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

Bakalářská práce

Grafické uživatelské rozhraní pro OpenSim plug-in automatického generování svalových vláken

Plzeň 2022 Kment Tomáš

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Tomáš KMENT**
Osobní číslo:	**A19B0091P**
Studijní program:	**B0613A140015 Informatika a výpočetní technika**
Specializace:	**Informatika**
Téma práce:	**Grafické uživatelské rozhraní pro OpenSim plug-in automatického generování svalových vláken**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se s interním projektem Muscle Wrapping 2.0.
2. Popište softwarovou architekturu systému OpenSim 4.0, přičemž se zejména zaměřte na možnosti jeho rozšiřování prostřednictvím uživatelských plug-ins.
3. Popište plug-in automatického generování svalových vláken pro OpenSim, vytvořeného v rámci projektu Muscle Wrapping 2.0.
4. Navrhněte a naimplementujte grafické rozhraní umožňující snadnou konfiguraci a spouštění plug-inu automatického generování svalových vláken přímo z vizuálního prostředí OpenSim.
5. Vytvořené softwarové vybavení uživatelsky otestujte na úloze flexe kyčelního kloubu pro vybrané svaly v oblasti stehenní kosti datového setu LHDL.

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

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: **4. října 2021**
Termín odevzdání bakalářské práce: **5. května 2022**

L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

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

V Plzni dne 14. října 2021

Prohlášení

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

V Plzni dne 5. května 2022

Kment Tomáš

Abstract

This bachelor thesis analyzes the structure of the open-source software architecture employed in the OpenSim system in order to extend it with a graphical interface for the Muscle Wrapping 2.0 plug-in. Based on the performed analysis as well as the plug-in analysis, possible interfaces are proposed. The selected graphical interface design is based on a survey of user preferences and its implementation is based on the Swing library of the Java platform. The work further tests the application (configuration module) on the problem of hip flexion for selected muscles in the femur of the LHDL data set.

Abstrakt

Tato práce analyzuje strukturu softwarové architektury open-source systému OpenSim za účelem jeho rozšíření o grafické rozhraní pro přídavný modul (plug-in) Muscle Wrapping 2.0. Na základě provedené analýzy a rovněž analýzy plug-inu jsou navržena možná rozhraní. Vybraný návrh grafického rozhraní vychází z průzkumu uživatelské preference a jeho implementace je postavena na knihovně Swing platformy Java. Práce dále testuje aplikaci (konfigurační modul) na úloze flexe kyčelního kloubu pro vybrané svaly v oblasti stehenní kosti datového setu LHDL.

Poděkování

Rád bych poděkoval Doc. Ing. Josefu Kohoutovi, Ph.D. za jeho vstřícný, obětavý přístup, cenné rady a připomínky, které mi významnou měrou pomohly při vypracování bakalářské práce.

Obsah

1 Úvod

Doktoři, mechanici a jiní přírodovědní vědci, kteří se zabývají muskuloskeletálním systémem využívají již delší dobu takzvané výpočtové modely svalů. Anatomie těchto svalů je založena na měření kadaverózních studií, kterých je ovšem pouze omezený počet. Tato měření se v dnešní době pokládají za obecné, čímž zkreslují výsledky pro svaly, které se morfologicky liší dle stavby těla člověka. Díky tomuto neosobnímu přístupu vědci pracují s nepřesnými daty. Tento problém se snaží vyřešit vědci z katedry informatiky a výpočetní techniky Západočeské Univerzity v Plzni ve spolupráci s vědci z katedry civilního a environmentálního inženýrství, Imperial College London.

Ze spolupráce vznikl interní projekt zvaný Muscle Wrapping 2.0, jehož cílem je vytvořit personifikované modely dle uživatelského vstupu. Výsledků dosahuje pomocí diskretizace svalů na jednotlivá vlákna (lines of action). Celý sval je tvořen libovolným počtem vláken, které se skládají z uživatelem definovaného počtu lomených čar. Z mechanického hlediska je tento způsob reprezentace v pořádku, dokud bereme v potaz, že segmenty přímek prochází těžišti rozložení sil v uvažovaných svalových úsecích. Jednotlivá vlákna jsou stejně jako reálné svaly připevněna ve svém počátku a směřují k oblasti kde dochází k pohybu svalu.

Hlavním cílem této práce je navrhnout a implementovat grafické rozhraní systému OpenSim. OpenSim je systém pro biomechanické modelování, simulaci a analýzu. Rozhraní bude zastávat efektivní konfigurační modul, pomocí kterého může uživatel lehce vytvářet nové konfigurační soubory pro již vytvořený plug-in Muscle Wrapping 2.0 přímo z vizuálního prostředí aplikace OpenSim.

2 Anatomická část

Jelikož je částí kvalifikační práce generování svalových vláken, musíme si nadefinovat, jak muskuloskeletální systém vypadá a funguje v reálném světě. V této kapitole budou vysvětleny a popsány důležité prvky anatomie lidského těla pro toto téma, muskuloskeletální systém, jeho stavba a rozdělení pro lepší pochopení tématu. Pokud nebude explicitně určeno, všechny informace uvedené v kapitole vychází z [13].

2.1 Rozdělení rovin těla

Z důvodu lepší orientace se dá tělo rozdělit pomocí rovin. Rovinou se v tomto smyslu myslí řez, jenž je veden skrz postavu člověka. Tělo můžeme rozdělit na čtyři různé části:

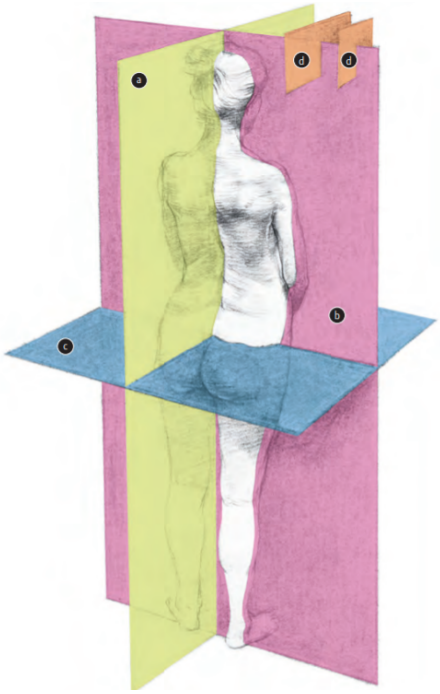
(a) **Rovina mediální** - vertikální řez z pohledu zepředu, rozděluje tělo na dvě identické poloviny.

(b) **Rovina frontální** - svislý řez z pohledu ze strany, rozděluje tělo na přední část a zadní.

(c) **Rovina transversální** - horizontální řez, rozděluje tělo na část horní a dolní

(d) **Rovina sagitální** - řez rovnoběžný s rovinou mediální

Pro lepší představivost jsou tyto roviny zobrazeny na Obrázku 2.1. V pozdějších kapitolách se budeme zabývat pohybem svalů, který je popsán pomocí zmíněných rovin.



Obrázek 2.1: Rozdělení lidského těla na jednotlivé roviny

2.2 Muskuloskeletální systém

Muskuloskeletální systém je systém lidského těla, který poskytuje našemu tělu oporu, stabilitu, tvar a pohyb[15]. Můžeme ho klasifikovat na muskulární a skeletální systém.

2.2.1 Skeletální systém

Systém je složen z 206 kostí. Kosti se dají pokládat za pasivní prvek pohyblivého aparátu, kdežto svaly zastupují pozici prvku aktivního. Jedná se o pevnou, tvarově neměnnou část těla, která je spojena klouby. Jejich multifunkčnost slouží především k ochraně, možnosti pohybu či podpoře držení celého lidského těla[1]. Vlastnosti jednotlivých kostí, jako je jejich tvar, pevnost, tloušťka a délka, jsou podmíněné mechanickými atributy kostry. Obecně můžeme kosti rozdělovat takto:

- **Dlouhé kosti** - jedná se o kosti trubicovitého tvaru s pohyblivým kloubním spojením.
- **Krátké kosti** - zde mluvíme například o obratlích nebo o kostech v oblasti zápěstí.

- **Ploché kosti** - slouží jako základ širokým svalovým úponům, hlavně v oblasti zad.
- **Sezamské kosti** - zabudovány v místě, kde svalová šlacha prochází přes kloub. Hlavním představitelem těchto kostí je češka.

Jak již bylo zmíněno, kosti jsou spojeny pomocí kloubů. Aby se dvě kosti daly spojit kloubem, musí mít konec první kosti tvar reliéfu a konec druhé kosti jeho negativní tvar. Umožňují tak kostem provádět jemné a hladké pohyby. Na hlavici kosti se nachází chrupavka, která pomocí dvou vrstev (zevní a vnitřní) tvoří kloubní pouzdro. Klouby se rozdělují dle směrnice pohybu na následující:

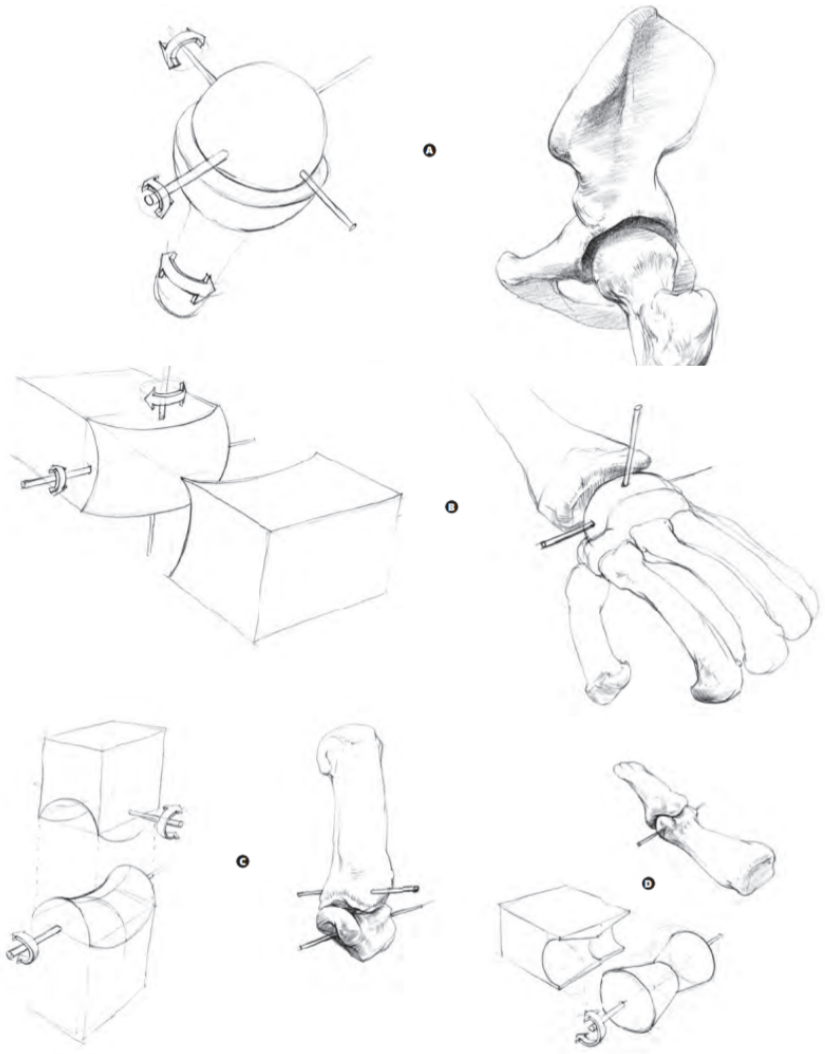
(a) **Kloub kulovitý** - zprostředkovává pohyby podél tří os společně s rotací.

(b) **Kloub elipsovitý** - vejcovitého tvaru, vytváří pohyb ve dvou směrech.

(c) **Kloub sedlový** - vyznačuje se tvarem jezdeckého sedla. Taktéž je pohyblivý podél dvou os.

(d) **Kloub kladkový** - podporuje pouze jeden směr, kolmý na osu kosti.

(e) **Kloub kolový** - podobný předešlému kloubu, zmíněná osa je souběžná s podélnou osou kosti.



Obrázek 2.2: Rozdělení kloubů dle směrnice pohybu[13]

2.2.2 Muskulární systém

Důležitým prvkem muskulárního systému je svalová tkáň. Dle kategorizace svalové tkáně rozdělujeme svaly do tří skupin. Jedná se o sval srdeční, hladký a kosterní. Pro naši tematiku je důležitá pouze třetí skupina a tudíž jakákoliv zmínka o svalech v dalších kapitolách bude poukazovat na svaly kategorie svalů kosterních. Kategorizační faktor svalů je jejich vnější tvar:

- **Svaly vřetenovité** - rozdělují se dle množství hlav. Příkladem muže být sval dvojhlavý, nebo trojbříškový.
- **Svaly ploché** - mají tenké bříško (hlavu) rozprostřenou do plochy. Mezi takové svaly řadíme například svaly břišní.

- **Svaly kruhovité** - představují snopce svalu stočeny do kruhu. Představiteli takových svalů jsou svaly obličeje, v oblastech očí a úst.

Struktura svalu

Sval je tvořen množinou příčně pruhovaných vláken společně s povázkou (vazivová vrstva, vytvářejicí šlachy), která obaluje povrch svalů. Sval můžeme rozštěpit na začátek(úpon na kost), hlavu a na úpon svalu. Hlava, někdy taky zmíněná jako bříško, je nejširší a nejviditelnější místo svalu. Pro správnou funkčnost musí sval splňovat několik kritérií. Sval musí být schopen kontrakce (smrštění, zkrácení) a excitace (natažení). Musí být elastický a musí být schopen excitace bez toho, aniž by došlo k přetrhnutí svalu.

Pohyb svalu

Hlavním účelem muskulárního systému je vytvářet pohyb. V závislosti na rovině či ose pohybu můžeme pohyb kategorizovat následujícím způsobem[3]:

- **Flexe a extenze** - slovní spojení popisuje nárůst či redukci úhlu mezi kostmi. Nachází se v sagitální rovině, jež byla zmíněna v kapitole 2.1 okolo popřední osy. Příkladem takového pohybu může být ohyb nohy v oblasti kolenního kloubu.

- **Abdukce a addukce** - vyobrazuje pohyb od/k ose těla. Takovým pohybem je například roznožení nebo přinožení v oblasti dolních končetin. Celý pohyb je taktéž prováděn v sagitální ose.

- **Rotace** - v tomto smyslu je rotace myšlena okolo vertikální osy v transversální rovině. Tu dále můžeme rozštěpit na rotaci vnitřní a vnější (zevní). Příkladem může být rotace stehenního svalu.

- **Supinace a pronace** - pohyb je znám především v oblastech předloktí, ale může se nacházet i v pásmovém okolí kotníku.

3 Reprezentace muskuloskeletálního systému

Muskuloskeletální systém, jehož modely se využívají v oblasti biomechaniky a biomedicíny, se skládá z dvou podružných systémů, muskulárního a skeletálního. Muskuloskeletální modely, v některých článcích známé pod zkratkou MSM, jsou modely lidského těla, založené na muskuloskeletáním systému. Pokud by byl vytvořen komplexní model, který odpovídá realitě, mohl by ušetřit čas jak vědcům a doktorům, tak i pacientům. Taková reprezentace by ale byla velice náročná a proto je potřeba model zjednodušit. Model má kosti, stejně jako lidské tělo v reálném světě, rigidní. Svaly se v MSM znázorňují mnoho způsoby. Jelikož se jedná o komplexní strukturu, tu jsme si popsali v podkapitole 2.2.2, reprezentujeme povrch svalu.

Jak již bylo zmíněno, kosti jsou rigidní prvek a jejich simulace není oproti svalům náročná. Jediný pohyb, který kosti provádí a je nutný simulovat, je rotace okolo určité osy ve třech dimenzích. Snímek kosti lze získat například pomocí rentgenu[14]. Díky němu se dá pak vygenerovat objekt kosti, nejčastěji tvořen sítí trojúhelníků nebo jiných polygonů. Klouby jsou umístěny stejně jako v lidském těle a reprezentují řídící jednotku pohybu kostí.

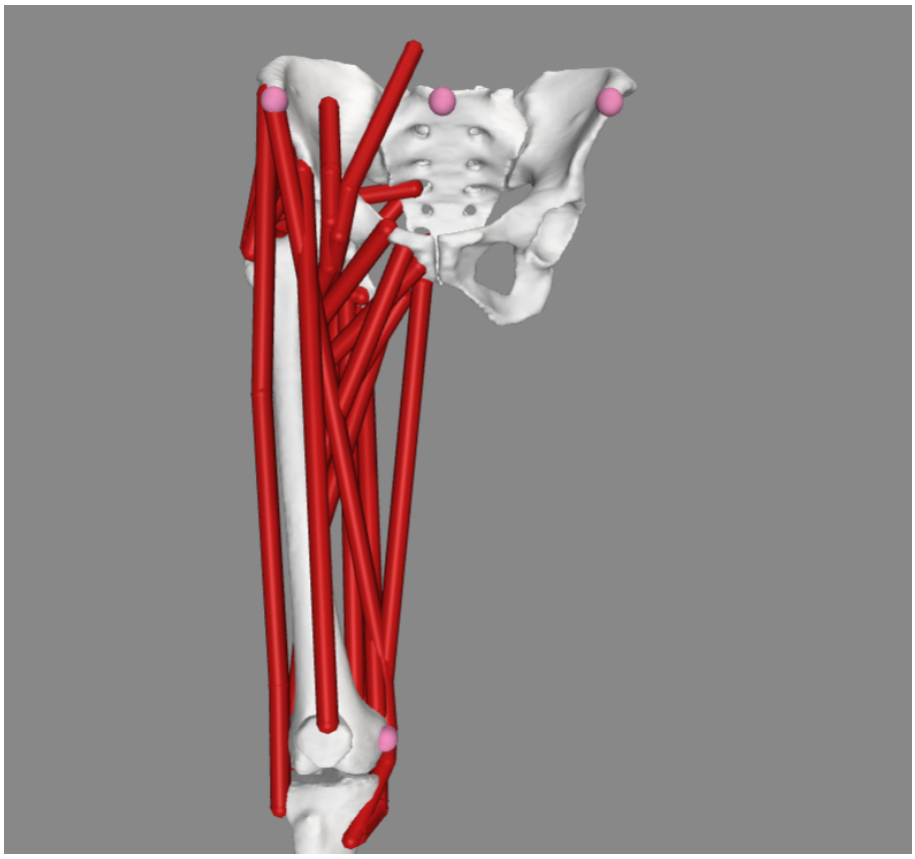
3.1 Geometrická reprezentace svalů

Reprezentace svalu v MSM je ten pravý oříšek. Jak již víme z kapitoly 2.2.2, sval je tvořen množinou příčně pruhovaných vláken společně s vazivovou vrstvou. Model musí být schopen tato vlákna nějakým způsobem reprezentovat a v případě simulačních animací musí být schopen sval dobře přichytit. Sval jako takový se dá zdigitalizovat několika způsoby:

- Můžeme ho popsat pomocí sítě polygonů, stejně jako kost. I když by na jednu stranu odpovídal realitě vizuálně lépe než níže zmíněné techniky, mechanická realističnost by zde mohla představovat problém.
- Lze místo svalu vybrat objekt jednodušší, třeba elipsoidy. Ty se využívají kvůli řízení zachování objemu.
- Nebo je nasnadě využít reprezentaci pomocí vláken. Tu si popíšeme detailněji neboť jeden ze dvou níže vypsaných způsobů využívá OpenSim a MuscleWrapping 2.0

3.1.1 Line of action

Line of action je lomená čára, kterou lze reprezentovat jednotlivá svalová vlákna a šlachy svalu[8]. Výhodou takové reprezentace je rychlost zpracovaní a jednoduchá deformovatelnost. Nenabízí ovšem pouze výhody. Svým způsobem reprezentuje výslednici síly, a i když většinou výslednice působí ve směru vláken svalu, což je správně, nemusí tomu být tak vždy. Problém by mohl nastat, pokud bychom použili jednu nebo dvě lomené čáry pro reprezentaci velkého svalu, jakým je například velký sval hýžďový (gluteus maximus). Byla provedena studie na IOR (Instituto Ortopedico Rizzoli), která poukazuje na použití pouze jedné lomené čáry pro již zmíněný sval. Výsledkem experimentu byla relativní chyba dosahující až 75 % pro velké svaly. Bavíme se zde pouze o použití jedné výslednice sil na velký sval. Studie dále prokazuje, že s rostoucím počtem lomených čar chyba asymptoticky klesá[17]. Line of action musí mít předem určený počátek (většinou na povrchu kosti) a konec (úponová část). Bonusem pak jsou takzvané via points, body skrz které lomená čára prochází. Tyto informace nám sdělují následující. Pro správnou reprezentaci většího svalu pomocí line on action je nutno vytvořit větší počet výslednic sil. Lomené čáry ale musí nadefinovat uživatel, většinou expert z oboru, a jejich definice bývá časově náročná. Kohout a Kukačka[8] argumentují, že časová náročnost je hlavním důvodem, proč se v běžné praxi nepoužívají více než dvě line of action. Na obrázku 3.1 můžeme vidět reprezentaci svalů pomocí line of action v Systému OpenSim.

