, běžně používaných postupů modelování, které budou postupně v této kapitole popsány.

Problém deformace svalových vláken lze řešit mimo jiné i rozdělením na dva jiné podproblémy – deformace povrchového modelu svalu a následný převod na svalová vlákna. O způsobu, jakým bude model převáděn (dekomponován), jsme se zmínili v předcházející kapitole. Dále se zaměříme na deformace povrchových modelů.

### 4.1 Deformační metody

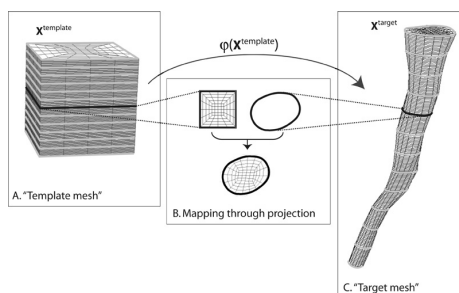
Deformační metody jsou takové metody, které dynamicky mění tvar objektu, Obecně se jedná o nerigidní transformace, které jsou na model aplikovány.

Nerigidní transformace je však nutné nějakým způsobem řídit. To závisí na podstatě řešené úlohy. V některých případech je dostačující například pouze zachovávat vzdálenosti bodů, v jiných aplikacích je nutné přidat další omezení. V případě deformací svalových modelů hledáme metodu, která zachovává vzdálenosti, dále by bylo vhodné aby zachovávala objem a aby co nejpřesněji zachovávala původní tvar původního povrchového svalu. Následují relevantní metody, které takové/některé možnosti poskytují.

#### 4.1.1 Metoda konečných prvků

První možností, jak modelovat sval je pomocí tzv. metody konečných prvků (FEM – Finite element method). Podstatou této metody je rozdělení svalu na co nejmenší (ideálně infinitezimální) části, kde každá část bude mít svůj stav. Tímto stavem může být například v případě zde řešeného problému rychlost nebo působící síla, v jiných aplikacích i jiné fyzikální veličiny jako například teplo. Každá část je dále svázána s několika sousedními prvky a dané fyzikální vlastnosti si pak dle předem stanovených pravidel předávají. Z matematického hlediska se jedná o aproximační metodu řešící soustavu nelineárních (typicky

diferenciálních) rovnic. Vstupem do této metody by byl kompletní sval (celý jeho objem, nejen povrch), což je problematické hned ze dvou důvodů. Prvním důvodem vychází z předchozí kapitoly o způsobu měření, zjištění vnitřní struktury je problém, s dostatečnou přesností lze získat pouze povrchový model. Druhý důvod je výpočetní složitost této metody. Povrchový model má mnohem méně bodů než je tomu v případě kompletního modelu svalu.



Obrázek 4.1: Ukázka převodu povrchového modelu svalu na objemový model [13]. Vnitřní struktura se „odhadne“ pomocí jedné z několika šablon svalu dle orientace vláken (paralelní, zpeřené atd.).

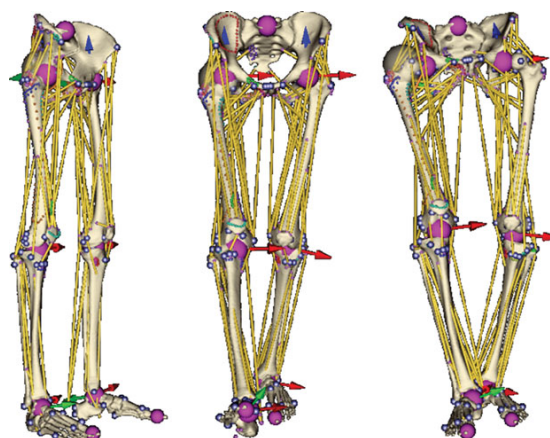
Dostatečně přesný výpočet touto metodou vyžaduje ruční předpracování, které je i pro experta velmi časově náročné a výpočty nad tímto modelem nedosahují ani zdaleka výpočtu v reálném čase ani na superpočítači [13]<sup>1</sup>. Objemový model lze samozřejmě aproximovat, příklad aproximace objemového modelu (který nelze měřením získat) z modelu povrchového je ukázán na obrázku 4.1. I s uvažováním relativní nepřesnosti měření vnitřní struktury svalu není nutné použít fyzikálně velmi přesný FEM k řešení této úlohy. Nepřesnost měření by v tomto případě byla řádově vyšší než nepřesnost výpočtů, což značí, že je použit příliš složitý aparát k řešení takového problému. Proto se pokusím nalézt jednodušší řešení s cílem provést výpočet pokud možno v reálném čase.

### 4.1.2 Aproximace křivkami

Na druhé straně škály co se výpočetní náročnosti týče je metoda aproximace křivkami. Nejjednodušší možností reprezentace svalu je nejprve zjistit, ve kterých směrech působí nejvýznamnější síly a tato místa nahradit předem definovanými křivkami. Některé svaly mají pouze jeden hlavní směr, ve kterém působí, to jsou zpravidla svaly s paralelními vlákny; jiné svaly mají vlákna zpeřená a je potřeba je aproximovat více křivkami.

Existují tedy modely, které aproximují svalová vlákna pouze jednou nebo více křivkami, počet se určí například empiricky podle typu a funkce určitého svalu. Problém takové aproximace je ale v tom, že svaly provádí často velmi komplexní pohyby a taková aproximace nemusí být přesná. Na modelu tohoto typu [14], kde byla svalová vlákna aproximována několika lomenými čarami Valente et. al. zjistili [15], že relativní chyba se může pohybovat až do 75% daného momentu síly. Jak takový model může vypadat je ukázáno na obrázku 4.2.

<sup>1</sup>Práce je z roku 2004, tehdy simulace dle autora trvala 5-10 CPU hodin. Odhadem dle Moorova zákona by tato metoda mohla být schopna běžet v reálném čase až v roce 2042.



Obrázek 4.2: Ukázka aproximace svalů lomenými čárami (žluté) uchycených ke kostem (pomocí modrých bodů) [7].

Postupně se ale ukazuje, že tyto modely nedosahují požadované přesnosti, dokud není křivek (lomených čar) dostatečné množství. Valente et. al. [15] ukázali, že v některých případech je potřeba až šest vláken pro dostatečně dobrou aproximaci. Nalezení těchto lomených čar není snadnou úlohou, typicky se provádí ručně. Výhodou tohoto přístupu je naopak jeho jednoduchost, z čehož vyplývají další výhody, například snadná implementace a rychlé výpočty.

### 4.1.3 Mass-spring systém

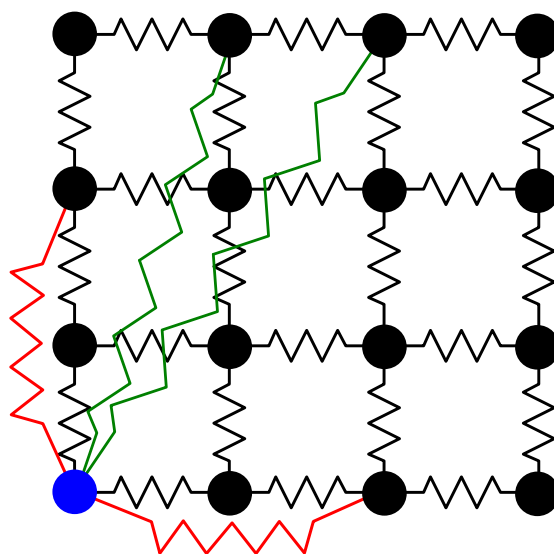
Další možností je systém *mass-spring* (doslovně hmotnost-pružina). Jako vstupní model se použije například trojúhelníková síť svalů. Každému bodu této sítě se přiřadí nějaká hmotnost (sval lze uvažovat jako homogenní, všechny body mohou mít stejnou hmotnost) a každá hrana se nahradí „pružinou“ – virtuální reprezentace hrany. Každá hrana/pružina se snaží zachovávat svojí původní délku, ale má také definovanou tuhost, je možné jí tedy natahovat a zkracovat, je-li k tomu použita dostatečná síla.

Příklad *mass-spring* systému (MSS) je ukázán na obrázku 4.3, vyobrazené body leží na povrchu svalů. Každý bod je spojen černou „pružinou“ se svým sousedem na povrchu svalů. To ale není dostatečné, protože to nezajistí stabilitu modelu – model by se samovolně zdeformoval. Proto je možné dodat další vazby (v případě modře vyznačeného bodu červené vazby), které zlepší stabilitu modelu. Přidáme-li i složitější zelené vazby (resp. spojíme každý bod s každým), model bude sice více stabilní, ale bude také výpočetně náročnější.

#### Matematická podstata pružiny

Pružina se chová dle Hookova zákona (vzorec 4.1).  $F$  vyjadřuje sílu, která je potřeba pro vychýlení pružiny o tuhosti  $k$  o délku  $x$  metrů oproti původní délce pružiny.

$$F = kx \quad (4.1)$$

Obrázek 4.3: Příklad *mass-spring* systému několika bodů.

Pohyb jednotlivých částic je dále možné definovat jako diferenciální rovnici druhého řádu (vzorec 4.2), kde  $x$  je pozice jednoho bodu,  $k$  je opět tuhost dané pružiny a  $m$  je hmotnost bodu. Proměnná  $\alpha$  reprezentuje tlumení pohybu pružiny.

$$m \frac{dx^2}{dt^2} + \alpha \frac{dx}{dt} + kx = 0 \quad (4.2)$$

Je zřejmé, že výpočty se začínají komplikovat. Pro každou dvojici pružina-bod bude existovat jedna rovnice 4.2, řešení celého problému se tím převede na problém řešení soustavy diferenciálních rovnic.

O tuto metodu deformace svalových vláken se pokoušel T. Janák [16], bohužel se ukázalo několik problémů s touto metodou. Navrhované řešení nebylo schopno běžet v reálném čase, v krajních případech bylo řešení nestabilní a další. Nejzásadnějším problémem však je, že postup řešení nezahrnuje zachování objemu svalu, což je jedna z klíčových vlastností které musí být respektovány. Neméně podstatný problém je špatné zachování tvaru, pokud je pružin příliš málo. Použitý způsob detekce kolizí měl také nedostatky a v některých krajních případech byl nefunkční.

Existuje několik možností, jak MSS upravit aby umožňoval zachování i jiných parametrů, než-li zachování vzdáleností. O tomto například pojednávají Hong et. al. [17], kteří do MSS přidávají zachování objemu. Postup zde uvedený však poměrně komplexní. Autor tvrdí, že se jedná o rychlou metodu zachování vzdáleností, není ale možné použít úplně jiný přístup, který by tyto výpočty ještě mnohem více zjednodušil?

#### 4.1.4 PBD

PBD (Position Based Dynamic - pozičně orientovaná dynamika) je způsob modelování, který se snaží pokud možno počítat přímo s pozicemi bodů, což je rychlejší přístup než v případě práce se silami v *mass-spring* systému nebo metodě konečných prvků.

Metoda byla poprvé představena roku 2007 Matthiasem Müllerem [18]. Metoda je zde popsána velmi obecně, řeší se zde obecné modelování manifoldních objektů a tkanin. Práce cílí na zachování hned několika vlastností objektů. První vlastností je vzdálenost mezi body (podobně jako v případě *mass-spring* systému), druhou je detekce kolize a její korektní oprava, třetí je zachovávání objemu během celé simulace a čtvrtou požadovanou vlastností je pokus o co nejlepší zachovávání tvaru vzhledem k ostatním omezením.

Jedná se tedy přesně o vlastnosti, které se v práci T. Janáka [16] ukázaly jako velmi problematické – zachovávání tvaru a objemu. I do MSS (který byl použit v práci T. Janáka) však lze další omezení přidat (viz předchozí sekce). Rozdílem mezi MSS a PBD je, že PBD je zpravidla rychlejší, ale ne tak fyzikálně přesné.

Nepřesnost je do PBD zavedena hlavně způsobem, jakým se řeší všechna omezení, tedy optimalizátorem. Ten na omezení pohlíží jako na systém lineárních rovnic, omezení ale obecně lineární nejsou. MSS používá z tohoto pohledu korektní fyzikálně podložené mechanismy, které jsou však pomalejší. PBD tedy není úplně fyzikálně přesné, ale vizuálně se jedná o dostatečně přesné řešení. Podrobnější detaily o PBD budou popsány dále v kapitole „Zvolené řešení“.

## 4.2 Detekce kolizí

Poté, co jsme prozkoumali některé z metod deformací systémem vázaných částic, mírně odbočíme a podíváme se na metody detekce kolizí. Detekce kolizí je klíčová, protože potřebujeme zajistit, aby například sval neprocházel skrze kost. Dále v této sekci se také podíváme na možnosti dělení prostoru, které jsou klíčové pro rychlou detekci kolizí.

### 4.2.1 Kolize elementárních prvků

Nejprve se zaměříme na kolizi elementárních prvků. Povrchový model je typicky trojúhelníková síť, je tedy tvořen z vrcholů a trojúhelníků.

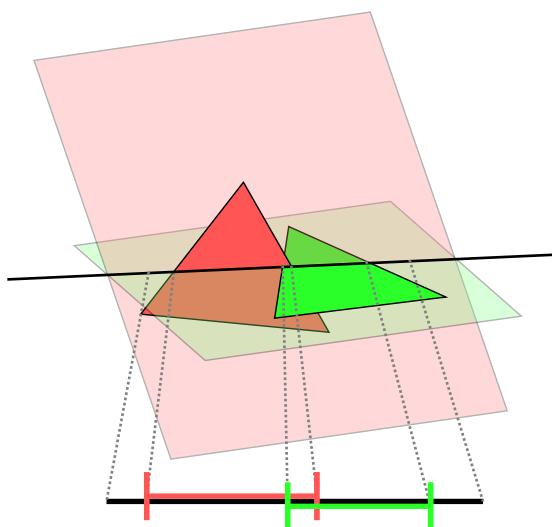
Zde existuje nespočet možností, jaké objekty mezi sebou testovat. Přesným řešením je, že budeme testovat mezi sebou dvojice trojúhelníků. Dále můžeme i uvážit některá zjednodušení, například detekci průchodu bodu trojúhelníkem a následně se podíváme na některé aproximační metody, které volí i jiná geometrická primitiva.

#### Kolize dvou trojúhelníků

Dokážeme-li s nějakou přesností a efektivitou zjistit dvojice trojúhelníků, které by mohly vzájemně kolidovat, je nutné exaktně ověřit, zda-li tato dvojice trojúhelníků opravdu koliduje, nebo jedná-li se pouze o planý poplach.

Přesný výpočet, jestli dva trojúhelníky kolidují, se může zdát triviální, ale úplně tomu tak není. Trojúhelníky mohou ležet libovolně v prostoru, mohou být

rovnoběžné, různoběžné nebo i koplanární – všechny takové možnosti je nutné ošetřit. Nejprve je potřeba určit obě roviny, na kterých oba trojúhelníky leží například jejich obecnými rovnicemi (tu zjistíme pomocí normálového vektoru a jednoho z bodů ležícího na trojúhelníku). Dále může následovat rozřazovací test, kde do první rovnice dosadíme body druhého trojúhelníka a obráceně. Bude-li mít levá strana obecné rovnice první plochy pro všechny body druhého trojúhelníka stejné znaménko, pak to znamená, že druhý trojúhelník leží buď celý před nebo celý za rovinou definovanou prvním trojúhelníkem (a obráceně), tzn. ke kolizi určitě nedošlo. Jsou-li levé strany pokaždé nulové, pak jsou trojúhelníky koplanární.



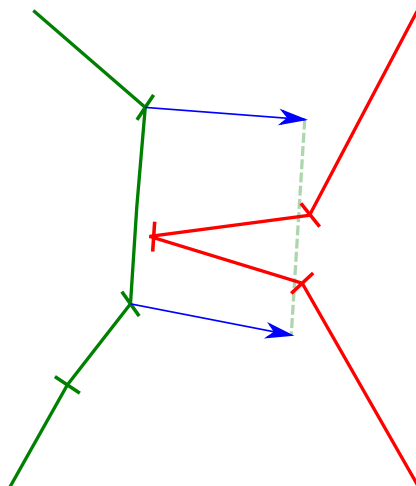
Obrázek 4.4: Detekce kolize dvou trojúhelníků.

Pokud tento test nezamítne kolizi, pak to znamená, že musí existovat přímka, která je průsečnicí obou rovin. Předchozí test také vynutil situaci, že tato přímka prochází oběma trojúhelníky. Na přímce lze nyní najít intervaly, ve kterých přímka leží uvnitř trojúhelníků – ty lze zjistit nejprve určením dvou hran každého trojúhelníka, které se protínají s přímkou (jsou to ty hrany, které vedou do bodu s odlišným znaménkem z rozřazovacího testu) a následným nalezením průsečíku dvou přímek v prostoru. Trojúhelníky kolidují, pokud je průnik intervalů pro oba trojúhelníky na přímce nenulový. Vše je vyobrazeno na obrázku 4.4. Z červeného a zeleného trojúhelníka se zjistí plochy, na kterých leží, poté se určí průsečnice (černá přímka). Na tuto přímku se vynesou intervaly, ve kterých protíná oba trojúhelníky. V tomto případě je průnik intervalů nenulový, což znamená že ke kolizi došlo. Minimálně z délky tohoto textu vyplývá, že detekce kolizí dvou trojúhelníků v euklidovském prostoru není triviální záležitostí. Proto se podíváme na různá zjednodušení, které urychlí výpočet.

### Kolize trojúhelníka a úsečky

Jednodušší možností je detekovat, zda-li existuje průsečík trojúhelníka a úsečky, než-li dvou trojúhelníků. V tomto případě nejprve určíme normálový vektor roviny na které leží trojúhelník a následně určíme průsečík úsečky s touto rovinou (neexistuje-li, pak objekty nekolidují). Nakonec je nutné zjistit, zda-li průsečík leží uvnitř trojúhelníka. Toho lze dosáhnout několika způsoby, jednou z možností je převést průsečík do barycentrických souřadnic tohoto trojúhelníka – kolize nastane pokud všechny tři souřadnice budou v ležet v intervalu  $(0; 1)$ .

Metoda pouze zaručí, že žádný bod pohybujícího se objektu neprojde do statického objektu, problémem však je, že to nic neříká o povrchu pohybujícího objektu. Problém je ukázán na obrázku 4.5. Oba zeleně vyznačené body se pohnou do svých cílových pozic, protože v jejich pohybu jim nic nebrání. Dojde však ke kolizi obou objektů částí trojúhelníku.

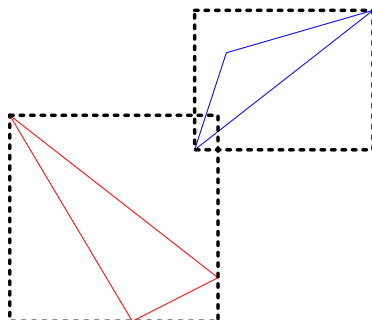


Obrázek 4.5: Problém zjednodušené kolize trojúhelníka a úsečky, objekty budou kolidovat (obrázek je v řezu, trojúhelníky jsou vizualizovány pomocí úseček).

