

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

## **Bakalářská práce**

# **Geometrické reprezentace svalových vláken a jejich vliv na ramena momentů sil**

# ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta aplikovaných věd  
Akademický rok: 2020/2021

## ZADÁNÍ BAKALÁŘSKÉ PRÁCE (projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení: **Lukáš RYPL**  
Osobní číslo: **A18B0304P**  
Studijní program: **B3902 Inženýrská informatika**  
Studijní obor: **Informatika**  
Téma práce: **Geometrické reprezentace svalových vláken a jejich vliv na ramena momentů sil**  
Zadávací katedra: **Katedra informatiky a výpočetní techniky**

### Zásady pro vypracování

1. Seznamte se s principy muskuloskeletálního modelování využívaných běžně v biomechanické praxi, např. v systémech OpenSim nebo AnyBody.
2. Seznamte se s modelovacím přístupem Luca2018 popsáním v článku Modenese & Kohout, 2020.
3. Navrhněte a implementujte adaptivní algoritmus pro generování svalových vláken reprezentovaných Catmull-Rom křivkou, který bude založen na původní metodě popsané v článku Kohout & Kukačka, 2014.
4. Navrhněte a implementujte algoritmus pro dodatečnou úpravu vláken v průběhu deformace metodou Luca2018, který zajistí, že vlákna neprotínají kosti.
5. Pro vybrané svaly oblasti pánve analyzujte vliv počtu segmentů, ze kterých je vlákno složeno, resp. které definují Catmull-Rom křivku, na ramena momentů sil v průběhu flexe, extenze a vnitřní rotace nohy.
6. Pro vybrané svaly oblasti pánve analyzujte vliv dodatečných úprav vláken na ramena momentů sil v průběhu flexe, extenze a vnitřní rotace nohy.
7. Dosažené výsledky kriticky zhodnoťte.

Rozsah bakalářské práce: **doporuč. 30 s. původního textu**  
Rozsah grafických prací: **dle potřeby**  
Forma zpracování bakalářské práce: **tištěná**

Seznam doporučené literatury:

Dodá vedoucí bakalářské práce.

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

Datum zadání bakalářské práce: **5. října 2020**  
Termín odevzdání bakalářské práce: **6. května 2021**

L.S.

---

**Doc. Dr. Ing. Vlasta Radová**  
děkanka

---

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

# Prohlášení

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

V Plzni dne 5. května 2021

Lukáš Ryppl

# Poděkování

Rád bych poděkoval svému vedoucímu doc. Ing. Josefu Kohoutovi Ph.D. za ochotu, kterou projevil v rámci naší spolupráce. I když mě v podstatě na měsíc odstavila nemoc, pomohl mi vrátit se zpět do rytmu a vše dohnat, abych stihl květnové odevzdání.



## Abstract

The thesis deals with the representation of muscle fibers through Catmull-Rom curves from their formation to deformation (motion simulation). The number of segments that make up a fiber is determined automatically using the adaptive method proposed in the work. The increased number of segments ensures a better representation of the fiber. To modify the fibers at runtime, a method for detecting the collision of the bone surface model and the fiber control point was introduced, based on the implementation of a signed distance field in the Discregrid library. The points of fiber located inside the bone were thus moved to its surface. Compared to the current approach, the proposed solution is significantly more accurate, although it does not guarantee that the individual fiber segments will not collide, as there are cases where the points are outside the bone, but line between them passes through the bone.

## Abstrakt

Práce se zabývá reprezentací svalových vláken prostřednictvím Catmull-Rom křivek od jejich tvorby až po deformaci (simulaci pohybu). Počet segmentů, ze kterých se vlákno skládá, je určen automaticky pomocí adaptivní metody navržené v práci. Zvýšený počet segmentů zajistí lepší reprezentaci vlákna. Pro úpravu vláken za běhu byla zavedena metoda pro detekci kolize povrchového modelu kosti a řídicího bodu vlákna, postavená na implementaci signed distance field v knihovně Discregrid. Body vlákna nacházející se uvnitř kosti byly díky tomu přesunuty na její povrch. V porovnání se současným přístupem je navržené řešení výrazně přesnější, přestože negarantuje, že jednotlivé segmenty vlákna nebudou kolidovat, neboť dochází k případům, kdy body jsou mimo kost, ale jejich spojnice kostí prochází.

## **Keywords**

Keywords for this bachelor thesis are:

Musculoskeletal modelling, Catmull-Rom curve, signed distance field, collision detection

## **Klíčová slova**

Klíčová slova pro tuto bakalářskou práci jsou:

Muskuloskeletální modelování, Catmull-Rom křivky, signed distance field, detekce kolizí

# Obsah

<b>1</b>	<b>Úvod</b>	<b>4</b>
1.1	MuscleWrapping project 2 . . . . .	5
1.2	Struktura práce . . . . .	5
<b>2</b>	<b>Muskuloskeletální modely</b>	<b>6</b>
2.1	Reprezentace svalů . . . . .	6
2.1.1	Line of action . . . . .	7
2.2	Reprezentace kostí . . . . .	8
2.3	Reprezentace kloubů . . . . .	8
2.4	Pohyb kostí . . . . .	8
2.5	Problém s deformací svalu . . . . .	9
<b>3</b>	<b>Metoda deformace vláken – Luca2018</b>	<b>11</b>
<b>4</b>	<b>Dekompozice svalů</b>	<b>13</b>
4.1	Vstup . . . . .	13
4.2	Úponové oblasti . . . . .	14
4.3	Tvorba skalárního pole . . . . .	15
4.4	Izočáry . . . . .	15
4.5	Šablony svalů . . . . .	16
4.6	Přiřazení vrcholů . . . . .	16
4.7	Prodloužení vláken . . . . .	17
<b>5</b>	<b>Catmull-Rom křivky</b>	<b>19</b>
5.1	Výpočet křivky . . . . .	19
5.2	Vliv hodnoty alfa . . . . .	20
<b>6</b>	<b>Návrh adaptivního algoritmu</b>	<b>22</b>
6.1	Obecný algoritmus . . . . .	22
6.2	Problém více vláken a jeho řešení . . . . .	23
6.3	Chybová funkce . . . . .	24
6.3.1	Zkroucení křivky . . . . .	24
6.3.2	Vzdálenost křivek . . . . .	24

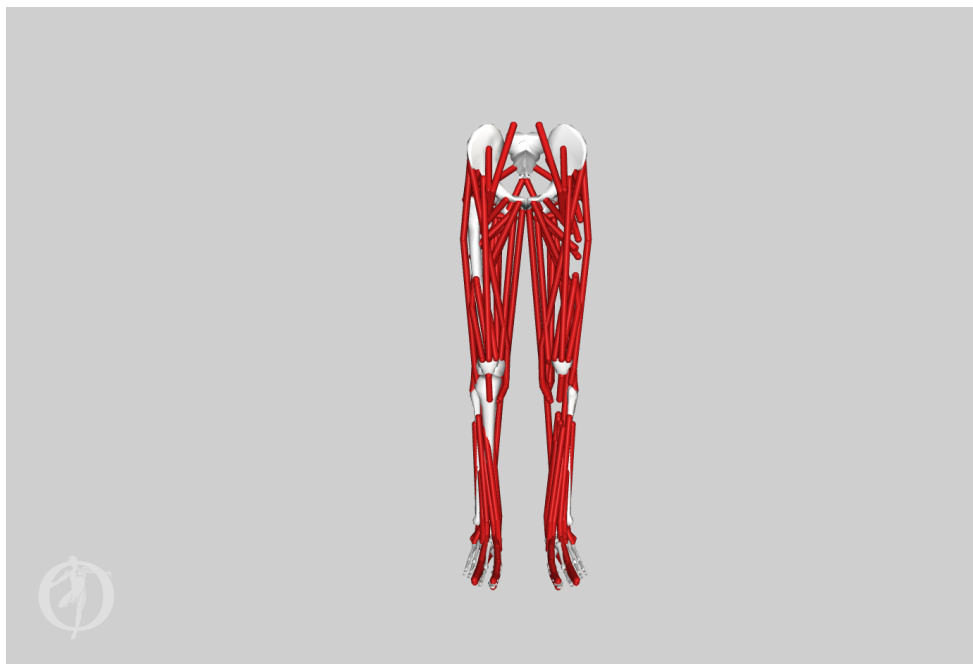
6.4	Dělení vláknů . . . . .	25
<b>7</b>	<b>Návrh detekce kolizí</b>	<b>26</b>
7.1	SDF . . . . .	26
7.2	Návrh detekce kolizí . . . . .	26
<b>8</b>	<b>Adaptivní algoritmus pro určení počtu segmentů</b>	<b>28</b>
8.1	Implementovaný algoritmus . . . . .	28
8.2	Souhrnná chybová funkce . . . . .	29
8.2.1	Vector computed a jeho obsah . . . . .	31
8.2.2	Pravidla dělení . . . . .	31
8.2.3	Příklad průběhu dělení . . . . .	32
8.2.4	Filtrování nežádoucích bodů . . . . .	33
8.2.5	Příklad filtrování . . . . .	33
8.3	Chybová funkce . . . . .	34
8.4	Filtrování . . . . .	36
8.5	Třída CatmullRomSpline . . . . .	36
8.5.1	Výpočet bodu na křivce . . . . .	37
8.5.2	Výpočet tečny . . . . .	37
8.5.3	Délka křivky . . . . .	37
8.5.4	Operace pro úpravu křivky . . . . .	37
<b>9</b>	<b>Úprava vláken za běhu</b>	<b>39</b>
9.1	Discregrid . . . . .	39
9.2	Integrace Discregridu . . . . .	39
9.2.1	Příprava struktur pro SDF . . . . .	39
9.2.2	Tvorba SDF . . . . .	40
9.2.3	Úprava deformace . . . . .	40
9.2.4	Úprava parametru w metody Luca2018 . . . . .	41
<b>10</b>	<b>Výsledky práce</b>	<b>42</b>
10.1	Testovací data . . . . .	42
10.2	Metoda testování . . . . .	43
10.3	Vliv parametru w . . . . .	43
10.4	Adaptivní vlákna . . . . .	45
10.5	Výsledný počet segmentů . . . . .	46
10.6	Úprava vláken . . . . .	46
10.7	Omezení aktuální implementace . . . . .	51
10.8	Počet kolizí . . . . .	52
10.9	Celkový rozdíl . . . . .	54
10.10	Časové nároky . . . . .	56

<b>11</b>	<b>Závěr</b>	<b>58</b>
	<b>Literatura</b>	<b>61</b>
<b>A</b>	<b>Uživatelská dokumentace</b>	<b>63</b>
A.1	Překlad a instalace . . . . .	63
A.1.1	Závislosti . . . . .	63
A.1.2	Překlad MWP2 . . . . .	64
A.1.3	Spuštění MWP2 . . . . .	65
A.1.4	Konfigurace . . . . .	65
A.1.5	Program za běhu . . . . .	66
<b>B</b>	<b>Obsah DVD</b>	<b>67</b>

# 1 Úvod

V posledních letech se vědci z oboru medicínské informatiky stále více a více zajímají o muskuloskeletální modely člověka. Tyto modely přináší velké výhody v oblastech prevence, diagnostiky, předoperačních vyšetření a rehabilitace, neboť dokážou nahradit drahá vyšetření. Jako příklad lze uvést nemoc zvanou osteoporóza, kdy dochází k řídnutí kostní tkáně. Modely mohou poskytnout lékaři informace o silách působících na dané (v případě osteoporózy oslabené) kosti. Tím lze určit, jaká je hranice sil působících na danou kost, než dojde k fraktuře. Díky tomu lze včasné zavést preventivní opatření a předejít nejen fraktuře, ale i nákladné léčbě.

Stávající modely (například model z Obrázku 1.1), které jsou dostupné, jsou však příliš obecné, zjednodušené, tím pádem neposkytují lékaři dostatečně personalizované informace. Tato zjednodušení jsou uskutečněna především kvůli časové náročnosti přesnějších modelů.



Obrázek 1.1: OpenSim [7] – BothLegs model

## 1.1 MuscleWrapping project 2

Tato bakalářská práce je vypracována v rámci MuscleWrapping 2 projektu (MWP2). Projekt navazuje na evropský projekt LHDL (Living Human Digital Library), jehož cílem bylo vytvořit digitální atlas člověka. Součástí tohoto projektu se během svého pobytu ve Velké Británii stal i vedoucí této práce doc. Ing. J. Kohout Ph.D., který do projektu vnesl svou inovativní myšlenku konstrukce a deformace svalů. V rámci LHDL se však nepovedlo dotáhnout projekt do použitelného stavu.

V roce 2018 o původní projekt projevil zájem Luca Modenese z ICL (Imperial College London) s novým pohledem na problém. Aktuální modely totiž nejsou ideální a přístup z LHDL by toto mohl výrazně změnit, jelikož automaticky umožní vytvořit modely přesnější, než jsou ty aktuální, které jsou navíc vytvořené ručně (což je nepraktické). Cílem MWP2 je tedy vytvořit plugin pro OpenSim, který umožní snadno vytvářet biomechanické modely.

Momentálně jsou v projektu rozvíjeny 2 přístupy pro deformaci svalů – Luca2018 a Cervenka2020. Luca2018 se zaměřuje především na biomechanickou povahu úlohy (výsledkem jsou momenty sil, geometrie tkání není podstatná) – soustředí se hlavně na generaci vláken a jejich přímou deformaci (z vláken se poté případně dá rekonstruovat povrch). Cervenka2020 se naopak zaměřuje na vizuální část (není podstatná vnitřní struktura svalu) – zabývá se tudíž deformací přímo povrchového modelu svalu (který lze případně dekomponovat na vlákna).

Cílem této bakalářské práce je seznámit se s danou problematikou a dále navrhnout a implementovat rozšíření stávajícího přístupu Luca2018. Rozšíření se skládá ze dvou dílčích částí – za prvé je potřeba za pomoci Catmull-Rom křivek zajistit adaptivní dělení vlákna na segmenty, druhým úkolem je zajistit, aby nedocházelo ke kolizím mezi vlákny svalu a kostmi.

## 1.2 Struktura práce

V následující kapitole bude čtenáři představena problematika muskuloskeletálního modelování. Poté budou představeny aktuálně používané metody – Luca2018 (pro deformaci svalu) a Kukačka (pro dekompozici svalu). Dále již budou následovat kapitoly týkající se návrhu implementace, které budou následované kapitolami popisujícími reálnou implementaci. Poslední kapitola práce zhodnocuje dosažené výsledky a v samotném závěru lze najít návrhy pro příští, navazující práci.

## 2 Muskuloskeletální modely

Muskuloskeletální modely (MSM) jsou, jak již název napovídá, modely lidského těla skládající se především ze svalů a kostí. Sval je v běžné praxi zobrazen pomocí vláken spojujících úponové oblasti svalu (místa, kde je sval uchycen na kosti). Dále mohou tyto svaly být definovány tzv. *via points* [3], body, které jsou obvykle v pevné vzdálenosti od jedné z kostí, a skrz které sval za každou cenu musí procházet. Je však zřejmé, že definovat takovéto body není jednoduché a vznikají při tom problémy.

Kromě svalů a kostí jsou v modelu definovány i klouby mezi kostmi a síly působící na celý model (například gravitační síla, případně síly pocházející z okolí modelu). Celý model by měl být pohyblivý a měl by co nejpřesněji odpovídat realitě. Realistické modely jsou schopny nahradit nákladná vyšetření a zabránit mnoha zraněním, a proto je o ně neustále rostoucí zájem. Ovšem, jak je již obvyklé, vytvořit takovýto komplexní model s nízkými nároky na hardware a čas není snadné. Problém je i se samotnou definicí svalů, neboť bychom si přáli, aby se neprotínaly. Toho však nelze jednoduše dosáhnout a do hry musí vstoupit složitější algoritmy. V této kapitole bude popsán obvykle používaný model a jeho problémy.

### 2.1 Reprezentace svalů

Svalem se obecně myslí svalově-šlachová jednotka, popřípadě i celá skupina svalů (např. biceps femoris je někdy rozdělován na dvě nezávislé hlavy, někdy nikoliv – to je způsobeno tím, že dokonce ani při disekci nejsou vždy zřejmé hranice svalů). Svaly v MSM jsou nejdůležitější, avšak zároveň i nejsložitější částí. Dle účelu modelu lze sval modelovat více způsoby:

- Lze využít elipsoidy nebo kvadratické Bézierovy objekty, avšak jejich tvar neodpovídá reálným svalům. Jejich výhodou je snadná deformovatelnost.
- Je možné sval popsat pomocí trojúhelníkových povrchových sítí, neboť ho mnohem přesněji reprezentují, na druhou stranu jejich deformování je velmi složité a výpočetně náročné.
- Pro medicínské účely je poměrně často podstatná i vnitřní struktura

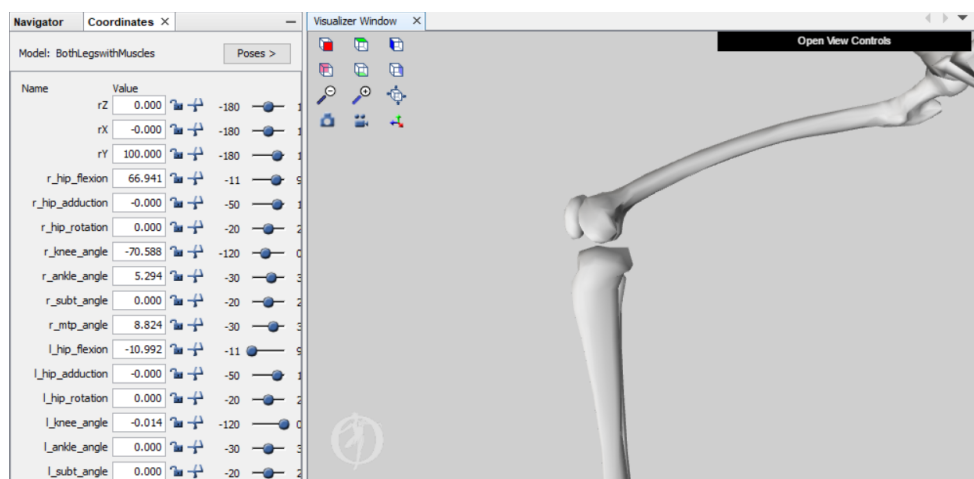


svalu (uspořádání vláken, ze kterých se každý sval skládá). Přesně pro tyto případy byla zavedena reprezentace svalů tzv. line of action [3].

### 2.1.1 Line of action

Line of action je křivka (případně je možno použít pouze přímku), která poměrně přesně reprezentuje pozorovatelnou výslednici síly. Ve většině případů působí tato výslednice ve směru vláken svalu, ale není tomu pravidlem. I přesto je line of action v literatuře často nazývána svalovým vláknem. Příkladem této reprezentace jsou Obrázek 1.1 a Obrázek 2.3. Křivka má definovaný počátek i konec (na povrchu kosti) a lze dodefinovat via points - body skrz které bude křivka procházet. Nespornou výhodou tohoto přístupu je rychlost a jednoduchost deformací, ale je otázka, jak přesně lze pomocí jedné až dvou křivek reprezentovat objemný sval [3].

Studie provedená na IOR (Istituto Ortopedico Rizzoli) v Boloni v Itálii ukazuje, že rozdíl v momentech sil, kterého se dopustíme použitím jedné výslednice síly místo použití více, může dosahovat relativní chyby až 75 % pro velké svaly (např. gluteus maximus – velký sval hýždový). Článek dále poukazuje na fakt, že s rostoucí hustotou vláken se tato relativní chyba asymptoticky snižuje [9]. Z toho je zřejmé, že více vláken přesněji popisuje MSM, ale tvorba nových vláken v klasických modelech je složitá a vyžaduje zásah uživatele – experta, který vlákna musí nadefinovat. To je bohužel ve větším měřítku v praxi nemožné, proto se běžně nepoužívají více než dvě line of action [3].



Obrázek 2.1: OpenSim [7], BothLegs model, ukázka možností ohybu v kolenním kloubu

## 2.2 Re prezentace kostí

Re prezentace kostí je jednodušší částí modelu. Předpokládáme, že kosti jsou tuhá tělesa a při pohybu se tudíž nijak nedeformují, pouze rotují o určitý úhel v určitém směru. Kostí jsou reprezentovány jako 3D objekty, nejčastěji pomocí sítí polygonů - obvykle trojúhelníků. Samotný tvar kosti lze poměrně snadno určit například pomocí magnetické rezonance a převést na 3D objekt. Dále se odstraní nechtěný šum a vyhladí se povrch kosti [8]. Tím vzniká kostra, na kterou se poté mohou upnout svaly (resp. svalová vlákna). Příklad, jak vypadá kost ze základního modelu programu OpenSim [7], lze vidět na Obrázku 2.1.