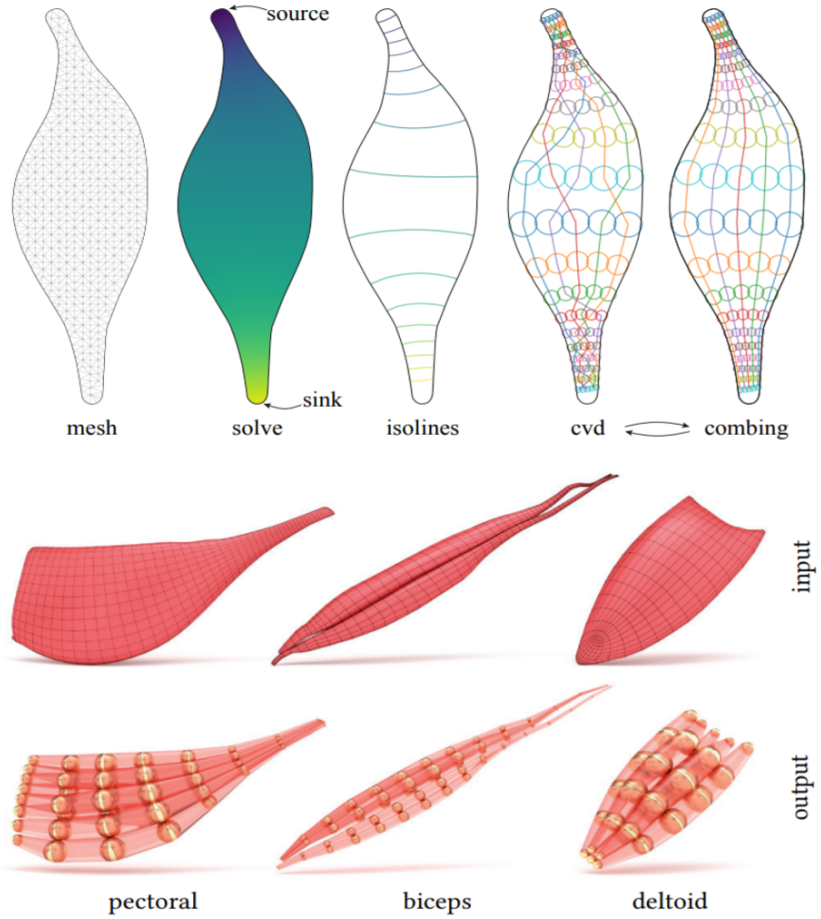


Obrázek 3.1: Reprezentace kloubů, kostí a svalů v OpenSim pomocí line of action

3.1.2 VIPER

Z podkapitoly 3.1.1 víme, že Line of action obsahuje řadu nevýhod, například již zmíněná nutnost definice lomených čar expertem z oboru nebo že úroveň diskretizace závisí na individuální anatomii jedince. Přístupů, kterými lze problémy odstranit je více a jedním z nich je VIPER. Volume Invariant Position-based Elastic Rods (objemově invariantní elastické tyče založené na pozici) neboli VIPER, je způsob, jakým lze simulovat dynamické chování svalu v čase. Algoritmy založené na PBD (Position-Based Dynamics) využívají řetězce částic spojené distančními omezeními k reprezentaci lan a vláken. Oproti PBD ropes, které simulují pozici, a Cosserat rods, které společně s pozicí simulují orientaci, VIPER přidává možnost úpravy velikosti. Díky tomu lze simulovat schopnosti jako je flexe a extenze, zmíněné v podkapitole 2.2.2. Automatickému procesu, ve kterém přeměníme sval na skupinu vláken se říká *Viperization*[5] a je graficky popsán na Obrázku 3.2. Uživatel vyznačí počáteční a koncovou pozici svalu. Proces vypočte harmonickou

funkci objemu a vyextrahuje diskrétní izo-plochy. Na každé ploše vytvoří stejný počet částic, které následně propojí svalovým vláknem tak, aby byla jeho délka co nejmenší.



Obrázek 3.2: Proces *Viperization*[5]

3.2 Simulace svalů

Oproti kostem, které jsou zpravidla reprezentovány jedním způsobem a při pohybu zachovávají svůj tvar, je simulace svalů podstatně komplikovanější, neboť svaly tvar při pohybu nezachovávají. Simulační metody se liší podle toho, jaká geometrická reprezentace svalů byla zvolena.

V případě, že se jedná o jednoduché Line of action, ať již s via points nebo bez nich, lze sval simulovat pomocí standardu OpenSim, rovnou v prostředí systému. Další možností simulace je použít algoritmus *Luca2018* [9]. V neposlední řade je zde algoritmus *Cervenka2019*, který už ale pracuje se sítí

polygonů popisující povrch svalu a k simulaci využívá PBD zmíněné v podkapitole 3.1.2.

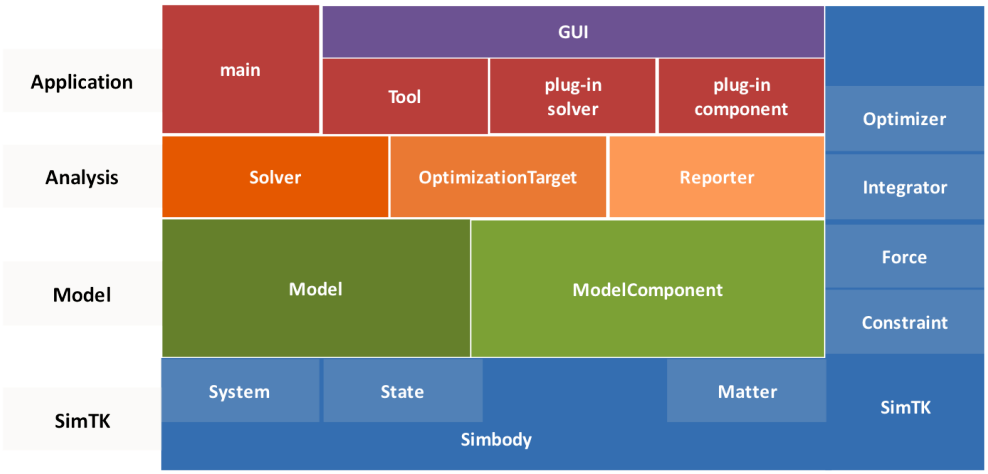
Simulace VIPER probíhá až po procesu *Viperization*[5] (předělání 3D objektu svalu na svalová vlákna). Koncové body vláken jsou kinematicky připojené ke kostem, zatímco mezisvalové kolize jsou detekovány a řešeny. Svaly lze aktivovat vložením vnitřních sil nebo zkrácením délky vláken.

3.3 Shrnutí

I přesto, že line of action může efektivně reprezentovat sval, vychází na povrch několik nevýhod. Bylo nalezeno, že úroveň požadované diskretizace závisí na individuální anatomii jedince[9]. Například vysoká míra diskretizace svalu obklopujícího kyčelní kloub je nezbytná k zajištění přesného odhadu. Další handicap byl zmíněn v podkapitole 3.1.1, tedy nutnost definice lomených čar expertem z oboru. Řešení některých nevýhod vede na automatické generování, které řeší projekt Muscle Wrapping 2.0. Pro simulaci Muskuloskeletálních modelů slouží řada programů. Mezi komunitní favority patří i OpenSim.

4 OpenSim

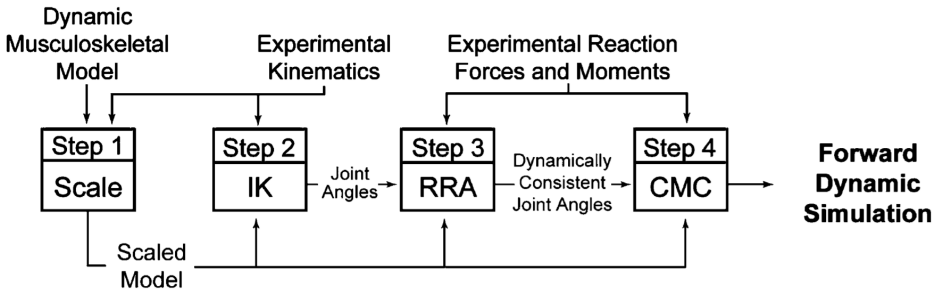
OpenSim je volně dostupnou platformou modulární struktury vytvořenou za účelem modelování, analýzy a simulace neuro-muskuloskeletálního systému. Díky své otevřenosti umožňuje výzkumníkům a vědcům z rozdílných laboratoří reprodukovat výsledky a společně pracovat na jejich zlepšeních. Aby mohli společně pracovat, OpenSim poskytuje dva možné nástroje, SimTrack a malou část SIMM. SimTrack, dále jen SimTK, bude zmíněn v kapitole 4.1. Co se SIMM týče, jedná se o muskuloskeletální modelovací prostředí vytvořené v 90. letech minulého století[6]. Dnes je v OpenSim už jen část jeho základní funkcionality, například možnost editace svalu. OpenSim podporuje MSM založené na line of action, které byly zmíněné v kapitole 3. Systém se poté postará o mechanickou deformaci modelu na základě nastavení nejrůznějších parametrů. Jak se OpenSim vyvíjel, vznikalo nejdříve jádro (core). Poté bylo vytvořeno uživatelské rozhraní (GUI) pro vizualizace. Pro oba projekty, jež jsou podmnožinou dnešní OpenSim aplikace, je možné vytvářet uživatelem napsané přídavné moduly (plug-ins). Všechny projekty jsou v této kapitole zmíněny více dopodrobna. Obrázek 4.1 popisuje strukturu systému OpenSim:



Obrázek 4.1: Struktura systému OpenSim[2]

4.1 SimTK

SimTK je nástroj pro generování dynamických simulací v OpenSim systému. Dosahuje toho pomocí diferenciálních rovnic, které popisují dynamickou kontrakci svalů, muskuloskeletální geometrii a dynamiku jednotlivých segmentů těla[6]. Celý proces vytváření dynamických simulací je přehledně popsán diagramem:



Obrázek 4.2: Diagram vytváření dynamické simulace[6]

Proces vytváření dynamické simulace lze rozdělit do čtyř kroků. Můžeme ho popsat například na simulaci chůze. Vstupem je dynamický model muskuloskeletálního systému společně s naměřenými reakčními silami, momenty a kinematikou. V prvním kroku je velikost SIMM modelu upravena tak, aby z antropometrického hlediska odpovídal realitě. Ve druhém kroku je nutné vyřešit inverzní kinematiku. Díky ní je možné určit hodnoty souřadnic pro klouby tak, aby co nejlépe odpovídaly naměřeným datům z simulace pohybu. Ve třetím kroku aplikujeme redukční reziduální algoritmus (RRA) na data vypočtené pomocí inverzní kinematiky. Kvůli aplikování inverzní kinematiky, která je vypočtena pro obecný model, může dojít k chybám. RRA algoritmus se snaží tyto chyby odstranit.V posledním kroku se použije vypočtená svalová kontrola (CMC). CMC je přístup pro generování dopředných dynamických simulací, které nabízí podstatné výkonnostní výhody oproti konvenčním technikám dynamické optimalizace[16].

4.2 Core

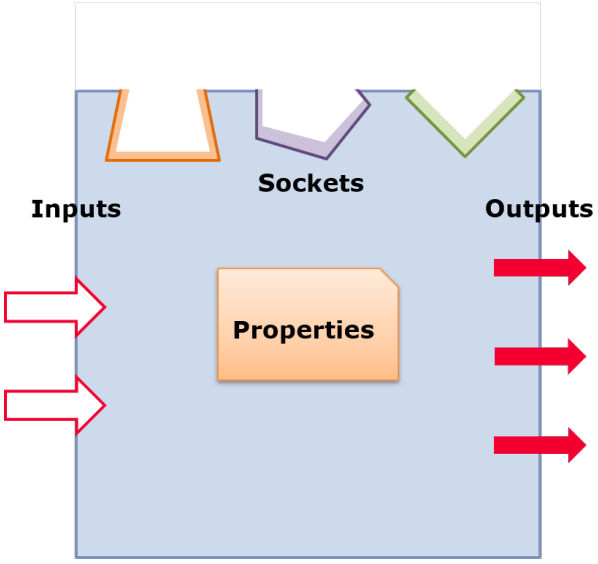
Jádro systému OpenSim funguje jako samostatný projekt. Software je napsán v ISO C++ 11 a běží na základních operačních systémech. Jádro zde zastupuje jednotku určenou pouze pro simulaci a výpočetní sílu, o vizualizaci se stará GUI, tudíž může být voláno přes příkazovou řádku, bez využití

GUI. Jelikož se jedná o komunitou vybudovaný systém, využívá mnoho volně dostupných knihoven. V dřívějších dobách využíval OpenSim dynamický engine SDFast, který je nyní nahrazován vývojáři vytvořeným Simbody[6]. Generickým prvkem všech tříd, ať popisujeme OpenSim komponenty, datové kontejnery či nástroje, je třída `Object`.

4.2.1 Komponenta

`Component` je základní "jednotkou"pro modelování v OpenSim[2]. Pro výpočet modelu se využívá systém rovnic, který třída `Component` využívá na úrovni abstrakce. Zastává roli základní třídy, kterou lze využít jako odrazový můstek k vybudování hierarchicky strukturovaného modelu. Taková komponenta je vyobrazena na Obrázku 4.3. Komponentu lze popsat atributy:

- **Property** - fixní parametr (nemění se v průběhu simulace). Funguje jako uložiště dat (ve formě XML), objektů nebo dokonce komponent.
- **Socket** - slouží k propojení komponent mezi sebou. Typ spoje musí být předem definován. Dále poskytuje status spojení.
- **Výstupy** - hodnoty generované komponentou (například svalem). Klasifikují se jako jednohodnotové (jeden kanál pro různé typy dat) a vícehodnotové (více kanálů pro více dat stejného typu).
- **Vstupy** - přijímají data od výstupu, ověřují jejich správnost a dovolují komponentám být závislé na výstupu, který dostanou, nikoliv na specifické komponentě. Rozdělují se taktéž na jednohodnotové nebo na vícehodnotové.



Obrázek 4.3: Struktura generické komponenty[2]

4.2.2 Kategorizace komponent

`Operator`

Matematická komponenta, jejímž cílem je manipulace s výstupy ostatních komponent, například získání maximální nebo minimální hodnoty, sečtením vstupu nebo použití jiných aritmetických operací. `Operator` pro svoji funkčnost potřebuje pouze množinu jednoho či více vstupů. Vstupy spočte a výsledek operace pošle dále na výstup. `Operator` není závislý na ostatních komponentách a tudíž nepotřebuje model.

`Source`

Komponenta slouží jako zdroj pro výstupy modelu. Sama žádné vstupy neobsahuje.

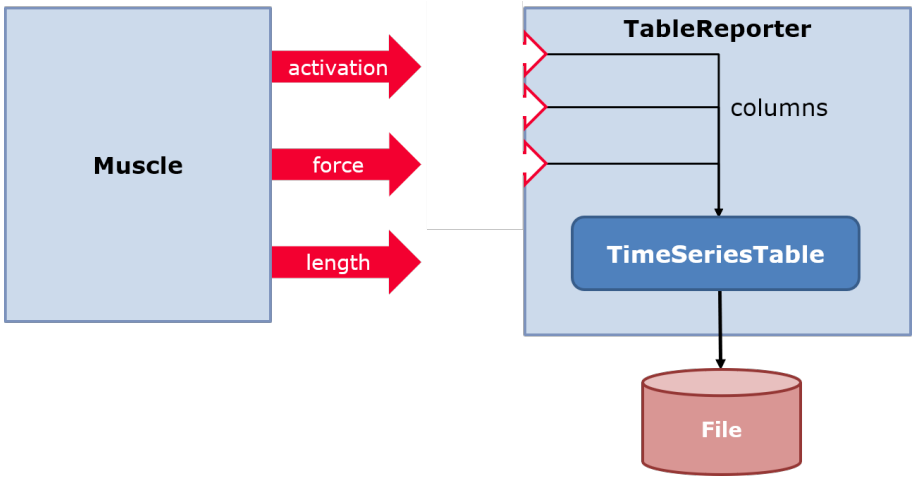
`ModelComponent`

Slouží jako generický typ všech komponent, ze kterých se může sestavit model. Komponenta je probrána do větší hloubky v navazující kapitole.

`Reporter`

Úkolem komponenty je sběr výsledků výpočtů modelu a jeho strukturu můžeme vidět na Obrázku 4.4. Vstupem komponenty je výstup libovolné komponenty modelu. Takovým vstupem může být soubor nebo výstup příkazové

řádky. `Reporter` může využívat pouze jednohodnotové výstupy, avšak model může těchto komponent vlastnit více.

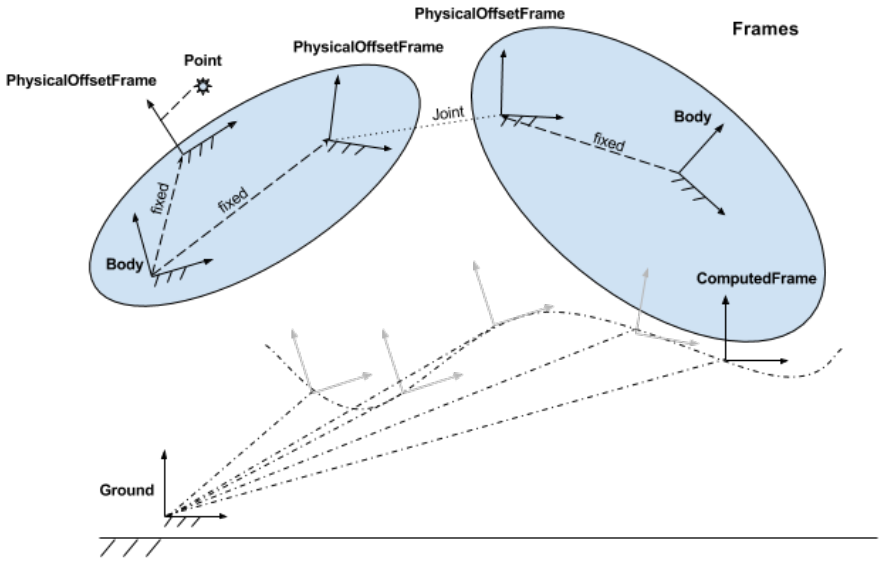


Obrázek 4.4: Struktura komponenty `Reporter`[2]

4.2.3 Prvky `ModelComponent`

Frames

`Frame` v systému OpenSim reprezentuje referenčí snímek, kterým lze popsat prostorovou orientaci a umístění ostatních `Frames`. Díky rámcům lze nacházet fyzické struktury, jako svalové úpony nebo klouby. Rozlišujeme rámce na `PhysicalFrame`, `Ground`, `Body` a `PhysicalOffsetFrame`. Pro lepší představu jednotlivých `Frames` je na Obrázku 4.5 zobrazen jejich diagram. Díky `Ground` lze efektivně vyjádřit pohyb všech snímků a bodů. `PhysicalFrame` podporuje různé typy fyzických spojení jako například klouby. `Body` se nachází ve `PhysicalFrame`, specifikované svou hmotností a těžištěm. Poslední `PhysicalOffsetFrame` určuje konstantní vzdálenost od jiného `PhysicalFrame`. Toho může být využito například k určení umístění a orientace kloubu na těle.