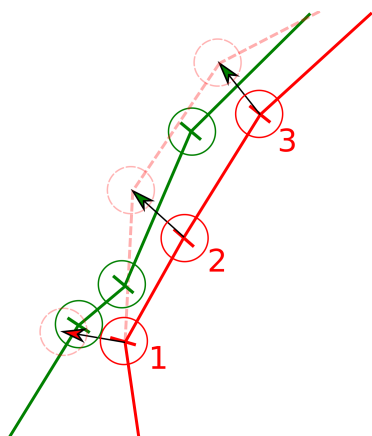
### Aproximační metody

Protože ani předcházející výpočet není úplně triviální (např. převod do barycentrických souřadnic), existují i aproximační metody, které výsledek pouze odhadují s nějakou přesností. Většina takových metod raději produkuje chyby 1. druhu, tedy falešně pozitivní kolize, než-li kolize falešně negativní. Příkladem může být například aproximace trojúhelníka nebo úsečky opsaným objektem – ve 3D například opsaný kvádr, jehož hrany jsou rovnoběžné s osami souřadnic (tzv. *bounding box* nebo také *AABB*) nebo například objekt opsaný koulí. Obě aproximace nebudou produkovat chyby 2. druhu, velmi často však budou produkovat chyby 1. druhu (viz obrázek 4.6, kde je problém zjednodušen do 2D).

Další možností je, že vytvoříme koule s dostatečně velkým poloměrem se středy ve všech vrcholech všech trojúhelníkových sítí. Poté stačí detekovat, jestli se neprotínají dvě koule. Takový test je velmi rychlý. Problémem je určení „dostatečného“ poloměru.



Obrázek 4.6: Problém kolize dvou trojúhelníků ve 2D pomocí tzv. *bounding-boxů*, kolize bude detekována, i když k ní nedošlo (chyba 1. druhu).



Obrázek 4.7: Chyba 2. druhu u aproximací koulemi. Dojde ke kolizi červeného a zeleného objektu, i když kolize mezi koulemi 2 a 3 detekována nebyla.

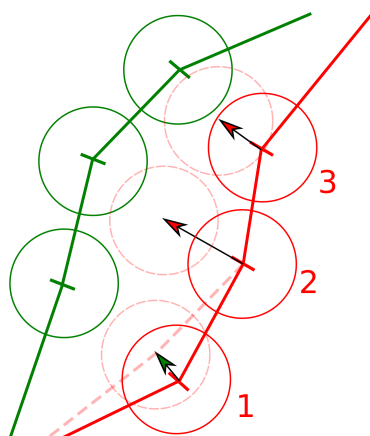
Je-li poloměr příliš malý, pak se lze dopustit chyby 2. druhu. To je vidět na obrázku 4.7, kde není detekována kolize koulí pro body 2 a 3, ale červený objekt projde do zeleného. Chyby 1. druhu vznikají velmi často z podstaty metody, viz obrázek 4.8. Ačkoliv by k přesunu dojít mohlo, červené body 2 a 3 si kvůli poloměru koulí budou držet dostatečný odstup od zeleného objektu.

Tato aproximace byla reálně použita v práci [16], situace je ilustrována obrázkem 4.9. V tomto případě však koule nebyly umístěny na povrchu (ve vrcholech) trojúhelníkové sítě, ale byly zanořeny směrem do středu objektu, což mělo řešit problém s chybovostí tohoto postupu.

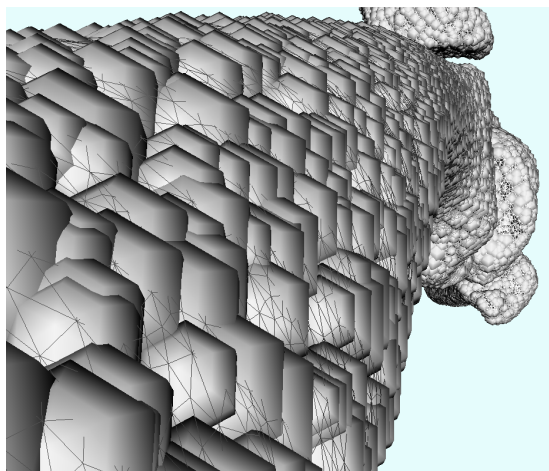
### 4.3 Reakce na kolizi

Poté, co se provede detekce kolize je potřeba provést takový opravný krok, aby ke kolizi pokud možno dále nedošlo. I v tomto případě existuje hned několik možností, jak tento problém řešit. Tyto možnosti lze rozdělit do dvou skupin podle toho, zda vyřeší problém okamžitě, nebo se k řešení dostanou až během několika dalších iterací.





Obrázek 4.8: Chyba 1. druhu u aproximací koulemi. K pohybu červených bodů 2 a 3 nedojde, i když by ke kolizi se zeleným objektem nedošlo.

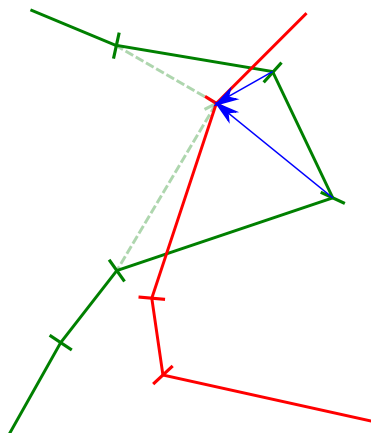


Obrázek 4.9: Nahrazení vrcholů trojúhelníkové sítě koulemi dostatečného poloměru [16].

### 4.3.1 Okamžitá oprava

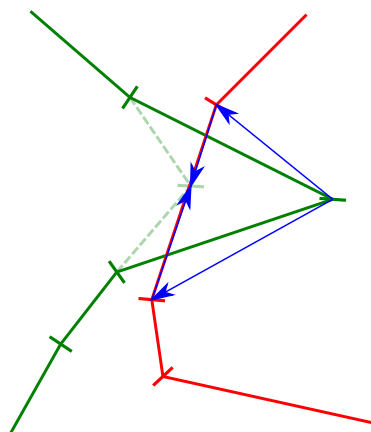
Kolize lze opravit okamžitě poté, co jsou detekovány. Jednou z nejjednodušších možností okamžité opravy je prosté nastavení souřadnic kolidujícího objektu (trojúhelníka, úsečky, ...) na souřadnice mimo objekt, například na některý z vrcholů druhého objektu (s případným posunem ve směru od objektu o nějaký konstantní krok). Problémů s tímto postupem je hned několik – několik bodů jednoho objektu se může namapovat na jediný bod objektu druhého, čímž dojde k zdegenerování (například trojúhelník takto může zdegenerovat do úsečky nebo dokonce do jediného bodu). Situace je ukázána na obrázku 4.10, kde zelený bod který se nejprve nacházel uvnitř červeného objektu je přesunut (dle modré šipky) na souřadnice nejbližšího vrcholu červeného objektu.

Lepší možnost než pouhé přiřazení na předem danou pozici je provedení nějakého druhu interpolace. Příklad uvedu na dvou kolidujících trojúhelnících.



Obrázek 4.10: Přesun dvou zelených bodů na jeden vrchol povrchu červeného objektu, čímž dojde k degradaci jedné hrany na bod.

Při opravě pozice bodu prvního trojúhelníka, který prošel skrz druhý trojúhelník se nyní vezme v úvahu nejen jeden cílový bod, ale celý druhý trojúhelník. Výsledná pozice pak může být promítnuta na druhý trojúhelník například za použití váženého průměru. Bod pak může být přemístěn kamkoliv na povrch trojúhelníkové sítě, nejen do vrcholů. Na obrázku 4.11 je opět uveden příklad ve 2D. Pro zelený bod, který se nachází uvnitř červeného objektu, se najdou dva nejbližší body. Podle obou vzdáleností se pak zelený bod umístí odpovídajícím způsobem na hranu mezi oba body. Pro jednoduchost zde byl uveden pouze příklad s váženým průměrem, různých způsobů interpolace existuje celá řada.

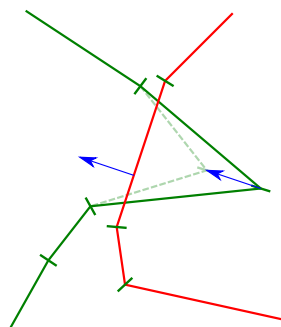


Obrázek 4.11: Přesun zeleného bodu na hranu červeného objektu dle vzdáleností ke dvěma bodům.

Existuje však problém s okamžitou opravou. Tím problémem je, že pokud se budou okamžitě a striktně opravovat kolize, všechny ostatní vlastnosti mohou být porušeny (zachování objemu, tvaru atd.). Toto lze uvážit tak, že použijeme metody, které neopraví kolizi hned, ale postupnými opravami.

### 4.3.2 Eventuální oprava

Striktní ošetření kolizí má svá úskalí (viz výše), proto existují i přístupy, které neopraví kolizi hned. Takové metody si lze představit například takovým způsobem, že pokud ke kolizi dojde, bude kolidující část pomalu vytlačována zpět silou určité velikosti.

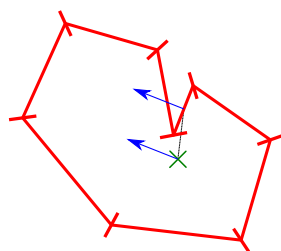


Obrázek 4.12: Vytlačení bodu zelené křivky pomocí modře vyznačené normály.

Pro příklad spolu budou opět kolidovat dva trojúhelníky. Nejjednodušší možností jak provést opravu je, že v každé iteraci posuneme bod prvního trojúhelníka o normálový vektor trojúhelníka druhého. V případě celých trojúhelníkových sítí tento bod „vytlačíme“ směrem k povrchu. Příklad viz obrázek 4.12 (zjednodušeno do 2D). Bod zelené křivky je vytlačen směrem z červeného objektu přičtením modré normály. Sice ani po přičtení není kolize ošetřena, ale je lépe zachován tvar zeleného objektu.

Lepší možností je provádět posun o nějaký násobek normálového vektoru. Násobek může být například přímo úměrný se vzdáleností kolidujícího bodu k povrchu, čímž se zajistí rychlejší „vytlačení“ hlouběji zanořených bodů.

Dále lze vzít v úvahu i více než jeden normálový vektor. Existují totiž případy, kdy volba pouze například normály nejbližšího trojúhelníka nevede k rychlé nápravě kolize. Situaci ilustruje obrázek 4.13, v tomto případě je zelený bod (pro jednoduchost) vytlačován z povrchu červeného objektu. Nejbližší normála však nesměruje ideálním směrem mimo objekt.



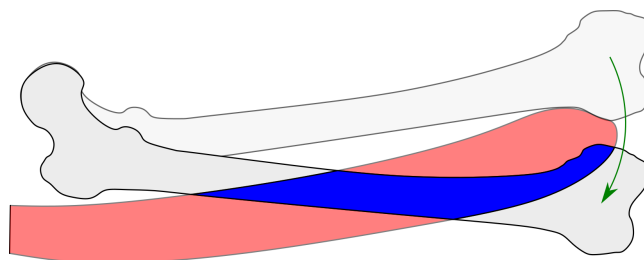
Obrázek 4.13: Problém výběru nejbližší (modré) normály, která posune zelený bod blíže středu červeného objektu.

Dále se podíváme na problém, který je pro modelování deformace svalů specifický, a to na problematiku pohybu kostí v okolí svalu.

### 4.3.3 Pohyb kostí

V úvodu této práce jsme se dozvěděli, že modelování probíhá v obráceném pořadí, než je tomu v realitě, tedy že se nejprve pohne kostí a až poté na tento pohyb reaguje sval. Oproti svalovým modelům se na kosti aplikuje pouze rigidní transformace (v ideálním případě, drobné reálně možné nerigidní transformace zde zanedbáme), což je značně jednodušší, než modelování výše uvedených nerigidních transformací. Dokonce tato transformace je i předem známa, proto není teoreticky žádný problém tuto transformaci provést.

Problémem však je, že pokud transformaci kosti provedeme, může se tím dostat část přilehlého svalu do kolize s touto kostí, bude nutné tedy i tuto kolizi detekovat. Co se týče reakce na tuto kolizi, kost na ni nesmí z podstaty věci reagovat, protože její pohyb je předem dán. Reagovat bude muset opět pouze sval. Příklad je ilustrován obrázkem 4.14. Při pohybu femuru se přilehlý sval dostane do kolize s touto kostí.



Obrázek 4.14: Kolize svalu a kosti způsobena pohybem kosti (extrémní případ).

### Reakce na kolizi s kostí

Výše uvedené metody opravy není možné přímo použít, protože model již na počátku detekce s kostí koliduje. Bude nutné tedy vymyslet jiný přístup.

Zde je několik možností, jak problém řešit. Opět jako v předcházejícím případě, kdy druhý objekt byl statický, lze provést opravu buď okamžitou, nebo eventuální.

Co se týče okamžitých oprav, zde je možné vymyslet mnoho přístupů. Mezi nejjednodušší patří nalezení nejbližšího bodu z množiny bodů specifikující trojúhelníkovou povrchovou síť kosti a s tímto bodem ztotožnit bod svalu. Tím se však může několik bodů svalu namapovat na jeden bod kosti, což není vhodné. Lépe je tedy provést interpolaci na povrch kosti, například nalezením nejbližšího bodu, který leží kdekoli na povrchu kosti. Problémem obou metod je, projede-li kost příliš hluboko do svalu. Příklad uvedu na obrázku 4.14. Zde se část kolidující modré části svalu namapuje na jednu stranu kosti a druhá část na druhou stranu. Červeně vyznačené části se zachovají. Sval se tedy prakticky rozpadne na dvě části.

Další možností je na detekované body aplikovat shodnou transformaci, jako byla použita na kost. Transformace by se ale použila pouze na body, jejichž počáteční bod leží uvnitř kosti, což značí, že kolize byla zapříčiněna pohybem kosti. Problém této metody je shodný s předešlými postupy.

Ze skupiny eventuálních oprav je možné provést vytlačování na povrch kosti ve směru nejbližším k povrchu. To by bylo možné zajistit například vektor-

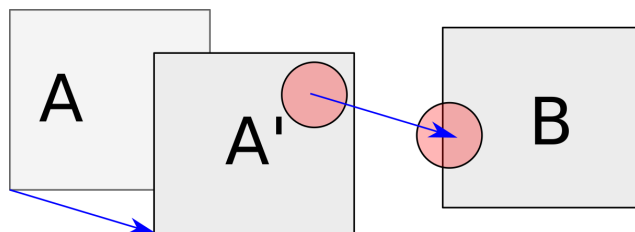
rovým polem, které by v kosti specifikovalo směr k povrchu. Metoda má opět stejný problém, pokud kost projde dostatečně hluboko, pak se sval rozdělí na několik částí. Problém však nebude tak signifikantní, protože k rozdělení nedojde okamžitě.

Samozřejmě zde je možné vymyslet mnoho řešení problému opravy kolize s pohybující kostí, proto jsou uvedeny pouze ta nejjednodušší.

### Pohyb více kostí

Problematika pohybu kostí v tomto případě nekončí. Pohybuje-li se více kostí v oblasti blízko svalu, nastávají další komplikace.

Problémem zde je, že ošetřením kolize s pohybující kostí se může stát, že dojde ke kolizi i s jinou kostí. Příkladem může být obrázek 4.15, kde je provedeno použití stejné transformace na sval, jako na kost. Po aplikování transformace se sval protne s kostí **B**. Tento problém může nastat pouze v případě, pokud je přítomno více kostí a alespoň jedna z nich se pohybuje.



Obrázek 4.15: Problém některých metod při opravě kolize s pohybující se kostí. Kost **A** při přesunu na pozici **A'** donutí sval, aby kolidoval s kostí **B**. Zde konkrétně jde o metodu aplikace shodné transformace.

Řešením by mohlo být spojení všech modelů kostí do jednoho, tím však ztratíme možnost snadno rigidně transformovat libovolnou z nich. Dalším řešením by mohlo být hledání kolize po cestě bodu interpolováním rigidní transformace. Interpolace rigidní transformace však přináší více problémů než užitek.

## 4.4 Dělení prostoru

Během detekce kolize je poměrně důležité zjistit, zda-li bude nutné kontrolovat všechny úsečky (pohyby všech bodů svalu) se všemi trojúhelníky okolních objektů (např. kostí) nebo jestli není možné nějakým způsobem například rychle vyřadit podmnožinu geometrických primitiv, které spolu kolidovat z podstaty věci nemůžou.

### 4.4.1 Brutální síla

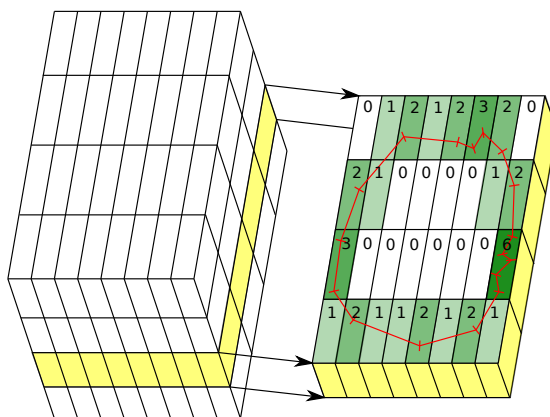
V dnešní době se v zásadě setkáme se třemi typy dělení prostoru. První, naivní metodou pro detekování kolize bez jakéhokoli dělení prostoru. Metoda detekce pak nedělá nic jiného než že testuje všechny dvojice trojúhelníků mezi sebou jestli spolu nekolidují. Kvůli tomu se musí provádět  $n$  nad dvěma kontrol, kde  $n$  je počet elementárních objektů, ze kterých je celek složen (trojúhelníky, body

apod.). Z toho plyne asymptotická složitost  $O(n^2)$ . Je pravděpodobné, že i pro běžná  $n$  nebude algoritmus běžet dostatečně rychle. Použití ale nalezneme v aplikacích, kde je menší  $n$  a režie sotsifikovanějších algoritmů se neamortizuje.

#### 4.4.2 Voxelizace

Druhou skupinou jsou postupy pracující se stejnou složitostí jako metoda brutální síly, snaží se však podělit hodnotu  $n$  nějakou konstantou tak, aby se špatná složitost projevila až déle. Postup může být takový, že se třídimenzionální prostor omezí na předem daný interval a ten se poté rozdělí do  $k^3$  „kvádrů“, kde  $k$  je konstanta nezávislá na  $n$ . Takovému přístupu dělení prostoru se pak říká voxelizace. Každému voxelu se pak například řekne, zda-li v něm leží nějaký objekt, kolik těchto objektů je, jaké jsou, nebo i jejich konkrétní výčet.