## 2.3 Re prezentace kloubů

Klouby se definují v místech doteku kostí a určují úhel a směr jejich pohybu. Úhel, o který je možno se pohybovat, je samozřejmě omezen dle fyziologických možností člověka. Obrázek 2.1, kde je v programu OpenSim otevřen model dolních končetin, ukazuje kolenní kloub jednoho ze základních modelů. V nabídce nalevo lze vidět možnosti pohybu v různých kloubech. Každá řádka reprezentuje jeden pohyb (o daný úhel v případě rotace, o danou vzdálenost v případě translace). Při vizualizaci modelu se klouby většinou nezobrazují, pouze umožňují předem daný pohyb.

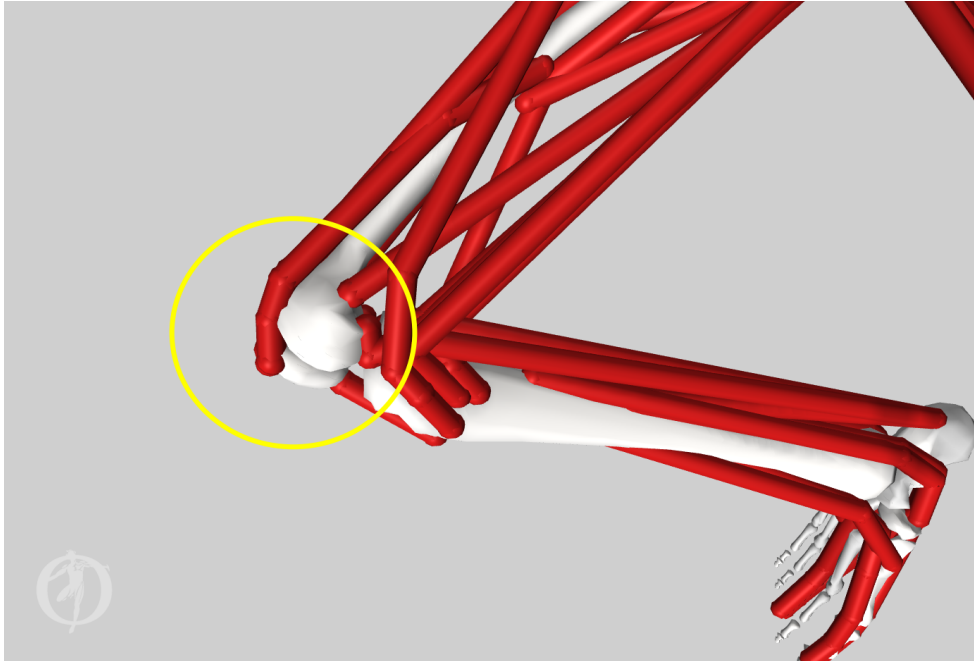
## 2.4 Pohyb kostí

I když je pohybem kostí myšlena pouze rotace o určitý úhel, který určuje kloub (viz Kapitola 2.3), musí během něj dojít i k deformaci svalu. Pokud je sval například reprezentován elipsoidem, tak jsou obecně koncové body jeho hlavní osy na povrchu kosti. Pohybem kosti se ve většině případů změní její délka a musí být opravena délka ostatních os tak, aby byl zachován objem původního elipsoidu.

V jiných případech zase mohou být ke kostem uchyceny i body, které leží mimo kost – např. vodící body u Bézierových ploch. Jako další příklad bodů uchycených na kost lze uvést i výše zmíněné *via points* využívané při reprezentaci svalu pomocí *line of action*. Díky tomu, že se *via points* deformují spolu s kostí, je v tomto přístupu usnadněna deformace svalů.

Ukázku *via points* lze pozorovat na Obrázku 2.2, kde jsou u kolena vidět krátké zlomy svalového vlákna způsobené právě dodefinovanými *via points*.

Díky nim se svalové vlákno neprotíná s kostí (resp. průnik je minimální vůči průniku, který by byl způsoben myslitelným vláknem mezi počátkem a koncem daného svalu).

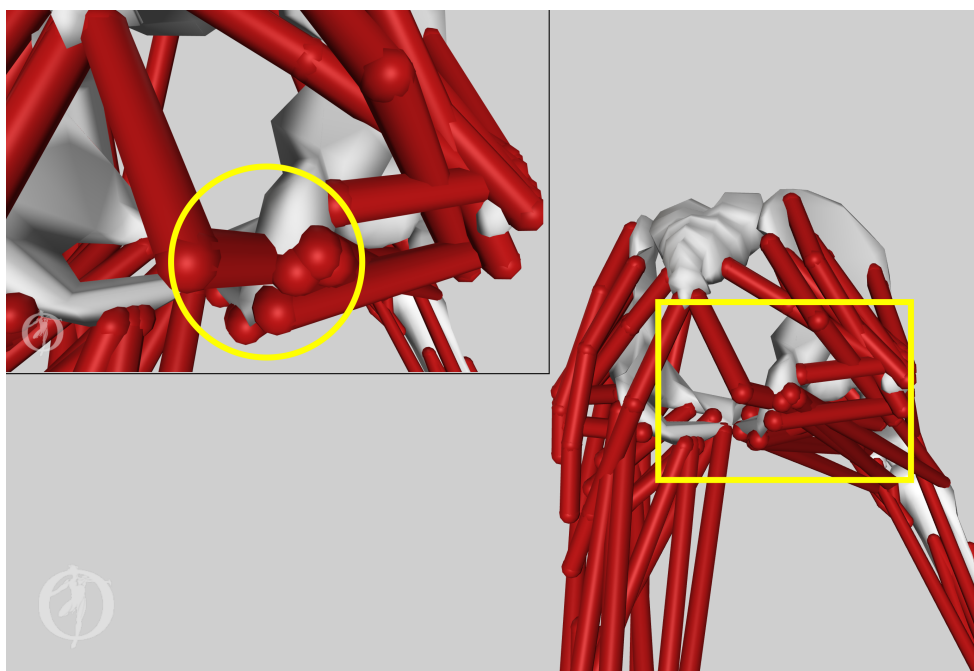


Obrázek 2.2: OpenSim [7], BothLegs model, via points u kolenního kloubu

## 2.5 Problém s deformací svalu

Nezávisle na reprezentaci mají však svaly jeden společný problém – bez dodatečné pomoci bude poměrně často docházet k průnikům kusu svalu a kosti. Svaly jsou totiž mnohem flexibilnější než kosti. Při pohybu v kloubu dojde k deformaci a při ní může lehce dojít k průniku. Problém nastává především při mezních pohybech – pohybech, kdy je v kloubu ohyb blízko maximálnímu možnému úhlu vůči klidovému stavu. Problém s průnikem kosti a svalu se vyskytuje v mnoha řešeních MSM. V reálném světě je to samozřejmě nemožné, ale v modelech, které se snaží být rychlé na úkor přesnosti, je výskyt tohoto problému častý.

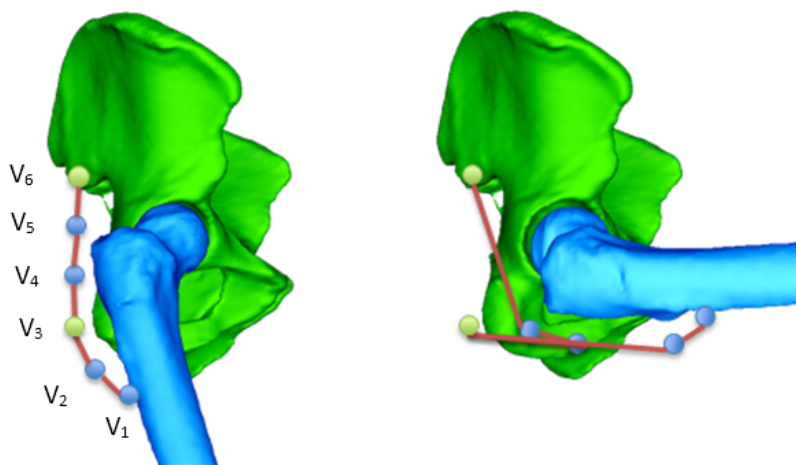
Příklad lze vidět na Obrázku 2.3, který zobrazuje tento problém na jednom ze základních modelů (model BothLegs) dostupných v rámci OpenSim. I přístup, který bude upravován v rámci této bakalářské práce (metoda Luca2018), tímto nešvarem trpí a oprava průniků je jedním ze dvou cílů této práce.



Obrázek 2.3: OpenSim [7], BothLegs model, ukázka průniku pánevní kosti a vlákna svalu, při ohybu dolní končetiny

### 3 Metoda deformace vláken – Luca2018

Metoda deformace vláken nazvaná Luca2018 byla popsána v článku Automated Generation of Three-Dimensional Complex Muscle Geometries for Use in Personalised Musculoskeletal Models [6]. Jedná se o metodu deformace svalu obecně použitelnou i pro modelování pomocí svalových vláken (line of action). Všechny body na vlákně, vyjma koncových, které jsou uchyceny pevně na kost, se musí podrobit transformaci, a tím dochází k jejich deformaci. Uchycení jen koncových bodů je zvoleno, protože zjistit, na kterou kost uchytit bod, není tak jednoduché, jak by se mohlo zdát. Nejjednodušší by bylo například bod uchytit na nejbližší kost, což se však rychle ukáže jako nesprávné (viz Obrázek 3.1). Právě proto jsou uchyceny jen koncové body, u kterých je kost, na kterou jsou přichyceny, jednoznačně daná.



Obrázek 3.1: Možné přichycení bodů na nejbližší kost a jeho důsledky [6]

Samotná metoda deformace je poměrně jednoduchá – pro každý bod vlákna máme informaci o jeho původní (rest-pose) poloze a známe jeho dvě nejbližší kosti. Výpočet aktuální pozice bodu lze nalézt v Rovnici 3.1. Z popisu a rovnice pro výpočet bodů si lze snadno domyslet, že i tento přístup bude

trpět na výše uvedené kolize svalových vláken s kostmi.

$$V'_i = \sum_{j=1}^2 w_{ij} \cdot [R_j \cdot T_j] \cdot V_i \quad i = 1 \dots n \quad (3.1)$$

$V'_i$  v Rovnici 3.1 je nová pozice bodu,  $V_i$  je rest-pose pozice bodu,  $R_j$  je matice rotace kosti na indexu  $j$  a  $T_j$  je transformační matice této kosti.

V rovnici se také vyskytuje  $w_{ij}$ , což je váha, která je každé kosti přidělena. Tato váha splňuje podmínku  $\sum_{j=1}^2 w_{ij} = 1$  pro každý bod  $i$ . Autoři článku dále uvádějí, že výpočet parametru lze provést mnoha způsoby, ale aktuálně je  $w$  pro bod na indexu  $i$  vypočteno pomocí kvadratické funkce  $f(t)$ :

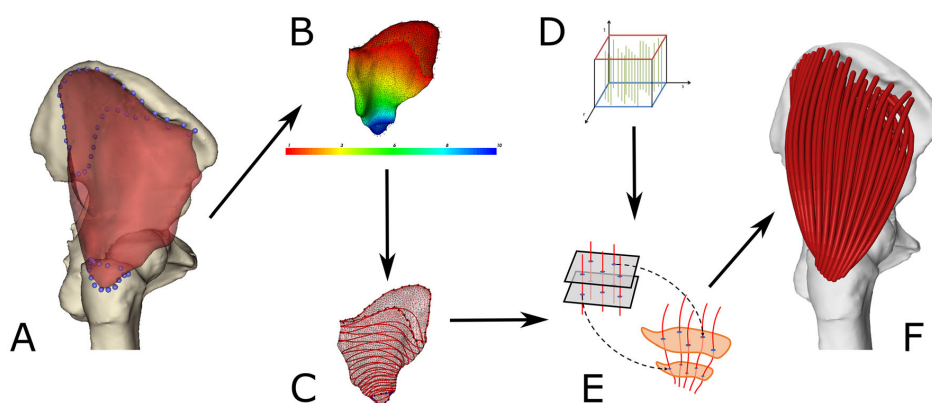
$$w_{i1} = f(t) = at^2 + bt + c; \quad w_{i2} = 1 - w_{i1} \quad (3.2)$$

Protože je kladen požadavek, aby  $f(0) = 1$  a  $f(1) = 0$ , autoři 3 neznámé parametry z Rovnice 3.2 dopočítali tak, že zbyla pouze jedna neznámá, což je parametr  $a$ . Uvádějí totiž, že aby funkce nabývala výše uvedených hodnot, je nutné položit  $b = -(a + 1)$  a  $c = 1$ . Parametr  $a$  bohužel přesně určit nelze a sami autoři ho pro svůj článek určovali metodou pokusu a omylu. Tento parametr je však poměrně důležitý, neboť jeho správné nastavení umožňuje minimalizovat množství průniků svalových vláken a kostí.

Článek dále poukazuje na fakt, že pro deformaci lze použít i vlákna vygenerovaná přístupem popsáním v článku Real-Time Modelling of Fibrous Muscle [3]. Tento přístup bude podrobně rozebrán v následující kapitole.

## 4 Dekompozice svalů

Jak již bylo zmíněno v Kapitole 2.1.1, při reprezentaci pomocí line of action nelze sval reprezentovat pomocí mnoha vláken, neboť definice každého jednoho vlákna vyžaduje zásah uživatele. Tento přístup je velmi nepraktický, a proto jsou vyvíjeny snahy o automatizaci tohoto procesu. V této kapitole bude popsán automatický přístup, který problém částečně řeší a vychází z článku Real-Time Modelling of Fibrous Muscle [3].



Obrázek 4.1: Postup modelování svalů [6]. A – vstup, B – skalární pole na povrchu, C – izočáry skalárního pole, D – šablona vláken, E – mapování řezů šablony na řezy svalu určené izočárami, F – výstup

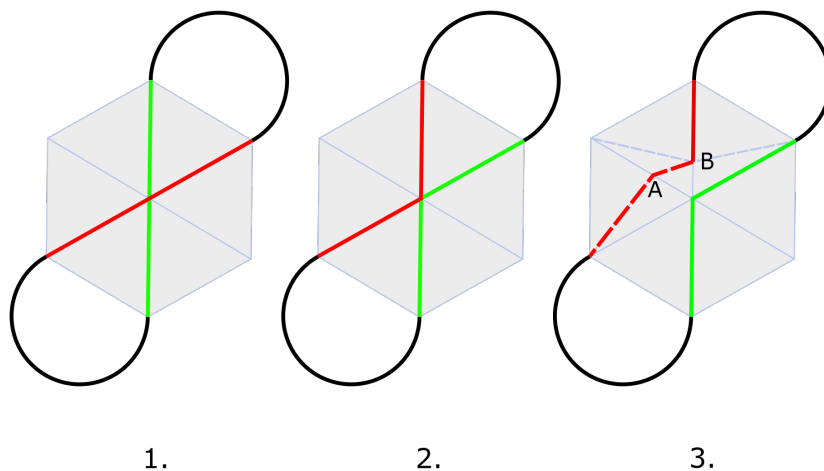
Celý algoritmus je znázorněn na Obrázku 4.1 a bude podrobně popsán níže. Stručně řečeno, algoritmus je založen na zpracování svalu reprezentovaného trojúhelníkovou sítí. Z té je poté vyříznuta úponová oblast (oblast, kde je sval upnut na kost), vytvořeno skalární pole a v něm nalezeny izočáry. Dále do procesu vstoupí předdefinované šablony svalových vláken a pomocí mapování jsou vytvořena vlákna podobná těm na všech obrázcích z OpenSim (viz výše).

### 4.1 Vstup

Algoritmus dokáže zpracovat libovolný sval reprezentovaný pomocí trojúhelníkové sítě, o kterém máme dvě podstatné informace – jeho úponové oblasti a jeho vnitřní strukturu (parallel, fanned, curved nebo pennate). Úponové

oblasti jsou popsány pomocí bodů tvořících uzavřenou neprotínající se plochu. Tuto oblast určuje expert. Ukázka úponových oblastí vyznačených body je vidět na Obrázku 4.1 v části A, kde jsou obě oblasti označeny modrými body. Největším problémem je však definovat počet úseků, ze kterých se každá line of action musí skládat, a dále i samotný počet line of action, ze kterých se bude skládat sval. I tyto dvě hodnoty však musí být definovány ručně na vstupu, což není jednoduché, neboť optimální hodnoty se liší sval od svalu.

## 4.2 Úponové oblasti



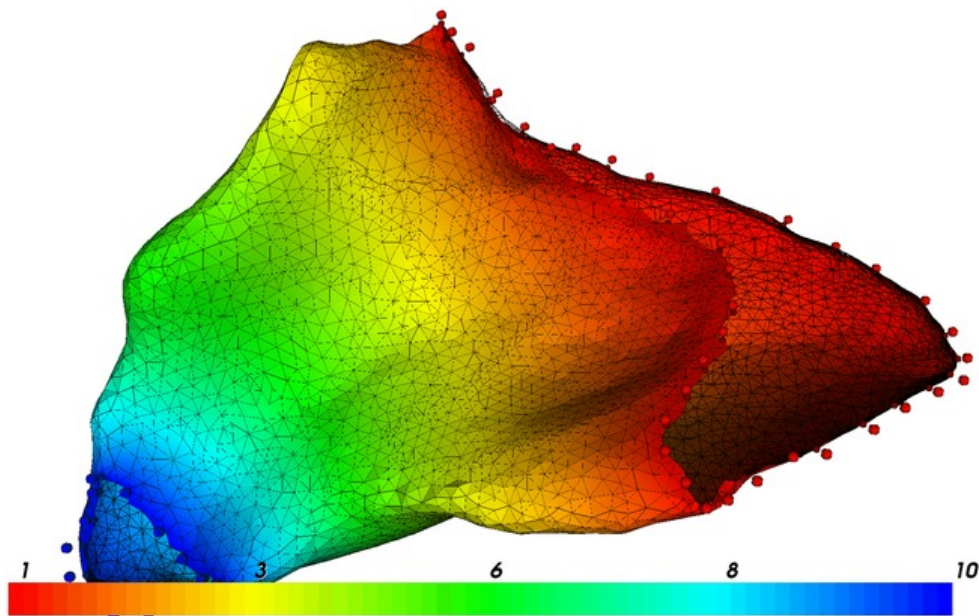
Obrázek 4.2: Protínající se (1), dotýkající se (2) a validní (3) cesta přes vrchol

Sval ze vstupu se však nemusí přímo dotýkat kosti. V tomto případě musí dojít k úpravě pozic bodů úponových oblastí tak, aby byla minimalizována vzdálenost mezi svalem a kostí. Jelikož je sval reprezentován za pomoci trojúhelníkové sítě, je výhodné, aby všechny body úponové oblasti byly vrcholy trojúhelníků sítě. Proběhne tedy posun bodů do nejbližšího vrcholu. Detaily algoritmu, který posun provede, jsou popsány v článku [3]. Dále je důležité najít úponovou oblast (resp. její hranici) – cestu skrz nové (posunuté) body. Avšak při pohybu bodů může dojít k deformaci původní hranice úponové oblasti (především k průniku/doteku), což je potřeba vyřešit. Algoritmus na opravu této cesty najde všechny průniky a opraví je na doteky (prohozením



pořadí bodů v jedné polovině cesty – viz změna v Obrázku 4.2-1 a 4.2-2), a poté opraví i doteky přidáním nového bodu do sítě (viz Obrázek 4.2-3, do sítě byly přidány nové body A a B). Poté, co proběhne tento algoritmus nad všemi body úponové oblasti, vznikne nám ohraničená oblast, která dělí síť na dvě části. Menší z těchto částí je považována za úponovou oblast a její trojúhelníky jsou zahozeny.

### 4.3 Tvorba skalárního pole



Obrázek 4.3: Pole skalárních hodnot na povrchu gluteus medius [3]

Poté, co jsou ze svalu oříznuty obě úponové oblasti, můžeme přejít ke tvorbě harmonického skalárního pole hodnot. Pro každý vrchol je spočtena hodnota  $u_i$  taková, že na hranici první úponové oblasti bude lineární funkce dosahovat svého minima  $k_{min}$  a v druhé svého maxima  $k_{max}$ . K jiným lokálním extrémům docházet nesmí. Příklad takového pole skalárních hodnot lze vidět na Obrázku 4.1B a v detailu na Obrázku 4.3.