Obrázek 4.5: Diagram všech `Frames` v OpenSim[2]

Points

Bod v systému OpenSim reprezentuje libovolnou lokaci v prostoru. Využití bodů můžeme najít v jejich schopnosti definovat a vypočítat lokaci fyzické struktury, stejně jako u rámců. Bodem můžeme popsat polohu, rychlost i zrychlení. Podmnožinou `Point` je `Station`, bod, který je fixní na `PhysicalFrame`. Podmnožinou `Station` je `Marker`, to je station, který reprezentuje značku pro zachycení pohybu z experimentu.
Mezi prvky komponenty patří:

- `Joint` - kloub, spojuje dva fyzické rámce. Určuje jejich relativní pohyb v závislosti na vnitřních koordinacích.
- `Constraint` - omezení, bývá děděno ostatními třídami. Například třídy `PointConstraint` nebo `PointOnLineConstraint`.
- `Force` - abstraktní třída, reprezentující sílu působící na model či na generické souřadnice během simulace. Každá dědící třída pak symbolizuje různé typy síly.

- `Actuator` - akční člen, požaduje vnější vstup pro generování síly.
- `Controller` - abtraktní třída, definuje rozhraní pro OpenSim ovladač. Vypočítává množinu hodnot pro podružný akční člen.

4.2.4 Datové třídy

`DataTable`

Uložiště dat, popřípadě i metadat (data o datech), nezávislé na formátu souboru. Dokáže uschovat data generovaná komponentami pomocí tabulky, ke které se přistupuje indexem a názvem sloupce. Obsahuje "nezávislý" sloupec, jehož typem může být skalár, a množinu závislých sloupců, jejímž typem lze být například `double`.

`DataAdapter`

Abstraktní třída definující rozhraní pro čtení a zápis dat z `DataTable`. Třída popisuje jak by se mělo číst (zapisovat) na různé zdroje dat. Zdroje dat jsou založeny na příponě souboru. `DataAdapter` rozlišuje čtyři základní adaptéry:

- `TRCFileAdapter`
- `STOFileAdapter`
- `CSVFileAdapter`
- `C3DFileAdapter`

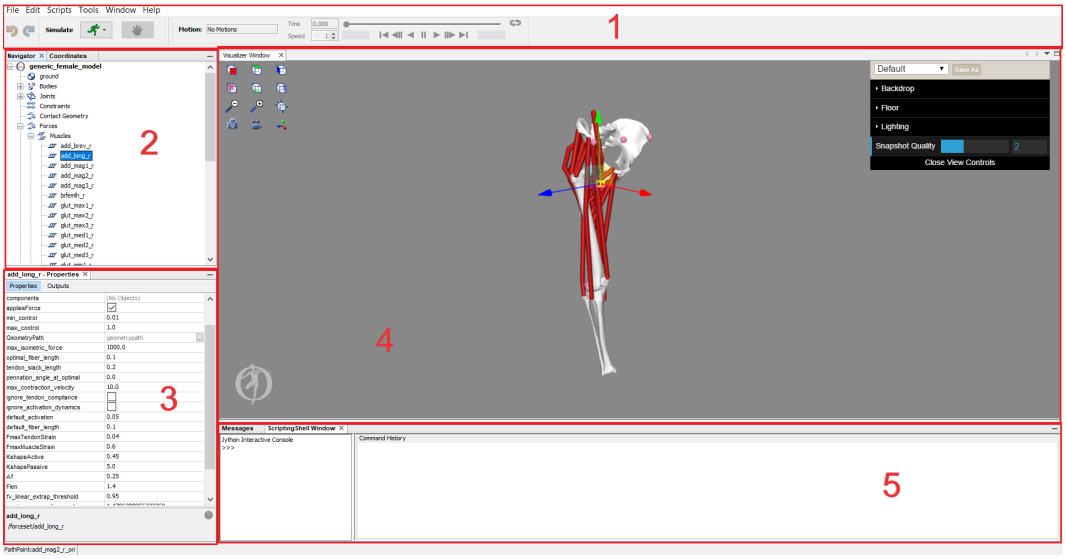
4.2.5 `Solvers`

Opět se jedná o abstrakci, tentokrát za účelem zachycení algoritmu pro výpočet neznámých modelů. Příkladem můžeme uvést `InverseKinematicsSolver`, který řeší problém inverzní kinematiky. `InverseKinematicsSolver` slouží k určení souřadnic, stavových proměnných, které splňují externí měření.

4.3 GUI

Grafické rozhraní vznikalo v pozdější době, postupně jak se systém OpenSim vyvíjel, a tudíž je představeno jako jeden prvek v tomto modulárním systému. Rozhraní je napsáno v jazyce Java, využívající balíky jako je Swing nebo AWT(Abstract Window Toolkit). Tyto balíky rozhraní využívá pro

tvorbu grafiky. Další nedílnou součástí je rozšíření SWIG(Simplified Wrapper and Interface Generator). SWIG umožňuje Java třídám přistupovat ke kódu napsaném v C++ a modifikovat ho. Důvodem používání rozšíření je již zmíněné jádro.



Obrázek 4.6: Ukázka grafického rozhraní OpenSim

Grafické rozhraní je rozděleno do několika částí, každá z nich zastává jinou funkcionalitu. Nyní si popíšeme jednotlivé části demonstrované na Obrázku 4.6. První část obsahuje lištu a simulační prvky. V liště můžeme nalézt klasické načítání a ukládání modelů a simulací, editace, skripty a pro tuto práci důležité menu - tools, díky němuž lze načíst plug-iny do systému. Simulační prvky nám dovolují spouštět různé simulace, zastavovat, přetáčet či nastavovat. V druhé části se nachází hierarchie. V té lze najít všechny objekty ve scéně. Můžeme zde například nalézt kosti, svaly či klouby. Třetí část obsahuje atributy a vlastnosti vybraného prvku. Jedná se většinou o biomechanické vlastnosti, například u svalu délka šlachy. Čtvrtá část obsahuje okno scény. Zde si může uživatel prohlížet model ve 3D. Může zde zkoumat, jak se jednotlivé části modelu chovají při simulaci a podobně. Poslední část je interaktivní konzole pro příkazy jazyka Python.Uživatel zde může zadávat příkazy, kterými může ovlivnit simulaci, ale celkově i celé grafické prostředí (příkladem takového příkazu může být stáhnutí výchozích modelů dodávaných přímo od systému).

4.3.1 Struktura projektu

K grafickému rozhraní nebyla ještě bohužel vytvořena API jako k jádru[2]. Celý GUI projekt je napsán v Javě a rozřazen celkem do 17 balíčků (valná většina z nich jsou knihovny). Na základě provedené vlastní analýzy těchto balíčků i celého projektu bylo zjištěno, že každý balíček zajišťuje jinou funkcionalitu, přičemž pro nás jsou nejdůležitější, přičemž pro nás jsou nejdůležitější *org.opensim.modeling*, *utils* a *view*. Prvně si rozeberme nejrozsáhlejší balík *org.opensim.modeling*. Balík obsahuje třídy a komponenty zmíněné v Kapitole 4.2. Třídy jsou pomocí rozšíření SWIG rozšířeny o C++ kód tak, že s nimi systém ovšem nadále pracuje jako s třídami jazyka Java. Druhý balík *utils*, jak už název vypovídá, obsahuje různé utility pro správnou funkčnost programu. Z nich lze vybrat dvě, které svojí funkčností vybočují z řady a operují nad větším celkem. `SwingWorker` je abstraktní třída, která provádí práci související s GUI v novém dedikovaném vláknu. Druhou je `TheApp` zastupující běžně používané utility a pomocné funkce. Má za úkol například prvotní načítání dll knihoven, nastavení základních parametrů, instalaci zdrojů a v neposlední řadě ukončení aplikace. Poslední balík *view* se dělí ještě na jednotlivé složky, tříděné dle funkcionality. Zde je poté kód, který se stará o jednotlivé grafické prvky, vytváří je a modifikuje.

4.4 Plug-ins

Jelikož je OpenSim volně dostupnou platformou, podporuje uživatelské projekty ve formě přídavných modulů. Plug-iny představují uživatelské dynamické knihovny, které se skládají z množiny uživatelem napsaných tříd. Kód systému je navržen tak, aby mohl pracovat s předem neznámými třídami. Vnitřní načítání je realizováno pomocí statické instance pomocné třídy na zásobníku. Operační systém po načtení knihovny do paměti pak automaticky zavolá konstruktor pomocné třídy. Kód v konstruktoru pak volá registrace tříd pro OpenSim. Třídy musí ovšem splňovat předepsané konvence. Nejdřív musí uživatel třídu registrovat. Toho dosáhne provedením následujícího příkazu hned po načtení plug-inů.

- `Object::RegisterType(My_Plugin_Class1();)`
- `Object::RegisterType(My_Plugin_Class2();)`

Třídy musí splňovat čtyři požadavky[4]:

1. Třída musí mít validní jméno podle standardu pro C++

2. Třída musí dědit od základní třídy OpenSim `OpenSim::Object`

3. Třída musí mít základní konstruktor bez argumentů

4. Pokud bude nutné třídu serializovat či upravovat v grafickém rozhraní, proměnné, kterých se to týká musí být označeny.

Plug-iny lze načíst jak z grafického rozhraní, tak i z příkazové řádky pro API, avšak je nutno nejdříve plug-iny otestovat. Příkazová verze nástrojů přebírá jako argument *"-L libraryName"*. Pro použití plug-inů přes grafické rozhraní je nutno postupovat následovně[4]:

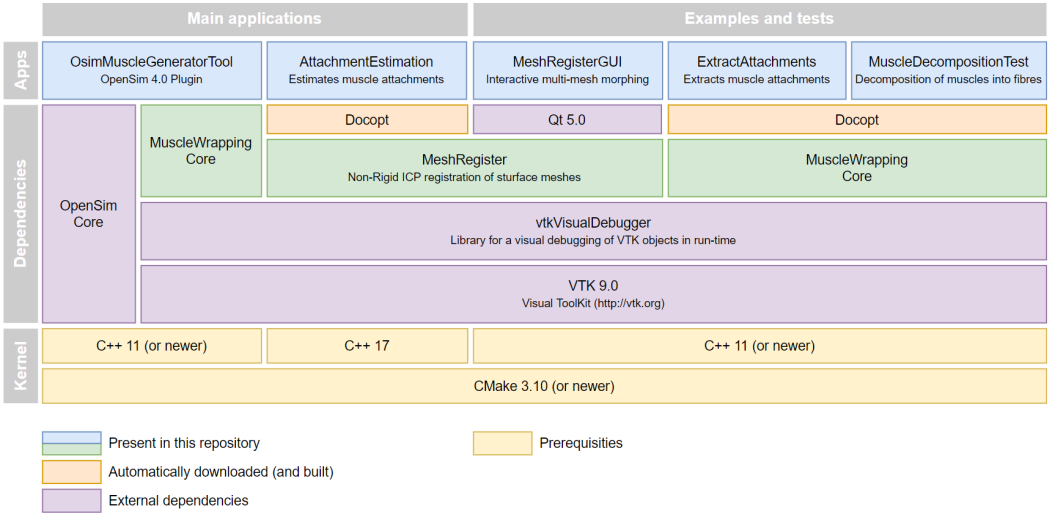
1. Před přidáváním plug-inu zavřeme OpenSim, abychom předešli chybám.

2. Přidáme plug-in do složky *plugins*, které se nachází v adresáři, kam se nainstaloval systém OpenSim. Takovou cestou může být například *C:\OpenSim<version number>\plugins*.

3. Spustíme OpenSim. V záhlaví grafického rozhraní vybereme *Tools*. Po kliknutí na *User Plugins*, bychom měli vidět náš plug-in. Poté stačí na plug-in kliknout, tím se načte. Je zde i nabídka automatického načítání přídavného modulu při spuštění systému OpenSim v podobě zaškrtávacího políčka. Vývojáři ale doporučují tuto nabídku využít až po řádném otestování modulu.

5 Muscle Wrapping 2.0

Jak bylo řečeno v úvodu, Muscle Wrapping 2.0 je přídavný modul pro systém OpenSim, jehož cílem je automaticky vytvářet personifikované modely. Projekt nyní obsahuje dvě hlavní aplikace. První je *OsimMuscleGeneratorTool* plug-in pro systém OpenSim. *OsimMuscleGeneratorTool* plug-in má na starost hned několik úkolů. Stará se o dekompozici svalů a jejich geometrickou reprezentaci. Dále obstarává inverzní kinematiku a deformaci svalů. *OsimMuscleGeneratorTool* umožňuje provádět simulaci prostřednictvím dvou algoritmů, které byly zmíněny v podkapitole 3.2. Tato kapitola se zaměřuje pouze na algoritmus *Luca2018*, neboť *Cervenka2019* nemá parametry. Druhou aplikací je *AttachmentEstimation*, nástroj pro odhad úponů svalů na kosti. Projekt dále obsahuje testy, ukázky a data. Softwarovou architekturu projektu můžeme vidět na Obrázku 5.1:

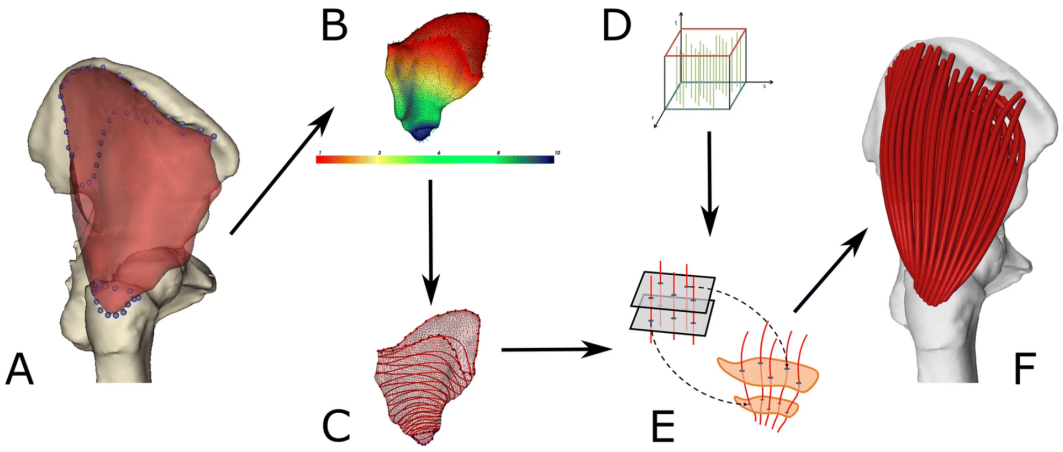


Obrázek 5.1: Struktura projektu [odkaz]

Žlutě vyznačené jsou nástroje důležité pro překlad a spuštění projektu. Fialově jsou pak externí knihovny, které *OsimMuscleGenerationTool* a *AttachmentEstimation* využívají. Oranžově jsou automaticky stahované knihovny. Celý projekt je automaticky překládán pomocí programu Cmake, který v průběhu konfigurace překladu stáhne zmíněné knihovny. Nakonec, modré a zelené jsou součástí repozitáře MuscleWrapping.

5.1 Dekompozice svalu

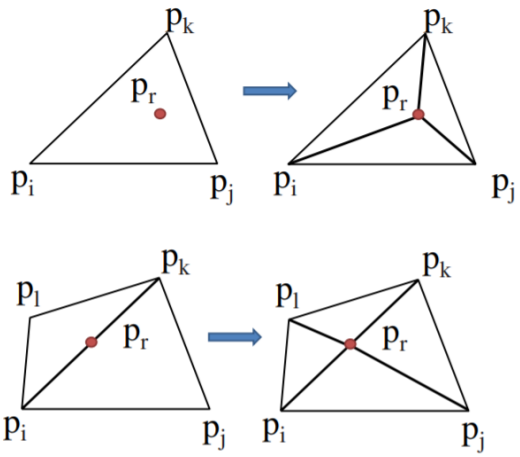
Algoritmus automatického generování zvládne zpracovat sval reprezentovaný pomocí trojúhelníkové sítě. Potřebuje k tomu znát vnitřní strukturu svalu (pennate, parallel, curved, fanned) a data o úponových oblastech. Úponové oblasti jsou na Obrázku 5.2 znázorněny modrými body. Body vytváří uzavřenou plochu, kterou určuje expert. Tato plocha je poté ve výsledném modelu vyplněna lomenými čárami. Problém nastává při definici počtu lines of action a jejich rozštěpení na N různých úseků. Tyto dvě hodnoty (počet lomených čar a jejich rozdělení na N úseků) musí být nadefinovány před začátkem procesu. Formulace hodnot se navíc liší sval od svalu. To platí i pro jeden speciální parametr. Parametr zastupuje koeficient, který mění středový bod svalu. Jak se sval natahuje, mění se i jeho středový bod. Tímto parametrem předcházíme protnutí či kolizi s kostí. Pro většinu svalů stačí parametr nastavit roven nule, v některých případech (například pro svaly v oblasti kyčlí) musí být koeficient různý. Současná implementace má tento parametr součástí konfigurace simulace, kde nelze specifikovat, pro jaké svaly se má parametr použít. Nelze tedy zpracovávat najednou svaly, které mají různé parametry. Vzhledem k tomu, že *Luca2018* neřeší vzájemné ovlivňování svalů, nepředstavuje toto z hlediska výsledku rozdíl, pouze to není uživatelsky přívětivé.



Obrázek 5.2: Proces automatické generace svalů[9]. A - vstupní data, B - skalární pole povrchu svalu, C - definice izočar, D - šablona vláken, E - mapování pomocí šablony a izočar, F - výstupní model

Dále se musí vyřešit problém úponových oblastí ležících na povrchu kosti. Ty potřebujeme dostat na povrch svalu, ale tak, aby vzdálenost mezi původní a

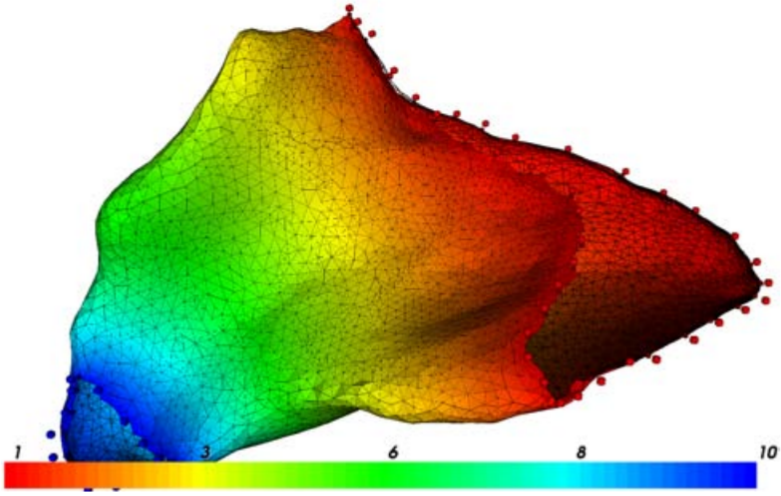
novou pozicí byla minimální. Toto se provede automaticky v prvním kroku algoritmu. Nyní když máme definovanou úponovou oblast na povrchu svalu, potřebujeme ji vyříznout. Mějme povrch svalu reprezentovaný trojúhelníkovou sítí a trojúhelník, jehož vrcholy p_i, p_j, p_k mohou být body úponu. Pokud je nový bod úponu, řekněme p_r, v oblasti trojúhelníku, spojí se se všemi vrcholy novou hranou (starý trojúhelník se rozdělí na tři nové). Pokud ale p_r leží na hraně mezi vrcholy, oba trojúhelníky sdílející tuto hranu musí být rozděleny – viz Obrázek 5.3.