Při hledání kolizí pak lze určit a prohledat pouze kolidující voxely. Složitost jak již bylo naznačeno je  $O(n^2/k^3) = O(n^2)$ . Pro uchování informace o tom, kde který trojúhelník leží bude navíc třeba alokovat paměť. Tato paměť má lineární složitost, protože se odvíjí nejen od konstanty  $k$  ale také od hodnoty  $n$  ( $k$  je sice konstanta, se zvyšujícím se  $n$  však v průměru přibývá počet elementů v každém voxelu). Příklad je uveden na obrázku 4.16. V prostoru je uložen model, který se ohraničí kvádrem a tento kvádr se rozdělí na  $8 \times 4 \times 4$  podkvádrů. V každé buňce pak může být uložena informace například o tom, kolik trojúhelníků zde leží (čísla na obrázku), nebo i komplexnější informace o tom jaké trojúhelníky to jsou apod.



Obrázek 4.16: Rozdělení prostoru na podkvádry (vlevo), řez jednou vrstvou kvádrů (vlevo).

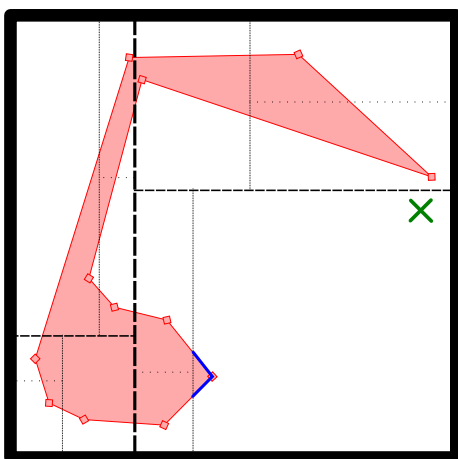
#### 4.4.3 Rekurzivní metody

Třetí část tvoří metody, které pracují s nižší výpočetní složitostí. Lze je identifikovat podle toho, že obvykle pracují se stromovou strukturou. Jako příklad uvedu nejjednodušší metodu BSP (binary space partitioning – dělení prostoru půlením). Myšlenka je taková, že rozdělíme zkoumaný prostor plochou tak, aby na každé straně plochy byl pokud možno stejný počet vrcholů (pro jednoduchost) detekovaného objektu. Vzniknou tím dva poloprostory, které se je možné

opět dělit na poloviny. Dělení je ukončeno, nachází-li se v daném podprostoru pouze jeden bod.

Prohledávání pak probíhá tak, že podrobíme druhý objekt stejnému dělení a zjistíme, do jakých elementárních podprostorů spadá. Složitost algoritmu je lineární ( $O(n \log_2 n)$ ), protože s logaritmickou složitostí procházíme podprostory pro  $n$  bodů. Bohužel shodná složitost bude i paměťová, protože ta závisí nejen na počtu bodů, ale také na hloubce vytvořeného binárního stromu.

Příklad stromu je uveden na obrázku 4.17, kde je pro jednoduchost problém převeden pouze do dvou dimenzí. Cílem je vytvořit strom nad červeně vyznačeným objektem. Nejprve se celý objekt ohraní kvádrem/obdélníkem a poté se začne vymezený prostor dělit. Je více kritérií podle kterých může být dělení provedeno, v tomto případě se dělí podle počtu bodů tak, aby byl strom co nejvyváženější. Dělení je ukázáno pomocí přerušovaných čar, tloušťka čáry odpovídá hloubce zanoření – nejvýraznější čára značí první dělení. Pro zjištění, jestli zeleně vyznačený bod koliduje nakonec stačí jen projít strom a zjistit, ve které buňce leží a provést kontrolu pouze s prvky objektu ve stejné buňce (modrá část křivky).



Obrázek 4.17: Dělení prostoru stromovou strukturou (konkrétně  $k$ -d stromem). Zeleně vyznačený bod se testuje pouze s modře vyznačenou částí celého objektu a to i přes to, že bod může ležet geometricky blíže k jiné části tohoto objektu.

Dále existují i metody s nižší složitostí, příkladem může být vzájemná detekce dvou objektů, když jsou oba objekty reprezentovány BSP (nebo jiným lineárním) stromem.

V této kapitole bylo mimo jiné ukázáno množství deformačních metod, metod detekcí kolizí a dělení prostoru s adekvátními rekacemi. Také zde byly ukázány i některé problémy, které s sebou popsání metody nesou. S uvážením těchto informací jsme nyní schopni zvolit vhodné řešení.

## Kapitola 5

# Zvolené řešení

V této kapitole bude popsán zvolený způsob řešení se všemi nutnými modifikacemi pro konkrétní problém deformace svalu.

Jako deformační algoritmus jsem zvolil PBD, který by měl odstranit nedostatky dříve použitých algoritmů [16]. Oproti původnímu algoritmu [18] bude nutné zohlednit i anizotropii svalů. Dále bude potřeba zjistit, který algoritmus detekce kolizí bude pro tento problém nejvhodnější použít.

### 5.1 Dostupné implementace

Tato sekce postupně prozkoumá již hotové implementace metody PBD, které by mohly být vhodné použít na problematiku modelování muskuloskeletárního systému.

#### 5.1.1 PositionBasedDynamics

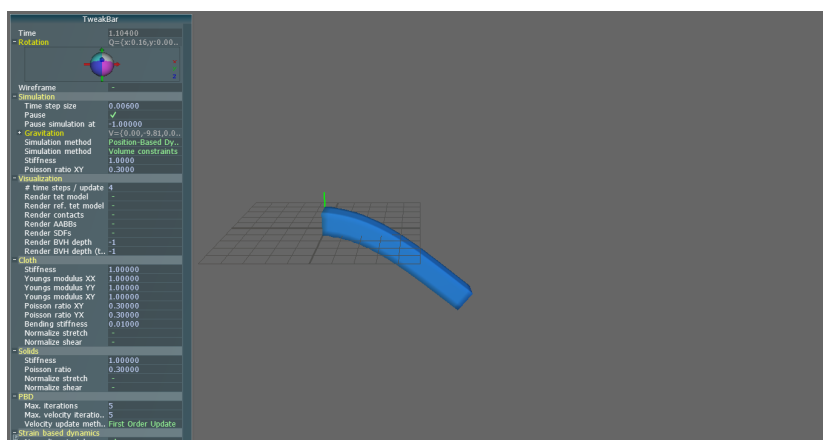
Existuje knihovna `PositionBasedDynamics`<sup>1</sup>, která provádí kompletní PBD deformaci. Knihovna je naprogramována v programovacím jazyce C++, takže by bylo možné tuto knihovnu snadno začlenit do již existujících projektů vyvíjených na Katedře. Knihovna je v licenci MIT, která by v rámci tohoto projektu mohla být použita. Tato knihovna obsahuje i několik testovacích scénářů, na kterých lze funkčnost otestovat. Po otestování několika scénářů se však ukázalo, že knihovna nedokáže provádět deformace v reálném čase, a to ani po degradaci kvality simulace za účelem jejího urychlení. Testovací scénáře měly také mnohem méně bodů, než je tomu v případě zde uvedených povrchových modelů svalů. Dostupný program také není stabilní a v některých případech se samovolně ukončí. Ukázka spuštěného programu je na obrázku 5.1.

Do knihovny by bylo nutné dále doprogramovat některé vlastnosti svalů, jakými je problematická anizotropie.

---

<sup>1</sup> J. Bender. `PositionBasedDynamics` <https://github.com/InteractiveComputerGraphics/PositionBasedDynamics> (2019)



Obrázek 5.1: Knihovna PositionBasedDynamics<sup>1</sup>.

### 5.1.2 NVFlex

Další knihovnou, která řeší mimo jiné i problematiku PBD je NVFlex<sup>2</sup>. Knihovna provádí výpočet PBD na grafické kartě, čímž dosahuje daleko většího výkonu než v případě předchozí knihovny PositionBasedDynamic, která vše počítá na CPU. Knihovnu vyvíjí společnost NVidia, a je dostupná pouze na jejich GPU (konkrétně je knihovna dostupná pro architekturu CUDA), dokonce jejich podmožině. Dalším omezením je, že knihovna je proprietární. Z těchto důvodů tato knihovna nebude použita, ačkoliv dle autorů je velmi rychlá a poskytuje realistické výsledky.

Obrázek 5.2: Knihovna NVFlex<sup>2</sup>.

Také do této knihovny by bylo nutné doprogramovat anizotropní vlastnosti svalů, které nejsou knihovnou v základu podporovány. Knihovna dle autorů dokáže vytvářet komplexní scény podobné té na obrázku 5.2.

<sup>2</sup>NVIDIA corporation. Knihovna NVFlex. <https://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/flex/>

### 5.1.3 Vlastní implementace

Z existujících řešení není ani jedno dostatečně vhodné. Případná úprava knihoven je buď složitější problém, než použít vlastní řešení, nebo je zakázána licencí. Proto bylo rozhodnuto o provedení vlastní implementace. Ta bude spočívat v implementaci níže uvedených omezení PBD algoritmu na vzdálenost bodů, lokální zachování tvaru, objemu, detekce kolize a vlastní implementaci simulace anizotropnosti modelu svalu. Nyní se však podíváme podrobněji na algoritmus PBD.

## 5.2 PBD algoritmus

Algoritmus PBD je uveden v pseudokódu 1. Nejprve se provede inicializace pozic všech bodů (například všech vrcholů trojúhelníkové sítě) na jejich počáteční pozici  $\mathbf{x}$ , nastaví se jejich počáteční rychlost  $\mathbf{v}$  a hmotnost (resp. její inverze  $w$ , je efektivnější počítat s inverzí).

Následuje hlavní nekonečná smyčka programu, kde se bodu provádět vnitřní iterace algoritmu. V té se nejprve upraví rychlost všech bodů uvážením vnějších sil působících na určité body (to je jediné místo algoritmu, které pracuje se silami). Dále je nutné uvážit, že v systému dochází i ke ztrátám způsobeným okolními vlivy. Ztráty lze simulovat například přenásobením vektoru rychlosti konstantou blížíící se zleva k jedné, např 0.99.

Pokračuje se provedením posunu bodů o zadanou rychlost. Posun nelze provést přímo do proměnné  $\mathbf{x}$  (pozice bodu), ale musí se nejprve spočítat do pomocné proměnné  $\mathbf{p}$  tak, aby bylo možné v dalších výpočtech znát jak původní tak i novou pozici.

Poté je provedena detekce kolizí a pokud daný pohyb bodu (z pozice  $\mathbf{x}$  na pozici  $\mathbf{p}$ ) způsobí kolizi, pak je tento pohyb označen jako nevalidní a přidán do seznamu omezení.

Následuje oprava všech omezení. Omezení jsou generována buď na začátku simulace (zachování objemu, tvaru, vzdáleností), nebo je generovala v této iteraci detekce kolizí.

Po nápravě všech omezení se zjistí nová rychlost všech bodů a nová pozice. Nakonec se modifikuje rychlost v závislosti na tření.

### 5.2.1 Pořadí optimalizace omezení

Na pořadí opravování omezení (viz řádky 15-17 Algorithm 1) je závislá rychlost konvergence tohoto algoritmu. V některých případech může algoritmus konvergovat velmi rychle, jindy pomaleji.

Představme si situaci, kdy je zachována vzdálenost (jak to lze provést viz další sekce) mezi body. Pro zjednodušení budou tyto body ležet na jedné přímce, jak je ukázáno na obrázku 5.3. Všechny body kromě posledního si zachovaly svoje původní pozice, poslední bod se posunul výrazně vpravo.

Začneme-li prohledávat zprava (na obrázku symbolizováno červeně označenými hranami), pak okamžitě nalezneme hranu, která má nesprávnou délku, proto oba body přiblížíme blíže k sobě, čímž provedeme opravu. Přejdeme-li na další hranu, tak je znovu co opravit. Po průchodu všech hran jednou není oprava dokonalá, ale je dostačující.

---

**Algorithm 1** PBD algoritmus [18], upraveno
 

---

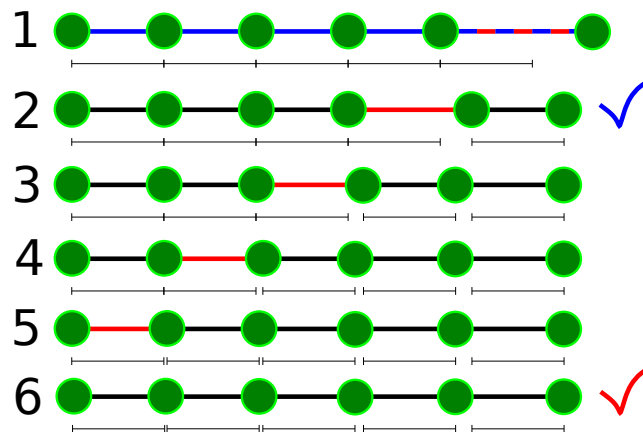
```

1: for all vertices  $i$  do
2:   initialize  $\mathbf{x}_i = \mathbf{x}_i^0$ ,  $\mathbf{v}_i = \mathbf{v}_i^0$ ,  $w_i = \frac{1}{m_i}$ .
3: end for
4: loop
5:   for all vertices  $i$  do
6:      $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t w_i \mathbf{f}_{ext}(\mathbf{x}_i)$ 
7:   end for
8:   dampVelocities( $\mathbf{v}_1, \dots, \mathbf{v}_N$ )
9:   for all vertices  $i$  do
10:     $\mathbf{p}_i \leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$ 
11:   end for
12:   for all vertices do
13:    generateCollisionConstraints( $\mathbf{x}_i \rightarrow \mathbf{p}_i$ )
14:   end for
15:   loop solverIterations times
16:    projectConstraints( $C_1, \dots, C_{M+M_{coll}}, \mathbf{p}_1, \dots, \mathbf{p}_N$ )
17:   end loop
18:   for all vertices  $i$  do
19:     $\mathbf{v}_i \leftarrow \frac{\mathbf{p}_i - \mathbf{x}_i}{\Delta t}$ 
20:     $\mathbf{x}_i \leftarrow \mathbf{p}_i$ 
21:   end for
22:   velocityUpdate( $\mathbf{v}_1, \dots, \mathbf{v}_N$ )
23: end loop

```

---

Představme si ale situaci, že prohledáváme zleva. Pro všechny kromě poslední hrany není třeba nic opravit, tzn. že po prvním průchodu se opraví pouze jedna hrana. Výsledek je tedy horší, použijeme-li tento směr prohledávání (na obrázku označeno modrými hranami).



Obrázek 5.3: Několik bodů ekvidistantně na přímce až na bod vpravo, který je posunut silou ještě více vpravo. Při prohledávání zprava doleva se dostane algoritmus až do kroku 6, v opačném případě pouze do kroku 2.

Na tento problém je třeba dávat pozor a volit počet iterací, který tento problém dostatečně minimalizuje, nebo zvolit adekvátní způsob prohledávání.

V následujícím textu se zaměříme na popsání jednotlivých omezení, konkrétně jak fungují a jaká je jejich matematická podstata.

### 5.2.2 Matematické pozadí

Základní matematická myšlenka skrytá za PBD je iterativní minimalizace penalizačních funkcí  $C$ , které budou definovány v dalších sekcích. Jedná se o funkce obecně  $n$  proměnných mající na výstupu skalární hodnotu, kterou se snažíme v optimálním případě vynulovat. Minimalizace probíhá metodou gradientního sestupu, je tedy nutné určit gradient funkce. Neuvažujeme-li hmotnosti bodů, pak změnu pozice bodu  $\Delta \mathbf{p}_i$  lze určit pomocí výpočtu 5.1.

$$\Delta \mathbf{p}_i = - \frac{\nabla_{p_i} C(\mathbf{p}_1, \dots, \mathbf{p}_n)}{\sum_j |\nabla_{p_j} C(\mathbf{p}_1, \dots, \mathbf{p}_n)|^2} \cdot C(\mathbf{p}_1, \dots, \mathbf{p}_n) \quad (5.1)$$

Tento vzorec lze jednoduše chápat jako posun proti směru gradientu (čitatel) o hodnotu penalizační funkce (člen za zlomkem). Derivace se musí dále normalizovat (jmenovatel) tak, aby byl směrový vektor posunu v rozmezí  $\langle 0; 1 \rangle$  a součet směrových vektorů všech parciálních derivací byl roven jedné.

Uvažujeme-li hmotnost bodů, pak místo jednoduchého sečtení všech posunů  $\Delta \mathbf{p}_i$  přes všechny penalizační funkce provedeme jejich vážený součet, kde váhami budou inverzní hodnoty hmotností bodů. Změna pozice je pak provedena v opačném směru (znaménko mínus) než směřuje derivace penalizační funkce tzn. provede se gradientní sestup.

Protože problém se bude skládat z mnoha penalizačních funkcí, jedná se o problém řešení soustavy nelineárních rovnic. Tato soustava bude řešena tak, že se budou jednotlivé rovnice řešit postupně s tím že se řešení všech rovnic provede několikrát po sobě, tím bychom měli konvergovat k cíli. Postup je podobný jako v případě Gauss-Seidelovy metody s tím rozdílem, že zde se nejedná o lineární rovnice.

### 5.2.3 Zachování vzdálenosti bodů

Zachování vzdálenosti mezi body je základní vlastnost, která by měla být v případě kvalitní deformace co nejvíce splněna.

Zachování vzdálenosti je provedeno velmi podobně jako v případě *mass-spring* systému (viz 4.1.3). Penalizace je nulová, pokud je vzdálenost mezi dvojicí bodů shodná s původní vzdáleností, pokud je větší nebo menší, penalizace roste.

Omezení je možné zapsat matematicky vztahem 5.2 [18]. funkce  $C$  je penalizační funkce dvou proměnných (vektorů), vracející skalár – hodnotu penalizace. Vektory  $\mathbf{p}_1$  a  $\mathbf{p}_2$  jsou dočasné pozice bodů (viz 1), hodnota  $d$  je původní vzdálenost mezi danou dvojicí bodů.

$$C(\mathbf{p}_1, \mathbf{p}_2) = |\mathbf{p}_1 - \mathbf{p}_2|_2 - d \quad (5.2)$$

V tomto okamžiku je nutné ještě určit gradient této funkce vzhledem k bodům  $\mathbf{p}_1$  a  $\mathbf{p}_2$ .