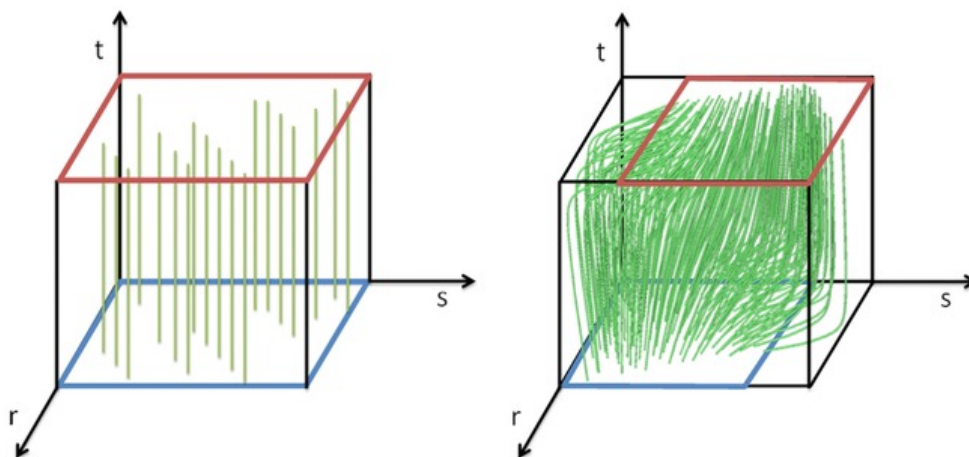
### 4.4 Izočáry

Pole obsahuje dle definice pouze hodnoty z intervalu  $\langle k_{min}, k_{max} \rangle$ , a tedy lze interval rovnoměrně rozdělit na  $y - 1$  částí, přičemž počet částí musí

definovat uživatel. Poté, co je interval rozdělen, vznikne  $y$  hodnot  $s_1$  až  $s_y$ . Ve skalárním poli najdeme izočáry, které odpovídají těmto hodnotám. Tím nám vznikne sada obrysových čar pro daný sval (viz Obrázek 4.1C).

## 4.5 Šablony svalů

Jako další do procesu vstoupí předdefinované šablony (viz Obrázek 4.4), ze kterých je jedna vybrána dle definované vnitřní struktury svalu (viz Kapitola 4.1). Tato šablona poté projde afinní transformací tak, aby se úponové oblasti šablony a našeho svalu ze vstupu co nejvíce shodovaly. Poté je i šablona rozdělena na stejný počet obrysů jako sval. Tím vznikne sada obdélníkových obrysů.



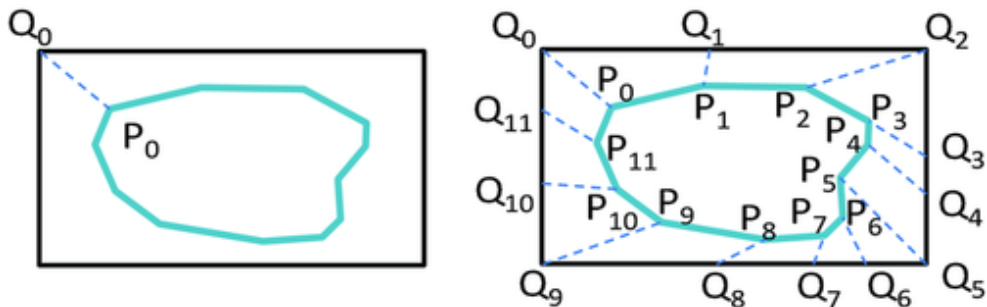
Obrázek 4.4: Příklad šablony pro vnitřní strukturu svalů – parallel (levá) a pennate (pravá) [3]

## 4.6 Přiřazení vrcholů

Aktuálně se algoritmus nachází ve fázi, kdy má sval s jeho obrysy (vytvořenými z pole skalárů) – označme tyto obrysy  $O_1$  a šablonu s jejími obrysy, ty označme  $O_2$ . Dále předpokládáme, že  $O_1$  jsou tvořeny  $m$  úseky, kde  $m > 4$  a  $O_2$  jsou tvořeny  $n$  úseky a bude platit, že  $n < m$ .

Dalším úkolem je najít 1 ku 1 přiřazení vrcholů  $O_1$  a  $O_2$ . Protože  $O_2$  se skládají z méně úseků, musí být strany těchto úseků rozděleny tak, aby vzniklo  $m$  úseků. Nyní platí, že  $n = m$  a ke každému vrcholu z  $O_1$  lze přiřadit vrchol z  $O_2$ .

Jak lze vidět na Obrázku 4.5, předpokládáme, že první vrchol z  $O_1 = Q_0$  odpovídá vrcholu z  $O_2 = P_0$ . Toto přiřazení je pro první řez provedeno v duchu Euklidovské vzdálenosti (snažíme se najít dva body s nejmenší vzdáleností – jeden z  $O_1$  a druhý z  $O_2$ ).

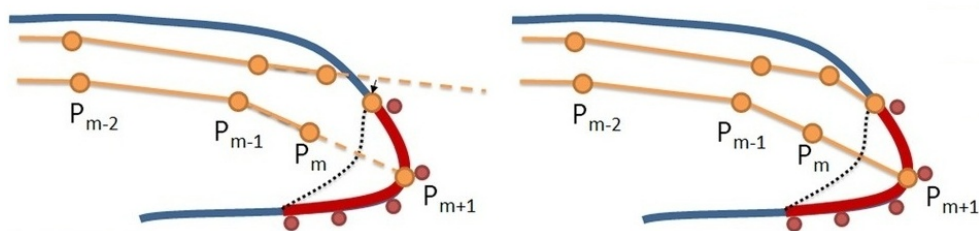


Obrázek 4.5: Příklad přiřazení [3]

Pro následující řezy se počáteční vrcholy hledají s pomocí Frechetovy vzdálenosti – tj. hledáme přiřazení dvojic vrcholů mezi dvěma obrysy tak, aby součet vzdáleností všech dvojic byl minimální a zároveň aby nedocházelo k průnikům spojnic těchto dvojic. Poté, co je nalezeno přiřazení počátečních vrcholů, je provedena parametrizace délkou oblouku – tzn. zajistíme, aby si body z obou obrysů proporcionálně odpovídaly (vzdálenost od  $P_0$  do  $P_i$  vydělená obvodem  $O_2$  bude stejná jako vzdálenost z  $Q_0$  do  $Q_i$  vydělená obvodem  $O_1$ ). Tím získáme přiřazení mezi polygony (obrysy)  $O_1$  a  $O_2$ .

## 4.7 Prodloužení vláken

Vlákna, reprezentovaná lomenou čarou  $P_0 \dots P_m$ , vzniklá metodou popsanou výše obvykle nemají koncové body přímo na povrchu úponové oblasti.



Obrázek 4.6: Proces prodloužení vláken [3]

Tato vlastnost je nežádoucí a je nutné vlákna prodloužit. To proběhne v postprocessingu, kdy je vlákno prodlouženo o nový bod takový, že je leží v úponové oblasti a je nejbližší k pomyslné přímce procházející body  $P_{m-1}$  a  $P_m$  –

tzn. jde o průsečík přímky a úponové oblasti nebo o bod z úponové oblasti, jehož vzdálenost od přímky je minimální (viz Obrázek 4.6 na předcházející straně).

## 5 Catmull-Rom křivky

V Kapitole 4 byla postupně popsána tvorba svalových vláken skládajících se z lomených čar. V Kapitole 4.1 bylo zdůrazněno, že počet lomených čar, ze kterých se vlákno musí skládat, aby bylo dostatečně reprezentativní, je pro každý sval odlišný (např. pro *Gluteus minimus* postačí i jedna část, pro *Gluteus maximus* je vhodné mít částí alespoň 14). Vystala tedy otázka, zda by se nedal tento počet stanovit automaticky a zda by se lomené čáry nedaly nahradit nějakou parametrickou křivkou. Jednou z možností je právě užití kubické Catmull-Rom křivky, která byla pro tuto práci vybrána díky svým vlastnostem popsaným níže.

Catmull-Rom křivky jsou parametrické křivky původně popsané E. Catmullem a R. Romem v článku CLASS OF LOCAL INTERPOLATING SPLINES [1]. Kubická Catmull-Rom křivka musí být popsána pomocí minimálně čtyř kontrolních bodů  $P_{1-4}$ . V základní formulaci je křivka v bodě  $P_i$  definována pomocí tečen vypočtených z dvou okolních bodů  $\tau(P_{i-1} - P_{i+1})$ . Platí, že jakákoliv Catmull-Rom křivka interpoluje všechny své kontrolní body (tzn. přímo jimi prochází) a splňuje podmínku  $C^1$  spojitosti. Obecně však Catmull-Rom křivky nespadají celé do konvexní obálky svých kontrolních bodů [12].

### 5.1 Výpočet křivky

Pro výpočet bodů křivky uvažujme, že kontrolní body jsou zadány ve 3D prostoru ve tvaru  $\mathbf{P}_i = [x_i \ y_i \ z_i]^T$ . Bod  $C$  Catmull-Rom křivky lze vypočítat rekurzivně pomocí následujícího vztahu [11, 12]:

$$\mathbf{C} = \frac{t_2 - t}{t_2 - t_1} \mathbf{B}_1 + \frac{t - t_1}{t_2 - t_1} \mathbf{B}_2$$

Kde:

$$\mathbf{B}_1 = \frac{t_2 - t}{t_2 - t_0} \mathbf{A}_1 + \frac{t - t_0}{t_2 - t_0} \mathbf{A}_2$$

$$\mathbf{B}_2 = \frac{t_3 - t}{t_3 - t_1} \mathbf{A}_2 + \frac{t - t_1}{t_3 - t_1} \mathbf{A}_3$$

$$\mathbf{A}_1 = \frac{t_1 - t}{t_1 - t_0} \mathbf{P}_0 + \frac{t - t_0}{t_1 - t_0} \mathbf{P}_1$$

$$\mathbf{A}_2 = \frac{t_2 - t}{t_2 - t_1} \mathbf{P}_1 + \frac{t - t_1}{t_2 - t_1} \mathbf{P}_2$$

$$\mathbf{A}_3 = \frac{t_3 - t}{t_3 - t_2} \mathbf{P}_2 + \frac{t - t_2}{t_3 - t_2} \mathbf{P}_3$$

$$t_{i+1} = [\textit{distance}(P_{i+1}, P_i)]^\alpha + t_i$$

$$\textit{distance}(P_{i+1}, P_i) = \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2 + (z_{i+1} - z_i)^2}$$

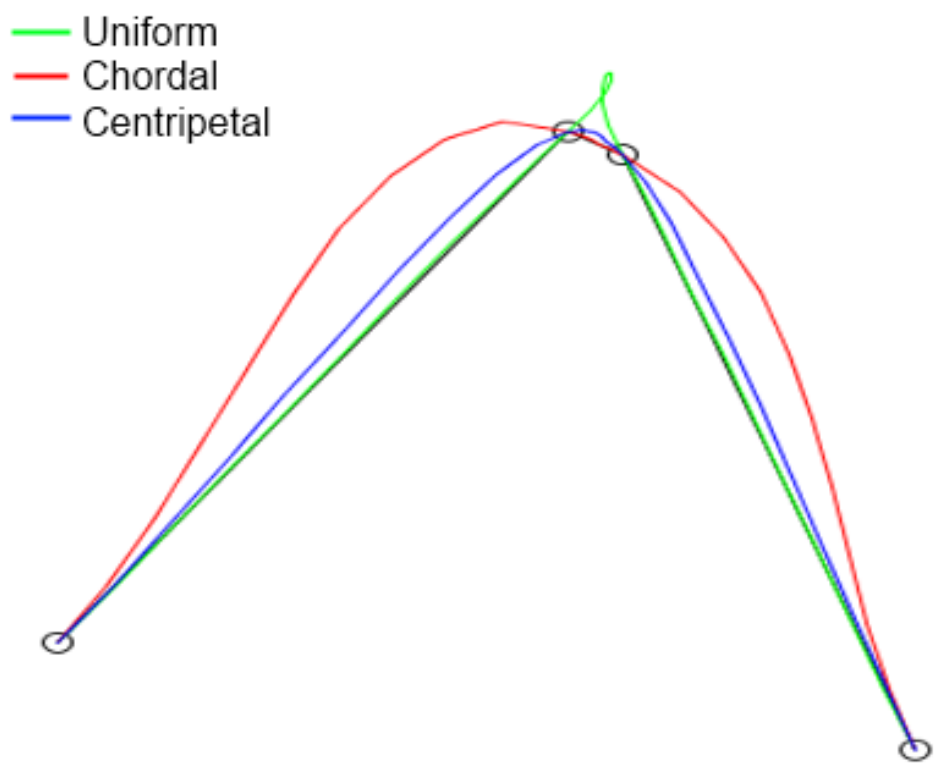
Ve vzorci se dále vyskytuje proměnná  $\alpha$ , která je z intervalu mezi 0 a 1, a určuje jí uživatel. Dále předpokládáme, že  $t_0 = 0$  a  $t$  je číslo z intervalu  $\langle t_a, t_b \rangle$ , kde  $t_a$  a  $t_b$  jsou hodnoty  $t_i$  náležící k prostředním kontrolním bodům křivky. Např. pro  $i = 0, 1, 2, 3$  je  $t$  z rozsahu  $\langle t_1, t_2 \rangle$ .

## 5.2 Vliv hodnoty $\alpha$

Dle volby hodnoty  $\alpha$  ve vzorci pro výpočet lze Catmull-Rom křivky dělit na tři základní druhy:

- uniformní –  $\alpha = 0$ ,
- chordální –  $\alpha = 1$ ,
- centripetální –  $\alpha = 0.5$ .

Pro tuto práci je nejzajímavější centripetální Catmull-Rom křivka, neboť při volbě parametru  $\alpha = 0.5$  lze dokázat, že křivka v daném úseku neprotne sama sebe (jako například uniformní křivka na Obrázku 5.1. Dále na rozdíl od chordální Catmull-Rom křivky je křivka blíže ke kontrolním bodům a nevzdaluje se od nich tolik. Jako poslední výhodu oproti ostatním Catmull-Rom křivkám lze uvést, že centripetální křivka nevytvoří ostrý zlom.



Obrázek 5.1: Závislost na parametru  $\alpha$  [11]

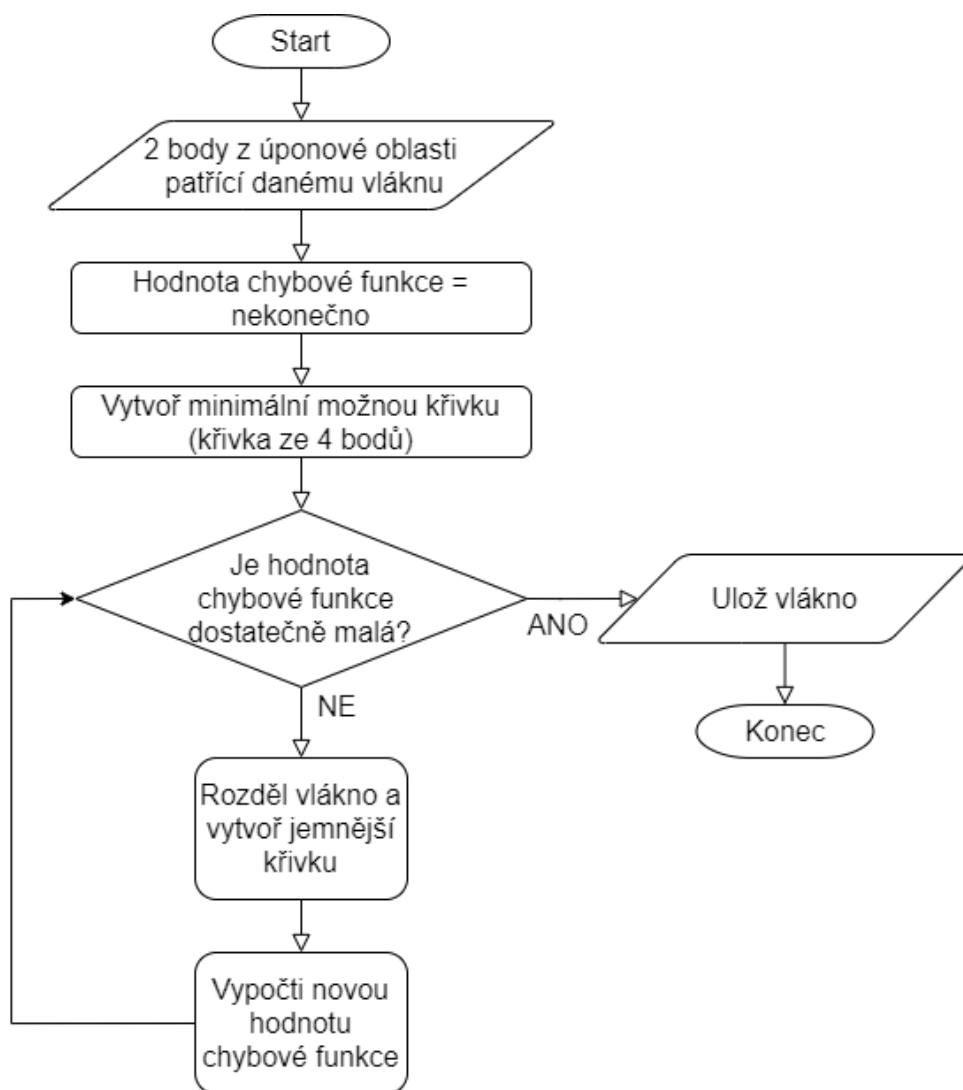
# 6 Návrh adaptivního algoritmu

V bakalářské práci bude jedním z úkolů navrhnout a implementovat algoritmus pro tvorbu svalových vláken z adaptivního počtu segmentů. K tomu bude využita i centripetální Catmull-Rom křivka, a to z výše uvedených důvodů. Pro určení správného počtu segmentů bude nutné využít adaptivní algoritmus. Adaptivní algoritmy jsou speciální algoritmy využívané například ve strojovém učení, které za běhu programu se mění své chování na základě aktuálních informací, například se snaží minimalizovat danou chybovou funkci. Chybová funkce určuje, jak špatné je aktuální řešení a tím, že snižujeme její hodnotu, se snažíme získat optimální řešení. Samozřejmě nalezení optimálního řešení velmi závisí na vhodném návrhu chybové funkce.

## 6.1 Obecný algoritmus

Algoritmus bude spočívat v adaptivním dělení úseku od jedné úponové oblasti k té druhé. V původní verzi jsou pro reprezentaci těchto úseků využity pouze lomené čáry, což by zde mohlo způsobit problémy. Proto pro adaptivní algoritmus budou vnitřně body svalových vláken tvořit množinu kontrolních bodů Catmull-Rom křivky. Adaptivní dělení bude probíhat následovně: máme 2 základní body a mezi nimi chceme najít optimální počet úseků, na které když dané vlákno rozdělíme, tak již bude vlákno dostatečně reprezentující. Algoritmus tudíž začne s minimem (křivka se 4 kontrolními body, které budou rovnoměrně rozloženy). Dále se bude křivka zjemňovat (budou přibývat nové kontrolní body) a bude vypočtena hodnota chybové funkce mezi dvěma po sobě jdoucími iteracemi. Pokud bude vypočtená hodnota pod určitým prahem, křivka se již dále dělit nebude. Vývojový diagram algoritmu pro jedno vlákno lze vidět na Obrázku 6.1.





Obrázek 6.1: Vývojový diagram

## 6.2 Problém více vláken a jeho řešení

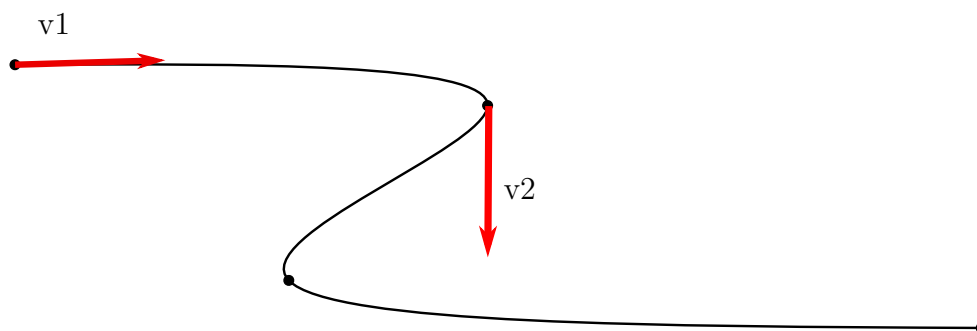
Sval se neskládá pouze z jednoho vlákna, a tudíž vyvstává problém s opakováním výpočtů. Pro každé vlákno bychom totiž opakovali to samé zjemnění, jen bychom se zajímali o jiný bod daného obrysu. Každé zjemnění vyžaduje nalezení izochar ve skalárním poli až po nalezení přiřazení vrcholů (viz. Kapitoly 4.4 až 4.6). Již jen z teoretického popisu je zřejmé, že tyto operace nejsou jednoduché a jejich opakování může zbytečně prodlužovat dobu běhu programu. Proto si již vypočtený obrys uložíme pro následující svalová vlákna. Tím se ušetří mnoho potenciálně zbytečných výpočtů za cenu paměti.