Obrázek 5.3: Úprava bodů v oblasti úponu[8]

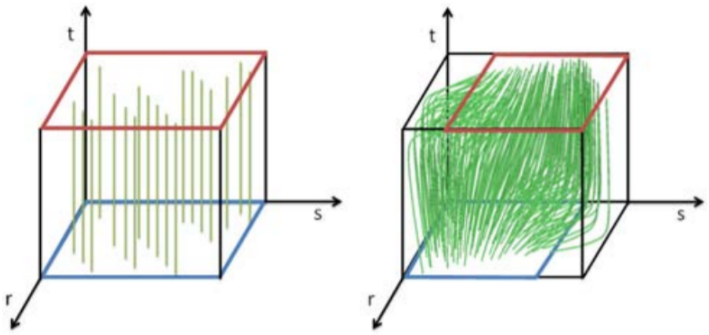
Po skončení procesu vzniká nová ohraničená oblast, která rozděluje trojúhelníkovou síť na dvě poloviny. Tu získáme pomocí nalezení nejkratší cesty na povrchu sítě mezi sousedními body úponu. Menší z nich je považována za úponovou oblast a její trojúhelníky jsou zahozeny.

Jakmile jsou oříznuty obě úponové oblasti, musíme vytvořit harmonické skalární pole hodnot (Obrázek 5.2B). Výsledku dosáhneme tím, že pro každý vrchol spočteme hodnotu u_i. Hodnoty lineární funkce musí na hranici první úponové oblasti docílit minima k_{min} a v druhé maxima k_{max}. Maximum je obecně značeno červenou barvou, minimum naopak modrou. Ostatní barvy symbolizují lineární přechod.
Interval, který jsme dostali můžeme nyní rovnoměrně rozdělit na y - *1* částí, přičemž proměnná y je uživatelem volitelná. Po rozdělení vznikne y nových hodnot s_1 až s_y, které odpovídají izočárám. Díky tomu máme nyní množinu obrysových čar pro uživatelem zvolený sval. Tato část procesu je znázorněna na Obrázku 5.2C.



Obrázek 5.4: Harmonické skalární pole pro střední sval hýžďový[8]

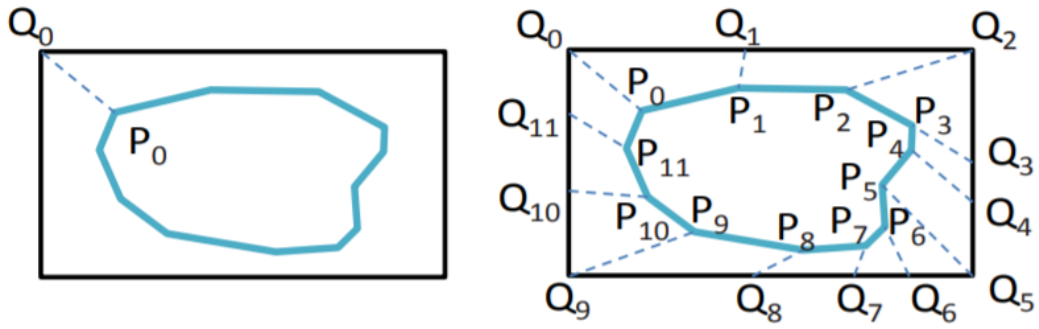
Následně se vybere jedna z předdefinovaných šablon. Prvek kategorizace je jejich vnitřní struktura podle které se také vybírají. Pro vybranou šablonu je provedena afinní transformace. Výsledek transformace je co největší shoda úponových oblastí šablony a vybraného svalu. Nakonec je šablona rozštěpena na stejný počet obrysů jako sval.



Obrázek 5.5: Ukázka šablony svalu - levá(parallel), pravá(pennate)

Nyní máme obrysy šablony a svalu. Zde mohou nastat dva signifikantní problémy. Prvním z nich je rozdílný počet vrcholů. Při rozdílném počtu není možné provést přiřazení vrcholů jedna ku jedné. Druhým z nich je rotace obrysu. Obrys může být otočen a tudíž nemáme jistotu, zda první vrchol obrysu šablony skutečně odpovídá prvnímu vrcholu obrysu svalu. Tyto problémy lze řešit různými způsoby. Jeden z nich je popsán ve článku [8]. Oproti němu se přiřazení v současné době provádí následujícím způsobem.

Nejprve se oba obrysy uniformě navzorkují. Používá se zde parametrizace délkou oblouku, tedy pro $t = 0$ nebo $t = 1$ získáme P_0 a Q_0. Pro $t = 0.5$ získáme bod Q_i, který leží přesně uprostřed obrysu O_2 a bod P_i ležící přesně uprostřed obrysu O_1. Počet vzorků může být do jisté míry libovolně velký, v praxi se volí stejný počet jako je počet vrcholů obrysu svalu. Po skončení procesu vzorkování dostáváme nyní dvě pole bodů a to $P = [P_0, P_1, ..., P_m]$ a $P = [Q_0, Q_1, ..., Q_m]$.



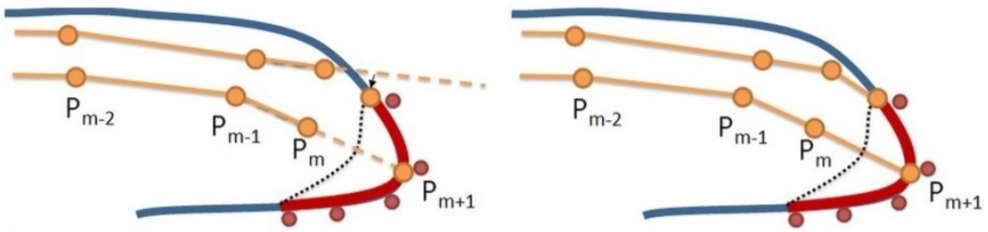
Obrázek 5.6: Princip přiřazení vrcholů

Dále předpokládejme, že jsou oba obrysy stejně orientované, například ve směru hodinových ručiček, stejně jako na Obrázku 5.6. Provádíme minimalizaci funkce součtu čtverců vzdáleností mezi P a Q:

$$argmin(k) \sum |P_i + k, Q_i|^2,$$

kde k je celé číslo nabývající hodnot 0 až m - 1 a určuje natočení P vůči Q. Výsledkem minimalizace je pak zjištění, že Q_0 z pole Q odpovídá vrcholu P_k z P. Posledním krokem je pak rotace bodů pole P tak, aby se P_k dostalo na index 0. Tím bude vyřešen druhý problém a tedy bude platit že Q_i odpovídá P_i. Důvodem změny algoritmu oproti algoritmu popsaném ve článku [8] je nejspíš problém zakroucení vláken, ke kterému mohlo docházet při nevhodně nastaveném parametru y.

Poslední částí je prodloužení vygenerovaných vláken. Lomené čáry skládající se z vrcholů $P_0...P_m$ obvykle nejsou zakončeny přímo na povrchu úponové oblasti. Vlákna jsou tím prodloužena v podobě post-processingu. Dosáhneme toho tím, že přidáme nový bod, jež leží v úponové oblasti a je nejblíže pomyslné přímce mezi P_{m-1} *a* P_m. Vznikne tak buď průsečík přímky a úponové oblasti nebo bod s minimální vzdáleností od přímky.



Obrázek 5.7: Metoda prodloužení vláken

6 Návrh řešení

Poté, co jsme si popsali systém OpenSim a jeho plug-in pro automatické generování svalových vláken, lze přistoupit k návrhu řešení grafického rozhraní.

6.1 Obecný návrh

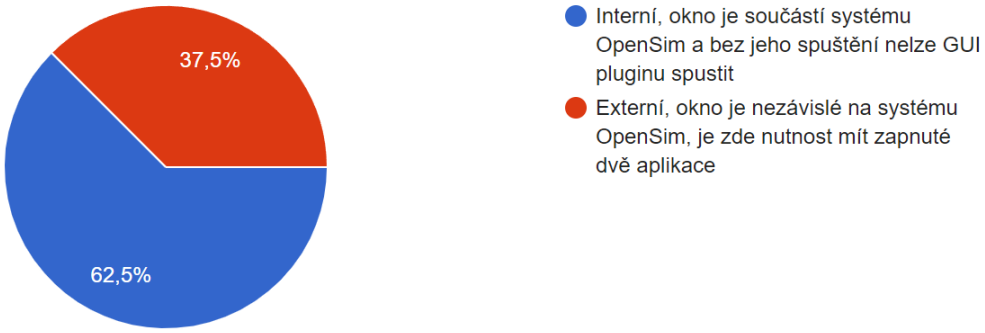
Výsledkem řešení by mělo být grafické rozhraní, které nám dovoluje měnit a nastavovat různá data či vlastnosti pro automatické generování svalových vláken. Nabízí se zde dva koncepty provedení, interní a externí.

Externí provedení by zahrnovalo vytvoření samostatné aplikace. Grafické rozhraní by nebylo plně závislé na systému OpenSim. Jednalo by se tedy o aplikaci s GUI, ve kterém by se nacházela konfigurace pro plug-in. Po konfiguraci by se spustil externí `osim-cmd.exe -L plug-in.dll`. Stejný princip využil například *Modenese L.* a *Ceseracciu E.* pro svůj *Muscle Optimazer Tool* plug-in ve verzi OpenSim 3.3. Takový návrh má své výhody i nevýhody. Výhodou je modularita řešení, implementace je zcela na rozmyšlení programátora. Programátor nemusí dodržovat konvence a předepsaná pravidla navržena vývojáři systému a řešení je zcela v jeho vlastní kompetenci. Nevýhodou je pak nutnost spuštění více aplikací najednou. Musel by být spuštěn OpenSim i grafické rozhraní.

Interní provedení je pak přesným opakem externího. Řešením by bylo rozšířit systém OpenSim o funkcionalitu v podobě nového rozhraní. Konkrétně, byly by rozšířeny některé vybrané třídy (například třída GUI pro menu tools) nebo by byla implementovány rozhraní, které se v systému starají o vykreslovaní grafických panelů. K těm by pak byly přidány třídy nové. Ty by dědily některé vlastnosti tříd systému, ale přidávaly by i nové, důležité pro tvorbu konfigurace plug-inu. Výhodou je zde uživatelská přívětivost a celistvost. Uživatel má vše zabudované v aplikaci, tudíž může vlákna generovat a nadále i provádět různé simulace či změny bez nutnosti přepínání mezi aplikacemi.Takový přístup využívají některé funkce již zabudované v systému OpenSim, jako například funkce Scale model, která mění velikost modelu a nastavuje další vlastnosti.

Aby nebyl výběr subjektivní, založen pouze na názorech tvůrce, byl vy-

tvořen formulář, který se ptá tázaných na několik otázek ohledně provedení grafického rozhraní. Na otázky odpovědělo celkem 24 lidí, z čehož převážnou část tvoří spolužáci nebo lidé znalí v oboru informačních technologií. Je zde avšak i procento lidí, kteří naopak přicházejí do styku s technologiemi minimálně a tudíž by výsledný graf neměl být odrazem určité skupiny lidí. Na otázku, zda by uživatelé preferovali provedení interní či externí na základě popisu a vlastností zmíněných výše, odpověděli tázaní následovně:



Obrázek 6.1: Graf reprezentující podíl odpovědí na otázku provedení

Téměř dvě třetiny tázaných uživatelů by preferovalo provedení interní, kde je vše pohromadě a zabudované v systému OpenSim. Zbylá jedna třetina by naopak preferovala provedení externí, kde grafické rozhraní reprezentuje samostatnou aplikaci.