### Gradient normy rozdílu dvou vektorů

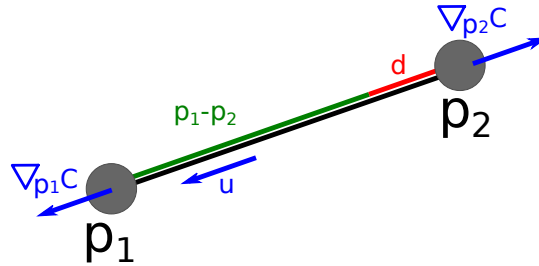
Nejprve bude potřeba určit gradient normy ze vzorce 5.2. Tu lze určit rozepsáním  $L_2$  normy, viz výpočet 5.3 pro gradient s respektem k bodu  $\mathbf{p}_1$ , pro  $\mathbf{p}_2$  je výpočet obdobný.

$$\begin{aligned}
 \nabla_{\mathbf{p}_1} |\mathbf{p}_1 - \mathbf{p}_2|_2 &= \nabla_{\mathbf{p}_1} \sqrt{(p_{1x} - p_{2x})^2 + (p_{1y} - p_{2y})^2 + (p_{1z} - p_{2z})^2} \\
 &= \frac{1}{2|\mathbf{p}_1 - \mathbf{p}_2|_2} \nabla_{\mathbf{p}_1} \left( (p_{1x} - p_{2x})^2 + (p_{1y} - p_{2y})^2 + (p_{1z} - p_{2z})^2 \right) \\
 &= \frac{\left[ \frac{\partial(p_{1x} - p_{2x})^2}{\partial p_{1x}} \quad \frac{\partial(p_{1y} - p_{2y})^2}{\partial p_{1y}} \quad \frac{\partial(p_{1z} - p_{2z})^2}{\partial p_{1z}} \right]}{2|\mathbf{p}_1 - \mathbf{p}_2|_2} \quad (5.3) \\
 &= \frac{2 \left[ \frac{\partial(p_{1x} - p_{2x})}{\partial p_{1x}} \quad \frac{\partial(p_{1y} - p_{2y})}{\partial p_{1y}} \quad \frac{\partial(p_{1z} - p_{2z})}{\partial p_{1z}} \right]}{2|\mathbf{p}_1 - \mathbf{p}_2|_2} = \frac{\mathbf{p}_1 - \mathbf{p}_2}{|\mathbf{p}_1 - \mathbf{p}_2|_2}
 \end{aligned}$$

Výsledný gradient penalizační funkce je s pomocí předešlého výpočtu určen v 5.4.

$$\begin{aligned}
 \nabla_{\mathbf{p}_1} C(\mathbf{p}_1, \mathbf{p}_2) &= \nabla_{\mathbf{p}_1} (|\mathbf{p}_1 - \mathbf{p}_2|_2 - d) = \frac{\mathbf{p}_1 - \mathbf{p}_2}{|\mathbf{p}_1 - \mathbf{p}_2|_2} = \mathbf{u} \\
 \nabla_{\mathbf{p}_2} C(\mathbf{p}_1, \mathbf{p}_2) &= \nabla_{\mathbf{p}_2} (|\mathbf{p}_1 - \mathbf{p}_2|_2 - d) = \frac{\mathbf{p}_2 - \mathbf{p}_1}{|\mathbf{p}_1 - \mathbf{p}_2|_2} = -\mathbf{u}
 \end{aligned} \quad (5.4)$$

Derivace těchto penalizačních funkcí jsou tedy směrové vektory hran (resp. jejich záporné hodnoty). Z podstaty věci tomu tak být musí, protože abychom změnil vzdálenost mezi body, je logické se pohybovat po směrovém vektoru. Výsledné veličiny jsou zobrazeny na obrázku 5.4.



Obrázek 5.4: Penalizace změny vzdálenosti. Gradient obou bodů je zobrazen modrými vektory.

#### 5.2.4 Zachování objemu

Další vlastností, které by bylo vhodné zachovat, je objem deformovaného svalu. Právě tato vlastnost chybí v navrženém systému *mass-spring* [16] a v případě modelování muskuloskeletárního systému se ukázala jako klíčová. I tak ale existují poměrně komplexní možnosti, jak do *mass-spring* systému zachování objemu [17].

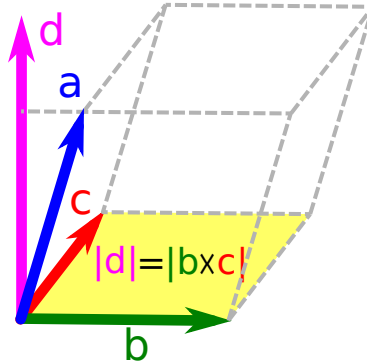
Nejprve je vhodné uvést, jakým způsobem se spočte objem manifoldní trojúhelníkové sítě. Problém se na první pohled jeví jako velmi složitý, s použitím vektorového počtu však lze problém vyřešit velmi elegantně.

Nejprve je třeba zvolit jeden libovolný bod, ze kterého budeme objem počítat. Obvykle se volí  $(0, 0, 0)$  nebo těžiště trojúhelníkové sítě. Z tohoto bodu budeme postupně počítat objemy tetrahedronů zformovaných z jednotlivých trojúhelníků a zvoleného bodu. Výpočet objemu tetrahedronu je uveden v 5.5. Pro vysvětlení tohoto vzorce se stačí podívat na tetrahedron jako jednu šestinu paralelogramu vytvořeného třemi hranami tetrahedronu sdílející jeden vrchol.

$$V_{tetra} = \frac{1}{6} \begin{vmatrix} a_x & b_x & c_x \\ a_y & b_y & c_y \\ a_z & b_z & c_z \end{vmatrix} = \mathbf{a} \cdot (\mathbf{b} \times \mathbf{c}) \quad (5.5)$$

Vektory  $\mathbf{a}$ ,  $\mathbf{b}$  a  $\mathbf{c}$  jsou hranami tetrahedronu sdílejícími jeden vrchol. V tomto případě je možné za společný bod zvolit počátek soustavy souřadnic. Pak se výpočet značně zjednoduší, protože vektory  $\mathbf{a}$ ,  $\mathbf{b}$  a  $\mathbf{c}$  je pak možno považovat za tři body  $\mathbf{p}$  jednoho trojúhelníka manifoldní sítě.

Příklad s těmito vektory je ukázán na obrázku 5.5. Vektory  $\mathbf{a}$  a  $\mathbf{b}$  ve vektorovém součtu vytvoří růžový vektor  $\mathbf{d}$  o délce shodné s obsahem žlutě vyznačené plochy. Následně skalárním součinem tohoto vektoru s vektorem  $\mathbf{c}$  získáme objem celého paralelogramu. Objem tetrahedronu tvořený zadanou trojicí vektorů pak tvoří šestinu objemu tohoto paralelogramu.



Obrázek 5.5: Výpočet objemu tetrahedronu tvořeného vektory  $\mathbf{a}$ ,  $\mathbf{b}$  a  $\mathbf{c}$ .

Z tohoto lze již určit penalizační funkci hodnotící změnu objemu. Vzorec je uveden v 5.6.

$$C(\mathbf{p}_1, \dots, \mathbf{p}_n) = \sum_{i=1}^m \left( \mathbf{p}_{t_1^i} \cdot (\mathbf{p}_{t_2^i} \times \mathbf{p}_{t_3^i}) \right) - kV_0 \quad (5.6)$$

Ve vzorci 5.6  $m$  značí počet trojúhelníků,  $\mathbf{p}_{t_i}$  označuje první vrchol trojúhelníka s indexem  $i$ . Hodnota  $V_0$  je počáteční objem, ke kterému se snažíme přiblížit, nakonec parametrem  $k$  lze definovat, zda-li se má objem zachovat  $k = 1$ , zmenšit  $k < 1$  nebo zvětšit  $k > 1$ . Pro tyto účely budeme uvažovat jen  $k = 1$ .

Derivaci této penalizační funkce lze provést buď z 5.6, nebo ze součtu determinantů v 5.5. Nejprve zvolíme jeden bod, pro který budeme derivaci počítat. Pro jednoduchost také zvolíme pouze jeden trojúhelník, pro který provedeme

derivaci (s využitím součtového pravidla). Výsledný gradient je uveden ve vzorci 5.7.

$$\nabla_{\mathbf{p}_i} C(\mathbf{p}_1, \dots, \mathbf{p}_n) = \begin{vmatrix} p_{ix} & p_{iy} & p_{iz} \\ p_{jx} & p_{jy} & p_{jz} \\ p_{kx} & p_{ky} & p_{kz} \end{vmatrix} + \dots = \mathbf{p}_j \times \mathbf{p}_k + \dots \quad (5.7)$$

Ze vzorce plyne, že derivace nad jedním trojúhelníkem je rovna vektorovému součinu zbývajících bodů trojúhelníka. Výsledná derivace daného bodu tedy bude součtem vektorových součinů dvojic ostatních bodů sdílející trojúhelník. To je vyjádřeno vzorcem 5.8.

$$\nabla_{\mathbf{p}_i} C(\mathbf{p}_1, \dots, \mathbf{p}_n) = \sum_{h=1}^t \mathbf{p}_j \times \mathbf{p}_k; \quad i \neq j \neq k \quad (5.8)$$

Proměnná  $t$  je počet trojúhelníků, které sdílejí bod s indexem  $i$ , body s indexy  $i$ ,  $j$  a  $k$  definují vždy stejný trojúhelník. Řídící proměnnou  $h$  v tomto případě vůbec nepoužijeme.

### 5.2.5 Zachování tvaru

Posledním důležitým omezením je zachování původního tvaru. To je řešeno zachováním úhlu mezi všemi dvojicemi sousedních trojúhelníků sítě.

Pro samotný výpočet je nejprve třeba uvést, jak se počítá úhel mezi dvěma sousedními trojúhelníky. Jsou známy čtyři body:  $\mathbf{p}_1$ ,  $\mathbf{p}_2$ ,  $\mathbf{p}_3$  a  $\mathbf{p}_4$ . Body s indexy 1 a 2 jsou sdíleny, body 3 a 4 leží pouze na jednom z dvojice trojúhelníků. Nejprve je potřeba určit normály obou trojúhelníků. To lze provést vektorovým součinem, viz 5.9.

$$\begin{aligned} \mathbf{u} &= (\mathbf{p}_2 - \mathbf{p}_1) \times (\mathbf{p}_3 - \mathbf{p}_1) \\ \mathbf{n} &= \frac{\mathbf{u}}{|\mathbf{u}|_2} \end{aligned} \quad (5.9)$$

Vektor  $\mathbf{u}$  je pomocný vektor, má stejný směr jako normálový vektor  $\mathbf{n}$  ale nemá jednotkovou délku. Body  $\mathbf{p}_1$ ,  $\mathbf{p}_2$  a  $\mathbf{p}_3$  leží na trojúhelníku, jehož normálu určujeme.

Pomocí tohoto výpočtu lze určit úhel mezi dvěma trojúhelníky. Nejprve určíme obě normály (například  $\mathbf{n}_1$  a  $\mathbf{n}_2$ ), pak z definice skalárního součinu lze určit úhel. Definice i s úpravami je uvedena ve výpočtu 5.10.

$$\begin{aligned} \cos \varphi &= \frac{\mathbf{n}_1 \cdot \mathbf{n}_2}{|\mathbf{n}_1|_2 |\mathbf{n}_2|_2} = \mathbf{n}_1 \cdot \mathbf{n}_2 \\ \varphi &= \arccos(\mathbf{n}_1 \cdot \mathbf{n}_2) \end{aligned} \quad (5.10)$$

Dále lze definovat penalizační funkci. Ta bude definována opět jako v předcházejících případech rozdílem nové a původní hodnoty, zde aktuálního a původního úhlu. Funkce je definována v 5.11.

$$\begin{aligned}
C(\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4) &= \\
&= \arccos(\mathbf{n}_1 \cdot \mathbf{n}_2) - \varphi_0 \\
&= \arccos\left(\frac{(\mathbf{p}_2 - \mathbf{p}_1) \times (\mathbf{p}_3 - \mathbf{p}_1)}{\|(\mathbf{p}_2 - \mathbf{p}_1) \times (\mathbf{p}_3 - \mathbf{p}_1)\|_2} \cdot \frac{(\mathbf{p}_2 - \mathbf{p}_1) \times (\mathbf{p}_4 - \mathbf{p}_1)}{\|(\mathbf{p}_2 - \mathbf{p}_1) \times (\mathbf{p}_4 - \mathbf{p}_1)\|_2}\right) - \varphi_0
\end{aligned} \tag{5.11}$$

V tomto výpočtu  $\varphi_0$  značí počáteční úhel (spočítaný pomocí 5.10).

Derivace v tomto případě bude složitější, než u ostatních omezení. V první řadě lze využít faktu, že omezení na tvar objektu je invariantní vůči posunu; lze například celou soustavu bodů přesunout tak, aby jeden z bodů měl nějaké vlastnosti, které usnadní výpočet.

Posuneme-li všechny čtyři body o  $-\mathbf{p}_1$ , vynulujeme tím  $\mathbf{p}_1$  a zároveň zjednodušíme výpočet normál obou trojúhelníků. Toto si lze dovolit, protože zachování tvaru by mělo být z podstaty invariantní vůči posunu celého objektu. Matematicky pak platí vztah 5.12, vektory s čárkami jsou posunuté vektory o  $-\mathbf{p}_1$ .

$$\begin{aligned}
\mathbf{n}_1 &= \frac{(\mathbf{p}_2 - \mathbf{p}_1) \times (\mathbf{p}_3 - \mathbf{p}_1)}{\|(\mathbf{p}_2 - \mathbf{p}_1) \times (\mathbf{p}_3 - \mathbf{p}_1)\|_2} = \frac{\mathbf{p}'_2 \times \mathbf{p}'_3}{\|\mathbf{p}'_2 \times \mathbf{p}'_3\|_2} \Leftrightarrow \mathbf{p}'_1 = 0 \\
\mathbf{n}_2 &= \frac{(\mathbf{p}_2 - \mathbf{p}_1) \times (\mathbf{p}_4 - \mathbf{p}_1)}{\|(\mathbf{p}_2 - \mathbf{p}_1) \times (\mathbf{p}_4 - \mathbf{p}_1)\|_2} = \frac{\mathbf{p}'_2 \times \mathbf{p}'_4}{\|\mathbf{p}'_2 \times \mathbf{p}'_4\|_2} \Leftrightarrow \mathbf{p}'_1 = 0
\end{aligned} \tag{5.12}$$

Nakonec se znalostí derivace  $\arccos$  ( $\frac{d}{dx} \arccos x = -\frac{1}{\sqrt{1-x^2}}$ ) lze získat předpisy pro jednotlivé gradienty penalizačních funkcí (výpočet 5.13). Bude zde záležet na tom, jaký bod má kterou roli, protože body  $\mathbf{p}_1$  a  $\mathbf{p}_2$  sice leží na jednom trojúhelníku, avšak pouze bod  $\mathbf{p}_1$  byl vynulován, proto se gradient podle tohoto bodu bude počítat jiným způsobem. V tomto případě totiž stačí pouze zajistit to, aby zachovávání tvaru neprovádělo posun celého objektu – tzn. výsledný posun čtveřice bodů musí být nulový.

$$\begin{aligned}
d &= \mathbf{n}_1 \cdot \mathbf{n}_2 \\
\nabla_{\mathbf{p}'_4} C &= -\frac{1}{\sqrt{1-d^2}} (\nabla_{\mathbf{p}'_4}(\mathbf{n}_2) \cdot \mathbf{n}_1) \\
\nabla_{\mathbf{p}'_3} C &= -\frac{1}{\sqrt{1-d^2}} (\nabla_{\mathbf{p}'_3}(\mathbf{n}_1) \cdot \mathbf{n}_2) \\
\nabla_{\mathbf{p}'_2} C &= -\frac{1}{\sqrt{1-d^2}} (\nabla_{\mathbf{p}'_2}(\mathbf{n}_1) \cdot \mathbf{n}_2 + \nabla_{\mathbf{p}'_2}(\mathbf{n}_2) \cdot \mathbf{n}_1) \\
\nabla_{\mathbf{p}'_1} C &= -\sum_{i=2}^4 \nabla_{\mathbf{p}'_i} C
\end{aligned} \tag{5.13}$$

Nyní jen stačí určit gradienty normál trojúhelníků dle ohraničujících bodů. Základní myšlenka za jejich vyjádřením je rozložení normály do vektorového součinu, který lze zapsat i maticově a každý prvek matice vyřešit samostatně. Matematicky je to vyjádřeno dále pro gradient normály  $\mathbf{n}_2$  dle bodu  $\mathbf{p}'_4$ . Provedeme tedy parciální derivaci normalizovaného vektorového součinu



### Gradient normalizovaného vektorového součinu

Pro tento výpočet si nejprve provedeme substituci vektorového součinu vektorem  $\mathbf{r}$  dle 5.14.

$$\mathbf{r} = \mathbf{p}'_2 \times \mathbf{p}'_4 \quad (5.14)$$

V dalším kroku spočítáme parciální derivaci nenormovaného vektorového součinu (viz 5.15). Vyjde matice  $\mathbf{A}$ , která se po vynásobení vektorem chová jako vektorový součin daného vektoru s vektorem  $\mathbf{p}'_2$ .

$$\begin{aligned} \nabla_{\mathbf{p}'_4} \mathbf{r} &= \begin{bmatrix} \frac{\partial p'_{2y} p'_{4z} - p'_{4y} p'_{2z}}{\partial p'_{4x}} & \frac{\partial p'_{2y} p'_{4z} - p'_{4y} p'_{2z}}{\partial p'_{4y}} & \frac{\partial p'_{2y} p'_{4z} - p'_{4y} p'_{2z}}{\partial p'_{4z}} \\ \frac{\partial p'_{4x} p'_{2z} - p'_{2x} p'_{4z}}{\partial p'_{4x}} & \frac{\partial p'_{4x} p'_{2z} - p'_{2x} p'_{4z}}{\partial p'_{4y}} & \frac{\partial p'_{4x} p'_{2z} - p'_{2x} p'_{4z}}{\partial p'_{4z}} \\ \frac{\partial p'_{2x} p'_{4y} - p'_{4x} p'_{2y}}{\partial p'_{4x}} & \frac{\partial p'_{2x} p'_{4y} - p'_{4x} p'_{2y}}{\partial p'_{4y}} & \frac{\partial p'_{2x} p'_{4y} - p'_{4x} p'_{2y}}{\partial p'_{4z}} \end{bmatrix} \\ &= \begin{bmatrix} 0 & -p'_{2z} & p'_{2y} \\ p'_{2z} & 0 & -p'_{2x} \\ -p'_{2y} & p'_{2x} & 0 \end{bmatrix} = \mathbf{A} \end{aligned} \quad (5.15)$$

Nyní určíme gradient normy skalárního součinu, výpočet viz 5.16. Výsledkem je opět vektorový součin, tentokrát však s normálovým vektorem.