## 6.3 Chybová funkce

Důležité pro správné fungování algoritmu je vhodné zvolení chybové funkce. Obecně se křivky mohou dopouštět dvou nežádoucích jevů – budou moc vzdálené od sebe, nebo dojde ke zkroucení křivky do tvaru S.

### 6.3.1 Zkroucení křivky

Zkroucení lze například určit pomocí tečen v bodech zlomu. Zde se opět uplatní Catmull-Rom křivka především proto, že je spojitá a má ve všech bodech derivaci. Pokud dvě po sobě jdoucí tečny budou svírat příliš velký úhel, znamená to, že se křivka kroutí, což může být nežádoucí, a pokud se tento jev objevil v nové iteraci, pravděpodobně se spokojíme s předchozí iterací (lepší nepřesná křivka než zkroucená). Ilustraci takového zkroucení, které je s vysokou pravděpodobností nežádoucí, lze vidět na Obrázku 6.2.



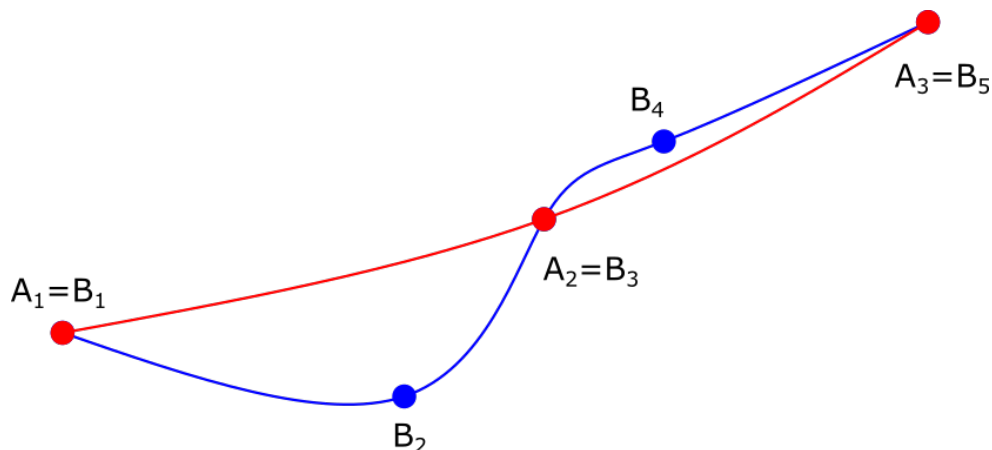
Obrázek 6.2: Příklad nevhodného zkroucení – úhel mezi tečnami ( $v_1$  a  $v_2$ ) v kontrolních bodech je příliš veliký

### 6.3.2 Vzdálenost křivek

Dále je potřeba do hodnocení zahrnout do chybové funkce i vzdálenost křivek. Díky tomu, že křivky mají shodné body (minimálně počáteční a koncový) můžeme například křivku navzorkovat a mezi jednotlivými vzorky spočítat vzdálenost. Suma čtverců těchto vzdáleností může být chybovou funkcí, kterou se budeme snažit minimalizovat. Dále k tomuto můžeme využít například i výše zmíněnou Frechetovu vzdálenost křivek.

## 6.4 Dělení vlákna

Je poměrně důležité a zajímavé zamyslet se nad dělením vlákna. Existuje několik logických možností, jak vlákno dělit – například jej vždy půlit (tzn. počet úseků by byl vždy 4,8,16,32...), pak ale přichází v úvahu i například dělit jej jen úseky, které nevyhovují požadavkům (tzn. hodnota chybové funkce je příliš vysoká). Příklad této situace lze vidět na Obrázku 6.3.



Obrázek 6.3: Obrázek ukazující dělení vlákna (viz text v Kapitole 6.4)

Na obrázku je pouze úsek vlákna, ale pro ukázkou postačí. Je zřejmé, že úseky  $A_1$  až  $A_2$  a  $B_1$  až  $B_3$  jsou velmi rozdílné, a tudíž (správně zvolená) chybová funkce je označí za nevyhovující a dělení na tomto úseku musí pokračovat. Jiná situace však nastává na úseku  $A_2$  až  $A_3$  a úseku  $B_3$  až  $B_5$ . Tyto úseky jsou velmi podobné a je otázkou, zda je nutné je dále dělit. Předpokládejme, že chybová funkce tento úsek označila již za vyhovující (její hodnota vyšla menší než stanovený práh), a proto bude nový úsek tvořen posloupností  $B_1$  až  $B_3$  a  $A_2$  až  $A_3$ . Část od  $A_2$  až k  $A_3$  bude navíc označena jako dále nedělitelná. Dělit se bude jen úsek  $B_1$  až  $B_3$ .

# 7 Návrh detekce kolizí

Vlákna, která budou vytvořena pomocí metody navržené v Kapitole 6, je dále potřeba deformovat, a tím simulovat pohyb modelu. Přesně pro tento účel slouží metoda Luca2018 popsaná v Kapitole 3. Jak již bylo zmíněno v dané kapitole, ani metoda Luca2018 nedokáže v jejím aktuálním provedení zabránit kolizím. V této kapitole bude popsán návrh, jak rozšířit tuto stávající metodu, aby byla schopna kolize nejen detekovat, ale i opravit.

## 7.1 SDF

SDF [4] neboli Signed distance field (případně function) je poměrně často využívaná struktura pro detekci kolizí. Jedná se o funkci, která nabývá následujících hodnot ( $x$  je zkoumaný bod a  $d$  je libovolná metrika určující vzdálenost):

$$f(x) = \begin{cases} d(x, \text{objekt}) & \text{if } x \text{ leží mimo objekt} \\ -d(x, \text{objekt}) & \text{if } x \text{ leží uvnitř objektu} \end{cases} \quad (7.1)$$

Právě podle znaménka, které nám SDF vrátí, můžeme o kterémkoliv bodu scény zjistit, zda leží uvnitř či mimo objekt a dokonce i jeho vzdálenost k objektu.

## 7.2 Návrh detekce kolizí

Druhou částí této práce je návrh a implementace algoritmu, který bude detekovat kolize. Jak již bylo popsáno výše, k vyřešení tohoto problému lze využít SDF. Pomocí této struktury by se daly detekovat kolize bodů s kostí. Postup detekce je tudíž jednoznačný – otestujeme bod proti SDF a ke kolizi došlo, pokud vzdálenost vyšla záporná. Pro opravu kolizí budeme potřebovat ještě jednu důležitou informaci, což je směr, kterým bod budeme muset posouvat. Již existující knihovny, které s SDF pracují, tuto informaci také umí vrátit, tudíž při volbě správné knihovny bude stačit pouze k bodu, u kterého byla detekována kolize, přičíst vektor směřující k povrchu vynásobený délkou.

Otázkou zůstává, jaké všechny body testovat. Je zde možnost testovat přímo body, ze kterých se svalové vlákno skládá. Tento přístup je snazší, body již

mít budeme, stačí je pouze otestovat. Druhá možnost spočívá v navzorkování jejich spojnic a testovat tím i body mezi jednotlivými body vlákna. Šlo by také zapojit do tohoto procesu Catmull-Rom křivku a pomocí ní rekurzivně dělit křivku napůl, dokud bychom neměli dostatečnou jistotu, že nedochází k průniku po celém oblouku křivky. Tento přístup by vyžadoval i úpravu řídicích bodů vlákna, neboť pokud by došlo k detekci kolize uprostřed úseku vlákna, jediným způsobem opravy by bylo úsek rozdělit tímto bodem na dva úseky, opravit kolizi v nově vzniklém bodu a pokračovat dále.

# 8 Adaptivní algoritmus pro určení počtu segmentů

Celý projekt je implementován v jazyce C++ (případně C) a pro vizualizaci je využito externích knihoven, kterými jsou například VTK [10] či OpenSim [7]. Tato práce bude respektovat původní jazyk celého projektu a bude také psána v jazyce C++.

## 8.1 Implementovaný algoritmus

Algoritmus 1 je jednou ze dvou hlavních částí bakalářské práce. Implementuje navržené řešení z Kapitoly 6, a především rozvíjí původní kód projektu. V původní verzi vše probíhalo pouze jednou a počet vláken musel být definován uživatelem. V aktuální verzi, která je výsledkem této bakalářské práce, došlo k přechodu na adaptivní tvorbu vláken. Prostřednictvím opakování kroků pro tvorbu svalu je dosaženo toho, že není nutné specifikovat výsledný počet vláken dopředu (především protože člověk tento počet nedokáže odhadnout přesně), ale stačí začít od jakéhokoliv kladného počtu a adaptivní algoritmus bude konvergovat k optimu (samozřejmě za předpokladu, že optimum je menší než zadaný počáteční počet – vlákna mohou jen přibývat).

Na počátku proběhne inicializace tří důležitých *std::vectorů*. První z nich (*computed*) obsahuje celá čísla a určuje, jak bude probíhat dělení (podrobně viz Kapitola 8.2). Zbylé dva *vector*y obsahují instance třídy *CatmullRomSpline* (podrobně v Kapitole 8.5) a jsou využívány především pro určení, zda je potřeba v dělení pokračovat (viz Kapitola 8.3), ale i při filtrování nevhodných bodů (viz Kapitola 8.4). Všechny tyto *vector*y jsou atributy třídy *vtkMAFMuscleDecompositionSlicing*, ve které je tento algoritmus implementován.

Jak již bylo výše naznačeno, dále v cyklu *do-while* probíhá tvorba svalu, dokud není počet vláken dostatečný. Všechny funkce použité v tomto cyklu zde nebudou podrobněji rozepsány, neboť jejich podrobný popis je již uveden v kapitolách, na které jsou u daných funkcí odkazy. Tyto funkce se využívaly už v původní verzi MWP2 a došlo jen k malým, především optimalizačním, úpravám. To zahrnuje například generování skalárního pole,

které je stejné nehledě na počet segmentů – tzn. není potřeba ho generovat v každém cyklu, stačí ho vygenerovat během prvního a uložit do paměti. Další optimalizace probíhaly za použití již zmíněného *vectoru computed* – díky němu bylo zabráněno počítání již spočtených částí a bylo zajištěno, že se budou počítat pouze segmenty, které minulou iteraci neexistovaly. Podrobnější popis i funkce, která říká, zda je již počet segmentů svalových vláken uspokojující, bude k nalezení v Kapitole 8.2.

---

**Algoritmus 1:** `AdaptivelyCreateFibres()` – Adaptivní dělení vláken

---

```
vygeneruj nové std::vector this->computed, this->splinesCurrent a
  this->splinesNew;
i = 0;
načti počáteční požadovaný počet segmentů;
do
  i = i + 1;
  vytvoř šablony vláken svalů (viz 4.5);
  rozděl šablonu každého vlákna na požadovaný počet segmentů;
  načti sval ze vstupu (viz 4.1);
  if skalární pole nebylo vytvořeno then
    | vytvoř a ulož skalární pole (viz 4.3);
  end
  najdi izočáry (viz 4.4);
  najdi přiřazení vrcholů mezi šablonami (viz 4.6);
  aktualizuj Catmull-Rom křivky v splinesNew;
  zdvojnásob požadovaný počet segmentů;
while !ErrFunction(i) // dosažen max. počet segmentů;
```

---

## 8.2 Souhrnná chybová funkce

Další uvedený pseudokód (Algoritmus 2) se týká souhrnné chybové funkce. Tato funkce má vcelku jednoduše znějící úkol – vrací *false*, pokud je zapotřebí nový cyklus, který vytvoří svalová vlákna z více segmentů. Není to však tak jednoduché. Z důvodu úspory času bylo zapotřebí zavést optimalizace, z nichž tou nejpodstatnější je *vector computed*. Dále je v této funkci na konci vždy provedeno prohození *vectorů splinesCurrent* a *splinesNew* a nepotřebné křivky jsou vynulovány, aby byly připraveny pro další použití.

---

**Algoritmus 2:** ErrFunction(*iRound*) – Souhrnná chybová funkce

---

```
// iRound – aktuální počet opakování
// computed – std::vector – hodnoty -2 až 3 (viz výše)
inicializuj std::vector new_vec a filter_vec;
if iRound < 2 then
    foreach i in computed do
        if i > 0 then
            | zapiš do new_vec 0 a 1
        else
            | zapiš do new_vec 0 a -1
        end
    end
else
    foreach i in computed do
        aplikuj přepisovací pravidla na dané i;
        //zde bude využito chybové funkce SegmentsSimilar()
        výsledek zapiš do new_vec;
        if FilterSegment() then
            | do filter_vec zapiš 2
        else
            if i != 1 then
                | do filter_vec zapiš i
            else
                | do filter_vec zapiš 3
            end
        end
    end
end
if filter proběhl alespoň 1x then
    | compute = filter_vec;
    | return false
end
if další dělení je potřeba then
    | compute = new_vec;
end
return další dělení je potřeba;
```

---



### 8.2.1 Vector computed a jeho obsah

*Vector computed* je *vector* čísel, který má stejně prvků, jako máme k dispozici obrysů. To znamená, že každému obrysu lze přiřadit jedno přidružené číslo z *vectoru*, které popisuje, jak se s tímto přidruženým obrysem bude zacházet. Čísla, která se v *computed* mohou vyskytovat, jsou celá čísla z rozsahu -2 až 3 a jejich přesný význam je následující:

- -2 přidružený obrys není platný (bude ignorován) a nebude ani dále zpracováván,
- -1 obrys není platný (bude ignorován), ale je potřeba ho zpracovat, aby dimenze *vectoru computed* byla dle očekávání (tzn. počet segmentů + 1),
- 0 obrys je platný, ale už byl jednou zpracován, není tedy potřebné jej opakovaně zpracovávat,
- 1 obrys je platný a byl vytvořen během aktuální iterace. Další postup bude určen funkcí pro filtraci (viz Kapitola 8.4) a chybovou funkcí (viz Kapitola 8.3),
- 2 obrys je platný, bohužel funkce pro filtraci ho označila jako nevhodný, neboť porušuje pravidla filtru, tudíž bude zopakována aktuální iterace, ale tento obrys z ní bude vyřazen,
- 3 obrys je platný, ale jiný obrys vynutil opakování iterace (byl vyřazen filtrem). Protože se chceme vyhnout opakovanému počítání tohoto obrysu a zároveň je potřeba ho po filtraci dále zpracovávat, je třeba ho odlišit od obrysů 0 a 1, které obě tyto vlastnosti nemají.

### 8.2.2 Pravidla dělení

Z výše uvedeného popisu je zřejmé, že každý obrys má definovaná i pravidla, jak se bude dále dělit. Byť nedochází přímo k dělení, proces úpravy *vectoru computed* podle pravidel reálnému dělení přímo předchází, a proto tak bude nadále nazýváno.

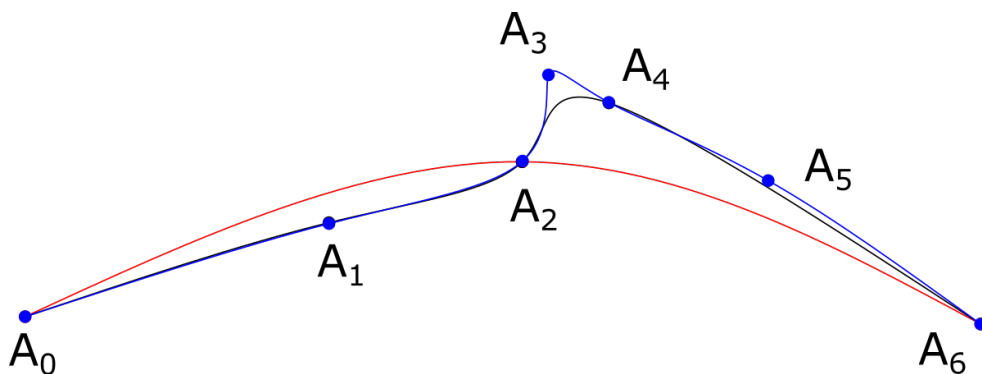
Tento proces úpravy probíhá právě v této funkci. V průběhu funkce je původní *vector computed* přepsán následujícími pravidly do pomocného *vectoru* tak, aby se jeho nová velikost odpovídala rozdělení všech předchozích úseků napůl.

Pravidla pro dělení jsou následující:

- -2 je přepsána na -2,
- -1 je rozdělena na -1, -2, -1,
- 0 je přepsána na 0,
- 1 (případ, kdy je potřeba další dělení) je rozdělena na 1, 0, 1,
- 1 (případ, kdy není potřeba další dělení) je rozdělena na -1, 0, -1,
- 2 je rozdělena na -1, -2, -1,
- 3 je identická s 1 (opět mohou nastat oba případy).

### 8.2.3 Příklad průběhu dělení

Pro lepší vizualizaci, jak dělení probíhá, lze uvést příklad. Je však důležité upozornit ještě na jednu věc. Předtím, než jsou aplikována pravidla uvedená výše, probíhá ještě jedna iterace (tzn. v první iteraci tato pravidla neplatí). Na začátku totiž *vector* obsahuje samé 1 a při dělení dle aktuálních pravidel by nerespektoval to, že krajní obrys nelze dělit z obou stran. V první iteraci je tedy pravidlo 1 se rozdělí na  $+1, 0, +1$  nahrazeno pravidlem 1 se rozdělí na  $0, 1$  (plus je přidáno pravidlo, které není využito, a to, že 0 se přepíše na  $0, -1$ ). Je však očividné, že např.  $[1, 1, 1]$  by se přepsalo na  $[0, 1, 0, 1, 0, 1]$ , i když bychom si přáli jen  $[0, 1, 0, 1, 0]$ . Nejsnazší je proto poslední prvek *vectoru* odebrat. Tvorba tohoto *vectoru* odpovídá dělení červené křivky ( $A_0, A_2, A_6$ ) a vzniku černé křivky ( $A_0, A_1, A_2, A_4, A_6$ ) na Obrázku 8.1.



Obrázek 8.1: Obrázek pro vizualizaci dělení křivky (viz text Kapitoly 8.2.3)

Nyní se můžeme přesunout k samotnému příkladu dělení. Pro jednoduchost použijeme předchozí *vector*  $[0, 1, 0, 1, 0]$  (černá křivka na Obrázku 8.1). Dále předpokládejme, že nedošlo k filtraci a první segment již není potřeba dělit a druhý bude potřeba dělit. Po aplikaci pravidel nám vznikne *vector*  $[0, -1, 0, -1, 0, 1, 0, 1, 0]$  (tento *vector* bude použit pro tvorbu modré křivky výše odkazovaného obrázku).

Pro upřesnění, -1 znamená, že tento obrys nepotřebujeme počítat, ale potřebujeme o něm vědět, aby dimenze *vectoru* zůstala správná. Hodnota -1 tudíž odpovídá myslitelnému bodu mezi  $A_0$  a  $A_1$  a mezi  $A_1$  a  $A_2$ . Hodnota 1 znamená, že tento obrys bude v další iteraci vytvořen a body z něj použity pro svalová vlákna (samozřejmě pokud v následující iteraci nedojde k filtraci). Tomu by odpovídaly body  $A_3$  a  $A_5$  z modré křivky. Hodnota 0 označuje obrys spočtený z některé z předchozích iterací. V tomto případě by bylo vytvořeno 7 obrysů a výsledné vlákno by se tudíž skládalo ze 7 bodů (tzn. mělo by 6 segmentů), přesně jak lze vidět na Obrázku 8.1.

## 8.2.4 Filtrování nežádoucích bodů

Druhá možnost, která může v průběhu této funkce nastat, je, že bod bude označen za nevyhovující pomocí filtru. Filtrovací funkce bude podrobněji popsána v Kapitole 8.4, důležité pro tento příklad je pouze to, že vrací *true*, pokud má dojít k aplikaci filtru.