6.2 Grafická struktura rozhraní

Grafická struktura rozhraní by měla být intuitivní, uživatelsky přívětivá a dobře organizovaná. Zmíněné faktory bývají často ale subjektivní a tudíž grafická struktura rozhraní může nabývat mnoha podob. Co se struktury týče, nechal jsem se inspirovat konfiguračním souborem, který je plug-inem *OsimMuscleGeneratorTool* zmíněným v kapitole 5.2 načítán a zpracováván. Konfigurační soubor je ve formátu XML (Extensible Markup Language). Tento formát je velice rozšířený v dnešní době díky své rozšiřitelnosti, mezinárodní podpoře, snadné konverzi a čitelnosti [7]. Abychom věděli, jak má vypadat rozhraní, musíme nejdříve znát data, s nimiž budeme manipulovat. Dle dat, která musí být schopen plug-in načíst, měnit a nastavovat, lze strukturu rozdělit celkem do tří částí, které se do sebe postupně vnořují. Příklad struktury takového konfiguračního souboru můžeme vidět na Obrázku 6.2.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<OpenSimDocument Version="20302">
    <MuscleGeneratorTool name="GenericFemale_WCB_MGT">
        <model_file>GenericFemale_WCB.osim</model_file>
        <motion_file>hip_flexion_kinematics_POS.mot</motion_file>
        <output_model_file>GenericFemale_WCB_Flex.osim</output_model_file>
        <MuscleGeneratorSet>
        <objects>
            <MuscleGenerator name="GluteusMaximus">
            <MuscleGeometry>
                <body>pelvis</body>
                <Mesh name="GluteusMaximus">
                    <scale_factors>0.001 0.001 0.001</scale_factors>
                    <mesh_file>S032_R_Gluteus_Maximus.obj</mesh_file>
                </Mesh>
                <attachment_areas>
                    <AttachmentArea name="origin Sacrum">
                        <type>origin</type>
                        <body>pelvis</body>
                        <point_file>S032_R_Gluteus_Maximus_Ori_SAC.vtk</point_file>
                       <scale_factors>0.001 0.001 0.001</scale_factors>
                    </AttachmentArea>
                </attachment_areas>
            </MuscleGeometry>
            <fiber_arrangement> uniform </fiber_arrangement>
            <num_of_lines> 100 </num_of_lines>
            </MuscleGenerator>
        </objects>
        </MuscleGeneratorSet>
        <kinematics_fibre_algorithm>
            <Cervenka2019PBDAlgorithm>
            </Cervenka2019PBDAlgorithm>
        </kinematics_fibre_algorithm>
    </MuscleGeneratorTool>
</OpenSimDocument>
```

Obrázek 6.2: Ukázka struktury konfiguračního souboru

Všeobecnou částí je nástroj pro generování svalů. Tento nástroj uchovává základní informace o souborech (konkrétně soubor modelu, pohybu a výstupní soubor). Následně rozhoduje, jaký algoritmus bude použit pro automatické generování svalových vláken a v neposlední řade obsahuje množinu generátorů svalu. Nástroj by měl zastávat funkci generování celého modelu, kdežto generátor svalu generuje svalová vlákna pouze pro konkrétní sval. Každý generátor svalu obsahuje informace o souboru, který vizuálně představuje sval. Zahrnuje také v jakém referenčním systému se sval nachází (v xml souboru lze nalézt referenční systém pod názvem muscle geometry a byl zmíněn v podkapitole 4.2.3). Poté uchovává doplňující informace ke svalovým vláknům svalu. Těmi jsou například jednotky, ať už se bavíme o metrech či milimetrech, dále počet vláken, ze kterých má být sval vytvořen, jejich rozlišení a závislost na sobě. V neposlední řade zde máme i vlastnosti momentálně ignorované nebo nedokončené, o které by mohlo být grafické rozhraní v budoucnu rozšířeno. Stejně jako má nástroj svou množinu generátorů, tak má generátor svou množinu úponových oblastí, které byly lehce zmíněny v ka-

pitole 3.1.1. Úponové oblasti představují poslední vnořený prvek nástroje. Každá úponová oblast může být pouze jedním ze dvou typů popsaným v 2.2.2 v podkapitole *struktura svalu* a to buď počátkem svalu nebo úponem. Následují, stejně jako u generátoru svalu, jednotky v nichž jsou body úponové oblasti a referenční systém ve kterém se oblast nachází. Nejdůležitější částí pro popis úponové oblasti jsou ale body. Body úponové oblasti lze získat dvěma způsoby, výčtem hodnot jednotlivých bodů, jež se skládají ze tří souřadnic, a nebo pomocí souboru. Soubory jsou vetšinou ve formátu *obj* nebo *vtk*, které můžeme získat z různých modelovacích programů, jako je například *Blender*, ale jinak jsou téměř totožné s výčtem hodnot.

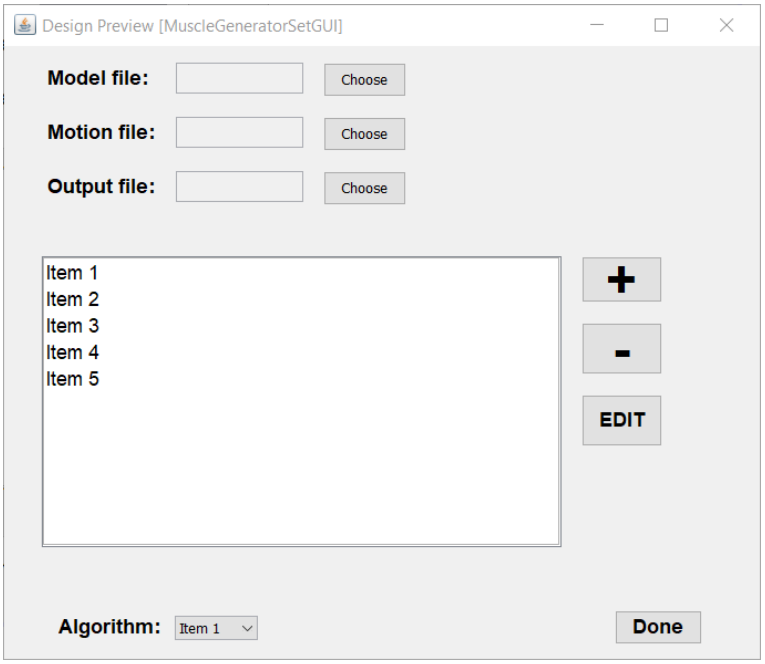
6.3 Návrh grafického rozhraní

Rozhraní může nabývat libovolných velikostí, vzhledů, uspořádání i barev. Líbivost rozhraní je čistě subjektivní prvek, který se liší od jedince k jedinci a nelze v tomto rozměru plně satisfikovat všechny uživatele. Co může návrhář rozhraní ovlivnit je ale intuitivnost a uspořádanost, která s grafickým rozhraním přichází a neliší se tolik v názorech veřejného publika. Ke strukturovanému přístupu vybízí i sama data, která se nám krásně člení a vnořují jak je popsáno v kapitole 6.2. Nabízí se tři možné provedení rozhraní, z čehož dvě podporují hierarchii XML souboru a využívají tohoto faktu. Třetí možné provedení je pak sloučením všeho do jednoho funkčního celku.

6.3.1 První návrh

Rozhraní je rozděleno celkem do tří panelů, každý panel zastává jeden z hlavních prvků XML souboru. Na samém začátku, při spuštění, se nachází všeobecný nástroj pro generování svalů. V nástroji se nastavují základní prvky nutné pro automatické generování svalových vláken a to konkrétně již zmíněný soubor modelu, pohybu a výstupní soubor. Dále množina generátorů svalu a algoritmus, který má být použit pro generování. Pro soubory lze využít libovolný průzkumník souborů, ať už zabudovaný nebo v podobě externí knihovny. Výstupem pak bude textové pole s názvem souboru, který uživatel vybral, pro kontrolu. Algoritmy v době psaní textu existují pouze dva, pevně dané a tudíž je ideální pro tento typ vybrat rozbalovací seznam. Množinu generátorů je možno reprezentovat výběrovým seznamem s posuvníkem, aby grafická velikost seznamu nehrála roli při libovolném počtu generátorů. Manipulaci s výběrovým seznamem by mohla zajistit tři tlačítka, jedno pro přidání nového generátoru do seznamu, druhé pro odebraní zvoleného generátoru ze seznamu a poslední, které by umožňovalo editaci ge-

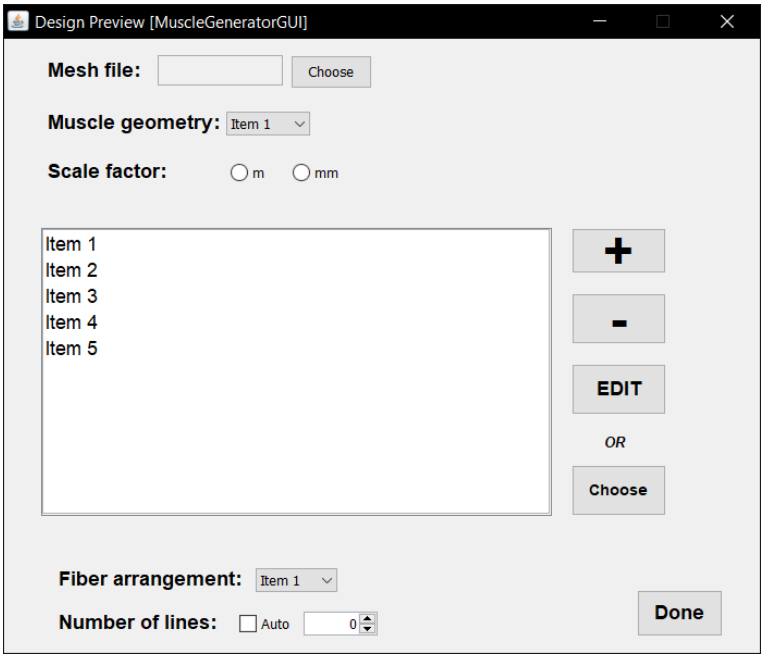
nerátoru a jeho vlastností. Závěrem panelu by bylo tlačítko, kterým uživatel stvrzuje nastavení nástroje. Celkový vzhled je vyobrazen na Obrázku 6.3.



Obrázek 6.3: Panel nástroje pro generování vláken

Již zmíněným tlačítkem editace by se uživatel dostal k vlastnostem a datům vybraného generátoru. Funkcí tlačítka by bylo otevření nového panelu, tentokrát specificky vyhrazenému pro generátor svalu. Členěním by se velmi podobal panelu předchozímu. Pro soubor s vizuální reprezentací svalu by se opět použil průzkumník ve spojení s textovým polem pro kontrolu. Data referenčního systému jsou součástí souboru modelu v prvním panelu. Tento fakt vybízí k přečtení souboru modelu při načítání a následné vložení dat referenčního systému do rozbalovacího seznamu. Všechny dostupné modely jsou v dnešní době buďto v metrech nebo milimetrech. Tomuto popisu odpovídají přepínače v grafickém rozhraní. V jedné skupině by se nacházeli dva, každý zastávající jinou měrnou jednotku. Další částí generátoru jsou vlastnosti vláken. Vlákna jsou uspořádána dvěma možnými způsoby, uspořádání náhodné a uspořádání uniformní. S ohledem na dosavadní informace nám postačí rozbalovací seznam. Druhá vlastnost vláken je jejich počet. Počet vláken je o něco málo komplikovanější než předešlá data. Pokud je počet vláken roven nule, plug-in pro automatické generování svalových vláken vytvoří počet vláken automaticky. Pokud je uspořádání vláken náhodné, je možné přiřadit libovolný počet. Jestliže by ale mělo být uspořádání vláken uniformní, musí být počet vláken roven druhé mocnině čísla kvůli výpočtům.

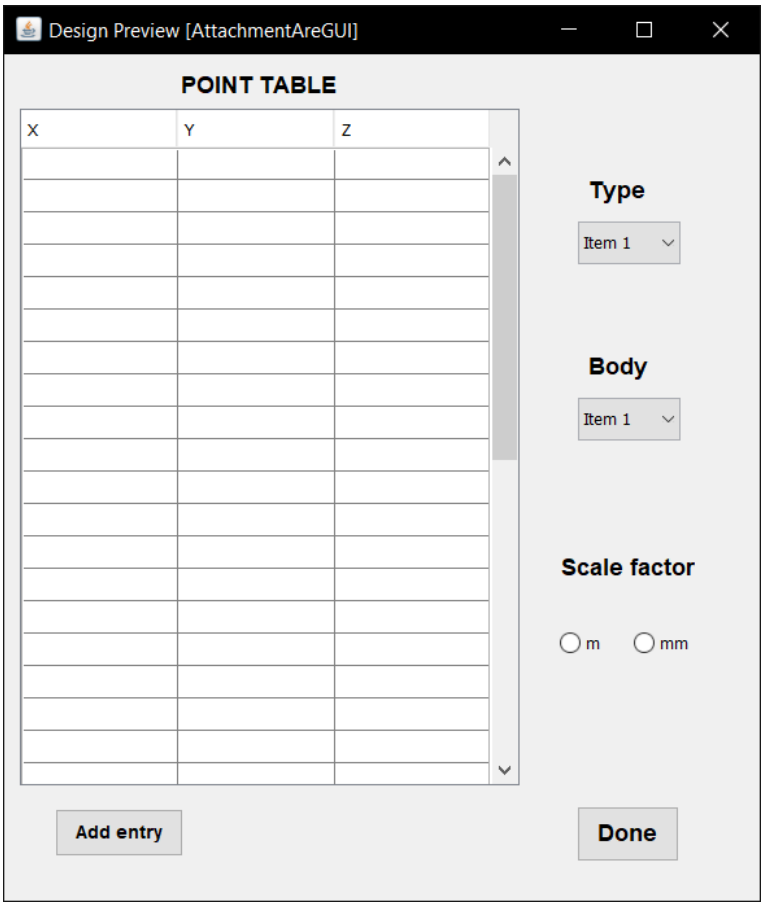
Tudíž, kvůli uživatelské přívětivosti, by mohl být počet vláken rozdělen do dvou částí. První by bylo zaškrtávací pole pro automatický počet a druhá část celočíselný inkrement. Při zaškrtnutí automatického počtu by se celočíselný inkrement zakázal. Inkrement při uniformním uspořádání vláken by byla druhá mocnina následujícího čísla, jinak jedna. Zbývá již pouze množina úponových oblastí svalu. Tu lze řešit totožně jako množinu generátorů v prvním panelu. Přidá se zde akorát tlačítko pro případný výběr souboru s vrcholy pro danou úponovou oblast. Nesmí samozřejmě chybět ani potvrzovací tlačítko pro uložení změn generátoru. Vzhled prvního návrhu generátoru svalu můžeme vidět na Obrázku 6.4.



Obrázek 6.4: Panel generátoru svalu

Pro bližší specifikaci se můžeme tlačítkem *edit* přesunout z panelu generátoru do posledního vnořeného panelu, tentokrát pro úponovou oblast. Úponová oblast je posledním vnořeným prvkem a zároveň nejméně objemným. Z kapitoly 6.2 víme, že existují pouze dva pevně dané typy. Tomuto popisu odpovídá v grafickém rozhraní rozbalovací seznam. Stejné provedení by mohl mít prvek *Body*, který má data totožná s referenčním systémem v generátoru svalu. Taktéž jednotky jednotlivých vrcholů lze řešit přepínači. Největším prvkem panelu je tabulka s body. Uživatel by měl být schopný vytvářet a přidávat nové body, které definují úponovou oblast. Tabulka by byla koncipována takovým způsobem, aby řádek tabulky představoval jeden vrchol oblasti. Sloupce by pak byly tři, každý by reprezentoval jednu sou-

řadnici v kartézské soustavě. U tabulky by se nacházelo tlačítko s možností zavést nový bod do tabulky. Závěrem celého panelu by bylo opět potvrzovací tlačítko změn. Rozhraní úponové oblasti prvního návrhu je vyobrazeno na Obrázku 6.5.

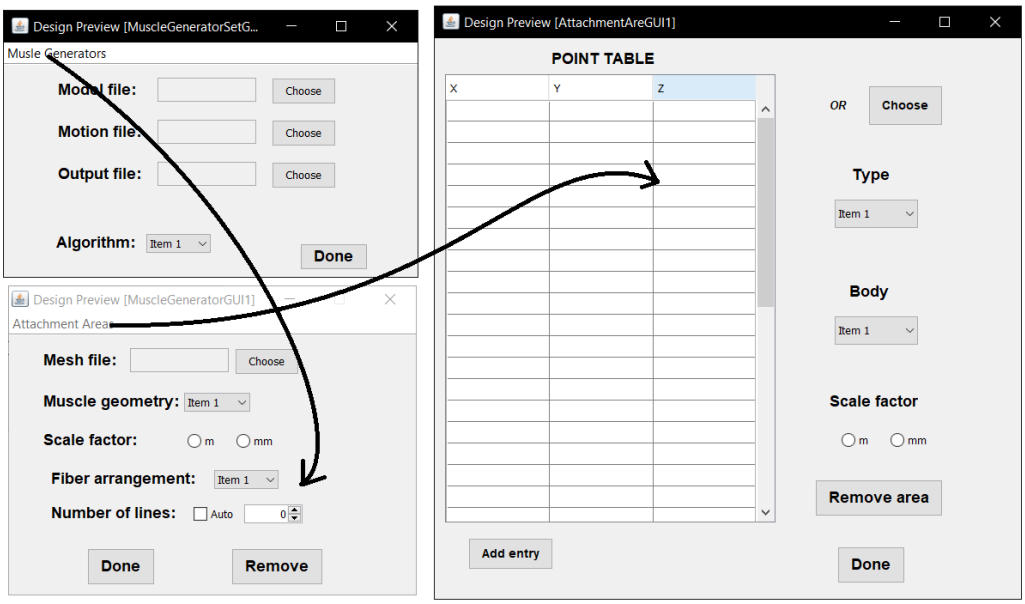


Obrázek 6.5: Panel pro úponovou oblast

6.3.2 Druhý návrh

Další návrh (viz Obrázek 6.6) je modifikací návrhu předešlého a napadl mě hned po náčrtu prvního. Pokud nebude zmíněno jinak, všechny doposud řečené uživatelské prvky rozhraní zůstávají stejné i pro návrh nynější. Myšlenkou současného návrhu je zmenšení panelů, které by mělo na první pohled přinést lepší orientaci a primitivnost. Všechny množiny prvků, ať už se bavíme o generátorech nebo o úponových oblastech jednotlivých generátorů, byly reprezentovány výběrovým seznamem, se kterým se manipulovalo pomocí okolních tlačítek. Logika výběrového seznamu by se dala

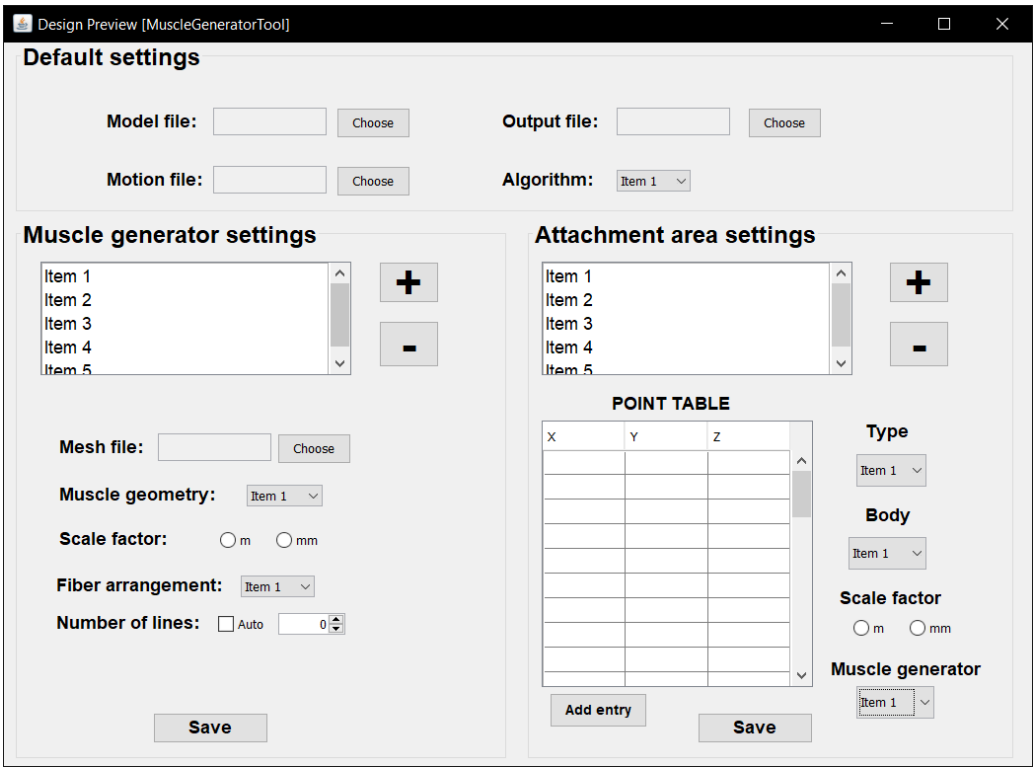