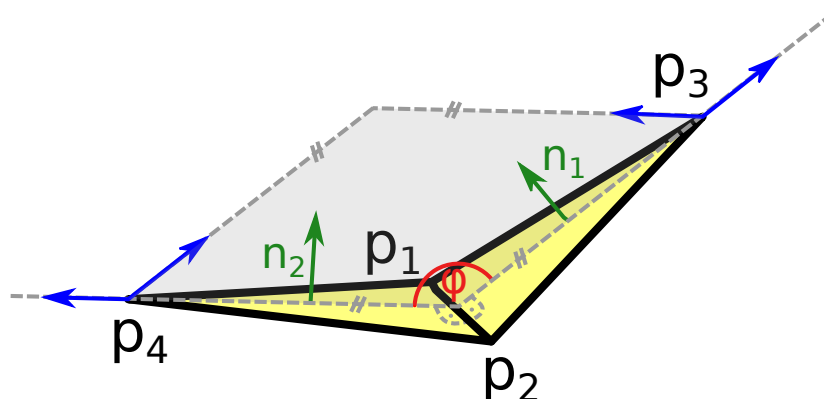
$$\begin{aligned} \nabla_{\mathbf{p}'_4} |\mathbf{r}|_2 &= \nabla_{\mathbf{p}'_4} \sqrt{r_x^2 + r_y^2 + r_z^2} = \frac{1}{2|\mathbf{r}|_2} \nabla_{\mathbf{p}'_4} (r_x^2 + r_y^2 + r_z^2) \\ &= \frac{1}{2|\mathbf{r}|_2} \begin{bmatrix} \frac{\partial r_x^2 + r_z^2}{\partial p'_{4x}} & \frac{\partial r_x^2 + r_z^2}{\partial p'_{4y}} & \frac{\partial r_x^2 + r_z^2}{\partial p'_{4z}} \end{bmatrix} \\ &= \frac{1}{|\mathbf{r}|_2} [r_y p'_{2z} - r_z p'_{2y} \quad -r_x p'_{2z} + r_z p'_{2x} \quad r_x p'_{2y} - r_y p'_{2x}] \\ &= \frac{\mathbf{r} \times \mathbf{p}'_2}{|\mathbf{r}|_2} = \frac{|\mathbf{r}|_2 \mathbf{n}_2 \times \mathbf{p}'_2}{|\mathbf{r}|_2} = \mathbf{n}_2 \times \mathbf{p}'_2 = \mathbf{b} \end{aligned} \quad (5.16)$$

Nezbývá než spojit obě parciální derivace z 5.15 a 5.16. Po použití pravidla o derivaci podílu získáme výsledek 5.17.

$$\begin{aligned} \nabla_{\mathbf{p}'_4} \frac{\mathbf{r}}{|\mathbf{r}|_2} &= \frac{\mathbf{A} |\mathbf{r}|_2 - \mathbf{r} \cdot \mathbf{b}}{|\mathbf{r}|_2^2} = \frac{1}{|\mathbf{r}|_2} (\mathbf{A} - \mathbf{n}_2 \cdot \mathbf{b}) \\ &= \frac{1}{|\mathbf{p}'_2 \times \mathbf{p}'_4|_2} \left( \begin{bmatrix} 0 & -p'_{2z} & p'_{2y} \\ p'_{2z} & 0 & -p'_{2x} \\ -p'_{2y} & p'_{2x} & 0 \end{bmatrix} - \mathbf{n}_2 \cdot (\mathbf{n}_2 \times \mathbf{p}'_2) \right) \end{aligned} \quad (5.17)$$

Obdobným způsobem by se pak vyjádřily i ostatní parciální derivace v 5.13, budou se prakticky lišit pouze indexy a znaménka. Výsledek parciální derivace dosadíme do vzorce 5.13 a tím určíme finální parciální derivaci penalizační funkce. Dále budeme postupovat stejně jako u ostatních omezeních – známe-li penalizační omezení a i její parciální derivaci, lze posun v každém kroku simulace provést dle vzorce 5.1.

Celou tuto situaci lze ukázat na obrázku 5.6, kde jsou pro jednoduchost znázorněny pouze dva trojúhelníky. Normály obou trojúhelníků jsou vyznačeny zeleně ( $\mathbf{n}_1$  a  $\mathbf{n}_2$ ). Gradient bodů  $\mathbf{p}_3$  a  $\mathbf{p}_4$  je vyznačen pro oba body dvěma vektory, jehož součtem vzniká (tzn. leží na ploše dané dvojicí vektorů). Z obrázku je zřejmé, že se oba tyto body mohou pohybovat pouze po rovině kolmé na oba trojúhelníky (šedá plocha), vychýlením bodu mimo tuto rovinu se úhel mezi trojúhelníky nezmění. Pro body  $\mathbf{p}_1$  a  $\mathbf{p}_2$  je gradient ještě složitější, proto nebyl zakreslen.



Obrázek 5.6: Zachování úhlu  $\varphi$  mezi dvěma trojúhelníky.

### 5.3 Detekce kolizí

Detekování kolizí hraje důležitou roli v celé simulaci. Nemělo by se během deformace stát, že například sval projde kostí.

Tomu lze zabránit zjištěním, zda-li na úsečce mezi body  $\mathbf{x}$  (předchozí pozice) a  $\mathbf{p}$  (nová pozice) nedojde ke kolizi s jinou trojúhelníkovou sítí. Počítání průsečku přímky (úsečky) a trojúhelníka je v tomto případě nezbytné. Druhou možností, jak detekovat kolize, je otestovat kolize vybraných dvojic trojúhelníků. V neposlední řadě existují i metody, které sice nedetekují kolize přesně, za to jsou ale rychlejší výpočetně.

Možnosti již byly podrobně popsány v předchozí kapitole, pro připomenutí lze mimo jiné detekovat kolize:

- dvojic trojúhelníků,
- trojúhelníka a úsečky,
- dvojici koulí,
- jinou aproximační metodou

Nejprve bylo rozhodnuto, že se použijí pro detekci primitiva trojúhelník a úsečka, protože poskytují vhodný kompromis mezi rychlou detekcí (například dvě koule) a přesností (dva trojúhelníky). Docházelo by tedy k problému uvedenému v sekci 4.2.1, testováním by se dále ukázalo, zda-li je tento problém vizuálně signifikantní či nikoliv.

Následně však byla detekce ještě více zjednodušena z důvodu urychlení výpočtu. Protože pro dělení prostoru byla použita voxelizace (zdůvodnění této volby viz sekce 5.3.3), bylo nutné do každého voxelu vložit informaci o všech trojúhelnících, které procházejí daným voxellem. Následné procházení tohoto seznamu a hledání přesného průsečíku bude poměrně časově náročné.

Proto se do každého voxelu uloží pouze informace, zda-li je daný voxel uvnitř nebo vně modelu, což značně sníží i paměťové nároky (teoreticky 1 bit na voxel). Následně se provede pouze detekce mezi voxellem a úsečkou, pokud úsečka koliduje s „vnitřním“ voxellem, pak je detekována kolize.

Nepřesnost této metody samozřejmě závisí hlavně na rozlišení voxelizace. Zvolené optimální rozlišení 64·64·64 voxelů se naštěstí ukázalo testováním jako dostatečné (viz dále).

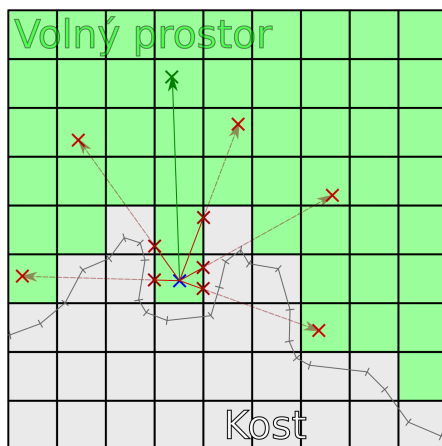
### 5.3.1 Směr detekce

Protože jako metoda dělení prostoru byla zvolena voxelizace a bude detekována kolize úseček a voxelů, budou se postupně procházet voxely, které úsečka protíná. Úsečka, na které se kolize detekuje vede dle PBD z bodu  $\mathbf{x}$  (předchozí pozice) do bodu  $\mathbf{p}$  (požadované pozice). Jedno z možných řešení je, že začneme procházet prostor z bodu  $\mathbf{x}$  s postupem do  $\mathbf{p}$ , pokud na cestě dojde ke kolizi, upravíme bod  $\mathbf{p}$  na pozici kolize. Druhou možností je provést procházení v opačném směru (tedy z bodu  $\mathbf{p}$  do bodu  $\mathbf{x}$ ) a v momentě, kdy se procházející bod nenachází uvnitř jiného objektu, ho prohlásíme za cílový bod  $\mathbf{p}$ .

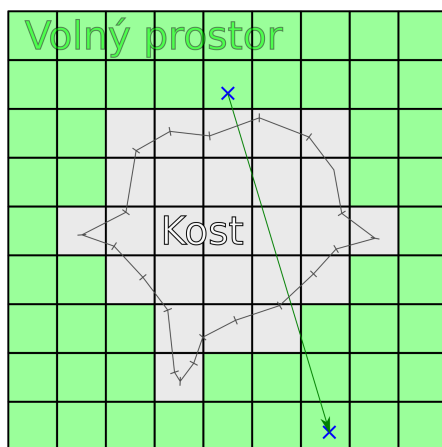
Směr z  $\mathbf{x}$  do  $\mathbf{p}$  samozřejmě více odpovídá realitě. Pokud se nějaký reálný objekt posouvá, pak samozřejmě narazí ve směru odkud míří. Problémem v tomto případě je však zvolené zjednodušení metody detekce kolize voxelu a úsečky. Může se totiž stát, že dvojice objektů se do sebe zaklesne. Problém je znázorněn na obrázku 5.7, kde se snaží modře označený bod pohybovat po povrchu objektu. Z důvodu hrubého povrchu objektu však bod uvízne a nemůže se dále pohybovat po povrchu.

Opačnou možností je se pohybovat v opačném směru. To by znamenalo, že by k uvíznutí nedocházelo, protože by se bod mohl pohnout do svého cíle, pokud přímo v cíli není detekována kolize, nikoliv po celé trase pohybujícího se bodu. V případě příkladu na obrázku 5.7 by to znamenalo, že bod by se mohl pohnout do libovolné z destinací. I tento přístup má však problém, bod v tomto případě může projít celým objektem, viz obrázek 5.8.

Největším problémem je zde určení prahu. Co je ještě povolený pohyb po povrchu? A co už je pohyb skrz objekt? Na to je těžké v tomto případě odpovědět, je-li znám pouze voxelizovaný povrch modelu. V této práci bude předpokládáno, že pohyb bodu nebude tak signifikantní, aby prošel napříč objektem, vše bude uvažováno jako pohyb po povrchu. Směr prohledávání je tedy z cílového bodu do počátečního.



Obrázek 5.7: Uvzlý modrý bod se nemůže pohybovat po povrchu kosti, protože naráží do sousedních voxelů hrubého povrchu voxelizovaného modelu. Má pouze málo možností se z tohoto uvíznutí dostat.



Obrázek 5.8: Modře vyznačený bod projde zkrz celý bílý model kosti, protože není testován žádný voxel na trase pohybujícího se bodu.

### 5.3.2 Další zjednodušení

Problém pohybu více kostí vzhledem ke komplexnosti nebude v této práci dále řešen. Provede se tedy transformace a bude se „doufat“, že sval při této operaci nevstoupí do jiné kosti.

Také v této práci nebude řešen problém vzájemné kolize více svalů a také problém sebezprotínání svalu. V případě sebezprotínání lze věřit například omezení pro zachování objemu nebo tvaru, že se bude výrazně eliminovat sebezprotínání.

### 5.3.3 Dělení prostoru

Prvním problémem je detekování kolize, konkrétně dělení prostoru. Původní algoritmus PBD ho uvádí jako obecný algoritmus dělení prostoru (*partial hashing*). V rámci této práce byly otestovány tři algoritmy.

#### Brutální síla

První zvolený algoritmus (který v podstatě není algoritmem dělícím prostor) je algoritmus brutální síly. Nebylo zprvu ani očekáváno, že by mohl být vhodný kandidát, ale alespoň bude sloužit jako měřítko pro další dva algoritmy.

#### Voxelizace

Dalším zvoleným algoritmem je voxelizace. Tento algoritmus je již dobře popsán v 4.2, dělí prostor na  $k^3$  kvádrů, kde  $k$  je konstanta, která je parametrem této metody.

#### Octree

Nejsložitějším algoritmem je algoritmus octree. Tento algoritmus postupně dělí prostor (kvádr) na osm stejně velkých podkvádrů. Strom vytvořený tímto algoritmem nebývá často optimálně vyvážen (není-li použit vyvažovací algoritmus), jeho sestavení je však relativně rychlé. Nevýhodou by mohla být jeho relativně velká spotřeba paměti.

K dělení dojde, pokud se v buňce již nachází  $k$  objektů (trojúhelníků, bodů apod.), tzn.  $k$  je parametr této metody.

#### Testování dělících algoritmů

Pro otestování zvolených metod dělení prostoru byl vytvořen prototyp programu, který dokázal pohybovat svalem v prostoru. V rámci těchto testů nebyla detekce nijak ošetřována, sval prostě procházel například kostí. Výhodou tohoto přístupu bylo zjištění, jak se ta která metoda bude chovat v krajních podmínkách.

#### Výsledky testování

V rámci testování byly zvoleny dva datasety. Jedním jsou reálná data svalu *gluteus maximus* (9878 vrcholů povrchového modelu svalu), kolidující s daty několika kostí (soustavou pánevních kostí spolu se stehenní kostí, 127211 resp.

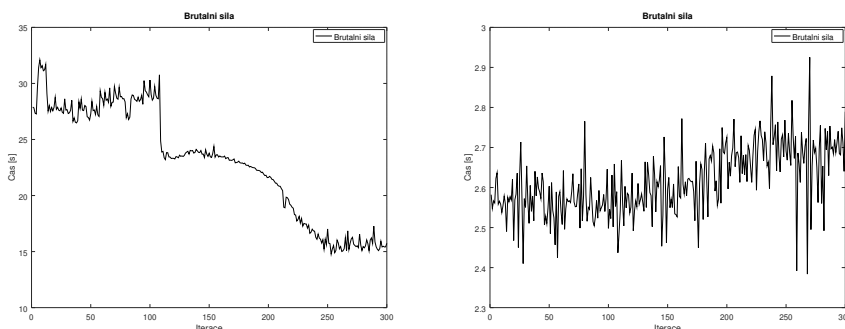
21230 vrcholů povrchových modelů kostí), druhou byla umělá data dvou kolidujících koulí (2210 vrcholů „svalu“, 89402 vrcholů „kosti“).

Tento test byl proveden na HW se specifikacemi uvedenými v tabulce 5.1. Pro informaci je uvedeno i GPU, i když se výpočet provádí pouze na CPU (paralelně).

CPU	Intel <sup>®</sup> Core <sup>™</sup> i3-5005U CPU @ 2.00GHz
RAM	4GB DDR3
GPU	Mesa DRI Intel <sup>®</sup> HD Graphics 5500 (Broadwell GT2)
HDD	128GB SSD

Tabulka 5.1: HW specifikace testovacího zařízení.

**Brutální síla** Výsledek brutální síly dopadl očekávatelně. Čas pro spočítání detekce jednoho snímku se pro umělá data pohyboval v rozmezí 2-3 sekundy, pro reálná byl dokonce v rozmezí 15-35 sekund. Výsledný graf je uveden na obrázku 5.9.



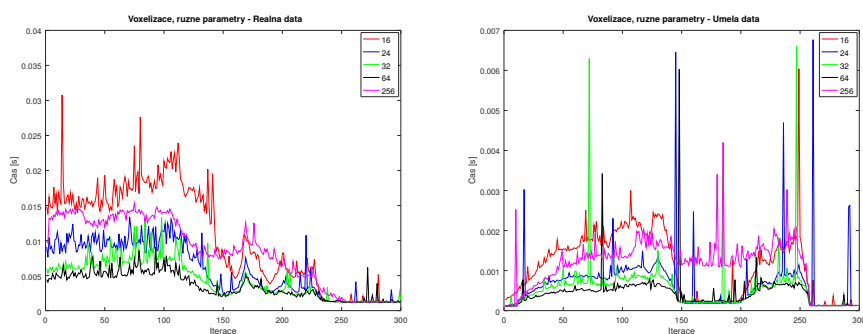
Obrázek 5.9: Doba detekce kolizí v jednotlivých iteracích. Vlevo pro reálná data, vpravo pro umělá data.

Z uvedených časů je zřejmé, že algoritmus není použitelný. Proto otestujeme další alternativy.

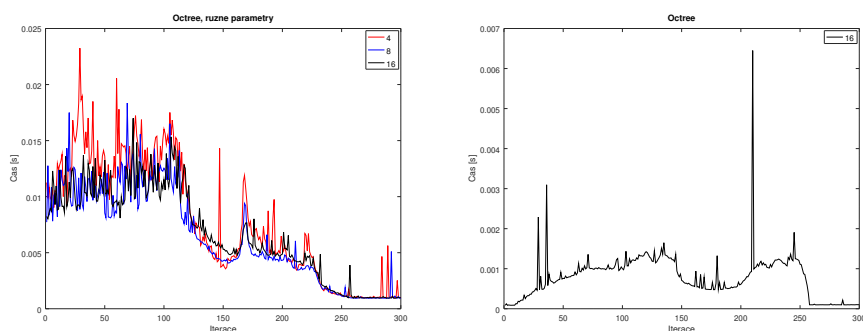
**Voxelizace** Před začátkem testování voxelizace bylo postupně zvoleno několik různých konstant  $k \in \{16, 24, 32, 64, 256\}$ . Následně byly měřeny časy výpočtu celé detekce kolize a výsledky mezi sebou porovnány. Výsledky jsou vyobrazeny na grafu 5.10.

Z těchto parametrů nejlépe dopadl  $k = 64$ , horší výsledky pro vyšší parametry jsou zapříčiněny velkým datovým objemem, na který se hůře přistupuje. Časy detekcí se však proti metodě brutální síly řádově snížili, jsou řádově v milisekundách, což lze použít pro reálnou simulaci.

**Octree** Poslední testovanou metodou dělení prostoru byl rekurzivní algoritmus octree. Opět bylo zvoleno několik parametrů  $k \in \{4, 8, 16\}$ , Výsledky jsou uvedeny na obrázku 5.11.



Obrázek 5.10: Doba detekce kolizí při dělení prostoru pomocí voxelizace. V jednotlivých iteracích. Vlevo pro reálná data, vpravo pro umělá data.



Obrázek 5.11: Doba detekce kolizí při dělení prostoru pomocí octree. V jednotlivých iteracích Vlevo pro reálná data, vpravo pro umělá data.

Algoritmus na první pohled vypadá, že by mohl běžet v reálném čase. Jeho problémem je však prostorová složitost. Pro umělá data nebyl algoritmus schopen alokovat paměť v RAM (4GB) pro parametry 4 a 8 a začal odkládat na disk. Proto čas těchto měření není uveden. I v ostatních měřeních případech byla spotřeba paměti vyšší. Z tohoto důvodu tento algoritmus zvolen nebude.

### Spotřeba paměti

V této sekci bude uvedeno, kolik který algoritmus použil maximálně paměti pro různé jeho parametry (má-li nějaké).