Jelikož filtrování je odděleno od dělení, má i vlastní *vector* nazvaný *filter\_vec*. Tento *vector* musí mít stejnou dimenzi jako původní *computed vector*, neboť se využívá k opakování iterace. Protože se využívá k opětovnému průchodu algoritmu tvorby vláken, pravidla pro tvorbu tohoto *vectoru* jsou velmi jednoduchá. Jedná se vlastně o kopii původního *vectoru computed*, akorát existuje jedna výjimka. Číslo 1 se nikdy nepřepíše opět na 1 (již z logiky věci – došlo by k opakování stejného výpočtu – a přesně tomu se snažíme vyhýbat). Toto číslo se tedy může přepsat na dvě jiná čísla, a jak již bylo uvedeno v 8.2.1, jedná se o čísla 2 a 3. Hodnota 1 se na 2 přepíše tehdy, pokud filtr pro daný obrys vrátil *true*, jinak se do *vectoru* zapisuje 3 (pro připomenutí – obrys označený číslem 2 bude vyřazen a obrysy s číslem 3 se chovají jako ty s číslem 1, ale v nové iteraci nebudou opětovně spočteny).

## 8.2.5 Příklad filtrování

Filtrování lze rovněž ukázat na příkladu. Vezměme opět *vector*  $[0,1,0,1,0]$ , ale tentokrát první 1 bude označena jako nevyhovující z pohledu filtru. Tento

*vector* by se běžně přepsal do *new\_vec* a ten do *computed* pro novou iteraci. Protože však došlo k aplikaci filtru, místo *new\_vec* bude do *computed* zapsán *filter\_vec*, ve kterém budou tato čísla [0, 2, 0, 3, 0]. Spustí se znovu již proběhlá iterace, ale s tímto *vectorem*. Tím vzniknou nová vlákna, která budou složena z bodů pouze čtyř obrysů místo pěti (obrys s číslem 2 bude vyřazen). Poté, co se opět dostaneme do této souhrnné chybové funkce, bude tento *vector* přepsán následovně (uvažujme, že obrys s číslem 3 vyžaduje dělení): [0, -1, -2, -1, 0, 1, 0, 1, 0].

## 8.3 Chybová funkce

Neméně podstatnou částí celého tohoto algoritmu je samotná chybová funkce popsaná Algoritmem 3. Tato funkce musí rozlišit mezi případy, kdy je změna mezi dvěma iteracemi stejného segmentu vlákna tak zanedbatelná, že již není potřeba na tomto segmentu provádět další dělení, a kdy je tomu naopak.

---

### Algoritmus 3: SegmentsSimilar() – Chybová funkce

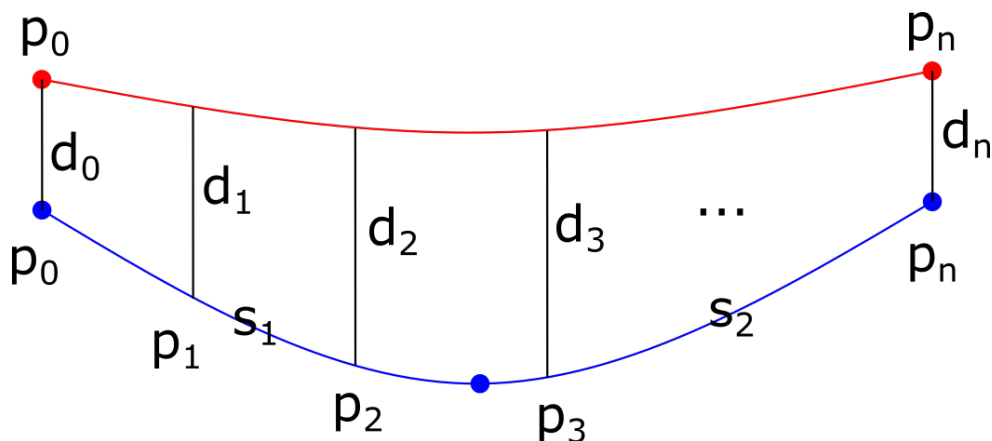
---

```
// splC – původní vlákna – Catmull-Rom křivky
// splN – nová vlákna – Catmull-Rom křivky
// SEG_COUNT – konstanta obsahující počet úseků, na které se
segment křivky bude dělit
// MAX – maximální povolená vzdálenost

for i = 0 to splC.length do
    arc_len_1 = délka zkoumaného úseku na splC[i];
    arc_len_2 = délka zkoumaného úseku na splN[i];
    rozděl obě délky arc_len_1 a arc_len_2 na SEG_COUNT dílů
    stejné délky;
    for j = 0 to SEG_COUNT + 1 do
        spočti vzdálenost mezi body ze splN[i] a splC[i] a přičti jí do
        celkové vzdálenosti
    end
    spočti průměrnou vzdálenost pro segment;
    if průměr > MAX then
        | return false
    end
end
return true
```

---

Jak již bylo popsáno v Kapitole 6.3.2, pro spočtení vzdálenosti je velmi výhodné, že počáteční a koncové body křivek (svalových vláken) jsou nezávislé na počtu segmentů a jsou vždy stejné. Díky tomu lze zvolit krok parametru  $t$  Catmull-Rom křivky a vytvořit body na křivkách, mezi kterými bude měřena vzdálenost. Celý následující segment je vizualizován na Obrázku 8.2.



Obrázek 8.2: Obrázek pro vizualizaci chybové funkce. Červené je vlákno z minulé iterace, modré je vlákno z nové iterace (skládající se ze dvou úseků  $s_1$  a  $s_2$ ). Vlákna jsou v reálu alespoň částečně přes sebe, ale pro názornost byla oddělena (tzn. reálně by  $d_0 = d_n = 0$ )

Jediný problém, který zde nastává, je ten, že jedna křivka (z minulé iterace) je v nové iteraci složena ze dvou křivek, a tím pádem nelze oběma křivkám jen dát stejný krok parametru  $t$ . Toto jsem vyřešil tím, že krok parametru  $t$  je určen pomocí délek křivky. Délka křivky je pouze aproximována pomocí vzdáleností několika bodů ležících na ní. Pro křivku z minulé iterace (červená) se tím nic nezmění –  $t$  bude mít krok  $\frac{1}{n-1}$ , kde  $n$  bude počet bodů (včetně koncových), ale pro segment z aktuální iterace (modrá křivka), tomu bude jinak.

Protože se tento segment skládá ze 2 křivek a každá z nich může mít  $t$  od 0 do 1, je potřeba tento parametr správně navzorkovat. Toho je docíleno následovně: Je určen krok jako  $\frac{s}{n-1}$ , kde  $s$  je délka segmentu ( $= s_1 + s_2$  – součet délky křivky 1 a 2). Tento krok je poté přenásoben čísly  $0..n$ . Tím vzniká posloupnost čísel  $p_n$  od 0 do  $s$  a umožňuje určit parametr  $t$  pro obě křivky takto: dokud je  $p_i$  menší než  $s_1$ , jedná se o parametr  $t$  první křivky, který se spočte jako  $\frac{p_i}{s_1}$ , jakmile je  $p_i$  větší než  $s_1$ , jedná se o parametr druhé křivky, který se spočte jako  $\frac{p_i - s_1}{s_2}$ .

## 8.4 Filtrování

Jak již bylo popsáno výše, při dělení vláken může dojít i k aplikaci filtru. Jak tento proces probíhá je popsáno v Kapitole 8.2.4. Co však ještě popsáno nebylo je jak je filtr implementován. Implementace filtru opět vychází z návrhu v Kapitole 6.3.1 a pseudokód této metody je možno vidět v Algoritmu 4. Tato metoda je vcelku jednoduchá. Jediné, co dělá je, že pro každý segment vezme jeho krajní body a spočítá jejich tečny. Výpočet tečen je metoda třídy *CatmullRomSpline* a bude popsána v Kapitole 8.5. Mezi spočtenými tečnami je poté vypočten úhel  $\theta$  podle obecně známého vzorce  $\cos(\theta) = \frac{t_1 \cdot t_2}{|t_1| \cdot |t_2|}$ . Pokud je  $\theta$  větší než definované maximum je aktuální segment označen za nevyhovující a proběhne jeho vyřazení, které bylo ukázáno výše.

---

**Algoritmus 4:** FilterSegment(iSegment) – Filtrování dle úhlu

---

```
// iSegment – číslo segmentu ke kontrole
// splinesNew – nová vlákna reprezentovaná pomocí Catmull-Rom
// křivek
// MAX – definovaná konstanta obsahující maximální úhel
foreach spline in splinesNew do
    spočti tečny v kontrolních bodech iSegment a iSegment + 1;
    spočti úhel mezi tečnami;
    if úhel > MAX then
        | return true
    else
        | return false
    end
end
```

---

## 8.5 Třída CatmullRomSpline

CatmullRomSpline je nová třída, která byla dodána do MWP2. Její implementace přímo vychází z popisu této křivky z Kapitoly 5. Třída poskytuje především tři metody vracející informace o křivce – *tangent()*, *getPoint()* a *getApproxLen()* a tři pomocné metody na nastavení parametrů/bodů – *addControlPoint()*, *clear()* a *setAlpha()*.

### 8.5.1 Výpočet bodu na křivce

Parametrem této funkce je  $t$ , což je parametr křivky, a dále index druhého vrcholu ze čtyř, pomocí nichž bude křivka (resp. její část) definována. Druhý vrchol je zvolen, neboť první vrchol nemusí v krajních úsecích existovat a bude dopočten pomocí středové souměrnosti. Před samotným výpočtem bodu je zkontrolováno, zda jsou parametry v pořádku (např. že  $t$  je reálné číslo mezi 0 a 1). Poté dochází k výpočtu bodu na křivce. Postup tohoto výpočtu je totožný s postupem popsáním v Kapitole 5.1 – tzn. proběhne výpočet koeficientů  $A_{1-3}$  a  $B_{1-2}$ , z nich jsou vytvořeny souřadnice bodu v parametru  $t$ .

### 8.5.2 Výpočet tečny

Výpočet tečny je kvůli aplikaci omezen pouze na výpočet tečny v kontrolních bodech, a byť by rozšíření na výpočet v jakémkoliv místě nebylo složité, bylo by zbytečné. Jediný parametr, který tato funkce vyžaduje, je tedy index kontrolního bodu, ve kterém se bude počítat tečna. Tečna je vypočtena pomocí limity dle vzorce  $\lim_{h \rightarrow 0} \frac{f(x+h) - f(x-h)}{2h}$ . Jediný problém zde nastává v krajních bodech, neboť nedokážeme pro bod  $x = P_0$  získat  $x - h$  (tento bod je mimo definovaný rozsah křivky), ten je však vyřešen využitím pouze jednostranné limity  $\lim_{h \rightarrow 0} \frac{f(x \pm h) - f(x)}{\pm h}$  pro počáteční, resp. koncový bod.

### 8.5.3 Délka křivky

Pro výše uvedené důvody (viz Kapitola 8.3) je podstatné, aby křivka znala svou (alespoň přibližnou) délku. Protože postačí aproximace délky, je křivka mezi dvěma kontrolními body rozdělena na 10 úseků (11 bodů celkem), což se ukázalo jako dostatečně přesné, a mezi nimi je spočtena euklidovská vzdálenost. Ta je uložena do *vectoru* délek jednotlivých úseků, aby bylo zabráněno zbytečnému opakování tohoto výpočtu. Výpočet délky úseku je proveden vždy, když přibude nový kontrolní bod.

### 8.5.4 Operace pro úpravu křivky

Nejdůležitější z těchto operací je přidání nového kontrolního bodu. Nové body se do křivky přidávají pomocí metody `addControlPoint()`. Body se vždy přidávají na konec křivky. Díky tomu lze okamžitě po přidání bodu vypočítat délku nového úseku a přidat ji do *vectoru* s délkami úseků. Další

metodou je *clear()*, což je metoda, která vymaže obsah této třídy, a tím umožní její opětovné použití v následující iteraci. Poslední metoda *setAlpha()* umožňuje nastavit parametr  $\alpha$  Catmull-Rom křivky, ale protože již v návrhu bylo avizováno, že bude použita pouze centripetální Catmull-Rom křivka, tato metoda není využívána.



# 9 Úprava vláken za běhu

Druhou částí této bakalářské práce bylo implementovat algoritmus, který by zajistil, že při deformaci svalů pomocí metody Luca2018 nebude docházet ke kolizím.

## 9.1 Discregrid

Discregrid [5] je statická C++ knihovna vytvořená D. Koschierem a je veřejně dostupná v rámci MIT licence. Tato knihovna umožňuje generování SDF z trojúhelníkové mřížky. Nad SDF je definována funkce, která umí pro každý bod uvnitř definovaného ohraničeného prostoru vrátit jeho vzdálenost k nejbližšímu bodu mřížky. Platí, že pokud je tato vrácená vzdálenost záporná, tak bod leží uvnitř objektu, pomocí kterého byla SDF vygenerována. Pokud je kladná, leží mimo. Dále tato funkce v rámci knihovny Discregrid umí vrátit i gradient, který ukazuje směr k nejbližšímu bodu na povrchu.

## 9.2 Integrace Discregridu

Jak bylo popsáno výše, knihovna Discregrid (viz Kapitola 9.1) umožňuje generaci SDF. Toho lze využít právě pro detekci a opravu kolizí. Bylo proto nutné upravit projekt tak, aby s Discregridem uměl pracovat. Dále bylo potřeba do projektu přidat i závislost Discregridu Eigen3 [2], což je opět veřejně dostupná knihovna (licence MPL2), která poskytuje rychlou a spolehlivou implementaci funkcí lineární algebry.

### 9.2.1 Příprava struktur pro SDF

V MuscleWrapping2 projektu je za veškerou deformaci svalů odpovědná třída *vtkMusculoSkeletalSystemDeformation* a její potomci. Tato třída zajišťuje inicializaci všech potřebných polí a transformačních matic. Dále vykonává i samotnou deformaci svalových vláken v daný čas. Důležité pro integraci Discregridu bylo to, že tato třída obsahuje i všechny informace o kostech – kosti jsou zde uloženy do *vtkPolyData* (což je struktura z knihovny VTK [10] pro uchování různých objektů – vrcholy, polygony, trojúhelníkové sítě,

atd.). Bohužel Discregrid neumí pracovat s touto strukturou a požaduje, aby informace o trojúhelníkové mřížce představující objekt, nad kterým bude generována SDF, byly uloženy do jeho vlastní struktury. To však nebyl zásadní problém, neboť *vtkPolyData* udržuje data v paměti v obdobném formátu jako je *.obj*, a díky tomu není složité jednotlivé vrcholy a k nim přilehlé trojúhelníky přepsat do struktury, kterou Discregrid používá. Pro každou kost reprezentovanou pomocí trojúhelníkové mřížky bylo tedy vygenerována a uložena její SDF.

Již v Kapitole 9.1 bylo zmíněno, že SDF je generována v uzavřeném prostoru. Tento prostor je také nutné definovat, postačí však použít minimum a maximum ze všech bodů v daném objektu (kosti). Tyto dvě hodnoty jsou tedy také získány během převodu *vtkPolyData* na strukturu Discregridu.

### 9.2.2 Tvorba SDF

Tvorba SDF je poměrně časově náročná úloha i při paralelizaci, kterou Discregrid samozřejmě podporuje. Pro svou náročnost je důležité zajistit, aby se struktura, která reprezentuje SDF, negenerovala vícekrát, neboť by to vedlo k neúnosnému časovému zdržení. Jako vhodné místo, kde bude možné tuto strukturu generovat, je metoda, která inicializuje celou deformaci, a tím pádem v aktuální implementaci probíhá jen jednou. Protože i tak je generování SDF náročné, je zde další možnost, kterou Discregrid nabízí. Tou je serializace a deserializace SDF. SDF pro každou kost lze uložit do samostatného souboru a z něj je možné ji opětovně načítat. Někteří se mohou domnívat, že využitelnost serializace a deserializace je v reálu mizivá, protože tento projekt se snaží vytvářet personalizované modely (a každý tím pádem bude mít jinou SDF). I ukládání SDF by se však dalo využít, například při opětovných vyšetřeních. Pro testovací účely je možnost SDF uložit k nezaplacení, protože to šetří mnoho času při opětovném spouštění simulace.

### 9.2.3 Úprava deformace

Deformace v rámci tohoto projektu probíhá v metodě *ExecuteDeformation*. Tato metoda provede deformaci z původní polohy do aktuální polohy pomocí transformační matice přilehlých kostí přesně tak, jak bylo popsáno v Kapitole 3. Pro připomenutí, je důležité znát původní pozici bodu, transformační matice kostí a váhu  $w$ . Pomocí těchto hodnot je vypočtena pozice bodu. Problém je, že tato pozice může být v kolizi s kostí. To lze však řešit pomocí SDF. Stačí jen aplikovat správnou transformaci na bod, tento

transformovaný bod otestovat pomocí SDF, a pokud dojde ke kolizi, tak vypočtenou pozici opravit pomocí gradientu, který nám Discregrid vrátí. Zůstává jediná otázka – jaká je správná transformace, kterou musíme aplikovat na vypočtený bod, abychom ho transformovali zpět do prostoru, ve kterém byla vypočtena SDF. Tato transformace musí být inverzní transformace kosti. Protože máme kosti dvě, provedeme celé testování vůči SDF dvakrát a výsledné gradienty sečteme. Tím zajistíme, že bod by neměl být uvnitř ani jedné kosti, ale maximálně na jejím povrchu.

### 9.2.4 Úprava parametru $w$ metody Luca2018

Při deformaci se používá metoda Luca2018 popsaná v Kapitole 3. Hlavní myšlenkou je, že pro transformaci bodu se využívá lineární kombinace dvou transformací kostí a parametr  $w$  určuje, jak velký vliv bude jednotlivá transformace na daný bod mít.

Úprava tohoto parametru spočívá v tom, že  $w$  se počítalo z  $f(t)$ , kde  $t$  bylo lineární od 0 do 1. Nyní je  $t$  do tohoto vzorce vypočteno jako poměr části svalu od počátku k aktuálnímu bodu ku celkové délce svalu, což je vcelku podstatné z pohledu adaptivních vláken. Před zavedením adaptivního počtu segmentů byl vždy bod vlákna zhruba stejně daleko od následujícího bodu – vlákno bylo děleno na několik podobných částí. Nyní je však možné dělit jen část vlákna a druhou již ne. Tím vznikají nepoměry částí a původní algoritmus počítání  $w$  by nefungoval dostatečně dobře.

# 10 Výsledky práce

V této kapitole budou shrnuty výsledky obou částí bakalářské práce. Napřed je však nutné popsat způsoby, jakými budou experimenty probíhat.

## 10.1 Testovací data

Protože MWP2 již nějaký čas běží a vyšlo ohledně něj již několik článků (např. [3] a [6]), testovací data již byla v projektu zahrnuta, když jsem se do něj připojil. Konkrétně se jedná o model dolních končetin ženy – *GenericFemale\_WCB.osim*.

Jednou z pozorovaných vlastností modelu je jeho chování během tří základních pohybů stehenní kosti – flexe (ohyb), rotace a abdukce/addukce (roznožení/přinožení). Proto jsou v testovacích datech i tři soubory, které definují tyto pohyby. Soubory jsou stejné jako soubory použité v článku [6] – tzn. flexe se pohybuje v rozsahu  $-10^\circ$  do  $60^\circ$ , rotace  $-30^\circ$  do  $30^\circ$  a abdukce/addukce od  $-40^\circ$  do  $40^\circ$ , všechny s krokem pohybu  $2^\circ$ . Tento krok je poměrně velký, ale protože byla snaha zachovat testovací soubory, které už byly v minulosti použity, krok jsem neměnil. Tyto soubory se specifikují uvnitř konfiguračního XML souboru (*setup\_MuscleGeneratorTool.xml*), díky čemuž lze lehko měnit nastavení a generovat data pro následující experimenty. Po domluvě s vedoucím práce byly určeny 3 svaly dolní končetiny, které budou analyzovány:

- Gluteus maximus (velký sval hýžďový) – tento sval je objemný a není přesně jasný ideální počet vláken pro jeho sestavení. Dále je problematická deformace tohoto svalu také při flexi, protože během tohoto pohybu dochází k průniku s kostí.
- Gluteus medius (střední sval hýžďový) – tento sval je zde především kvůli jeho netečnosti vůči jakémukoliv ze tří vybraných pohybů. Nemělo by v něm tudíž docházet k podstatným rozdílům bez ohledu na testovací parametry.
- Iliacus (kyčelní sval) – tento sval obecně v rámci projektu působí problémy. Při pohybu se zkracuje a často dochází k jeho „zmuchlání“. Také při jeho pohybu dochází k mnoha kolizím s kostí.

## 10.2 Metoda testování

Kromě testovacích dat jsou v projektu zahrnuty i metody, které umí vypočítat délku svalových vláken a ramena momentů sil v průběhu celé simulace. Ramena momentů sil ( $M$ ) jsou v testovací metodě počítány pomocí vzorce 10.1, kde  $dL$  je změna délky vlákna a  $du$  je změna úhlu. Protože změna úhlu je po celou dobu v testovacích souborech konstantní ( $2^\circ$ ), rameno momentu síly je závislé pouze na změně délek vláken.