nahradit dalším grafickým prvkem, nabídkou. Rozhraní by se ovládalo následujícím způsobem. U prvních dvou panelů by zmizel výběrový seznam s ovládacími tlačítky. V obou panelech by se v horní části nacházela lišta nabídek, ve které by byl pouze jeden prvek, ten by představoval množinu generátorů/úponových oblastí. Po kliknutí na prvek lišty nabídek by se rozbalila místní nabídka všech dostupných generátorů/oblastí. Na samém konci rozbalovací místní nabídky by se nacházel prvek, kterým lze přidat nový generátor/oblast. Po kliknutí na již existující nebo na tlačítko přidat by se otevřel nový panel, velice podobný těm popsaným v kapitole 6.3.1. Změn by zde bylo pouze pár. Protože zmizel výběrový seznam s ovládacími tlačítky, muselo se ovládání udělat jinak. Editace je na kliknutí na vybraného generátoru/oblasti v rozbalovací místní nabídce. Přidaní je pak posledním prvkem nabídky. Odebrání generátoru/oblasti by se poté muselo nacházet na panelu samotném. Všechny prvky by tedy zůstaly stejné jako v prvním návrhu, akorát by se vedle potvrzujícího tlačítka objevilo tlačítko nové, které by mazalo právě vybraný generátor/oblast. Pro úponovou oblast by se navíc stejným způsobem přidalo tlačítko pro vybrání souboru s vrcholy, které bylo v prvním případě součástí ovládacích tlačítek u výběrového seznamu úponových oblastí v panelu generátoru.



Obrázek 6.6: Druhý návrh možného zpracování GUI

6.3.3 Třetí návrh

Poslední návrh (viz Obrázek 6.7) je taktéž z velké části odrazem návrhu prvního, jen postrádá členitost v podobě panelů. Hlavní myšlenkou třetího návrhu je spojit všechny uživatelské prvky do jednoho velkého, celistvého panelu. I přesto, že budou všechny grafické prvky v jednom panelu, neměly by ztratit strukturu. Důvodem toho bude rozdělení panelu na tři menší části. Části budou od sebe viditelně odděleny a budou představovat zmíněné tři stavební bloky celého rozhraní, nástroj pro generování svalových vláken, generátor a úponovou oblast. Na rozdíl od druhého návrhu se zde nenachází žádná lišta nabídek. První část lze nazvat částí všeobecnou a uchovávala by stejná data jako nástroj v předchozích návrzích, akorát bez výběrového seznamu. Zbytek panelu by se rozdělil na dvě poloviny. Levá by sloužila k operování s daty generátoru, pravá pro manipulaci s úponovými oblastmi. Začneme částí levou. Nacházela by se zde stejná data jako v panelu generátoru v prvním návrhu. Rozdíl by zde byl v tom, že výběrový seznam pro generátory, který se dříve nacházel v panelu nástroje, by se nacházel zde, konkrétně v horní části levé podčásti. Zůstala by tlačítka pro přidání a odebrání, editační tlačítko by zmizelo. Při vybrání generátoru v seznamu by se načetla data do uživatelských prvků, které by se nacházely pod ním. Tedy, při vybrání různých generátorů by se jejich data objevila v grafických prvcích a uživatel by je mohl dle libosti měnit. Totožně by fungovala i pravá část. V horní části by se nacházel výběrový seznam s úponovými oblastmi. Jelikož je seznam úponových oblastí vázán na generátor, jeho obsah by závisel na vybraném generátoru z levé podčásti. Po vybrání jedné z nich či vytvoření nové, by se data načetla do uživatelských prvků pod ní, kde by uživatel editoval změny. Obě podčásti by měly své vlastní potvrzovací tlačítko.



Obrázek 6.7: Třetí návrh možného zpracování GUI

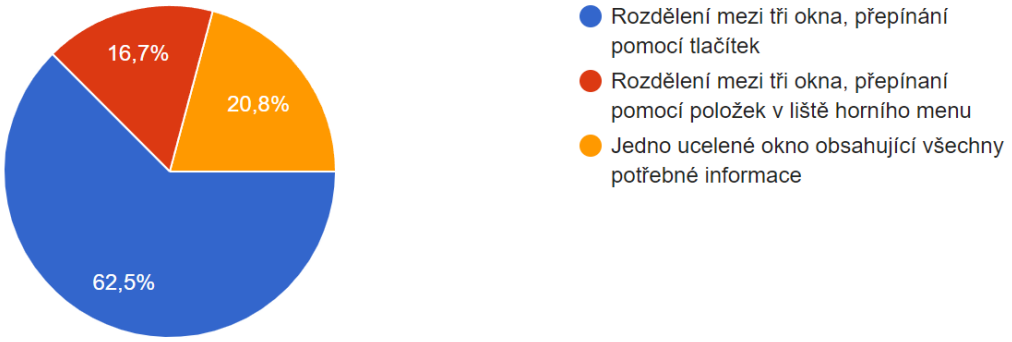
6.3.4 Výhody a nevýhody návrhů

Žádný z představených návrhů není dokonalý. Každý má své pro a proti, záleží tudíž na subjektivním názoru jedince. První návrh je dle mého nejlépe strukturovaný, z čehož vyplývá i jeho intuitivnost. Uživatel není vystaven hromadě dat naráz a rozdělení na jednotlivé panely podporuje myšlenku vnořování, která je i v souboru s daty. Výběrovým seznamem uživatel ovládá, který prvek chce vybrat a svůj výběr musí navíc ještě potvrdit okolními tlačítky. Samozřejmě, že se splést může, je tu však menší prostor pro chybu než například v návrhu druhém. Tam se zobrazí generátor/úponová oblast hned po vybrání. Hierarchie zde může zastávat i pozici nevýhody. Vyvstávají zde otázky jako: Co kdyby chtěl uživatel editovat více generátorů/úponových oblastí najednou? Není neustálé přesouvání se z jednoho panelu do druhého otravné, možná třeba i náročné na vykreslování grafických prvků neustále dokola? Stejnou nevýhodou trpí i návrh druhý. Návrh je sice ořezán o několik grafických prvků, nachází se zde ale otázka přívětivosti, zda uživatele vůbec napadne, že se v liště nabídky může objevit něco jako množina generátorů/úponových oblastí. Tato možnost sice přináší na pohled jednodušší rozhraní, otázkou ovšem zůstává, zda přehnané zjednodušování nakonec spíš

neuškodí než pomůže. Nad velikostí třetího návrhu by se dalo polemizovat. Pravdou je, že dat k nastavení není tolik, aby se nevměstnala do jednoho panelu. Tím by se přeskočilo opětovné vykreslovaní grafických prvků a měnila by se pouze data, která uchovávají. To se dá považovat za výhodu. Nevýhodou pak může být nekonzistentnost dvou podčástí, které obsahuje. Při změně jedné se změní zároveň i druhá a to by mohlo potencionálního uživatele zmást. Zkrátka, co může být pro jednoho nevýhodou je pro druhého výhodou a obráceně. Proto jsem nedbal jen na názor svůj, ale také na názory ostatních.

6.4 Názory tázané skupiny

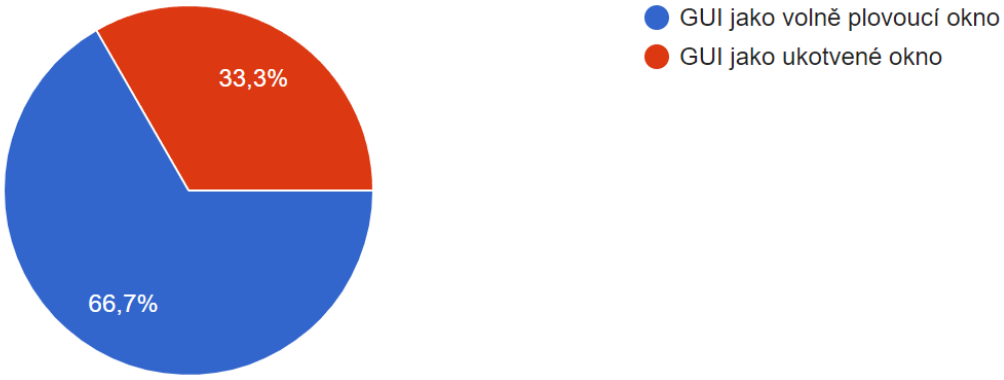
V kapitole 6.1 byla zmíněna skupina lidí (24), která vyplnila dotazník týkající se návrhu uživatelského grafického rozhraní. Tázaní odpovídali celkem na čtyři různé otázky (graf odpovědí na jednu z nich je na obrázku 6.1). První a nejdůležitější otázka byla na jednotlivé návrhy rozhraní. Skupině byly návrhy představeny totožným způsobem jako právě zde, tedy popsány a ukázány nástřely budoucího rozhraní. Obrázek 6.8 ukazuje graf odpovědí na zmíněnou otázku.



Obrázek 6.8: Graf reprezentující podíl odpovědí na otázku uživatelské přívětivosti návrhu

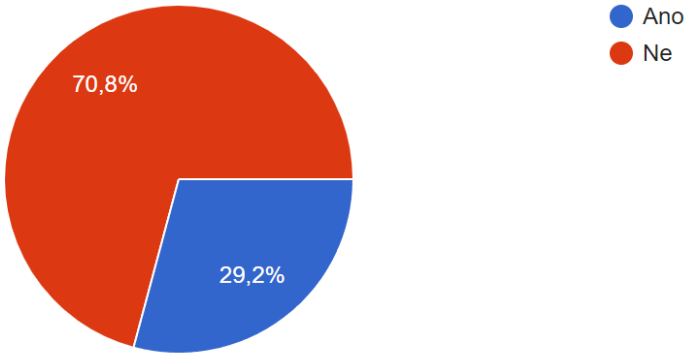
Z grafu lze vyčíst, že více než polovina tázaných by preferovala rozdělení mezi tři okna a přepínání pomocí tlačítek, tedy návrh číslo jedna. Na druhém místě se umístil třetí návrh, jeden celistvý panel. Tento návrh zvolilo pět lidí. Na posledním místě pak skončil návrh s lištou nabídek, který skupina označila za nejméně přívětivý. Zbylé dvě otázky byly mířeny na prostředí systému OpenSim a na chování rozhraní plug-inu v něm. Tázaným byla položena otázka, zda by preferovali okno plug-inu (jednotlivé panely či jeden

celistvý panel) volně plovoucí či okno ukotvené. Důvodem této otázky je fakt, že při vytváření rozhraní se tvůrce může rozhodnout, zda se bude panel chovat jako panely OpenSim, tedy ukotvené v určitém místě systému. Pokud tak vývojář nezvolí, panel se chová jako obecný volně plovoucí okenní prvek.



Obrázek 6.9: Graf reprezentující podíl odpovědí na otázku pohybu okna

Graf na Obrázku 6.9 vykazuje převahu 2:1 pro volně plovoucí okno grafického rozhraní. Na otázku proč většina lidí odpověděla, že mají rádi, když mohou hýbat rozhraním jak je v daném okamžiku potřeba. Nelimituje je struktura systému a mohou uzpůsobit pracovní plochu svým potřebám. Poslední otázka se opět váže na vlastnosti OpenSim. Vývojář si může zvolit, zda se okno rozhraní má zobrazit automaticky při startu aplikace či nikoliv. Názor tázané skupiny můžeme vidět na Obrázku 6.10.



Obrázek 6.10: Graf reprezentující podíl odpovědí na otázku automatického spuštění

Sedmdesáti procentům tázaných se nelíbí myšlenka automatického zobra-

zení grafického rozhraní při spuštění systému OpenSim. Domnívám se, že by příčinou mohl být rozsáhlý repertoár systému v podobě nejrůznějších funkcí a metod. Uživatel, který využívá aplikaci pouze za účelem využití automatického generování svalových vláken by takové vylepšení uvítal. Pokud se ale bavíme o uživateli, který s OpenSim pracuje běžně a používá ho na řadu jiných úloh než je automatické generování, pak by mohlo být automatické zobrazování plug-inu přinejmenším otravné.

6.5 Knihovny

Při vytváření grafického rozhraní se musí brát zřetel i na knihovny, které je k této činnosti nutno využít. Hlavní myšlenka je taková, že grafické rozhraní, které vznikne, bude mít na starost knihovna. Stejně tak práce s XML soubory. Navržená rozhraní musejí být schopna ukládat i načítat soubory s daty zmíněné v kapitole 6.2. Celé grafické rozhraní systému OpenSim je vyvíjeno v jazyce Java a tomu musí být přizpůsobeno hledání vhodné knihovny pro zmíněné dva úkoly.

6.5.1 XML

Java nabízí mnoho knihoven určených pro práci s XML dokumentem. Valná většina je součástí Javy a tudíž není potřeba stahovat či nějakým způsobem manipulovat s externími knihovnami. Pod drobnohledem se nachází dvě nejznámější, SAX a DOM.

Simple API for XML neboli SAX, je algoritmus pro práci s XML dokumenty řízený událostmi [11]. SAX pracuje s dokumentem pouze na úrovni jednotlivých elementů a tudíž je velice přívětivý po stránce vytížení paměti počítače. Díky tomu, že je proces algoritmu sekvenční a řízený událostmi, SAX zpracovává dokumenty rychleji než druhý potencionální analyzátor, DOM. SAX má bohužel i své nevýhody, jelikož teoreticky každá validace XML dokumentu potřebuje dokument celý. Například, aby bylo možné ověřit, že každý element má přijatelnou posloupnost elementů podřízených, musí být informace o tom, jaké podřízené elementy byly viděny u každého elementu nadřazeného, uchovány, dokud se nadřazený element neuzavře. Dalším možným záporem je samotná sekvenčnost. Pokud potřebujeme vytvářet nad XML dokumentem složitější operace než sekvenční hledání či čtení, lépe nám poslouží analyzátor DOM.

DOM, celým názvem Document Object Model, je analyzátor, který vyu-

žívá hierarchie XML dokumentu a chová se k němu jako k datové struktuře stromu, kde každý uzel představuje objekt určité části dokumentu [10]. Každá větev stromu pak končí elementem, kterým začala a každý uzel obsahuje objekty, pokud element v dokumentu obsahuje elementy podřízené. Vývojář pak může provádět nad dokumentem stejné operace, jako nad datovou strukturou stromu. Nevýhodou zůstává fakt, že aby byl analyzátor schopen vytvořit datovou strukturu stromu, musí dokument nejdříve celý načíst do paměti. Paměťové vytížení je ekvivalentní délce XML dokumentu a načítání rozsáhlého souboru může trvat řádové i desítky sekundy. Druhou stranou mince je skutečnost, že jakmile je soubor načten do paměti a zhotoven strom nad ním, lze přistupovat k jakékoliv části dokumentu téměř instantně. Vzhledem k velikosti konfiguračních souborů pro plug-in, který jsem měl možnost vidět však DOM nepředstavuje problém a lze ho tedy upřednostnit.