Pro měření byl použit nástroj *massif*, dostupný v programu *valgrind*. Tento nástroj dokáže analyzovat haldu a zásobník za běhu programu a dělat tzv. snapshoty, tedy snímky paměti v různých okamžicích vykonávání programu. Výsledky jsou uvedeny v tabulce 5.2.

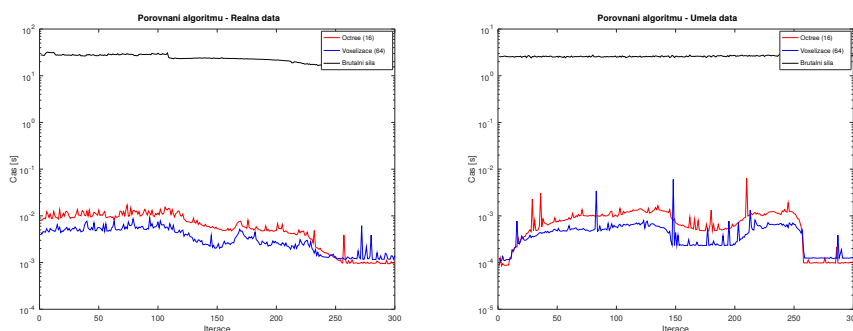
### Porovnání

Nyní porovnáme všechny tři algoritmy, přesněji řečeno jejich jejich verze s nejlepšími parametry. Nejlépe v testech dopadl voxelizační algoritmus, který běžel zhruba dvakrát rychleji než octree (a mnohonásobně rychleji než algoritmus

Algoritmus	Reálná data [B]	Umělá data [B]
Brutální síla	23.59 M	18.87 M
Voxelizace 16	42.08 M	21.97 M
Voxelizace 24	46.19 M	24.63 M
Voxelizace 32	54.03 M	25.14 M
Voxelizace 64	95.92 M	49.36 M
Voxelizace 256	1.90 G	946.29 M
Octree 4	1.58 G	>4 G
Octree 8	703.00 M	>4 G
Octree 16	431.88 M	1.56 G

Tabulka 5.2: Spotřeba paměti jednotlivých algoritmů nad použitými daty.

brutální síly). Výsledky jsou ukázány grafem 5.12. Voxelizace také nemá takové problémy se spotřebou paměti jako je tomu v případě octree. Proto tento způsob dělení volím jako vhodný pro další použití.



Obrázek 5.12: Porovnání nejlepších výsledků všech zvolených metod dělení prostoru. Vpravo pro reálná data, vlevo pak pro umělá data. Osa Y je v logaritmickém měřítku.

## 5.4 Anizotropie

Pro účely této práce je nutné zohlednit, že sval má různé vlastnosti v závislosti na směru působících sil (sval je tzv. anizotropní). Jednou z možností, jak takovéto vlastnosti simulovat je, že v případě *mass-spring* systému nastavíme každé „pružině“ různé konstanty tuhosti, v případě PBD lze přenásobit vzdálenostní penalizační funkci hodnotou  $k \in \langle 0; 1 \rangle$ . Takový parametr se pak bude chovat za určitých podmínek podobně jako parametr tuhosti v *mass-spring*.

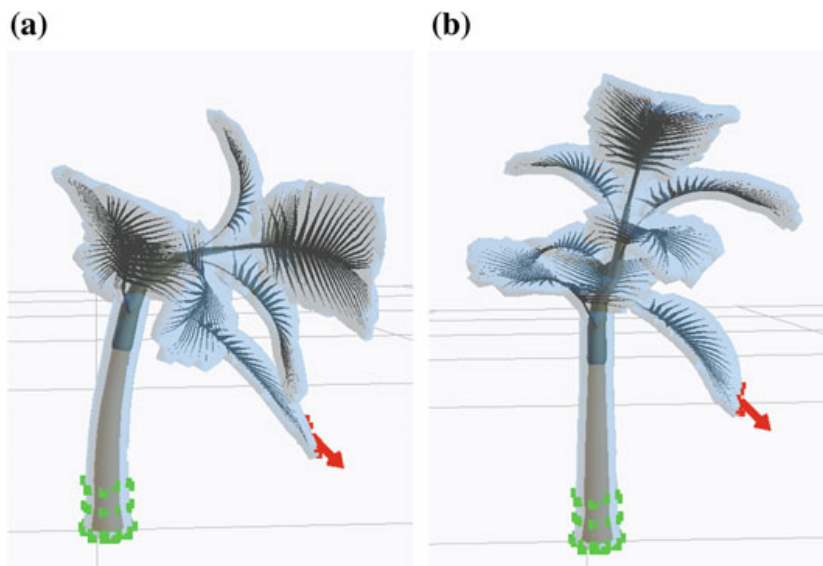
Dalším problémem je však jak tyto parametry určit. V práci [19] je ukázáno, jak lze problém podobného charakteru řešit. Bohužel autor předpokládá ortotropii která je podmnožinou anizotropních materiálů, často se jedná o materiál s paralelními vlákny (svaly toto obecně nesplňují, viz sekce 2). Dalším problémem je, že autor tuto metodiku používá s FEM (Finite element method – metoda konečných prvků), protože jsme však již zvolili PBD, bylo by nutné provést úpravy.



Lze se ale pokusit o nějaké vlastní řešení. Nejprve bychom mohli anizotropii ignorovat a následně zjišťovat, zda-li má její ignorování zásadní vliv. Další možností je, že pro každou hranu, která má zachovávat vzdálenost si určíme v předzpracování její úhel k nejbližší hraně lomených čar povrchových vláken (které v tomto momentě bude nutné znát), výše zvolený parametr  $k$  by pak mohl být pouze doplněk skalárního součinu směrových vektoru hrany a vlákna k jedničce, tedy dle vzorce 5.18, kde vektor  $\mathbf{u}_i$  je směrový vektor hrany s indexem  $i$  a vektor  $\mathbf{v}_i$  je směrový vektor vlákna. Doplněk do jedničky proto, že chceme ve směru vláken (kde se bude skalární součin směrových vektorů blízko jedné) povolit vyšší roztahování – tj. chceme nižší parametr  $k$ .

$$k_i = 1 - \mathbf{u}_i \cdot \mathbf{v}_i \quad (5.18)$$

Výpočet by bylo možné pak dále upravovat podle toho, jak moc je nutné anizotropii zachovávat. To by bylo možné například skalární součin umocnit vhodnou mocninou nebo i jinými možnostmi v závislosti na kvalitě výsledné deformace. Mocnitel by pak mohl být parametrem anizotropie.



Obrázek 5.13: (a) - deformace bez anizotropie, (b) - s uvážením anizotropie. Strom se v případě (b) ohne přirozeněji [20].

V tomto stádiu máme diskutovány všechny teoretické problémy, které by mohly stát v cestě otestování zvoleného způsobu deformace. V následující kapitole se tedy na samtonou implementaci detailně podíváme.

# Kapitola 6

## Implementace

Implementace byla provedena v jazyce *C++*. Tento jazyk byl doporučen vedoucím práce, protože ostatní podobně tématicky zaměřená výskumná činnost již byla v tomto jazyce na Katedře implementována. Pro vizualizaci výsledků byla použita knihovna *VTK* (Visualization Toolkit)<sup>1</sup> a to hlavně z důvodu snadného a rychlého použití. I přes to však byla implementace provedena tak, aby byla co nejméně závislá na této knihovně. Žádné jiné knihovny použity nebyly, což zajistí větší nezávislost programu na okolním software.

### 6.1 Třídy a moduly

V této sekci si uvedeme jednotlivé třídy a moduly, ze kterých je program komponován a naznačíme jejich základní účel.

- **main** – Tento modul má na starost spuštění celého programu. Načte vstupní data a ta ve formátu *VTK* předá ke zpracování dalším třídám.
- **data** – Třída **data** je v podstatě rozhraním mezi datovými strukturami knihovny *VTK* a vnitřními strukturami programu, stará se tedy o převod mezi těmito formáty. To umožňuje další nezávislost následujících tříd a modulů na knihovně *VTK*.
- **graph** – Tato třída poskytuje funkcionalitu pro práci s datovou strukturou grafu. Uchovává v sobě informace o vrcholech a hranách grafu, který je vytvořen z povrchových sítí. Dále je rozšířen o možnosti práce s povrchovými sítěmi, tzn. například uchovává a pracuje se stěnami, počítá *bounding-box* grafu apod.
- **point\_3D** – Třída poskytující metody pro práci s body a vektory v trojrozměrném prostoru. Veškeré operace, které je nutné často a opakovaně využívat (vektorové operace typu součet, rozdíl apod.) jsou napsány pomocí *maker*, aby byly rychle vykonány. V případě metod/funkcí se musíme spolehat na překladač, že toto zajistí. Dále pro každý bod uchovává informace potřebné pro algoritmus PBD, tedy počáteční a koncovou pozici, rychlost pohybu apod.

---

<sup>1</sup>Kitware, Inc. The visualization Toolkit. <https://vtk.org>

- `pbd` – Tato třída je hlavním jádrem programu, provádí totiž výpočet deformace pomocí PBD. Implementuje tedy celý algoritmus a počítá všechny penalizační funkce, kromě detekce a reakce na kolize, které zasloužilo kvůli komplexnosti zařadit do vlastní třídy.
- `collision` – Tato třída poskytuje základní funkcionalitu, kterou potřebuje většina metod detekcí kolizí, poskytuje také rozhraní pro zvolenou metodu voxelizace.
- `voxel` – Tato třída má za úkol dělení prostoru, detekci kolize a následnou opravu nad každým ze svalových modelů.

### 6.1.1 Správa paměti

Pro dynamicky alokovanou paměť bylo rozhodnuto, že se využijí prostředky pouze tradičního C++ (ISO/IEC 14882:1998), tedy že nebude řešena pomocí tzv. smart pointerů. Prvním důvodem je, že kód bude přeložitelný i pomocí starších překladačů, dalším důvodem je, že smart pointery by v tomto případě zavedly až zbytečně vysokou úroveň abstrakce, kterou není příliš vhodné zavádět v aplikacích, které se mají vykonávat rychle. V případě komplexnějších datových struktur (např. `std::vector` apod.), které jsou v C++ 98 přítomny, je vhodnější samozřejmě použít standardní knihovny, v případě jednodušších alokací je však vhodnější se o alokaci a dealokaci postarat tzv. ručně, než-li se spoléhat na jiné mechanismy.

### 6.1.2 Formát a načítání dat

Vstupní data modelů povrchových sítí jsou uloženy ve formátu VTK a jsou k programu i přiloženy. Formát těchto souborů je poměrně jednoduchý. Na první řádce najdeme komentář, který specifikuje verzi datového souboru. Následuje řádka, na které se může nacházet libovolný (maximálně 256 znaků dlouhý) text, například popis o jaká data se jedná. Následující řádka říká, v jakém kódování jsou dále uvedena data. Možnosti jsou buď RAW, kdy budou data ukádána binárně, nebo ASCII, kde jsou data ukládána v uživatelsky čitelné podobě. Příklad, jak může vypadat tetrahedron, je uveden v následujícím výpisu 6.1.

Na další řádce se nachází typ dat. Program akceptuje pouze data typu POLYDATA. Na další řádce je řečeno, z kolika bodů je objekt složen a jakého datového typu, po čemž následuje výčet těchto hodnot. Na konec je v případě dat o povrchové síti uvedeno, z kolika polygonů (POLYGONS) je povrch vytvořen (a kolik paměťového místa je třeba na uchování všech indexů), následováno samotným seznamem polygonů. Program akceptuje pouze trojúhelníky, tedy polygony o třech bodech. V případě dat o svalových vláknech k modelování anizotropie je místo sekce POLYGONS uvedena sekce LINES, která specifikuje lomené čáry, kudy přibližně vedou svalová vlákna.

Data jsou pak načtena knihovnou VTK, která všechny body interně uloží do jednoho pole, kam se body poskládají za sebe ( $x_1, y_1, z_1, x_2, y_2 \dots$ ). Informace o polygonech taktéž uloží do jednoho pole, kam si uloží vždy velikost polygonu a pak indexy do pole bodů, například dva polygony, jeden čtverec spojující body 1 4 5 a 12 a trojúhelník spojující body 2 8 a 12 by byly uloženy v poli v pořadí 4, 1, 4, 5, 12, 3, 2, 8, 12.

```

# vtk DataFile Version 2.0
Příklad tetrahedronu
ASCII
DATASET POLYDATA
POINTS 4 FLOAT
0.9428090415820634 0 -0.3333333333333333
-0.4714045207910317 0.816496580927726 -0.3333333333333333
-0.4714045207910317 -0.816496580927726 -0.3333333333333333
0 0 1
POLYGONS 4 16
3 0 2 1
3 0 1 3
3 0 3 2
3 1 2 3

```

Obrázek 6.1: Příklad *VTK* datového souboru.

Data z výpisu 6.1 by tedy byla uložena v případě bodů ve formátu:

```
0.94 0 -0.33 -0.47 0.82 -0.33 -0.47 -0.82 -0.33 0 0 1
```

data o polygonech/trojúhelnících pak ve formátu:

```
3 0 2 1 3 0 1 3 3 0 3 2 3 1 2 3
```

Po načtení dat ze souborů se spustí samotný výpočet. Uživateli se otevře okno, ve kterém bude schopen dynamicky měnit parametry simulace (parametry PBD). Uživatel uvidí za optimálních podmínek aktuální stav všech zvolených povrchových modelů. V demonstrační aplikaci si však uživatel nebude schopen provést vlastní rigidní transformaci kosti, scénář pohybu musí být vytvořen pouze programově.

Na pozadí mezitím poběží simulace. Simulace běží výhradně na CPU, v kritických místech i multivláknově (za pomoci OpenMP). Hodnotu FPS (počet snímků za sekundu) je možné vidět v běžící simulaci v levém horním rohu. Tato hodnota lze také možné zhora omezit (např. na maximální hodnotu 30 FPS) makrem `FPS_LIMIT`, což zajistí plynulejší simulaci a ušetří výkonnostní zdroje, to však za cenu rychlosti simulace. V programu bylo také implementováno měření času jednotlivých operací, například detekce kolizí, reakce na kolize a jiná omezení. To je možné aktivovat makrem `TIME_MEASUREMENT`. Pro otestování jednotlivých omezení je možné i některá z nich deaktivovat, to lze provést nastavením jedné z proměnných `m_no_edge`, `m_no_volume`, `m_no_bending`, `m_no_collision` resp. `m_no_anisotropy` třídy `pb`.

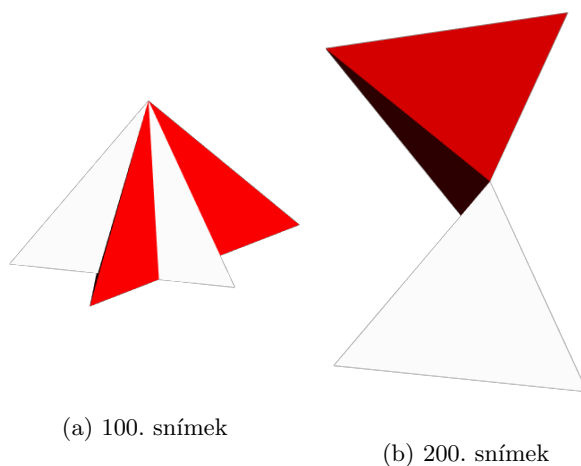
## 6.2 Základní testy funkčnosti

Pro základní otestování jednotlivých omezení PBD algoritmu byly navrženy jednoduché (jednotkové) testy, na kterých je možné empiricky ověřit funkčnost jednoho omezení. Ostatní omezení jsou zakázány. Z tohoto důvodu byly vytvořeny dva tetrahedrony (jeden z nich je uveden ve výpisu 6.1), protože se jedná o simplex, tzn. nejjednodušší tvary v 3D. Tetrahedrony byly také navrženy jako

pravidelné pro další zjednodušení. V následujících testech budu hovořit o dvou tetrahedronech, proto je rozliším tak, že jeden bude „reprezentovat“ sval a druhý kost. Všechny následující testy budou přiloženy i na CD. Jakým způsobem je lze spustit je uvedeno v příloze A.

### 6.2.1 Test zachování vzdáleností

Při tomto testu jsou oba tetrahedrony spojeny ve dvou vrcholech. Tím získáme prakticky pouze jeden stupeň volnosti. Na „svalový“ tetrahedron je poté působeno silou v jednom z dvojice volných vrcholů. Cílem je, aby svalový tetrahedron prováděl kyvadlový pohyb s respektováním obou bodů uchycení. Výsledek toto testu je uveden na snímku 6.2.



Obrázek 6.2: Test zachování vzdáleností mezi body.

Tetrahedron v testu vizuálně zachovává vzdálenosti mezi body, proto lze přistoupit k dalším testům.

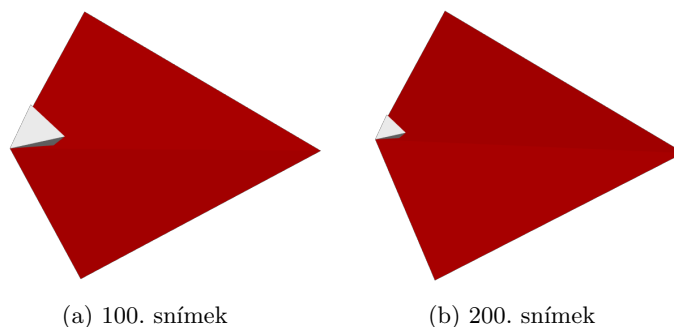
### 6.2.2 Test zachování tvaru

Dalším testem je test zachování tvaru. Tetrahedrony v tomto případě budou ukotveny navzájem pouze jedním vrcholem, aby „svalový“ tetrahedron měl více volnosti. Poté se začne působit silou na jeden z vrcholů, přičemž je použito pouze omezení na tvar.

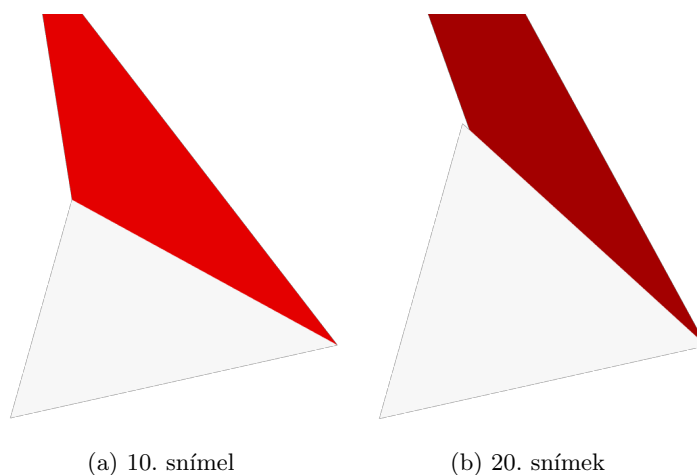
Tetrahedron tedy nemá jinou možnost, než aby se začal zvětšovat, protože to je jediná možnost, jak zachovat v tomto případě tvar (viz obrázek 6.3). Nyní přejdeme na poslední test, test objemu.