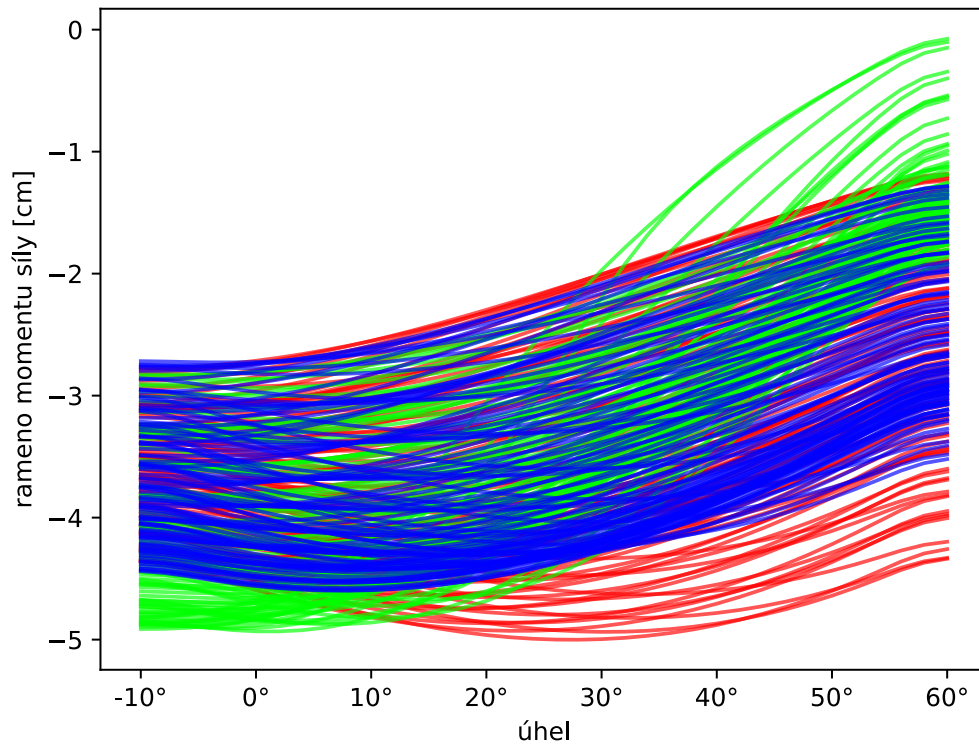
$$M = \frac{dL}{du} \quad (10.1)$$

Pro spuštění testovacích metod bylo potřeba správně upravit konfigurační soubor (přidat složku, do které bude vypisován výstup) a poté data zpracovat. Metody generují data do textového souboru s příponou *.sto*, který obsahuje tabulátory oddělená vypočtená data s příslušným časem. Díky tomu lze data vynést do grafu a zjistit, jak se změnilo chování svalů, resp. ramena jejich momentů sil a délky vláken. Byť je metodami exportována závislost na čase, pro lepší představu bude na ose x vždy uváděn úhel (převod mezi časem a úhlem je přímo popsán v souborech *.mot* pro daný pohyb).

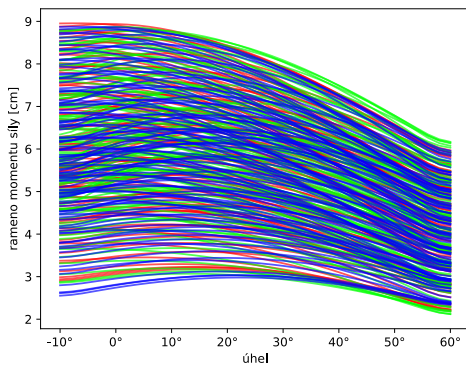
## 10.3 Vliv parametru $w$

Předtím, než budou srovnány staré a nové metody generování vlákna, bych rád na příkladu ukázal, jak parametr  $w$ , jehož úprava byla popsána v Kapitole 9.2.4, ovlivňuje sval. Význam tohoto parametru byl detailně popsán v Kapitole 3. Pro připomenutím tento parametr ovlivňuje deformaci tím, že je používán pro lineární kombinaci transformací působících na deformovaný bod. Na Obrázku 10.1 jsou zobrazeny 3 barevně odlišené grafy ramen momentů sil pro sval Iliacus. Červenou barvou jsou znázorněna ramena momentů sil pro původní verzi, zelená jsou ramena pro adaptivní vlákna bez úpravy parametru  $w$  a modře zvýrazněna jsou ramena pro adaptivní vlákna spolu s upraveným výpočtem  $w$ . Je zde poměrně jasně zřetelný rozdíl mezi zelenou a modrou částí grafu, především u konce simulace. To je s vysokou pravděpodobností způsobeno parametrem  $w$ . U tohoto svalu bude dělena pouze jedna část a zbylé se dělit nebudou. Tím vznikne výše zmíněný rozdíl mezi  $w$  vypočteným lineárně a  $w$  vypočteným pomocí délek části ku celku. Z tohoto důvodu bude dále používána pro srovnání pouze metoda s parametrem  $w$  vypočítaným pomocí délek. Lineární výpočet  $w$  často nezpůsobil

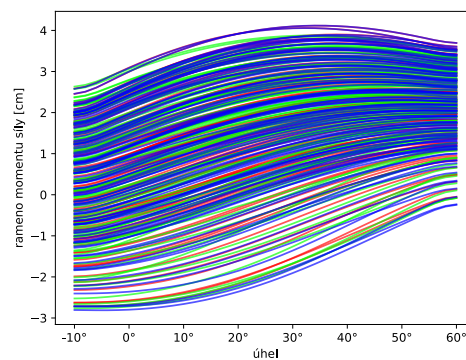
žádný zásadní rozdíl (viz Obrázek 10.2) či pouze zhoršoval chování (viz výše zmíněný Iliacus).



Obrázek 10.1: Vliv parametru  $w$ , Iliacus, flexe – červená jsou ramena původní metody, modrá jsou adaptivně vytvořená vlákna s novým výpočtem  $w$  a zelená je adaptivní metoda bez úpravy výpočtu  $w$  – více viz Kapitola 10.3



(a) Gluteus maximus



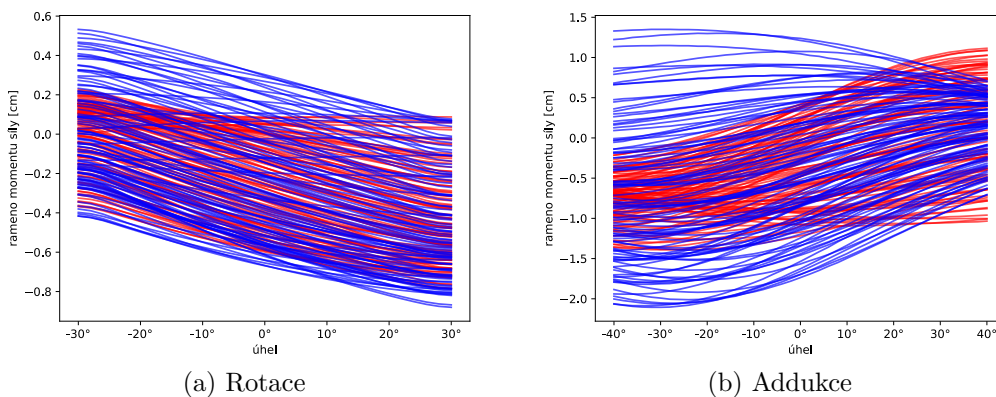
(b) Gluteus medius

Obrázek 10.2: Srovnání ramen momentů sil pro původní metodu (červená), metodu adaptivních vláken s novým výpočtem  $w$  (modrá) a pro metodu adaptivních vláken bez úpravy výpočtu  $w$  (zelená) – více viz text Kapitoly 10.3

## 10.4 Adaptivní vlákna

Prvním úkolem bylo porovnat vliv zavedení adaptivní tvorby vláken. Následovat budou grafy ramen momentů sil, případně grafy délek svalových vláken, které budou vždy okomentovány, o jakou metodu se jedná a jak ovlivnila tato metoda sval. Porovnávána bude vždy originální verze MWP2 a verze, kde jsou zavedena adaptivní vlákna s upraveným parametrem  $w$  (viz Kapitola 10.3), ale není povolena dodatečná úprava pomocí Discregridu.

Na Obrázku 10.2 lze vidět srovnání obou metod (červené a zelené čáry) při ohybu (flexi) pro svaly Gluteus maximus a Gluteus medius. Je zřetelné, že v tomto případě nedošlo k žádné významné změně (obě barvy se prolínají). Naopak v případě svalu Iliacus na Obrázku 10.1 je zjevné, že metoda adaptivních vláken sval ovlivnila – díky tomu, že se vlákno skládá z více segmentů, které jsou navíc lépe rozloženy, sval při deformaci méně koliduje s kostí. To se samozřejmě promítá i do ramen momentů sil. Pro ostatní pohyby (rotace a addukce) jsou výsledky obdobné. Během aplikace metody na Gluteus maximus a medius se žádné výrazné změny neprojeví, a proto zde ani nebudou ukázány – krom jiných hodnot na osách dochází k podobnému překryvu červené a modré jako lze vidět na Obrázku 10.2. Pro Iliacus však došlo ke změnám ramen momentů sil – viz Obrázek 10.3, a jak již bylo napsáno výše v tomto odstavci, tyto změny jsou opět způsobeny změnou délky vláken.



Obrázek 10.3: Iliacus – srovnání ramen momentů sil pro původní metodu (červená) a metodu adaptivních vláken (modrá)

## 10.5 Výsledný počet segmentů

Díky metodě adaptivních vláken lze nyní určit ideální počet vláken pro každý sval. Tento počet musel být dříve odhadován a byl uniformně rozložen. Pro vybrané svaly dolních končetin byla počáteční hodnota nastavena na 1, filtr byl nastaven na úhel  $90^\circ$ . Maximální povolená vzdálenost byla 10 mm, která byla určena na základě výsledků získaných během vývoje algoritmu. Při výrazném snížení této vzdálenosti (pod 7 až 8 mm) se tvořilo příliš mnoho segmentů pro každé vlákno, ale rozdíl vůči verzím s nižším počtem nebyl pozorovatelný. Kolem hodnoty 10 mm sval vypadá lépe než původní sval a zároveň počet segmentů není příliš vysoký. Optimum, které určila takto nastavená adaptivní metoda, lze najít v Tabulce 10.1.

Tabulka 10.1: Počet vláken

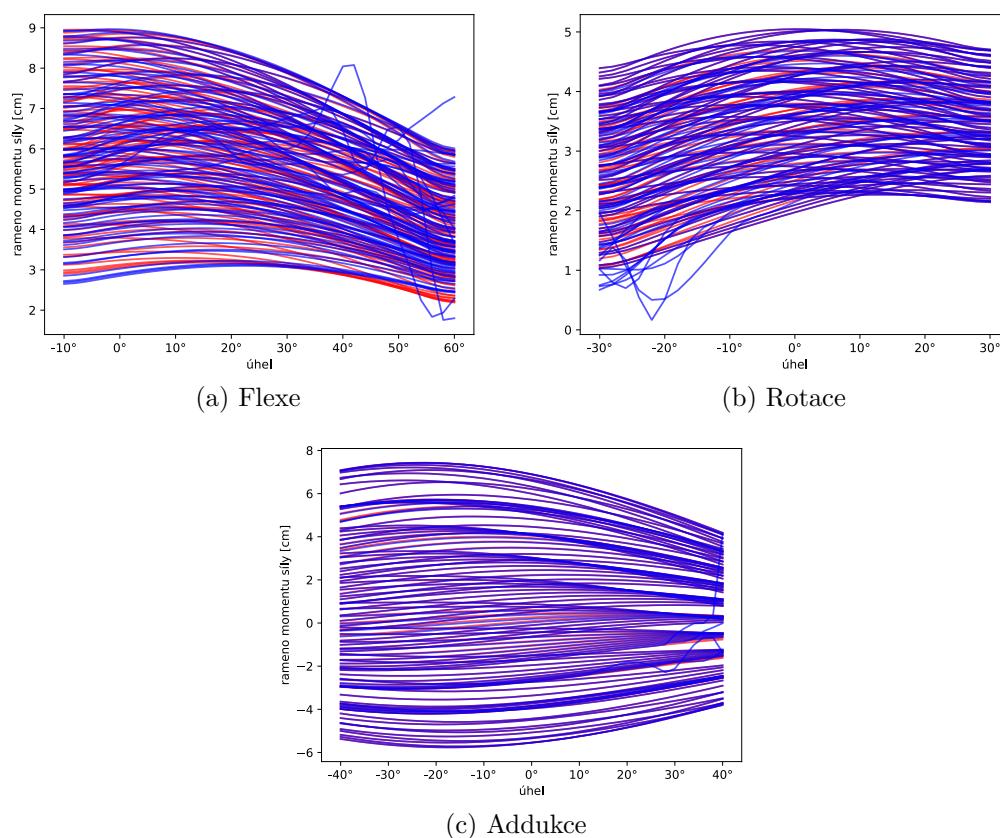
	Původní hodnoty (uniformně)	Adaptivní metoda
Gluteus maximus	15	22
Gluteus medius	15	18
Iliacus	15	20

## 10.6 Úprava vláken

V této kapitole budou prezentovány běhy programu s povolenou úpravou deformace pomocí knihovny Discregrid (více viz Kapitola 9) a zároveň nebudou použita adaptivní vlákna – tzn. všechna vlákna budou z 15 segmentů.

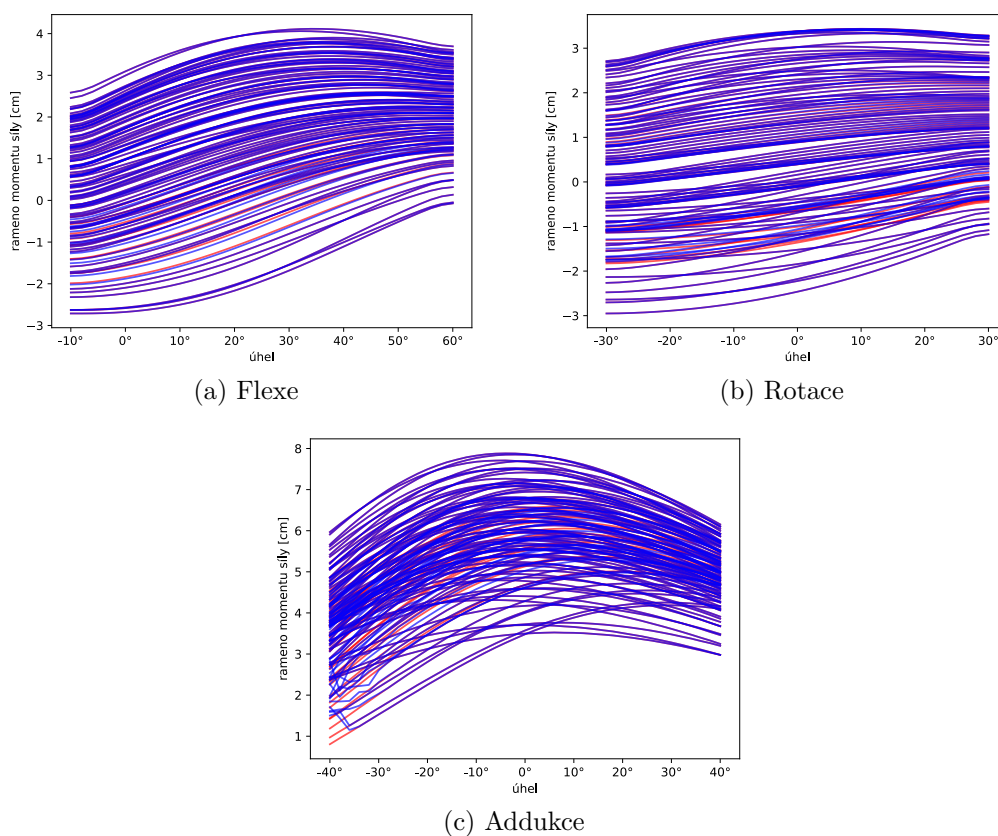
V následujících třech srovnáních ramen momentů sil (Obrázek 10.4, 10.5 a 10.6) je vidět obecný trend, že ramena jsou v absolutní hodnotě delší. To je způsobeno tím, že se více mění délka vlákna (při detekci kolize), a jak bylo popsáno ve Vzorci 10.1, velikost ramene je přímo úměrně závislá na změně délky vlákna.





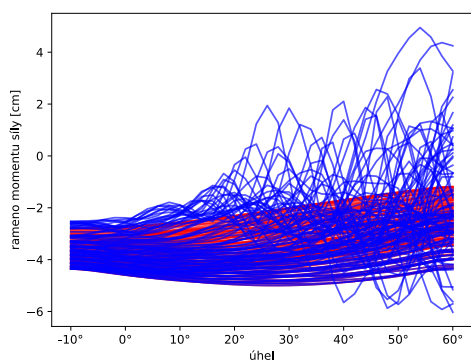
Obrázek 10.4: Gluteus maximus – srovnání ramen momentů sil pro původní metodu (červená) a metodu úpravy vláken (modrá)

Např. pro sval Gluteus maximus (Obrázek 10.7) lze na oranžově obarvených vláknech, která byla v původní verzi v kolizi, vidět poměrně velké zlepšení mezi originální metodou (10.7a) a metodou s úpravou vláken (10.7c). Jelikož vlákna nejsou v kosti, jsou samozřejmě delší a výkyvy ramen momentů sil (viz Obrázek 10.4) jsou tudíž také větší. Tyto výkyvy jsou velké i z důvodu, že byl zvolen krok  $2^\circ$  – kdyby byl krok jemnější, lze předpokládat, že i ramena momentů by netvořila takovéto skoky.

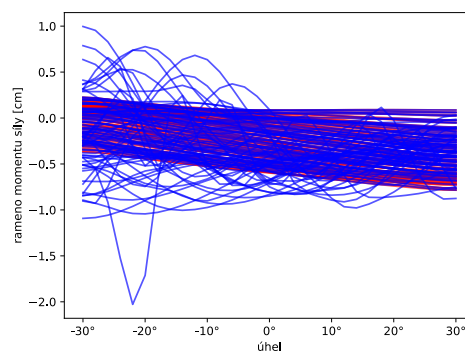


Obrázek 10.5: Gluteus medius – srovnání ramen momentů sil pro původní metodu (červená) a metodu úpravy vláken (modrá)

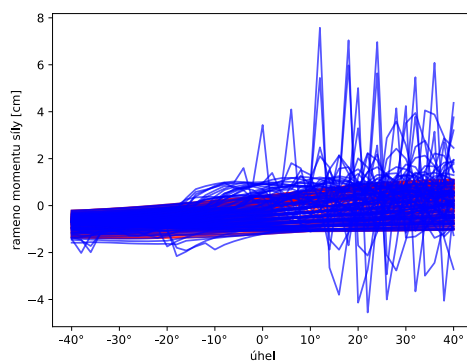
Pro jistotu, že tyto výkyvy nejsou způsobeny chybou v implementaci lze prozkoumat Obrázek 10.7 a Obrázek 10.8. Vlákná v nich se chovají dle očekávání - body vlákna nejsou v kolizi s kostí a maximálně jsou na povrchu kosti. Lze zde vidět i nedostatek aktuální implementace popsany v Kapitole 10.7 – pokud jsou body mimo kost, spojnice jí stále může projít a aktuální metoda tomu nezabrání.