6.5.2 GUI

Grafické uživatelské rozhraní bude hlavním prvkem, které uživatel uvidí a bude modifikovat po celou dobu jeho práce. Tudíž je na vývojáři, aby vybral správné, pro danou práci vhodné, uživatelsky přívětivé prostředí ve kterém uživatel stráví většinu času. Java, jakožto vyšší programovací jazyk nabízí takových knihoven hned několik. Mezi nejznámější a zde podrobněji popsané patří AWT, Swing a JavaFX.

Nejstarší a dnes již nejméně používaný je Abstract Window Toolkit, zkráceně AWT. AWT samotné se v dnešní době pro grafické rozhraní téměř nepoužívá a když ano, tak dohromady se svým nástupcem Swingem. Důvodem proč Swing nahradil v pozdějších letech AWT byly chyby. AWT například nepodporovalo multiplatformní chování na kterém si Java zakládá nebo nevyužívalo třívrstvou architekturu, pouze dvouvrstvou.

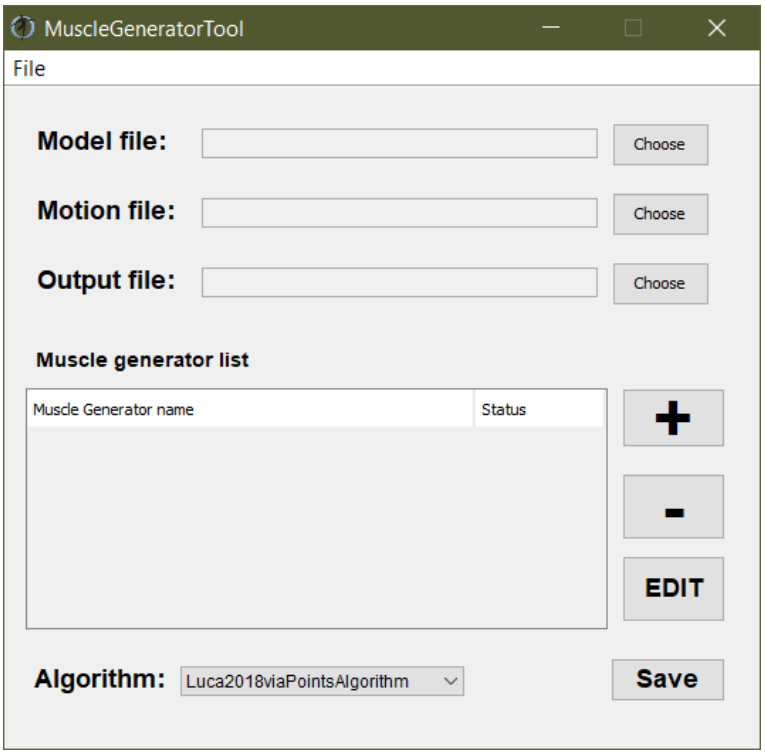
Nástupcem AWT je již zmíněný Swing. I přesto, že už také nepatří mezi nejnovější knihovny, je stále ve velké míře využíván ve spojení s AWT pro určité prvky. Swing v porovnání s AWT obsahuje více funkcionality, podporuje třívrstvou architekturu, je platformně nezávislý a jeho vykonávací doba je nižší než u jeho předchůdce. Další výhodou může být i fakt, že grafické rozhraní systému OpenSim je taktéž vytvořeno knihovnou Swing.

Poslední a zároveň nejmladší knihovnou pro vývoj grafického rozhraní v Javě je JavaFX. Co se integrity týče, JavaFX je zabudována ve starších ver-

zích Javy. Od JDK (Java Development Kit) verze 11 už však není součástí a pro její využití je nutno ji specificky zahrnout [12]. Využití knihovny v posledních letech rapidně roste a nahrazuje využití Swingu. Hlavním důvodem může být fakt, že i přesto, že je Swing novější než AWT, funkcionalitou zaostává. Swing například nepodporuje dotykové zařízení, neumí vytvářet a pracovat s animacemi a navíc nepodporuje Data binding. V těchto ohledech JavaFX předchází Swing. Pro účel této práce nic takového ale není potřeba.

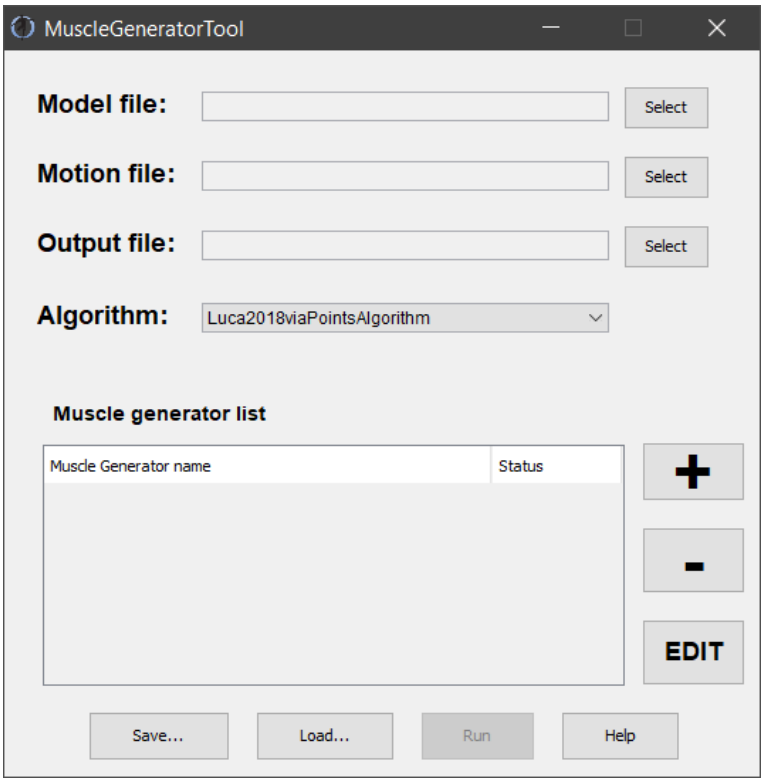
6.6 Postupný vývoj GUI

V našem případě se názor většiny tázaných ztotožňuje s názorem vývojáře a první návrh se zdá býti jako nejlepší řešení. Budeme se tedy nadále soustředit pouze na návrh první, ostatní vypustíme. Prvotní návrh bylo možno vidět na Obrázku 6.3 pro nástroj, Obrázku 6.4 pro generátor a 6.5 pro úponovou oblast. Návrh byl vytvořen bez jakékoliv konzultace a tudíž byl limitován pouze mojí znalostí a pochopením zadání. Po zkonzultování a druhém pohledu bylo jasné, že rozhraní by mělo umět nabídnout uživateli více než pouhé základní nastavení. Kromě vytvoření nastavení by mělo být rozhraní schopno i konfiguraci naimportovat. Zároveň by bylo uživatelsky přívětivé, kdyby grafické rozhraní sdělovalo uživateli, v jakém stavu se nachází jednotlivé generátory/úponové oblasti. Proto nastalo pár změn a přišla modifikace návrhu, kterou můžeme vidět na Obrázku 6.11.



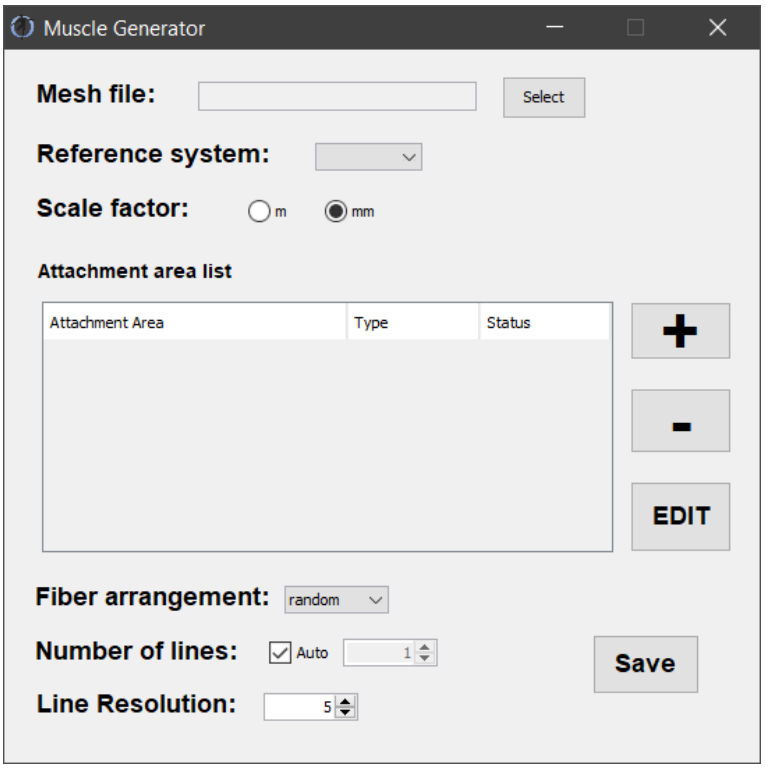
Obrázek 6.11: Modifikace prvního návrhu

Na pohled největší změnou je roztažení a ukotvení jednotlivých prvků oproti obrázku 6.3. Největší změna přichází ale v horní liště nabídek. Zde přibyly možnosti jako je načtení hotové konfigurace v podobě XML dokumentu a nastavení základní cesty průzkumníku. Zároveň byl seznam generátorů/úponových oblastí nahrazen tabulkou. Panely generátoru a úponové oblasti byly taktéž modifikovány, ale jen z hlediska vzhledu (natáhnutí textové oblasti, struktura mezer a jiné), žádná nová funkcionalita přidána nebyla. Později přišla ještě jedna modifikace, rozsáhlejší, která zasáhla všechny panely a dokonce přidala jeden nový. Důvodem byla další konzultace plus inspirování se od samotných tvůrců systému OpenSim. Do panelu nástroje byly přidána čtyři tlačítka, *Save*, *Load*, *Run* a *Help*, která do jisté míry nahradila lištu nabídek. Identický přístup použili vývojáři systému například pro nástroj *Scale Tool*. Obrázek 6.12 reprezentuje finální vzhled panelu nástroje.



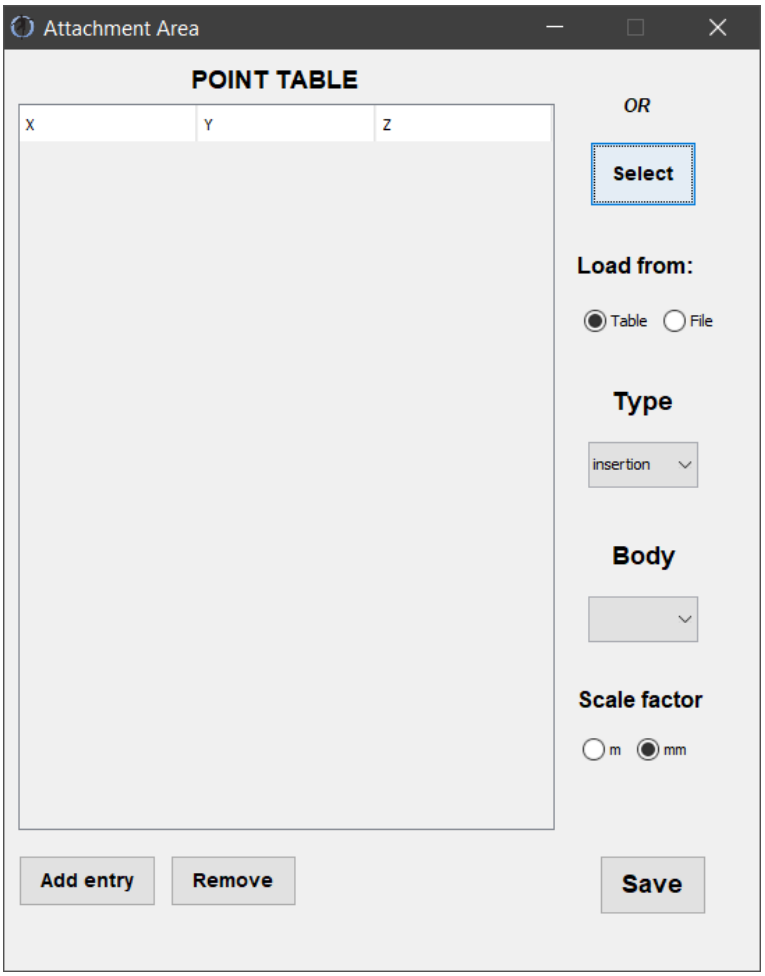
Obrázek 6.12: Konečný návrh rozhraní nástroje

Panel generátoru prošel také řadou změn a výsledný vzhled je vyobrazen na Obrázku 6.13. Struktura prošla změnami ve vzhledu, seznam byl taktéž nahrazen tabulkou a tlačítko pro výběr souboru s vrcholy bylo přesunuto do panelu úponových oblastí. *Muscle geometry* bylo přejmenováno na *Reference system* a tlačítko potvrzení je nyní *Save* z předešlého *Done*. Zároveň bylo přetvořeno tlačítko průzkumníka z *Choose* na *Select*.



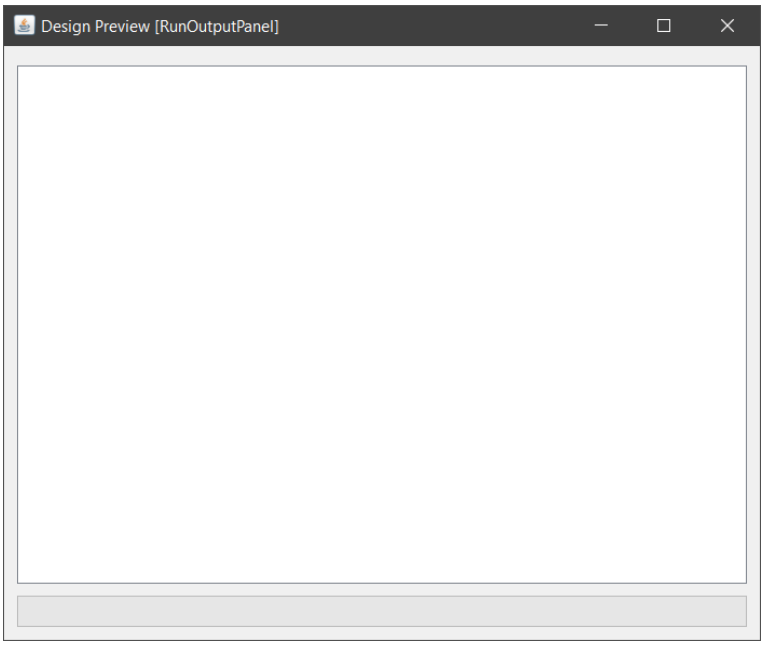
Obrázek 6.13: Konečný návrh rozhraní generátoru

Rozhraní úponové oblasti zůstává v jádru stejné. Jak již bylo zmíněno, tlačítko pro výběr souboru s vrcholy bylo přesunuto z generátoru sem. Zároveň přibyly přepínače pro rozhodování, zda použít data z tabulky rozhraní či ze souboru. Objevily se žádosti na automatické generování následující řádky poté, co byla poslední vyplněna. To by mělo za následek rychlejší a pohodlnější přidávání vrcholů než neustálé klikání na přidání vrcholu nového. Zároveň jsem chtěl ale vyhovět žádosti, kde má uživatel vše pod kontrolou a přímo rozhoduje co kam přidá nebo kde co ubere. Nechal jsem proto v návrhu oba přístupy a je jen na uživateli, který mu sedne víc. Konečný vzhled panelu úponové oblasti je reprezentován Obrázkem 6.14.



Obrázek 6.14: Konečný návrh rozhraní úponové oblasti

Novým přírůstkem do kolekce panelů je panel s výpisem výstupu na Obrázku 6.15. Jak už bylo řečeno dříve, grafické rozhraní bude umět i pustit samotný algoritmus pro automatické generování svalových vláken. Uživatel musí být o takové skutečnosti řádně informován a proto vznikl panel nový. Součástí panelu jsou pouze dva prvky. Textová oblast, kam se vypisuje progres algoritmu a následný vizuální indikátor průběhu. Po úspěšném běhu algoritmu se uživateli zobrazí hláška hlásící úspěšný běh a po stisknutí zavře celý panel. V opačném případě vrátí návratovou hodnotu algoritmu a nechá panel otevřený, aby si mohl uživatel přečíst, kde nastala chyba.



Obrázek 6.15: Návrh panelu s výpisem výstupu

6.7 Požadavky

Implementace řešení by měla mít bázi třívrstvé architektury. Uživatelské rozhraní by měl být určitý prvek, který zařídí a vykreslí jedna z knihoven probraných v kapitole 6.5.2. Datový model aplikace pak představuje třídu, například přepravku, která vyobrazuje určitý prvek pro generování svalových vláken, třeba instanci generátoru svalu. Řídící logiku pak mohou zastupovat různé manipulátory či naslouchači GUI prvků. Kvůli tomu je nutno specifikovat, jakých hodnot může daný prvek nabývat a jaké chovaní je od něj očekáváno.

- `TextField` - Zobrazuje název vybraného souboru. Textové pole je pouze informativní, tudíž by mělo být needitovatelné.
  - Model File
  - Motion File
  - Output File
  - Mesh File
- `ComboBox` - Slouží pro výběr z určité množiny možností. Některé množiny jsou předem dané (P), jiné získávají data dynamicky z načtených souborů (D).

  - Algorithm (P)
  - Reference system (D)
  - Fiber arrangement (P)
  - Type (P)
  - Body (D)

- `Table` - Prvek zastupuje tabulku. Tabulka může mít jednu a více vlastností (sloupců), stejně tak i objektů (řádků).
  - **Seznam generátorů** - Tabulka s názvem a stavem, ve kterém se generátor nachází. Název může být libovolné podoby, stav může nabývat pouze *INCOMPLETE* nebo *COMPLETE*.
  - **Seznam úponových oblastí** - Stejný případ jako seznam generátorů.
  - **Tabulka vrcholů** - Sloupce představují souřadnice XYZ, řádek pak instanci jednoho vrcholu. Zadávaná data mohou být pouze číselné podoby. Prázdné řádky se ignorují. Na nedokončené vrcholy aplikace upozorní.

- `RadioButtons` - Skupina dvou přepínačů, představují výběr možností, z nichž může být v daný okamžik aktivní právě jedna.
  - **Scale factor** - Přepínače se nachází v generátoru i v úponové oblasti a v obou rozhodují, zda jsou data v milimetrech či v metrech.
  - **Load from** - Dva přepínače v panelu úponové oblasti. Uživatel pomocí nich rozhoduje, zda se mají data vrcholů použít z vybraného souboru nebo z tabulky v rozhraní.

- `Spinner` - Celočíselný inkrement v panelu generátoru. Nastavuje počet vláken a jejich rozlišení. Lze zadat celé, nezáporné číslo. Inkrement je v takovém případě roven jedné. Pokud je `ComboBox` *Fiber arrangement* nastaven na uniform, mění dle specifikací inkrement tak, aby byl počet vláken roven druhé mocnině následujícího čísla.

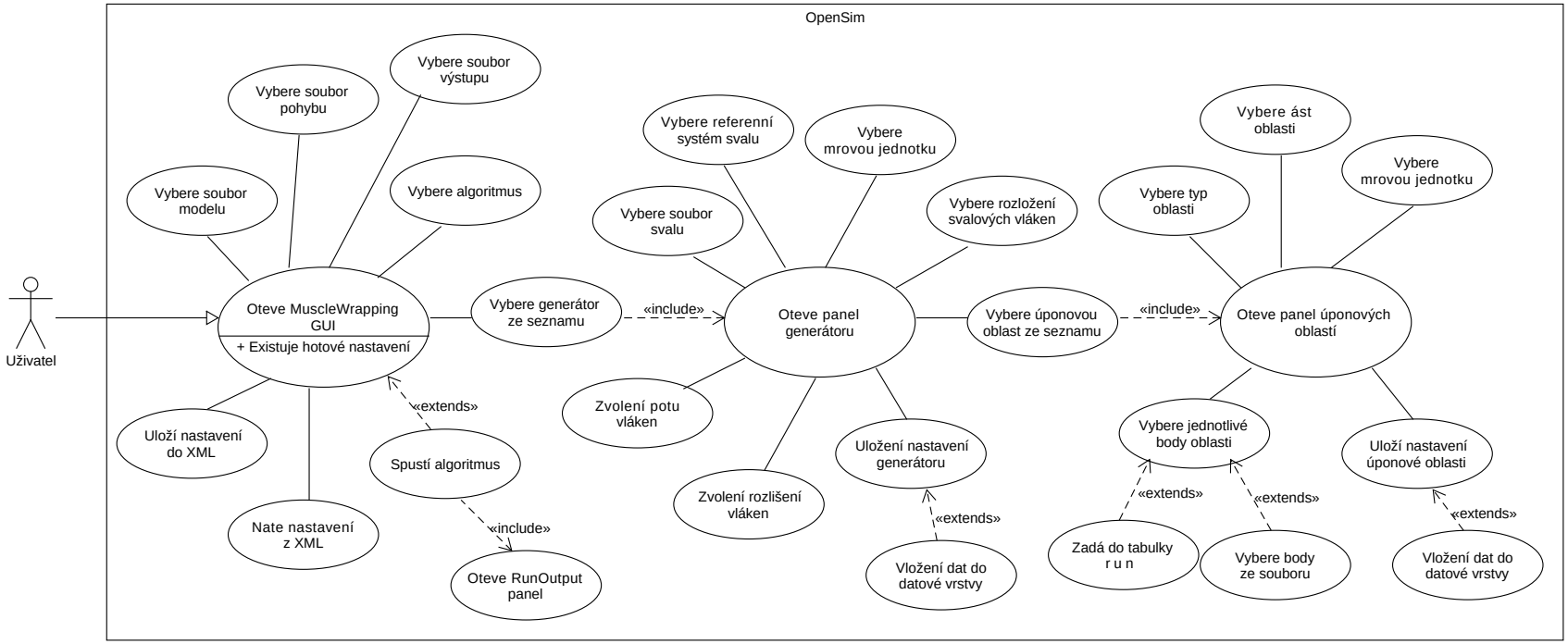
- `Checkbox` - Zaškrtávací políčko, může nabývat pouze dvou stavů, vybráno či nevybráno. Nachází se pouze v panelu generátoru a představuje použití automatického počtu svalových vláken. Pokud je vybráno, `Spinner` pro počet vláken ztrácí význam a je deaktivován.

- `Button` - Tlačítek je v konečném návrhu celá řada a proto budou popsány obecně. Některá tlačítka dělají to samé, akorát s jinými daty, proto nebudou explicitně popsána.
  - **Select** - Tlačítko vyvolá zabudovaný průzkumník souborů a přiřadí vybraný soubor do textového pole.
  - **Add (+)** - Přidá nový prvek (generátor, úponovou oblast, vrchol) do dané datové struktury.
  - **Remove (-)** - Odebere vybraný prvek z datové struktury.
  - **Edit** - Upraví vybraný prvek tím, že pro něj otevře nový panel s jeho daty.
  - **Save** - Verifikuje nastavení a pokud je validní, uloží nastavená data do datového modelu.
  - **Load** - Načte nastavení z validního XML souboru do rozhraní.
  - **Help** - Zobrazí webovou stránku s návodem
  - **Run** - Pomocí uloženého nebo načteného nastavení spustí modul pro automatické generování svalových vláken. Dokud není nastavení kompletní, je zakázáno. Pokud nastane modifikace při načteném či již jednou uloženém nastavení, aplikace se uživatele zeptá, zda chce změny před spuštěním uložit.
- `TextArea` - Textová oblast se využívá při výpisu stavu po spuštění algoritmu pro automatické generování svalových vláken. Nachází se v panelu výpisu, vyobrazeném na Obrázku 6.15. Jedná se o informativní výpis, tudíž je interakce s textovou oblastí zrušena.
- `ProgressBar` - Nachází se taktéž v panelu výpisu. Reprezentuje vizuální progres automatického generování svalových vláken a je závislý na výpisu textové oblasti.

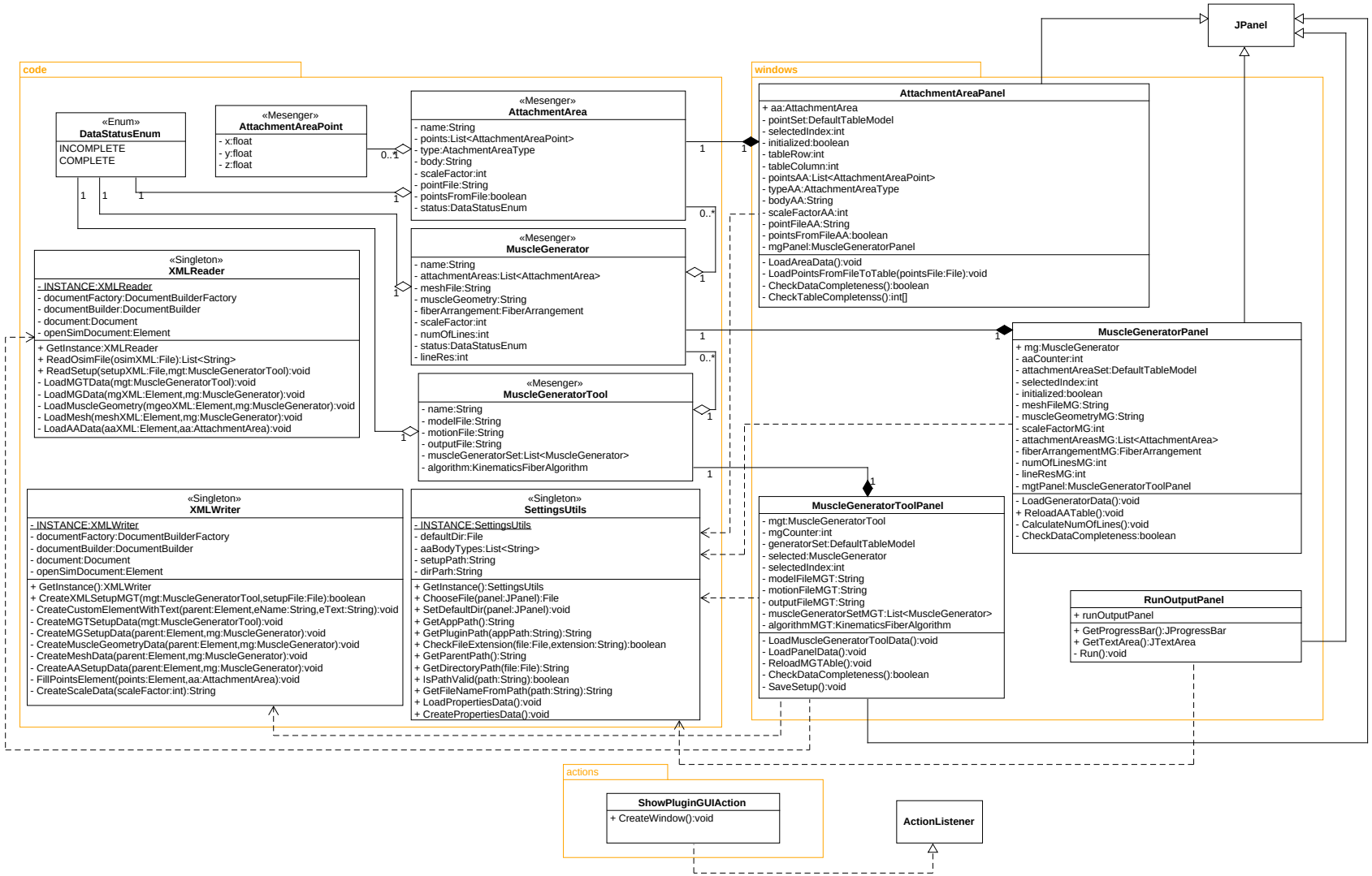
7 Implementace