### 6.2.3 Test zachování objemu

Posledním individuálním testem je test na zachování objemu tetrahedronu. Tetrahedron reprezentující sval je nyní uchycen opět dvěma body, na třetí působí síla. Tento příklad je na obrázku 6.4.



Obrázek 6.3: Test zachování tvaru tetrahedronu. „Svalový“ tetrahedron se škáluje, protože je to jediná možnost, jak zachovat tvar, pokud je tetrahedron jedním bodem ukotven a na druhý působí síla.



Obrázek 6.4: Test zachování objemu tetrahedronu. „Svalový“ tetrahedron postupně mění tvar a konverguje k ploše, protože na obrázku horní bod je vzdalován od ostatních působící silou.

Správné chování je takové, že tetrahedron se bude postupně měnit a nekonečně se blížit svým tvarem k ploše, aby zachoval svůj konstantní objem. A právě toto tetrahedron činí.

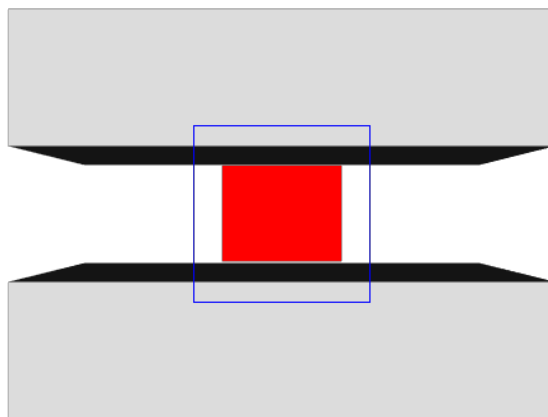
Toto byla minimální sada jednotkových testů, která otestovala jednotlivá omezení. V následující sekci je integrujeme dohromady a přidáme detekci kolizí spolu s anizotropií. Také se pokusíme zvolit vhodné parametry těchto omezení pro konkrétní aplikace.

## Kapitola 7

# Dosažené výsledky

V textu následující sekce se podíváme na výsledky zvolené metody deformace svalového modelu. Nejprve si musíme ujasnit parametry, kterými lze deformaci řídit. Jako testovací data použijeme uměle vytvořená data dvou ploch, které se přiblíží a oddálí, čímž se stlačí sval umístěný mezi plochami. Mezera mezi plochami se zmenší až na 10% původní délky, což dostatečně otestuje schopnost algoritmu napravit sval do původního stavu. Ve všech zde uvedených případech se plochy již přiblížili na minimální vzdálenost a už se vracejí zpět. Pro jednoduchost budeme dále zanedbávat vzájemné vazby mezi parametry, i když samozřejmě existují.

Počáteční scéna bude vypadat jako na obrázku 7.1. Parametry (nebude-li uvedeno jinak) budou nastaveny na hodnoty z tabulky 7.6 (tedy stejné jako pro dále uvedená reálná data).



Obrázek 7.1: Testovací scénář „slisování“ svalu mezi dvě desky. V dalších testech bude ukázán již pouze oblast modrého výřezu.

## 7.1 Volba parametrů



















Uživatel má možnost za běhu simulace dynamicky měnit parametry PBD. Těmito parametry jsou počet iterací během jednoho snímku (*Iteration/frame*), nastavení „přísnosti“ omezení zachování tvaru (*Bending*), nastavení setrvačnosti bodů (*Velocity damp*), vnitřního tlaku ve svalu/objemu (*Pressure*), hodnoty gravitace (*Gravity*) a nastavení anizotropie (*Anisotropy*). Poslední posuvník (*Pause*) umožňuje zastavit simulaci přesunem na hodnotu menší než 0.5. Nyní se podíváme postupně na jednotlivé parametry, jakým způsobem jejich hodnoty ovlivňují simulaci.

V následující sekci se podíváme na počet iterací a jaký má vliv na simulaci. V první řadě je však nutné vymezit pojem „iterace“, protože se zde setkáváme s problémem dvojích iterací. Jedny iterace jsou vnitřní iterace algoritmu, mezi kterými se objekt stabilizuje a v této době jsou vnější síly konstantní. Pak jsou zde ještě vnější iterace, to jsou iterace, mezi kterými se působící vnější síla může měnit. Dále v textu práce budou vnitřní iterace označovány pouze jako „iterace“, vnější iterace budou označovány jako „snímky“, protože jedna vnější iterace odpovídá jednomu snímku vizualizované simulace.

### 7.1.1 Počet iterací

Počet iterací udává, kolikrát dokola se všechna omezení vykonají, aby se objekt stabilizoval v rámci jednoho snímku. V GUI programu je možné tuto hodnotu nastavit v rozsahu 1-100. Čím nižší hodnota je, tím se simulace jeví uživateli plynulejší. Naopak pro vysoké hodnoty klesá FPS, jenže mezi snímky se provede mnohem přesnější výpočet, protože není zdržován překreslováním. Mezi jednotlivými snímky se také aplikují vnější síly působící na sval (např. gravitace), to je vlastnost kterou je také nutné při nastavení parametrů zvážit.

Výsledky pro některé ukázkové parametry počtu iterací jsou uvedeny na následujícím obrázku 7.1.

Snímek:	1	50	100	150	200	300
1 iterace:						
3 iterace:						
10 iterací:						

Tabulka 7.1: Výsledky pro různý počet iterací v různých snímcích simulace.

### 7.1.2 Zachování tvaru

Dalším parametrem, který lze nastavit, je přesnost zachování tvaru. Hodnoty se pohybují v intervalu  $(0; 1)$ , přičemž hodnota 0 znamená nulové zachování tvaru, 1 co nejlepší možné.



Snímek:	1	50	100	150	200	300
<i>Bending=0:</i>						
<i>Bending=0.5:</i>						
<i>Bending=1:</i>						

Tabulka 7.2: Chování svalu při různých úrovních zachování tvaru.

Výsledek pro některé parametry je zřejmý z obrázku 7.2. Pro parametr 0 zafungovala jiná omezení (vzdálenostní a objemové), které dokázaly nepřítomnost zachovávání tvaru částečně nahradit.

### 7.1.3 Setrvačnost

Parametrem setrvačnosti lze nastavit úbytek rychlosti mezi iteracemi. Účel tohoto parametru je simulovat ztráty. Hodnota je opět v intervalu  $\langle 0; 1 \rangle$ . Touto hodnotou se přenásobí rychlost po každé iteraci simulace. Nastavením příliš vysokého parametru bude docházet k zbytečným pohybům a povrch svalu se rozvlní. V opačném případě bude algoritmus konvergovat k optimálnímu tvaru pomaleji.



















Výsledky pro různé parametry úbytku jsou vidět na obrázku 7.3.

Snímek:	1	50	100	150	200	300
Úbytek=0.2:						
Úbytek=0.5:						
Úbytek=1:						

Tabulka 7.3: Různé hodnoty úbytku rychlosti.

### 7.1.4 Tlak


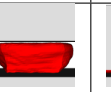

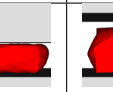
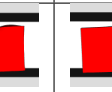


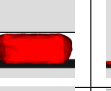

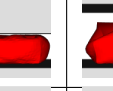
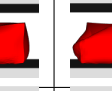


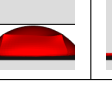




Dalším parametrem, který lze nastavit je vnitřní tlak ve svalu. Pochopitelně zvyšování a snižování tlaku ve svalu prakticky nedává smysl, uživatel ale přesto má možnost otestovat, jak by se sval choval při změně jeho objemu (např. „nafukování“). Parametr tlaku je možné volit z intervalu  $\langle \frac{1}{2}; 2 \rangle$ , přičemž hodnota 1 je normální tlak, a vyšší hodnota je přetlak. V dalším testování již tento parametr bude ignorován, stejně jako parametr na pozastavení simulace. Pro ukázkou jsou opět ukázány výsledky na obrázku 7.4.

Snímek:	1	50	100	150	200	300
Tlak=0.6:						
Tlak=1:						
Tlak=2:						

Tabulka 7.4: Různé hodnoty tlaku uvnitř svalů.

### 7.1.5 Gravitace

Posledním parametrem je nastavení gravitace. Tato hodnota je v exponenciálním tvaru  $10^x$  a značí vnější sílu použitou na všechny svalové body povrchové sítě před každou iterací. Síla směřuje záporným směrem na ose  $z$ . Výsledky pro různé hodnoty gravitace jsou vyobrazeny na obrázku 7.5.

Snímek:	1	50	100	150	200	300
Gravitace=0:						
Gravitace=4:						
Gravitace=5:						

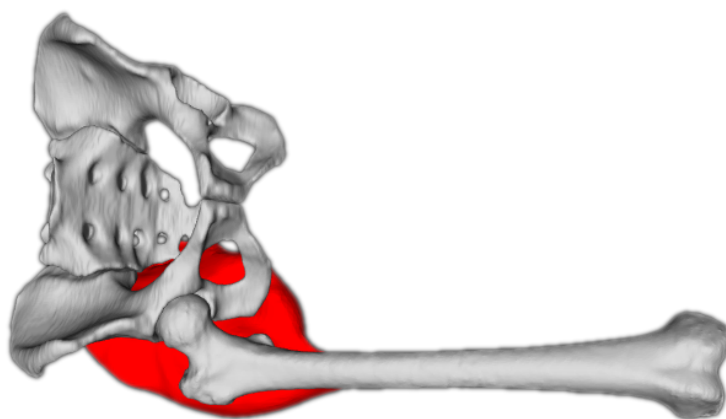
Tabulka 7.5: Chování svalů při různé gravitaci.

## 7.2 Reálná data

Poté co jsou otestována uměle vytvořená data, je vhodné provést i test nad reálnými daty, která nejsou z pravidla tak systematická.

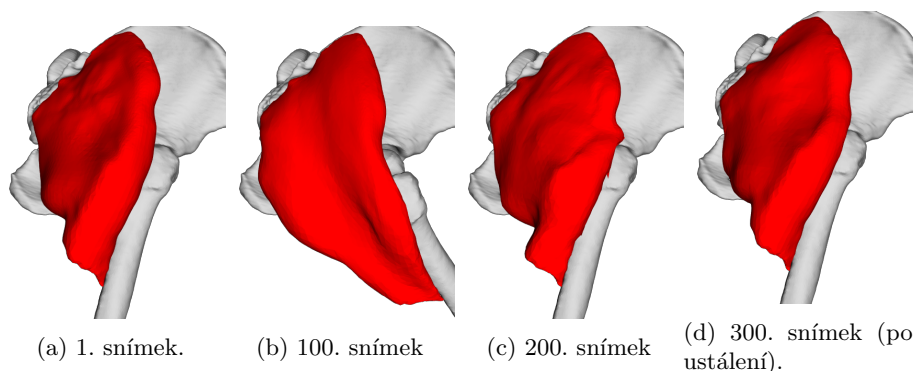
### 7.2.1 Gluteus maximus

Prvními otestovanými daty bude povrchový model svalu *gluteus maximus* a modely pánve a stehenní kosti. Modely byly pořízeny z projektu LHDL [8] a nepřesnosti v datech byly upraveny prací [7] v rámci projektu VPHOP (FP7-ICT-223865). Tato data pak vypadají jako na obrázku 7.2.



Obrázek 7.2: Reálná data: kost stehenní (*femur*) a pánev (bíle) spolu s velkým svalem hýžděovým (*gluteus maximus*, červeně).

Testovacím scénářem je pohyb *femuru* o 1 radián vpřed a zase zpět během 200 snímků. Kost se bude pohybovat reálným způsobem, tzn. bude rotovat v kyčelním kloubu. Nejprve si ukážeme výsledek, kde se podaří sval navrátit do velmi podobného stavu, jako byl na začátku simulace. Porovnání je na obrázku 7.3. Parametry simulace byly nastaveny na hodnoty uvedené v tabulce 7.6.



Obrázek 7.3: Chování algoritmu nad reálnými daty (*gluteus maximus*).

Tyto parametry byly nalezeny empiricky, výběr parametru, který by měl být změněn byl proveden podle toho, jak se parametr choval v případě umělých dat.

Anisotropy	0.5
Pressure	1
Iteration / frame	3
Bending	0.9
Velocity damp	0.99
Gravity	0

Tabulka 7.6: Vhodné parametry pro reálná data.

### Diskuze optimálních parametrů

Tyto parametry byly zvoleny empiricky, což samozřejmě nemusí být optimální. Nyní se budeme věnovat jednomu parametru po druhém, proč má právě hodnotu kterou má.

Prvním parametrem je parametr anizotropie (*Anisotropy*). Tento parametr byl nastaven na hodnotu 0.5, proto aby bylo dosaženo vhodného poměru podélného a příčného roztahení. V případě nižšího parametru se sval roztahoval příliš do šířky, v případě vyššího se naopak moc protahoval.

Dalším parametrem je parametr tlaku. Tento parametr není vhodné měnit, protože sval nemůže měnit svůj objem, proto byl ponechán na hodnotě 1.

Dále následuje volba počtu iterací. Hodnota 3 byla zvolena jako kompromisní řešení mezi přesností a rychlostí simulace. Hodnota 1 například provádí rychlou simulaci, sval však nedokáže dostatečně rychle reagovat na pohyb stehenní kosti. V obráceném případě vyšší hodnoty získáme jen málo přesnosti navíc za cenu výrazného snížení rychlosti simulace.

Dalším parametrem je zachovávání tvaru (*Bending*). Hodnota 0.9 byla zvolena z toho důvodu, že hodnota 1 byla až moc přísná a omezovala ostatní omezení (protože cílem algoritmu je optimalizovat současně všechna omezení). Nejdůležitější omezení zde bylo zachovávání vzdálenosti mezi body, proto byl tento parametr snížen. Není možné ho ovšem úplně vynulovat, protože tím ztratíme tvar úplně. Jedná se tedy opět o kompromisní řešení (jako u téměř všech parametrů).

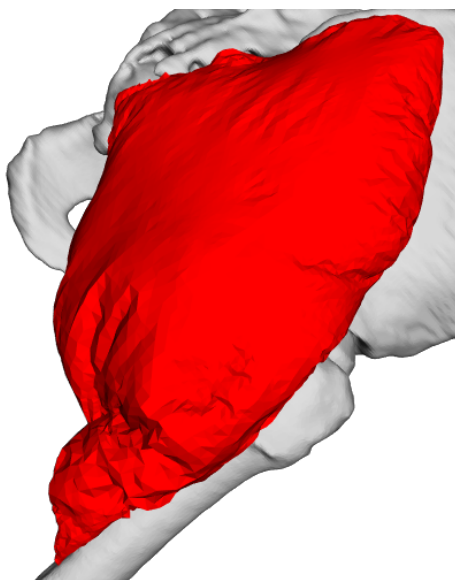
Předposlením voleným parametrem je snižování rychlosti. Tuto hodnotu teoreticky chceme mít co nejvyšší v případě, abychom se dostali do původní pozice, jenže vysoká hodnota (např. 1) znamená že algoritmus konverguje příliš dlouho. To je zřejmé z „rozvlnění“, které nastane na povrchu modelu svalu. Hodnota 0.99 tedy znamená poměrně dobré obnovení tvaru a zároveň ustálení do 100 snímků (z 200. snímku, kdy se zastaví pohyb kostí do 300. snímku, ze které je pořízen obrázek 7.3).

Protože jsme se snažili vrátit sval do původní polohy, byla hodnota gravitace vynulována (sval je totiž na vstupu uvažován v relaxovaném stavu, použijí na něj gravitaci, pak se zdeformuje). V praktických případech je ale velmi pravděpodobné, že nebude požadována nulová gravitace.

### Nevhodné parametry

Na konec je dobré se podívat i na parametry, které nejsou úplně vhodné. To ukáže hned několik věcí – za prvé slabiny zvoleného algoritmu a za druhé to poskytne vzor pro další modely, které by byly pomocí tohoto algoritmu deformovány.

**Nulové zachování tvaru** V případě, že nastavíme nulové zachování tvaru, se v zásadě z počátku nestane nic úplně kritického. Model pouze nebude zachovávat svoje lokální charakteristiky. Model bude připomínat zmuchlaný papír (viz obrázek 7.4). Při mnoha iteracích a mnoha pohybech se však lokální tvar zdeformuje natolik, že již například ani nepůjde poznat, o jaký tvar se původně jednalo.

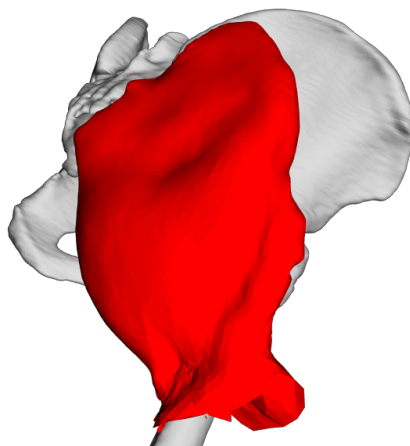


Obrázek 7.4: Reálná data: z důvodu nulového zachování tvaru je povrch svalu zdeformován (300. snímek po ustálení).

**Nízký počet iterací** Při nastavení příliš nízkého počtu iterací sval nedokáže dostatečně dobře reagovat na pohyb kostí. Algoritmus bude stále konvergovat, jen mu to bude trvat mnohem déle. Příklad je na obrázku 7.5, kde je 200. snímek (kost se právě přestala pohybovat).

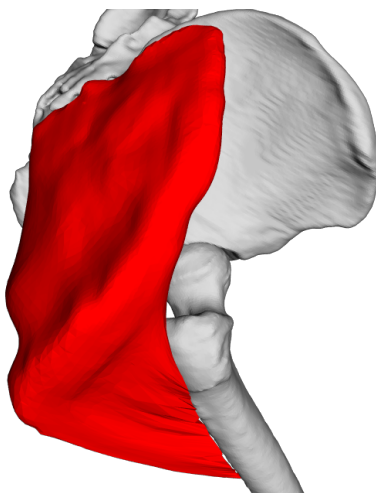
Při jedné iteraci na snímek bylo nutné přes 1000 simulačních snímků, aby dokonvergoval k rozumnému výsledku. V případě 3 iterací na snímek k tomu stačilo 100 snímků.

**Nízká hodnota zachování rychlosti** Omezíme-li zachování rychlosti na minimum, pak se budou body pohybovat, pouze budou-li k tomu dostatečně donuceny. Body totiž ztratí veškerou setrvačnost. V případě tohoto testu to má jednu výhodu. Většina bodů se během simulace viditelně nepohne, což zaručí vysokou podobnost tvaru před a po pohybu kosti.



Obrázek 7.5: Reálná data: algoritmus má před sebou ještě mnoho snímků, než dokonverguje k rozumnému výstupu (Iteration/frame: 1, 200. snímek).