(a) Flexe

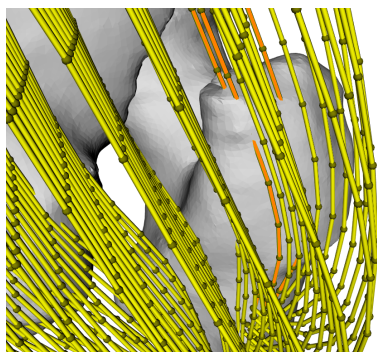


(b) Rotace

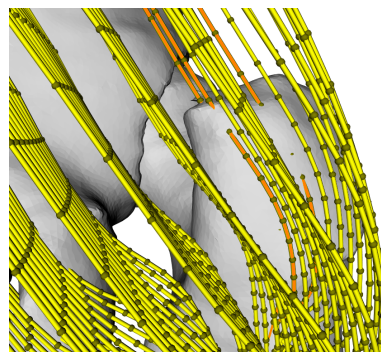


(c) Addukce

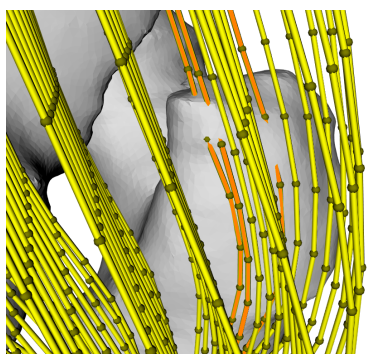
Obrázek 10.6: Iliacus – srovnání ramen momentů sil pro původní metodu (červená) a metodu úpravy vláken (modrá)



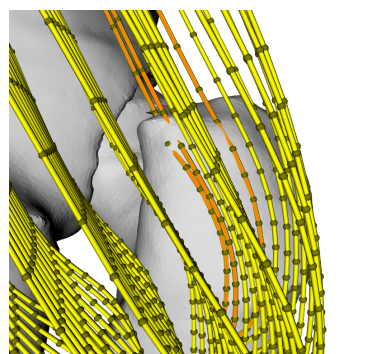
(a) Originální metoda



(b) Pouze metoda adaptivních vláken

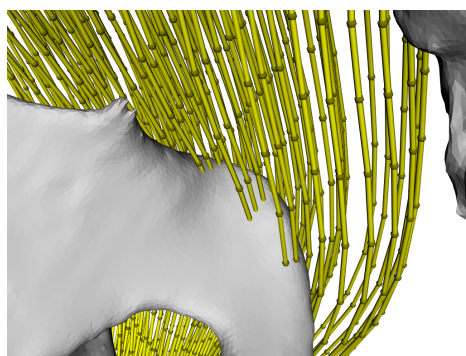


(c) Pouze úprava vláken

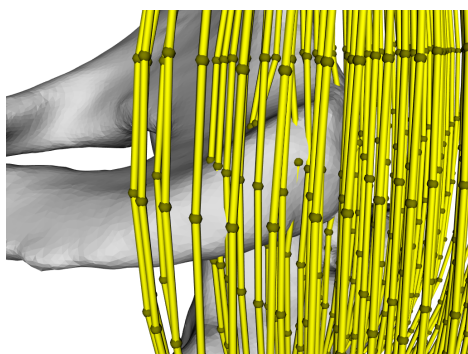


(d) Výsledná verze

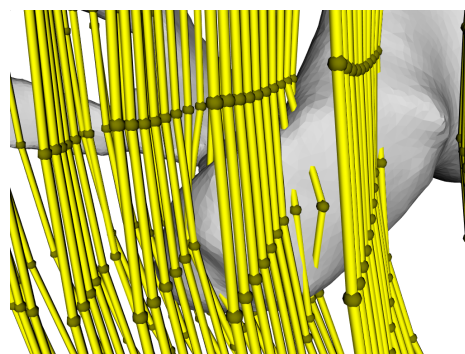
Obrázek 10.7: Gluteus maximus (rotace,  $-22^\circ$ ) – vizuální srovnání všech metod s vyznačenými body vláken



(a) Originální metoda



(b) Úprava vláken, pohled 1



(c) Úprava vláken, pohled 2

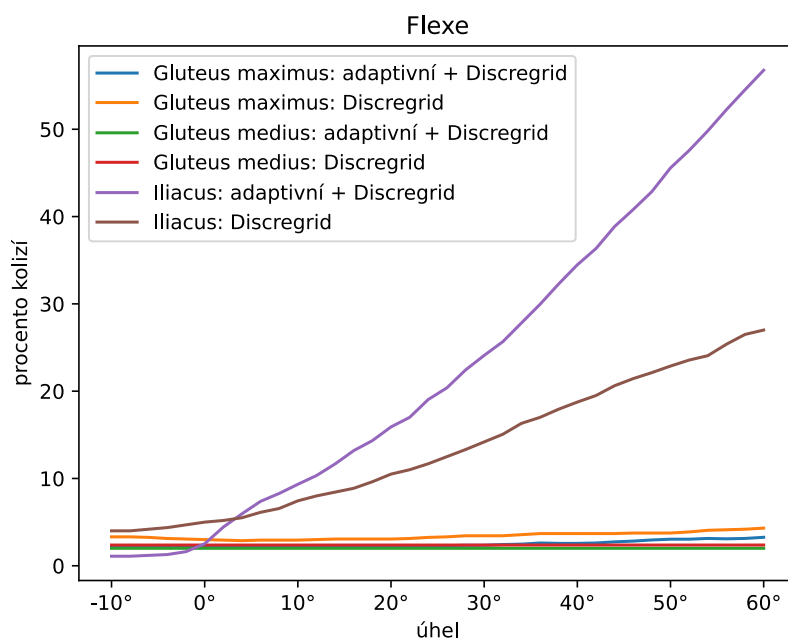
Obrázek 10.8: Gluteus maximus (flexe,  $42^\circ$ ) – vliv úpravy metod – vlákna, která na (a) protínala kost jsou na (b) resp. (c) posunuta blíže k okraji kosti. Dochází k zabránění kolizí bodů a kostí. Body vláken jsou vyznačeny. Bohužel aktuální implementace neřeší kolize celé spojnice bodů a i to se zde vyskytuje. Více viz Kapitola 10.7

## 10.7 Omezení aktuální implementace

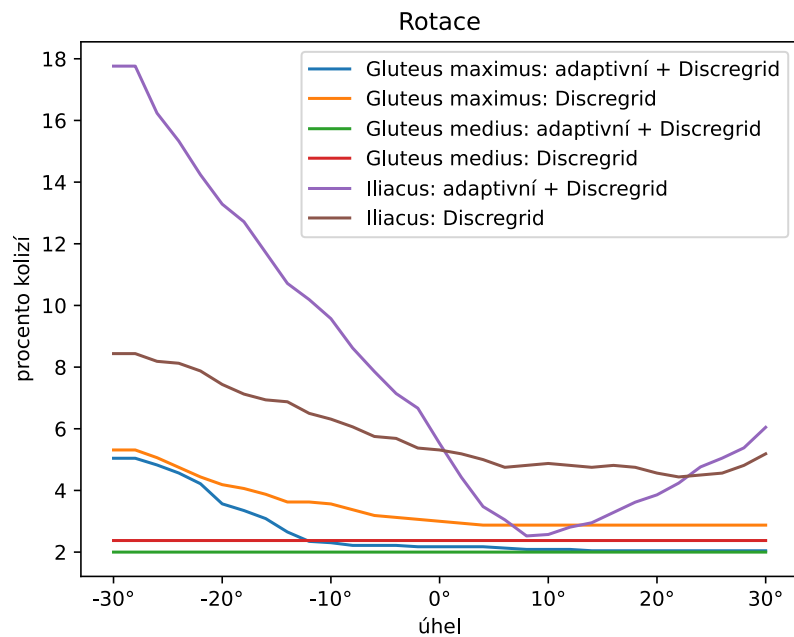
Byť návrh detekce kolizí byl respektován, v průběhu implementace se objevil problém s tím, že aktuálně implementovanou metodou lze pomocí Discregridu posoudit pouze kolizi bodu svalového vlákna, nikoliv křivky. Pokud by se stalo, že oba body budou mimo kost (resp. na její povrch mohou být posunuty), stále je možné, že jejich spojnice kost protne. Bohužel k tomuto jevu dochází ve všech 3 analyzovaných případech. Tento jev se vyskytuje především u svalu Iliacus, protože jeho svalová vlákna jsou velmi blízko kosti a Discregrid jejich body sice posune na povrch, ale kvůli nerovnostem na kosti dojde k výše zmíněnému průniku spojnice dvou bodů vlákna s kostí.

## 10.8 Počet kolizí

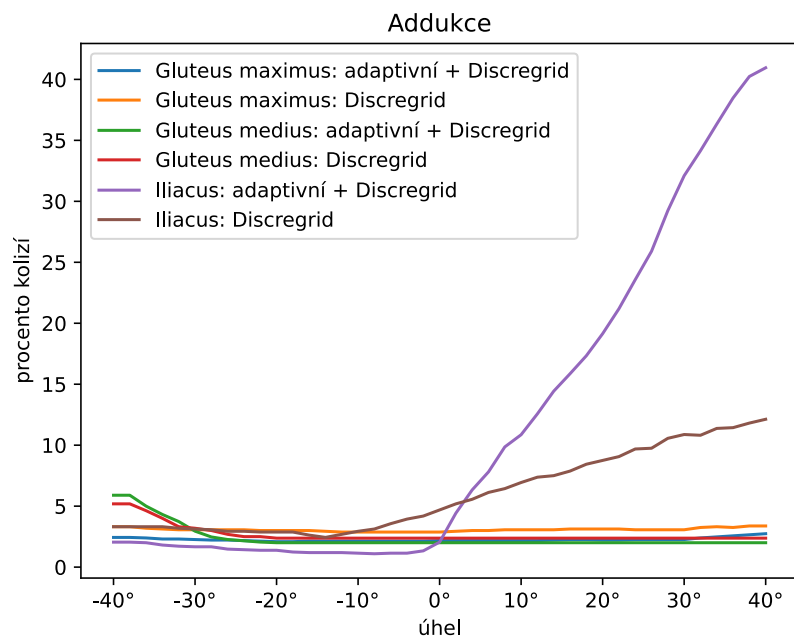
Do následujících grafů v Obrázcích 10.9, 10.10 a 10.11 byl vynesena počet detekovaných kolizí. Kolize se vždy detekují pro každý bod. To znamená, že pokud bychom uvažovali celkem 100 bodů, tak 50 % kolizí bude odpovídat 50 bodům detekovaným uvnitř kosti. Počty bodů lze zjistit jako stonásobek počtu bodů, který lze odvodit z Tabulky 10.1 (počet bodů je počet segmentů plus jedna). Metoda „Discregrid“ ve výše odkazovaných grafech používá originální počet segmentů vláken a metoda nazvaná „adaptivní + Discregrid“ využívá adaptivní počet segmentů vláken.



Obrázek 10.9: Procento bodů z celkového počtu, u kterých byla detekována kolize – flexe



Obrázek 10.10: Procento bodů z celkového počtu, u kterých byla detekována kolize – rotace



Obrázek 10.11: Procento bodů z celkového počtu, u kterých byla detekována kolize – addukce

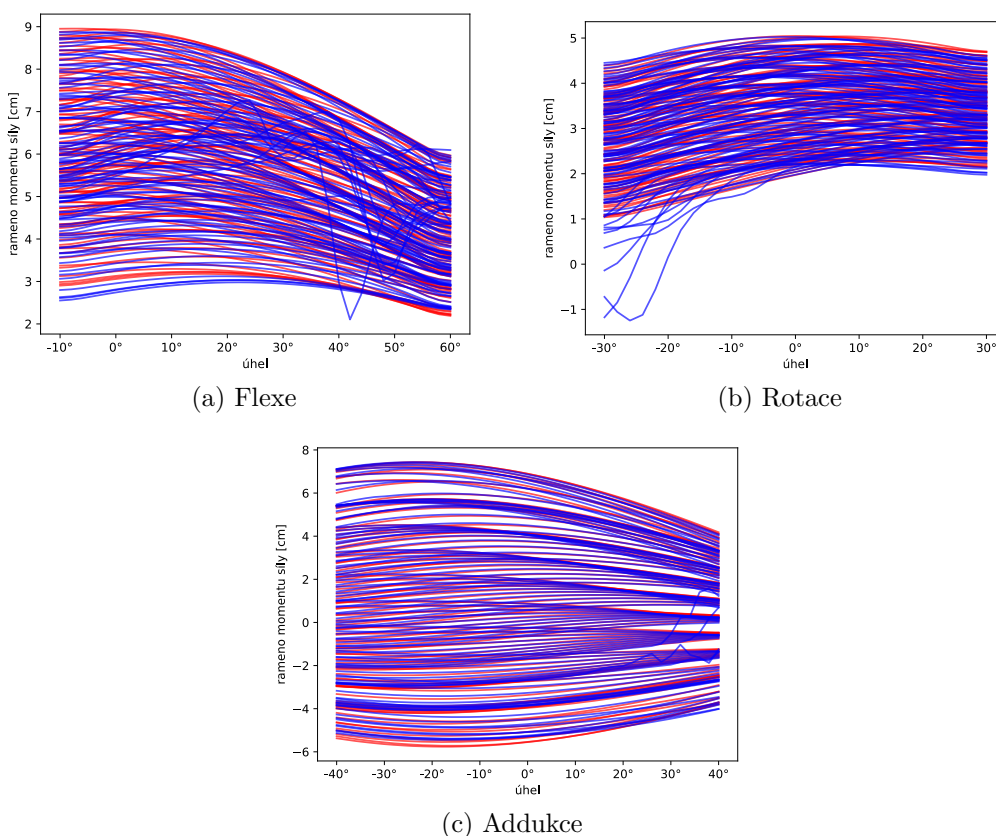


## 10.9 Celkový rozdíl

Tato kapitola ukazuje celkové výsledky práce – pro následující grafy byly povoleny obě úpravy prezentované v této práci (adaptivní vlákna – Kapitola 8 a deformace pomocí Discregridu – Kapitola 9). V předchozích kapitolách byla porovnávána vždy jen jedna metoda a nebylo tudíž zatím možno vidět souhrnný dopad práce na MWP2.

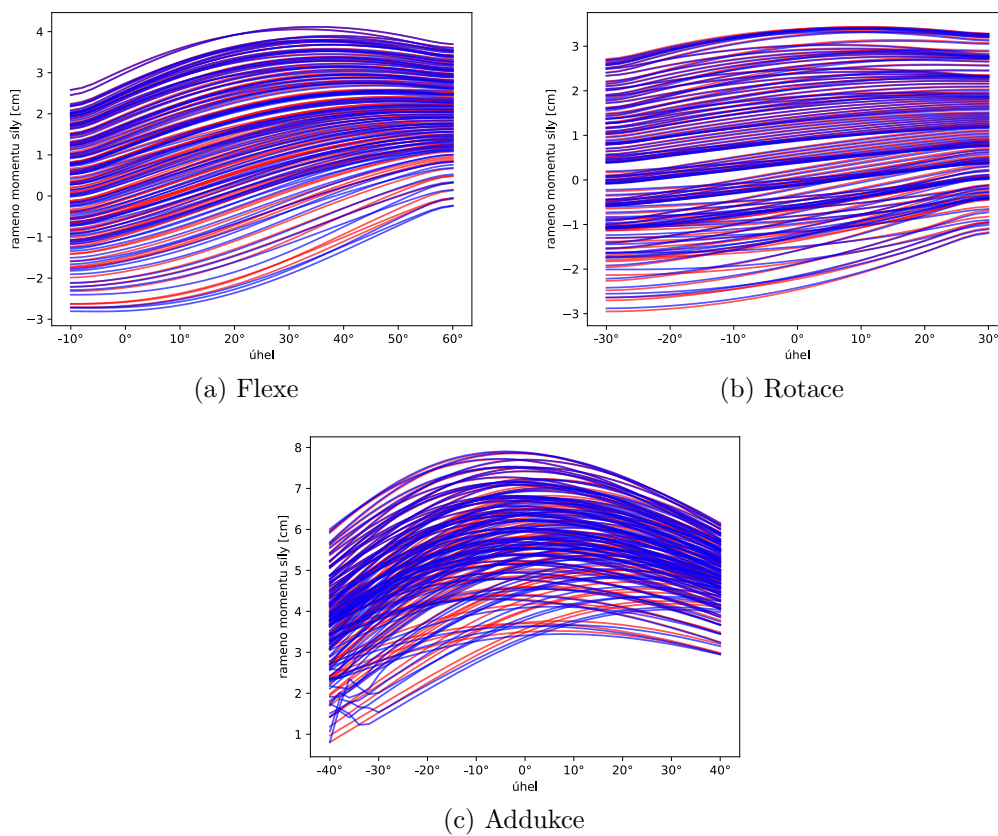
Na Obrázku 10.12 lze vidět výsledný vliv na sval Gluteus maximus. Celková změna ramen pro tento sval není nijak výrazná, pouze opět dochází ke zvlnění křivek kvůli změnám délek svalů. K podobnému chování dochází i u svalu Gluteus medius (viz Obrázek 10.13).

K největším rozdílům dochází u svalu Iliacus (viz Obrázek 10.14). To poměrně dobře reflektuje fakt, že Iliacus byl sval, u kterého docházelo k největším rozdílům i v obou oddělených případech.

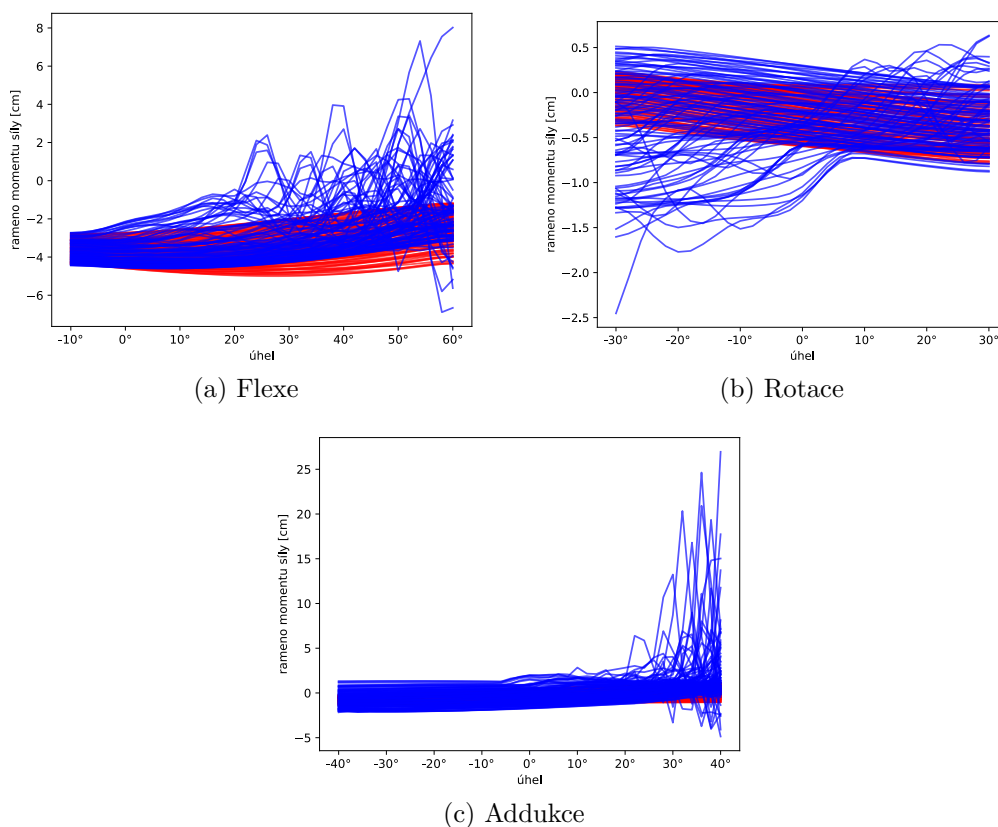


Obrázek 10.12: Gluteus maximus – srovnání ramen momentů sil pro původní metodu (červená) a výslednou verzi této BP (modrá)





Obrázek 10.13: Gluteus medius – srovnání ramen momentů sil pro původní metodu (červená) a výslednou verzi této BP (modrá)



Obrázek 10.14: Iliacus – srovnání ramen momentů sil pro původní metodu (červená) a výslednou verzi této BP (modrá)

## 10.10 Časové nároky

Jednou z důležitých vlastností, která je u tohoto projektu sledována, je časová náročnost. Jak již bylo naznačeno v kapitolách rozebírajících implementaci, doba běhu je pro tento program důležitá. Snahou proto bylo, aby provedené úpravy nezpůsobovaly zbytečné časové prodlevy.

U metody adaptivních vláken je tohoto dosaženo poměrně dobře – celá metoda je jen cca 2,7krát pomalejší – originální metodě trvalo vytvoření vláken v průměru 0,835 s (průměr z 25 běhů, spouštěno na Win10 v MS Visual Studiu, Intel i7-9750H, 16 GB RAM, sval Iliacus, addukce) a adaptivní metoda trvala v průměru 2,26 s.

Dodatečná úprava vláken za běhu má bohužel horší bilanci. Nelze přehlédnout, že generování SDF trvá okolo 35 minut – v této práci byla SDF poté serializována a uložena do souboru – tím jsem při každém běhu ušetřil daných 35 minut, protože deserializace je okamžitá. Když pomínu tvorbu SDF, i sa-

motná detekce kolizí trvá déle, než pokud se program spouští bez Discregridu – program je přibližně 7,2krát pomalejší. Původní metoda trvá v průměru pouze 2,6 s a kontrola pomocí Discregridu zabere v průměru 18,7 s. Všechna výše uvedená měření proběhla na stejném stroji a nad stejnými daty.

# 11 Závěr

V rámci zpracovávání bakalářské práce jsem se seznámil s fungováním muskuuloskeletálních modelů (viz Kapitola 2), dále jsem detailně prostudoval metodu použitou pro automatickou dekompozici (viz Kapitola 4) a deformaci (viz Kapitola 3) svalových vláken. Pro implementaci adaptivních vláken jsem také musel prozkoumat chování Catmull-Rom křivek (viz Kapitola 5), aby z nich nakonec byla vybrána pro tuto práci ta nejlepší – centripetální. Tím byla zakončena teoretická část této práce a na řadu přišel samotný návrh a implementace obou požadovaných metod.

Nejprve byla implementována metoda adaptivní tvorby vláken. Návrh této metody je popsán v Kapitole 6. Poměrně detailní popis implementace lze nalézt v Kapitole 8. Tato metoda byla v Kapitole 10.4 porovnána s původní metodou a je zde ukázáno, že zavedením této metody se chování svalu buď zlepšilo (Iliacus), nebo zůstalo v podstatě stejné (Gluteus maximus a medius). V rámci tvorby adaptivních vláken proběhla i úprava parametru  $w$  (viz Kapitola 9.2.4). Bez této úpravy by adaptivní vlákna mohla způsobovat problémy v průběhu deformace. Výsledky této metody bych označil za uspokojivé a poměrně očekávané – více vláken a jejich lepší rozložení přispělo k přesnější reprezentaci a díky tomu se zlepšil i průběh deformace.

Druhým cílem bakalářské práce bylo navrhnout dodatečnou úpravu svalových vláken během deformace (viz Kapitola 7) a implementovat navrženou metodu (viz Kapitola 9). Opět proběhlo porovnání této metody a původního stavu projektu (viz Kapitola 10.6). Již v průběhu implementace této metody se postupně začaly objevovat problémy – nejprve šlo o dobu běhu tvorby SDF a poté o opravu kolizí. Metoda je sice funkční, ale má svá omezení – pokud body nejsou v kolizi s kostí, ale jejich spojnice kostí projde, není v aktuální implementaci způsob, jak tomuto zabránit. Dále sval Iliacus činil problém, protože se sval deformoval i do kloubní jamky, což je samozřejmě fyziologicky nemožné. I přes tato omezení metoda prokazuje mírné zlepšení, ale ne tak dobré, jako metoda tvorby adaptivních vláken.

Nakonec bylo představeno srovnání originální implementace a implementace vytvořené v rámci této bakalářské práce. Toto srovnání lze vidět v Kapitole 10.9.

Tato práce má dozajista potenciál k budoucímu rozvoji, neboť se jedná o stále ne ideálně vyřešené téma a domnívám se, že projekt, v rámci kterého byla tato práce vypracována, má rozhodně šanci na úspěch. Pro praktickou

využitelnost metody je ale nutné další zlepšení jejího fungování. Na tuto práci lze v budoucnu navázat tím, že se dořeší detekce kolizí v případě, kdy oba body jsou mimo kost, ale jejich spojnice kostí prochází. Možným řešením je druhý přístup navržený v Kapitole 7.2. Poté by bylo vhodné zlepšit ukládání SDF, aby se ukládalo pro jiný model pod jiným názvem, což zatím nebylo řešeno. Dále ještě v budoucnu bude důležité navržení druhé adaptivní funkce – v této práci byl navržen a implementován algoritmus upřesňující počet segmentů vlákna, ale samotný počet vláken byl definován uživatelem (viz Kapitola 4.1). I tento počet by však bylo ideální určovat adaptivně, protože se opět jedná o člověkem těžko určitelný parametr.

# Přehled zkratk

MWP / MWP2 – Musclewrapping Projekt  
MSM – Muskuloskeletální model(y)  
HDL – Living Human Digital Library  
SDF – Signed distance field (resp. function)  
XML - Extensible Markup Language

# Literatura

- [1] CATMULL, E. – ROM, R. A CLASS OF LOCAL INTERPOLATING SPLINES. In BARNHILL, R. E. – RIESENFELD, R. F. (Ed.) *Computer Aided Geometric Design*. : Academic Press, 1974. s. 317 – 326. doi: <https://doi.org/10.1016/B978-0-12-079050-0.50020-5>. Dostupné z: <http://www.sciencedirect.com/science/article/pii/B9780120790500500205>. ISBN 978-0-12-079050-0.
- [2] *Eigen* [online]. [cit. 16/04/2021]. Dostupné z: [https://eigen.tuxfamily.org/index.php?title=Main\\_Page](https://eigen.tuxfamily.org/index.php?title=Main_Page).
- [3] KOHOUT, J. – KUKAČKA, M. Real-Time Modelling of Fibrous Muscle. *Computer Graphics Forum*. 2014, 33, 8, s. 1–15. doi: 10.1111/cgf.12354. Dostupné z: <https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12354>.
- [4] KOSCHIER, D. et al. An hp-Adaptive Discretization Algorithm for Signed Distance Field Generation. *IEEE Transactions on Visualization and Computer Graphics*. oct 2017, 23, 10, s. 2208–2221. ISSN 1941-0506. doi: 10.1109/TVCG.2017.2730202.
- [5] KOSCHIER, D. *Discregrid* [online]. [cit. 15/04/2021]. Dostupné z: <https://github.com/InteractiveComputerGraphics/Discregrid>.
- [6] MODENESE, L. – KOHOUT, J. Automated Generation of Three-Dimensional Complex Muscle Geometries for Use in Personalised Musculoskeletal Models. *Annals of Biomedical Engineering*. 2020, 48, 6, s. 1793–1804. doi: 10.1007/s10439-020-02490-4. Dostupné z: <https://doi.org/10.1007/s10439-020-02490-4>.
- [7] *OpenSim* [online]. SimTK, 2020. [cit. 05/09/2020]. OpenSim project site. Dostupné z: <https://simtk.org/projects/opensim>.
- [8] RAHMAN, M. et al. Musculoskeletal Model Development of the Elbow Joint with an Experimental Evaluation. *MDPI*. April 2018, 5(2), 31. doi: 10.3390/bioengineering5020031. Dostupné z: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6027184/>.
- [9] VALENTE, G. et al. 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*. February 2012, 226, s. 161–9. doi: 10.1177/0954411911425863. Dostupné z:

[https://www.researchgate.net/publication/223981126\\_Muscle\\_discretization\\_affects\\_the\\_loading\\_transferred\\_to\\_bones\\_in\\_lower-limb\\_musculoskeletal\\_models](https://www.researchgate.net/publication/223981126_Muscle_discretization_affects_the_loading_transferred_to_bones_in_lower-limb_musculoskeletal_models).

- [10] *VTK* [online]. [cit. 10/04/2021]. Dostupné z: <https://vtk.org/>.
- [11] *Centripetal Catmull–Rom spline* [online]. Wikipedia, the free encyclopedia. [cit. 22/11/2020]. Dostupné z: [https://en.wikipedia.org/wiki/Centripetal\\_Catmull%E2%80%93Rom\\_spline](https://en.wikipedia.org/wiki/Centripetal_Catmull%E2%80%93Rom_spline).
- [12] YUKSEL, C. – SCHAEFER, S. – KEYSER, J. Parameterization and applications of Catmull–Rom curves. *Computer-Aided Design*. 2011, 43, 7, s. 747–755. ISSN 0010-4485. doi: <https://doi.org/10.1016/j.cad.2010.08.008>. Dostupné z: <https://www.sciencedirect.com/science/article/pii/S0010448510001533>.



# A Uživatelská dokumentace

Na přiloženém DVD se nachází zdrojové kódy aplikace a soubory pro překlad pomocí CMake, použité konfigurační a vstupní soubory. Zdrojové soubory včetně *CMakeLists.txt* se nacházejí uvnitř složky *Sources* a soubory použité pro spuštění programu lze nalézt ve složce *Data*. Celý program je napsán v jazyce C++. Použité knihovny nejsou k této práci přiloženy, ale níže se nachází podrobný popis instalace i s příslušnými odkazy na potřebné knihovny.

## A.1 Překlad a instalace

### A.1.1 Závislosti

Pro správný překlad a instalaci programu, v rámci kterého byla tato bakalářská práce vypracována, je nejprve potřeba nainstalovat knihovny, na kterých je tento projekt závislý. Všechny knihovny využívají pro svůj překlad CMake (verze alespoň 3.10). Jimi jsou:

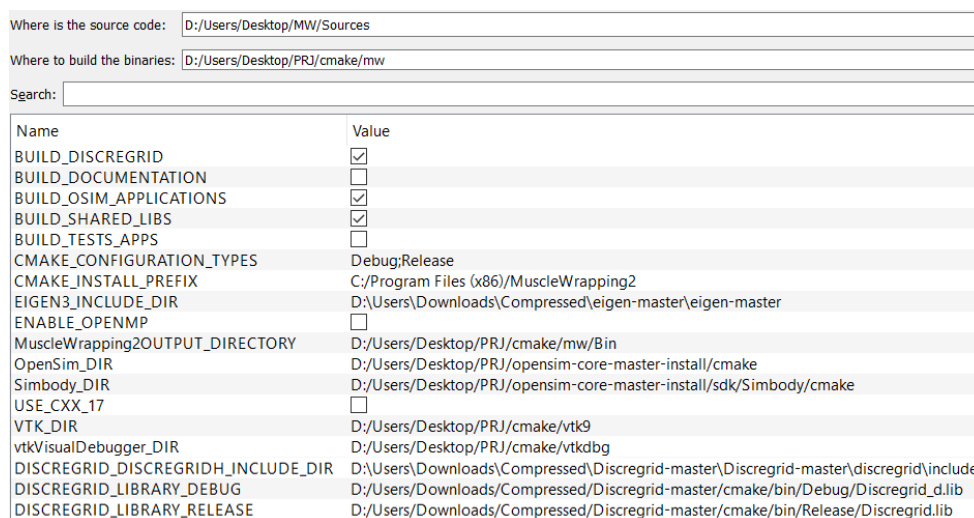
- VTK [odkaz] – je nutné stáhnout nejnovější verzi 9.x. Mezi verzemi 8 a 9 došlo k podstatným změnám, které by zapříčinily možnou nefunkčnost. Dále je pro tuto knihovnu nutné aplikovat patch, který opravuje podstatný bug. Tento patch bohužel stále nebyl přidán do samotné knihovny a je nutno ho provést manuálně. Je nutno provést změny z toho merge requestu [odkaz].
- vtkVisualDebugger [odkaz] – jedná se o knihovnu závislou na VTK, tudíž je potřeba napřed nainstalovat samotné VTK.
- OpenSim [odkaz] – detailní popis instalace je popsán v README.md této knihovny, doporučuji využít superbuid, který doinstaluje všechny závislosti OpenSim automaticky.
- Eigen3 [odkaz].
- Discregrid [odkaz] – tato knihovna je závislá na Eigen3, je opět potřeba nejprve mít nainstalovaný Eigen3 než bude spuštěn překlad této knihovny.

## A.1.2 Překlad MWP2

Samotný překlad tohoto projektu probíhá opět pomocí CMake. Celý tento překlad byl testován v CMake 3.16 na platformě Win10. Soubory byly překládány pro MS Visual Studio (2019). Následující popis předpokládá stejné specifikace.

Pro správný překlad je potřeba mít nainstalované všechny knihovny z Kapitoly A.1.1. CMake je potřeba navést do složky *Sources*, kde se nachází soubor *CMakeLists.txt*. Při překladu pomocí CMake bude uživatel postupně dotazován na složky s knihovnami. První je zapotřebí specifikovat knihovny VTK a VtkVisualDebugger. Po jejich zadání je důležité zkontrolovat, že je zaškrtnutá možnost *BUILD\_OSIM\_APPLICATIONS*. Pokud tomu tak není, je nutno ji zaškrtnout. Poté lze již specifikovat umístění instalace knihovny OpenSim.

Pozor, narozdíl od VTK a VtkVisualDebugger, OpenSim vyžaduje cestu do složky, kam byl instalován, nikoliv do složky, kam byl přeložen pomocí CMake (tato složka je specifikována při překladu OpenSim). Dále je nutné zaškrtnout *BUILD\_DISCREGRID* a specifikovat cesty k Eigen3 a Discregrid. Pokud je vše nastaveno, ještě doporučuji zkontrolovat, že je zaškrtnuto *BUILD\_SHARED\_LIBS*. Příklad konfigurace, která byla použita po celou dobu práce na této bakalářské práci, lze vidět na Obrázku A.1.

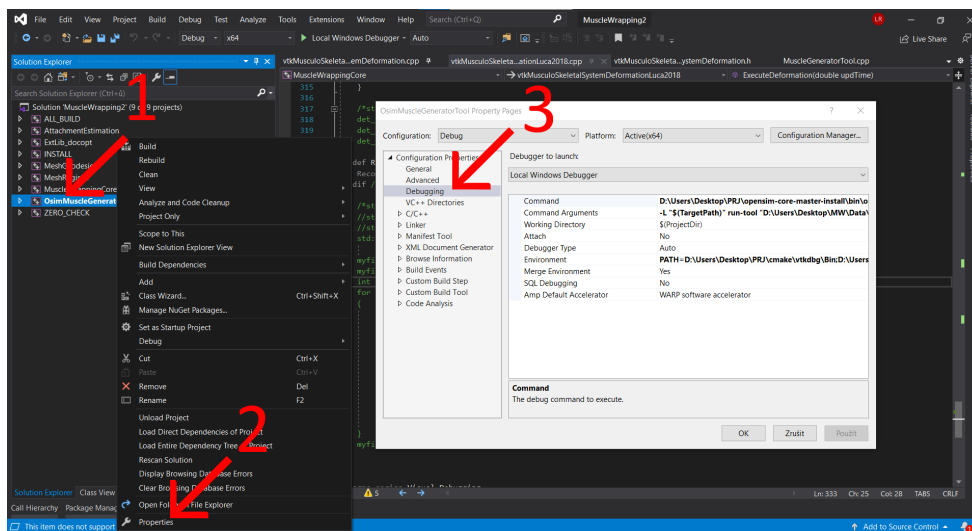


Name	Value
BUILD_DISCREGRID	<input checked="" type="checkbox"/>
BUILD_DOCUMENTATION	<input type="checkbox"/>
BUILD_OSIM_APPLICATIONS	<input checked="" type="checkbox"/>
BUILD_SHARED_LIBS	<input checked="" type="checkbox"/>
BUILD_TESTS_APPS	<input type="checkbox"/>
CMAKE_CONFIGURATION_TYPES	Debug;Release
CMAKE_INSTALL_PREFIX	C:/Program Files (x86)/MuscleWrapping2
EIGEN3_INCLUDE_DIR	D:\Users\Downloads\Compressed\eigen-master\eigen-master
ENABLE_OPENMP	<input type="checkbox"/>
MuscleWrapping2OUTPUT_DIRECTORY	D:/Users/Desktop/PRJ/cmake/mw/Bin
OpenSim_DIR	D:/Users/Desktop/PRJ/opensim-core-master-install/cmake
Simbody_DIR	D:/Users/Desktop/PRJ/opensim-core-master-install/sdk/Simbody/cmake
USE_CXX_17	<input type="checkbox"/>
VTK_DIR	D:/Users/Desktop/PRJ/cmake/vtk9
vtkVisualDebugger_DIR	D:/Users/Desktop/PRJ/cmake/vtkdbg
DISCREGRID_DISCREGRIDH_INCLUDE_DIR	D:\Users\Downloads\Compressed\Discregrid-master\Discregrid-master\discregrid\include
DISCREGRID_LIBRARY_DEBUG	D:/Users/Downloads/Compressed/Discregrid-master/cmake/bin/Debug/Discregrid_d.lib
DISCREGRID_LIBRARY_RELEASE	D:/Users/Downloads/Compressed/Discregrid-master/cmake/bin/Release/Discregrid.lib

Obrázek A.1: Ukázka CMake konfigurace

### A.1.3 Spuštění MWP2

Pokud proběhl CMake v pořádku, měl by se ve složce, kam byl uskutečněn překlad, nacházet MS Visual Studio projekt (.sln). Jeho spuštěním se otevře celý projekt. Pro překlad a spuštění je nejprve nutno specifikovat parametry. Postupem uvedeným na Obrázku A.2 (OsimMusleGeneratorTool – Properties – Debugging) se lze dostat do konfiguračního dialogu.



Obrázek A.2: Cesta k nastavení před spuštěním

Zde je nutno specifikovat *Command* na *opensim-cmd.exe*. Ten byl nainstalován v rámci OpenSim, jen je potřeba ho specifikovat se správnou cestou. Dále je nutno nastavit *Command Arguments* na *-L \$(TargetPath) run-tool <cesta\_ke\_xml>*. Cesta ke XML představuje cestu ke jednomu z konfiguračních souborů nacházejících se ve složce Data. Více o těchto souborech bude popsáno v Kapitole A.1.4. Dále je potřeba do proměnné *Environment* přidat *PATH=<vtk\_bin>; <vtkvisualdbg\_bin>; <opensim\_bin>; <discregrid\_bin>*. Jednotlivé *<xxx\_bin>* parametry jsou cesty do složek s .dll soubory jednotlivých knihoven. Poté, co jsou všechny parametry a cesty nastaveny, je důležité nastavit jako spouštěný projekt právě *OsimMusleGeneratorTool*, a poté je již možné spustit samotný překlad a vizualizaci.

### A.1.4 Konfigurace

Složka data obsahuje mnoho konfiguračních souborů, které je možno spouštět. Soubory začínající na *setup\_MuscleGeneratorTool\_xxx.xml* nacházející se uvnitř složky Data jsou základní soubory použité pro konfiguraci. Místo

*xxx* je vždy jméno (resp. zkratka) svalu. Každý soubor je specifický pro jeden sval a jeho nastavením ve výše uvedeném konfiguračním dialogu Visual Studia lze mezi svaly přepínat. Soubory pro *GMax*, *GMed* a *Iliacus* byly využity v průběhu této práce a s jejich pomocí vznikly výše uvedené grafy a obrázky. Uvnitř těchto souborů lze nalézt druhé nastavení podstatné pro tuto práci – *<motion\_file>*. Tato položka specifikuje soubor, ve kterém je popsána kinematika. V rámci práce byly použity soubory *hip\_flexion.mot*, *hip\_rotation.mot* a *hip\_adduction.mot*. Každý z těchto souborů odpovídá pohybu v názvu (flexe, rotace, addukce).

Parametr *<kinematics\_fibre\_algorithm>* určuje, že se bude používat metoda Luca2018 (viz Kapitola 3), jeho změnou by byla vypuštěna celá polovina této práce, tudíž je nutné ho neměnit.

Za zmínku stojí i parametr *<line\_res>*, který je nyní nastavený na 1 – tento parametr je právě oním parametrem, který nyní specifikuje až metoda adaptivních vláken (viz Kapitola 8).

Dále se zde nachází parametr *<num\_of\_lines>*, který specifikuje počet vláken, ze kterých se sval skládá. Tento parametr je do budoucna také nutno nahradit adaptivní metodou (viz rozbor práce do budoucna v Kapitole 11).

Důležitým parametrem je také *<output\_muscle\_analysis\_file\_prefix>*, který specifikuje složku, do které se uloží soubory s analyzovanými vlákny. Pouze upozorním, že při spuštění s tímto parametrem je nutno také správně nastavit parametr *<coordinate>* na správný pohyb, jinak generované soubory nebudou obsahovat správné hodnoty. Parametr *<coordinate>* musí obsahovat pohyb, který je nastaven v parametru *<motion\_file>* a k němu dopsané *\_r* (např. *hip\_flexion\_r*).

Zbylé parametry jsou víceméně nepodstatné pro tuto práci, jen je potřeba nechat je nastavené tak, jak jsou.

### A.1.5 Program za běhu

Po spuštění programu se zvolenou konfigurací se otevře konzole, do které se budou vypisovat informace o otevíraném modelu atd. Poté, co proběhne deformace a tvorba (resp. načtení) SDF, je otevřeno okno s vizualizací. Vizualizace odpovídá svalů zvolenému pomocí XML souboru a pohybu v něm uvedenému. Vizualizace jde krokovat pomocí klávesy ESC. Po ukončení vizualizace je případně provedena analýza vláken, pokud byl uveden parametr *<output\_muscle\_analysis\_file\_prefix>* v konfiguračním souboru.

# B Obsah DVD

Přiložené DVD obsahuje následující soubory:

- Sources – složka se zdrojovými soubory a *CMakeLists.txt*.
- Documentation – složka se zdrojovými soubory a textem této bakalářské práce.
- Data – složka s konfiguračními soubory.