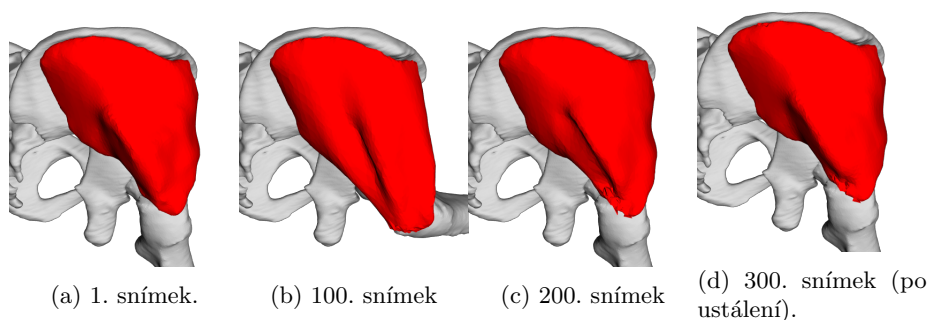
Samozřejmě takový pohyb je ale nepřirozený, když bude stehenní kost nejvzdálenější od svalu (100. snímek), model svalu vypadá nepříjemně (viz obr. 7.6).



Obrázek 7.6: Reálná data: setrvačnost bodů je vypnuta (Velocity damp: 0), takže body se pohybují pouze v případě, jsou-li k tomu donuceny (Snímek: 100).

### 7.2.2 Gluteus medius

Dalším svalem v oblasti kyčelního kloubu, který lze otestovat, je *gluteus medius*. Tento sval je (jak název napovídá) menší než předcházející a také je umístěn na jiném místě než předcházející *gluteus maximus*, proto je vhodné tento sval také otestovat. Parametry simulace budou stejné jako v předcházejícím případě (tabulka 7.6).

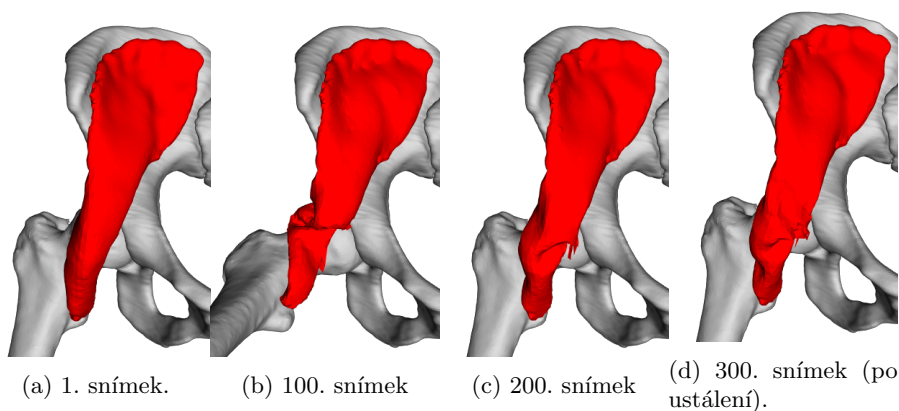


Obrázek 7.7: Chování algoritmu nad reálnými daty (*gluteus medius*).

Pohyb svalu je plynulý bez výraznějších problémů (viz obrázek 7.7). Problémy vznikají pouze v oblasti uchycení svalu na stehenní kost, kde dochází k drobným problémům se zachováním tvaru v průběhu simulace.

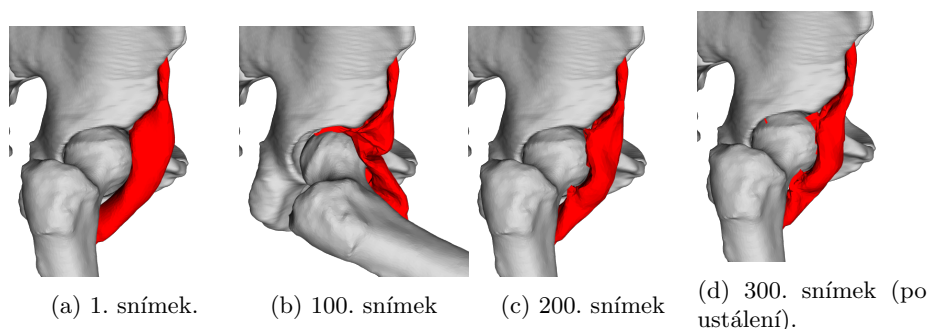
### 7.2.3 Iliacus

Posledním svalem je *iliacus*. Tento sval je velmi problematický co se týče jeho pozice a funkce, protože působí opačným směrem, než oba předchozí svaly. Dalším problémem, že model se nachází velmi blízko kyčelního hloubu.



Obrázek 7.8: Různé snímky deformace svalu *iliacus*.

Z obrázku 7.8 je zřejmé (hlavně ze 100. snímku), že deformace nedopadla až tak úspěšně jako v předcházejících případech. Hlavně oblast blízko kloubu je problematická. Podíváme-li se detailněji na kloub, zjistíme co je za problém (obrázek 7.9).



Obrázek 7.9: Různé snímky deformace svalu *iliacus* – detail na kyčelní kloub.

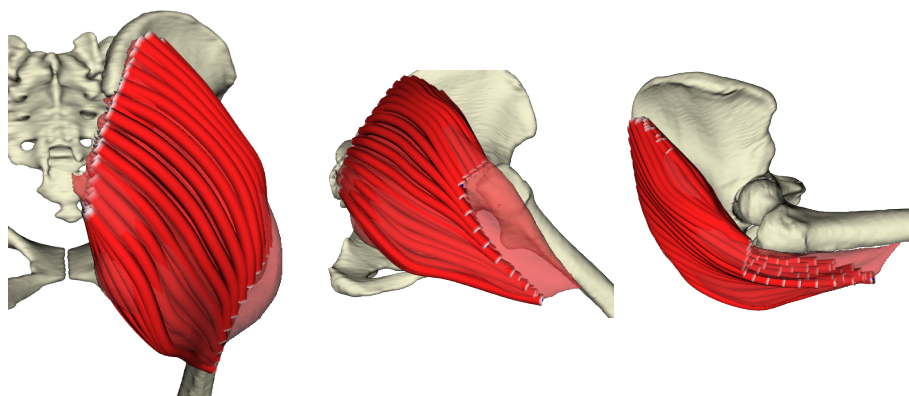
Problémů je hned několik. První (hlavní) problém je, že kloub v simulaci je tvořen pouze dvěma kostmi, chybí zde jakékoliv vazy, které by zabraňovaly vstupu svalu do kloubu. Problémem je ještě umocněn tím, že sval i přes to, že by měl zachovávat tvar, je nuceně vtlačěn (kolizemi) do kloubní jamky mezi kosti. Tento problém vznikl mimo jiné i zjednodušenými (nepřesnostmi) v detekci kolizí a reakcích na ně (viz sekce 5.3.2). Tento problém je hlavně patrný na 100. snímku.

Dalším problémem je jeden bod, který pronikl skrz stehenní kost (na obrázku jde o snímek 300). Problém vznikl opět zjednodušením, které bylo uvedeno v sekci 5.3.1, kde jsme tento problém prakticky i přepověděli.



### 7.3 Výsledek dekompozice

Posledním krokem je převést povrchový model svalu na model svalových vláken. K dekompozici na svalová vlákna z povrchového modelu jsem použil práci D. Cholty [12] a výsledek na svalu *gluteus maximus* je vidět na obrázku 7.10. Dekompozice byla v tomto případě provedena na 100. snímku simulace (tedy když byla stehenní kost nejdále od své výchozí polohy).



Obrázek 7.10: Dekompozice svalu *gluteus maximus* na svalová vlákna pomocí algoritmu D. Cholty [12].

Metoda D. Cholty vyžaduje model svalových vláken na povrchu svalu, aby dokázala dekompozici provést. V našem případě taková data jsou k dispozici (byla například použita i pro anizotropii), ale vlákna jsou pouze ve výchozí pozici, bylo nutné je zdeformovat podobně jako sval, aby Choltyova metoda byla korektní (dle autora je na polohu vláken citlivá).

Deformace byla provedena jednoduchým způsobem – body lomených čar modelů vláken byly na počátku simulace přiřazeny k nejbližším bodům na povrchu svalu a během simulace se nastavuje pozice bodů vláken na pozice bodů na povrchu svalu, čímž je zajištěno, že vlákna leží (ve smyslu bodů) na povrchovém modelu svalu.

Existují i jiné, přesnější postupy registrace, například [10] nebo [9], pro účel otestování bylo však zvoleno toto zjednodušení. Zjednodušení je zřejmé z toho, že vlákna se neroztáhla po celém povrchu svalu, takže část svalu není vlákní pokryta. Cílem zde bylo ukázat, že není problém provést dekompozici na svalová vlákna v jakémkoliv snímku simulace. Mimo jiné i touto metodou lze získat deformovaná svalová vlákna z povrchového modelu, který byl modelován zde navrženým postupem za pomoci systému vázaných částic.

## Kapitola 8

### Závěr

Cílem této práce bylo prozkoumání metod pro modelování a deformaci svalových vláken. Bylo zvoleno několik postupů relevantních pro problematiku modelování muskuloskeletálního systému, ať již state-of-the-art, nebo metod dosud v této problematice nevyzkoušených.

Z těchto metod byla zvolena metoda známá jako PBD, která za cenu částečného fyzikálního a matematického zjednodušení měla umožňovat provedení deformace efektivně a rychle, což se touto prací i potvrdilo.

Prostor pro řešení problémů použitím PBD pro modelování muskuloskeletálního systému je však stále velký. V práci bylo nalezeno a diskutováno několik problémů, které je nutné pro použití PBD v této problematice vyřešit. Mezi tyto problémy patří hlavně reakce na detekci kolizí, která je poměrně komplexní záležitostí. Řada z problémů zde byla vyřešena, stále zbývá vyřešit problém pohybu více kostí blízko sebe, kdy stále může docházet k neošetřené kolizi. Také byl vysloven předpoklad, že pohyb kostí nebude natolik signifikantní, aby sval prošel skrz celý objem kosti. Tento předpoklad byl však vyvrácen posledním testem na svalu *iliacus*. Dalším problémem je nepřirozená deformace svalu v blízkosti kloubu.

Algoritmus PBD byl dále podrobně popsán i z matematického hlediska, aby bylo evidentní jakým způsobem vlastně funguje. Dále byla diskutována i specifika modelování muskuloskeletálního systému, která je nutné přidat ke zvolenému algoritmu. Jedná se hlavně o anizotropii, která je klíčová v případě svalů, složených z různě orientovaných vláken.

Na závěr byl algoritmus PBD otestován implementačně a byly demonstrovány jeho schopnosti.

Zadání této práce je tímto splněno. Algoritmus PBD doporučen vedoucím této práce považuji za vhodného kandidáta pro použití v problematice modelování pohybu svalů a kostí, je však nutné ještě dořešit některé (zde v práci uvedené) nedostatky.

# Literatura

- [1] Wade, S.W., Strader, C., Fitzpatrick, L.A. et al. Estimating prevalence of osteoporosis: examples from industrialized countries. *Arch Osteoporos* (2014) 9: 182. <https://doi.org/10.1007/s11657-014-0182-3>
- [2] G. Bergmann, G. Deuretzbacher, M. Heller, F. Graichen, A. Rohlmann, J. Strauss, G. N. Duda. Hip contact forces and gait patterns from routine activities. *Journal of Biomechanics*. Vol. 34. p. 859-871 (2001)
- [3] Carol A. Oatis. *Biomechanics of skeletal muscle*. ch. 4 p. 45-68 (2017)
- [4] J. Zhang, J. Fernandez, J. Hislop-Jambrich, T. F. Besier. Lower limb estimation from sparse landmarks using an articulated shape model. *Journal of Biomechanics*, Vol. 49. p. 3875-3881. <https://doi.org/10.1016/j.biomech.2016.10.021> (2016)
- [5] S. L. Delp, J. P. Loan, M. G. Hoy, F. E. Zajac, E. L. Topp, J. M. Rosen. An Interactive Graphics-Based Model of the Lower Extremity to Study Osteopaedic Surgical Procedures. *Biomedical Engineering*. Vol 37. No 8. (1990)
- [6] D. Testi et al. PhysiomeSpace: digital library service for biomedical data. *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences* vol. 368,1921: 2853-61. <https://doi.org/10.1098/rsta.2010.0023> (2010)
- [7] J. Kohout, G. J. Clapworthy, Y. Zhao, Y. Tao, G. Gonzalez-Garcia, F. Dong, H. Wei, E. Kohoutová. Patient-specific fibre-based models of muscle wrapping. 3. *Interface Focus*. <http://doi.org/10.1098/rsfs.2012.0062> (2013)
- [8] The Living Human Digital Library  
<https://projects.kmi.open.ac.uk/lhdl/>
- [9] M. Červenka. Nerigidní registrace povrchových svalových a šlachových snopců. Bakalářská práce. Západočeská univerzita v Plzni, Fakulta aplikovaných věd. Vedoucí práce Doc. Ing. Josef Kohout, Ph.D.. (2017)
- [10] H. Li, R. W. Sumner, M. Pauly. Global Correspondence Optimization for Non-Rigid Registration of Depth Scans. *Computer Graphics Forum* 27(5), *Proceedings of the Sixth Eurographics Symposium on Geometry Processing* (2008).

- 
- [11] J. Kohout, M. Kukačka. Real-Time Modelling of Fibrous Muscle. *Computer Graphics Forum*. vol. 33, p. 1-15. <https://doi.org/10.1111/cfg.12354> (2014)
- [12] J. Kohout, D. Cholt. Automatic Reconstruction of the Muscle Architecture from the Superficial Layer Fibres Data. *Computer Methods and Programs in Biomedicine*. 150. <https://doi.org/10.1016/j.cmpb.2017.08.002>. (2017)
- [13] S. S. Blemker, S. L. Delp. Three-Dimensional Representation of Complex Muscle Architectures and Geometries. *Annals of Biomedical Engineering*. vol. 33, p. 661-673 <https://doi.org/10.1007/s10439-005-1433-7> (2004)
- [14] B. A. Garner, M. G. Pandy. The Obstacle-Set Method for Representing Muscle Paths in Musculoskeletal Models. *Computer Methods in Biomechanics and Biomedical Engineering*. 3:1, 1-30, <https://doi.org/10.1080/10255840008915251> (2000)
- [15] G. Valente, S. Martelli, F. Taddei, G. Farinella, M. Viceconti. Muscle discretization affects the loading transferred to bones in lower-limb musculoskeletal models. *Proceedings of the Institution of Mechanical Engineers. Part H, Journal of engineering in medicine*. 226. 161-9. <https://doi.org/10.1177/0954411911425863>. (2012)
- [16] T. Janák. Fast soft-body models for musculoskeletal modelling. Technický report DCSE/TR-2012-5. Západočeská univerzita v Plzni, Fakulta aplikovaných věd (2012)
- [17] M. Hong, S. Jung, M. Choi, S. W. J. Welch. Fast Volume Preservation for a Mass-Spring System. *IEEE Computer Graphics and Applications*. vol. 26. p. 83-91. Sept.-Oct. <https://doi.org/10.1109/MCG.2006.104>. (2006)
- [18] M. Müller, B. Heidelberger, M. Hennix, J. Ratcliff. Position based dynamics. *Journal of Visual Communication and Image Representation*. vol. 18. p. 109-118. <https://doi.org/10.1016/j.jvcir.2007.01.005> (2007)
- [19] J. Cai, F. Lin, Y. T. Lee et al. Modeling and dynamics simulation for deformable objects of orthotropic materials. *The Visual Computer* 33: 1307. <https://doi.org/10.1007/s00371-016-1221-4> (2017)
- [20] J. Cai, F. Lin, H. S. Seah. Graphical Simulation of Deformable Models. <https://doi.org/10.1007/978-3-319-51031-6> (2016)

## Příloha A

# Uživatelská příručka

Na přiloženém CD se nachází složka `bin`, ve které jsou spustitelné soubory programu. Ve složce `bin` se nachází několik podsložek, každá pro nějakou platformu. Proto je třeba zvolit vhodnou platformu, na které se program spouští (např. `Lin_64` znamená 64-bitový operační systém Linux).

V těchto podsložkách se dále nalezne vlastní program (`pbid` a případně i přípona) a spolu s ním i skripty, které spustí program s určitými daty. Skripty jsou bezparametrické, aby bylo možné je snadno spustit. Jejich názvy jsou (`run_` + číslo scénáře + platformě závislá přípona (například `.sh`)). Například tedy skript s názvem `bin/Lin_64/run_1.sh` spustí 1. scénář. Možné scénáře jsou uvedeny v následující tabulce:

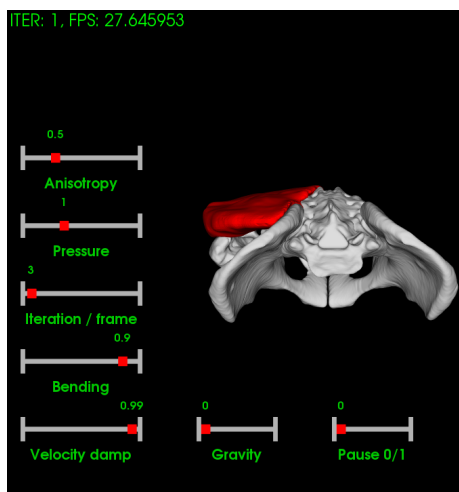
Název scénáře	Číslo scénáře
Reálná data, <i>gluteus maximus</i>	1
Reálná data, <i>gluteus medius</i>	2
Reálná data, <i>gluteus iliacus</i>	3
Umělá data, stlačení svalu mezi dvě plochy	4
Tetrahedrony, test zachování vzdáleností	5
Tetrahedrony, test zachování tvaru	6
Tetrahedrony, test zachování objemu	7

Dále je v dané složce i přítomný program (`pbid` + případná přípona), kterému lze předat modely, se kterými má pracovat. To lze provést například takto:

```
./pbid \  
  --muscles ../data/1/m* \  
  --bones ../data/1/b* \  
  --fibres ../data/1/f*
```

Příkaz je samodokumentující, proto nebude dále popsán. Zatím neexistuje možnost, jak programu předat dynamicky scénář (způsob pohybu kostí, svalů apod.), proto jediný způsob jak je možné svaly rozhýbat je parametry simulace.

Po spuštění příkazu či skriptu se zobrazí okno s pozastavenou simulací (viz obrázek A.1).



Obrázek A.1: Okno aplikace (scénář 1).

Simulace musí být spuštěna ručně přesunem posuvníku *Pause 0/1* do polohy 0. Simulace se poté rozběhne. Kdykoliv během simulace je možné změnit některý z parametrů simulace, a simulace na takovou změnu okamžitě reaguje.

Na model se lze dívat z různých úhlů. To lze provést držením levého tlačítka myši a současného pohybu. Pro posun se k této kombinaci podrží klávesa *Shift*, pro rotaci kolem osy kamery se úkon provede za současného držení klávesy *Ctrl*.