V kapitole 6.5 byly navrženy knihovny pro práci s XML dokumenty a GUI. Pro XML jsem si vybral knihovnu DOM, kvůli přívětivému zpracování dat pomocí stromu a faktu, že při načítaní mám celý dokument jako objekt po ruce. Nevýhody DOMu by se neměly projevovat, neboť načítané dokumenty by neměly mít nikdy takovou velikost, která by byla potřebná pro zpomalení systému. Pro GUI jsem šel cestou spojení Swingu s prvky AWT. Důvodem je, že celý systém OpenSim je taktéž ve Swingu s prvky AWT, tudíž bude modul splňovat konvence systému. Navíc paleta grafických prvků vývojového prostředí Netbeans je předem naplněna prvky ze Swingu a není tedy potřeba nic importovat. Na Obrázku 7.1 je znázorněn diagram případů užití aplikace.

Z návrhu vyplývá, že by měla být aplikace členěna po vzoru třívrstvé architektury. Toho lze dosáhnout na logické úrovni, ale v tomto konkrétním případě implementace je aplikační vrstva společně s prezenční vrstvou na společném místě. Základním stavebním blokem jsou tedy dva balíky. Jeden balík pojmenovaný `code` obsahuje třídy představující datovou vrstvu a třídy pro práci s XML soubory. Druhý balík `windows` reprezentuje spojení prezenční vrstvy s vrstvou aplikační. Důvodem je samotná struktura kódu. Prezenční vrstva zde představuje pouze instance tříd z grafické knihovny. Ty mohou být obohaceny o různé vlastnosti, třeba co se vzhledu týče, faktem ale zůstává, že se jedná pouze o vizuální prvek bez jakékoliv funkcionality. Tu dodává vrstva aplikační pomocí svých naslouchačů, řadičů a jiných ovládacích prvků. Ty poté naslouchají na událost na grafickém prvku. Právě proto se úzce spjaté vrstvy v implementaci slučují do jedné větší, reprezentované balíkem `windows`. V celém modulu se nachází ještě jeden balík `actions`. Balík obsahuje pouze jednu třídu, pomocí které se celé grafické rozhraní registruje do systému. Na Obrázku 7.2 můžeme vidět diagram tříd celého projektu, rozřazeného do jednotlivých balíku.



Obrázek 7.1: Diagram případu užití aplikace



Obrázek 7.2: Diagram tříd aplikace, neobsahuje gettery, settery a metody typu `actionPerformed` pro lepší přehlednost

7.1 actions

Aby bylo možné pracovat se systémem OpenSim a přidávat do něj nové rozšíření, je ho potřeba o takové skutečnosti informovat. OpenSim využívá přídavných modulů platformy Netbeans pro účel přidávání rozšíření. Při vytváření nového souboru se pak vývojové prostředí zeptá, zda chceme využít kategorii vývoje modulu. Ta nabízí různé typy, pro náš účel potřebujeme typ *action*. Zmíněný typ nám dá na výběr, v jaké nabídce z lišty nabídek se má naše akce objevit, jakou má mít pozici a další možné nastavení. Díky tomu se nám vytvoří třída typu *action*, která se po spuštění systému registruje do grafického rozhraní. Zde se poté volá samotné vytváření panelu. Je nutno podotknout, že vytváření nového grafického okna za normálních okolností vzniká pomocí `JFrame`, nikoli `JPanel`. Při takovém přístupu ale systém nedokáže zaregistrovat grafické okno a tím pádem ho ani nevykreslí. Řešením je využít službu knihovny `DialogUtils` od vývojářů OpenSim. Místo vytváření `JFrame` vytvoříme `JPanel` a ten předáme službě knihovny. Ta nám vrátí `JFrame` podle svých předpisů, který už systém dokáže zobrazit.

7.2 code

Balík obsahuje celkem osm tříd. Čtyři z toho jsou přepravky datové vrstvy. Každá přepravka představuje objekt dat, které přepravuje. Hierarchicky, pro celý nástroj je zde `MuscleGeneratorTool`, pro jednotlivé generátory `MuscleGenerator`, pro jejich úponové oblasti `AttachmentArea` a pro jednotlivé body úponových oblastí `AttachmentAreaPoint`. Dále balík uchovává jeden výčtový typ `DataStatusEnum`, který signalizuje, zda jsou data kompletní nebo ne. Poslední je trojice jedináčků. Jedná se o dvě třídy pro práci s XML, `XMLReader` a `XMLWriter`, a o `SettingsUtils` pro utility. Třídy pro práci s XML dokumentem mají velice podobný základ, obsahují proměnné pro práci s dokumentem a liší se pouze v manipulování s daty. Jedna elementy dokumentu čte a jejich hodnoty předává zmíněným přepravkám, druhá data z přepravek zapisuje do elementů. `SettingsUtils` je pak jedináčkem, který obstarává různá nastavení aplikace. Obsahuje metody pro práci se soubory, cestami a jiné metody, které se využívají na více místech v kódu.

7.3 windows

Na každý panel připadá jedna třída, všechny dědí od třídy `JPanel`. Každá obsahuje automaticky vygenerovanou metodu `initComponents()`, která slouží

k registraci grafických prvků rozhraní. Metoda je vykonávána v konstruktoru při vytváření tříd. Těmito třídami jsou `MuscleGeneratorToolPanel`, `MuscleGeneratorPanel`, `AttachmentAreaPanel` a `RunOutputPanel`. Všechny obsahují naslouchače na jednotlivé grafické prvky typu `actionPerformed()` nebo `valueChanged()`. Třídy uchovávají privátní data datové vrstvy, které se pomocí grafického rozhraní nastavují. Metoda `CheckDataCompleteness` pak validuje data a při úspěchu je uloží do přepravek datové vrstvy.

8 Testování

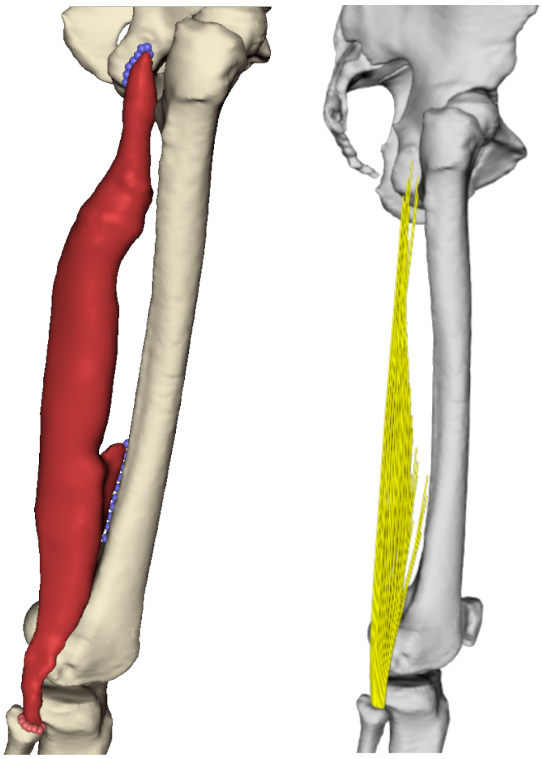
Testování aplikace bylo rozděleno do tří kategorií: Unit testy, uživatelské testování a testovaní na množině dat LHDL. Kategorie Unit testů není příliš obsáhlá jelikož je většina tříd návrhového vzoru přepravka, kde se pouze nastavují hodnoty bez nějakého výpočtu. Unit testy byly vytvořeny konkrétně pro metody tříd `XMLReader`, `XMLWriter` a `SettingsUtils`. Pokrytí testů není stoprocentní, ale postačuje pro testování při přidání nové funkcionality do aplikace.

8.1 LHDL set

Jedním z úkolů této práce bylo otestovat vytvořenou konfigurační aplikaci na množině dat LHDL(Living Human Digital Library). LHDL byl evropský projekt, jehož cílem bylo vytvořit digitální atlas člověka. Aplikace byla otestována na datech čtyř různých svalů plus na jednom již vytvořeném konfiguračním souboru, který se skládá ze svalů Gluteus medius, Gluteus maximus a Illiacus. Jedná se o data svalů Biceps femoris, Rectus femoris, Vastus lateralis a Vastus medialis. Pro data svalu Rectus femoris se mi nepodařilo vytvořit funkční konfigurační soubor, pomocí kterého by plug-in pro automatické generování zvládl vytvořit svalová vlákna. Bohužel se mi nepodařilo zjistit zdroj problému, domnívám se ale, že jde o špatně vyextrahovaná data, se kterými si plug-in nedokáže poradit. Pro každý sval jsem dle mého úsudku vytvořil nejlepší možnou konfiguraci, kterou jsem následně porovnal s její předlohou. Následně jsem vytvořil různé modifikace konfiguračního souboru a jejich výsledky jsem mezi sebou porovnal.

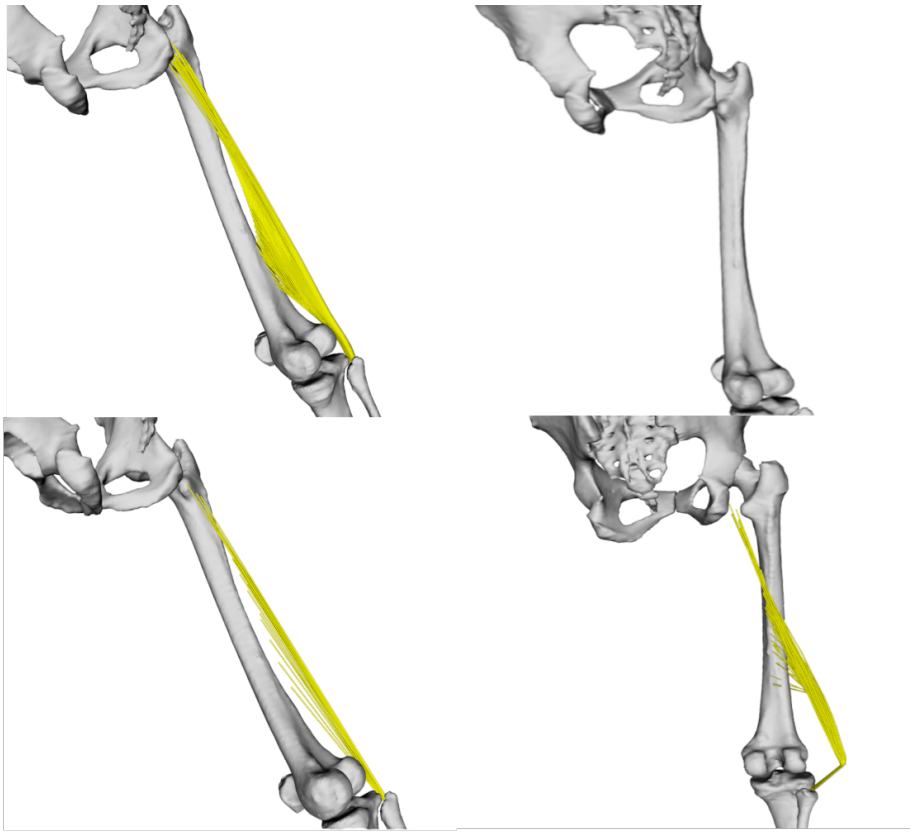
8.1.1 Biceps femoris

Biceps femoris jako jediný ze zmíněných svalů pro které byly generovány konfigurační soubory obsahuje více než dvě úponové oblasti, konkrétně dva počátky a jednu úponovou část. Na Obrázku 8.1 můžeme vidět porovnání předlohy svalu s výstupem plug-inu, jenž výsledek vytvořil pomocí konfiguračního souboru. Pro takový výstup bylo nastaveno uspořádání vláken uniformně, jejich počet roven stovce a rozlišení vláken na patnáct.
Vizuálně výsledek v celkem pěkné míře odpovídá předloze. Avšak při generování může nastat problém, pokud při konfiguraci prohodíme počátky svalu. Pokud tak učiníme, plug-in vykreslí pouze jednu část svalu (počá-



Obrázek 8.1: Biceps femoris - vlevo šablona, vpravo výsledek generování

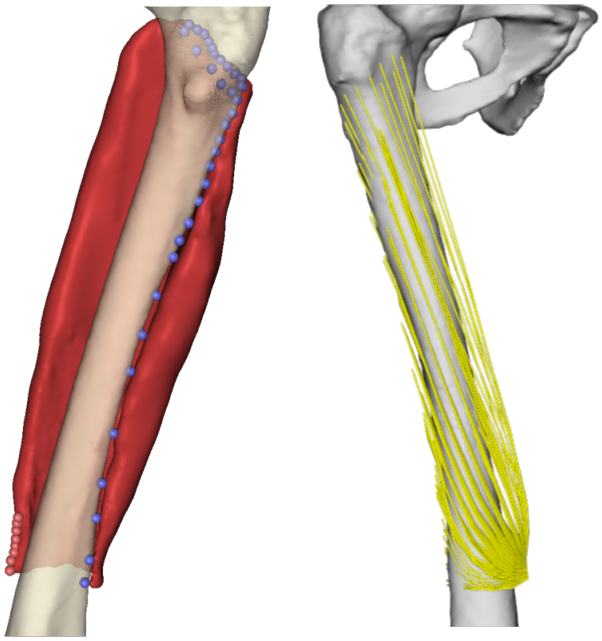
tek - úponová oblast), konkrétně tu kratší. Dále byly vytvořeny čtyři různé konfigurační soubory, ve kterých se měnily použité algoritmy a vlastnosti generátoru, tedy uspořádaní, počet vláken a jejich rozlišení. Porovnání výsledků můžeme vidět na Obrázku 8.2. V levé polovině obrázku se nachází generace pomocí algoritmu *Luca2018*, na pravé naopak *Cervenka2019*. V horní polovině je poté uspořádání uniformní, počet vláken roven stovce s rozlišením patnáct. V dolní polovině se nachází uspořádání náhodné s počtem vláken dvacet a rozlišením pět. Můžeme si povšimnout, že v pravém horním rohu nebyla vygenerována žádná vlákna, algoritmus sice nespadl na chybě, ale nedokázal dle konfigurace vlákna vygenerovat.



Obrázek 8.2: Porovnání čtyř různých výsledků konfigurací pro Biceps femoris

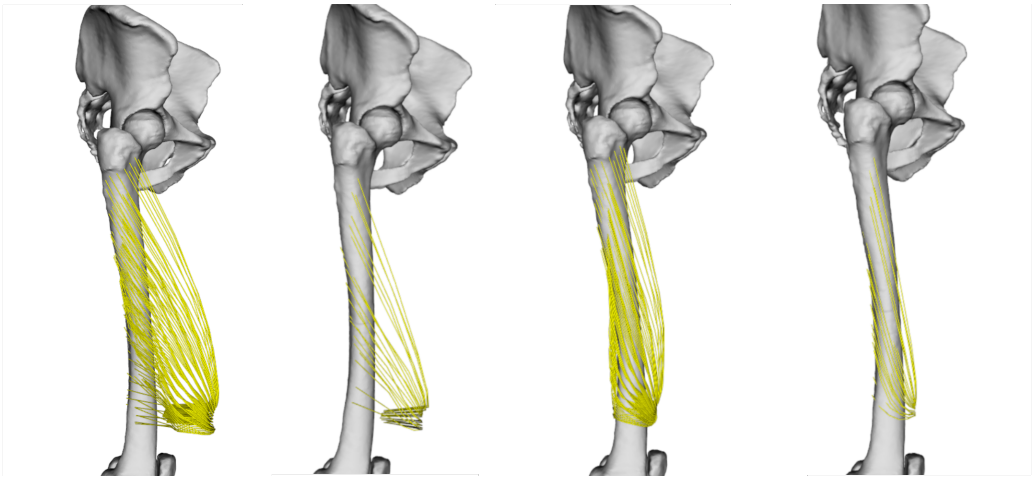
8.1.2 Vastus lateralis a medialis

Vastus lateralis a medialis jsem spojil do jedné podkapitoly, neboť jsou oba součástí čtyřhlavého svalu stehenního. Na Obrázku 8.3 je vyobrazeno porovnání výstupu plug-inu s předlohou pro lateralis a na Obrázku 8.5 pro medialis. Oba výstupy jsou nakonfigurovány stejným způsobem jako Biceps femuris v kapitole 8.1.1.



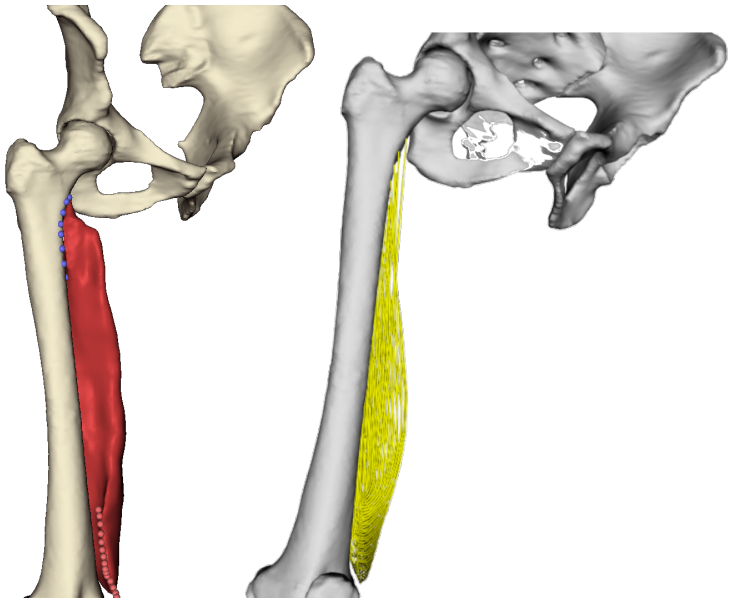
Obrázek 8.3: Vastus lateralis - vlevo šablona, vpravo výsledek generování

Poté jsem stejně jako u Biceps femoris vytvořil čtyři různé konfigurační soubory a dle nich vygenerované výsledky porovnal, tentokrát ale v jiném pořadí. Porovnaní pro Vastus lateralis můžeme vidět na Obrázku 8.4 a pro Vastus medialis na Obrázku 8.6. Pro oba případy platí, že se obrázek dělí na levou a pravou polovinu dle použitého algoritmu. Na levé straně se na-

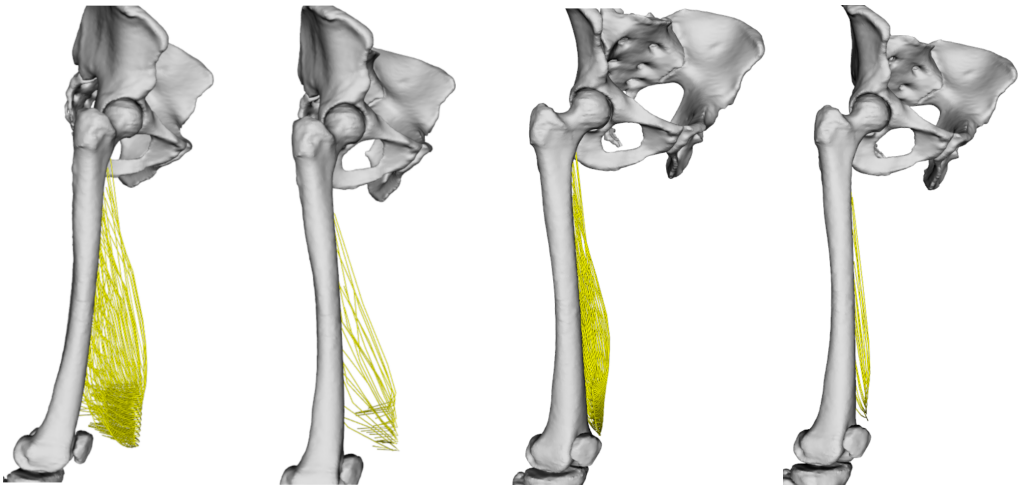


Obrázek 8.4: Porovnání čtyř různých výsledků konfigurací pro Vastus lateralis

chází výsledky vygenerované algoritmem *Cervenka2019*. Konkrétně, první zleva obsahuje uniformní uspořádání vláken, počet vláken roven sto s rozlišením patnáct. Druhý pak uspořádání náhodné s dvaceti vlákny a rozlišením pět. Stejná konfigurace platí i pro pravou polovinu výsledků, vygenerovanou algoritmem *Luca2018*. U obou svalů si můžeme povšimnout, že má *Cervenka2019* problém se správným uchycením svalových vláken v dolní části stehenní kosti.



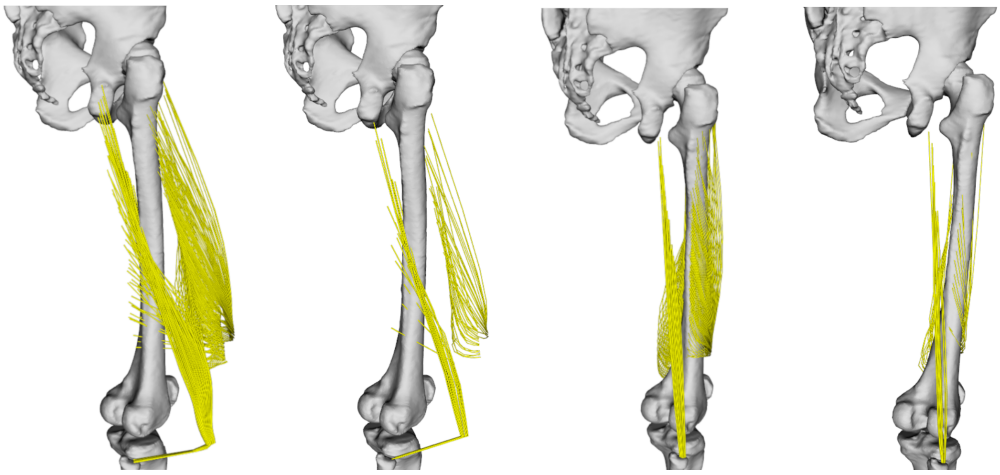
Obrázek 8.5: Vastus medialis - vlevo šablona, vpravo výsledek generování



Obrázek 8.6: Porovnání čtyř různých výsledků konfigurací pro Vastus medialis

8.1.3 Sloučení svalů

Předchozí svaly lze sloučit dohromady a vytvořit tak jeden větší konfigurační soubor, který spravuje všechny tři zmíněné svaly. Výsledkem takové konfigurace je vygenerování všech svalů zároveň. Na Obrázku 8.7 pak můžeme vidět již známe nastavení. Levá polovina je vygenerována pomocí algoritmu *Cervenka2019*, první část je uniformní uspořádání vláken s počtem sto a rozlišením patnáct, druhá část je uspořádání náhodné, počet vláken dvacet a rozlišení pět. Pravá polovina taktéž, akorát pro algoritmus *Luca2018*.



Obrázek 8.7: Porovnání čtyř různých výsledků konfigurací pro sloučené svaly

8.2 Uživatelské testování

Aplikace byla testována celkem na pěti různých zařízeních (včetně vývojářského). Nutno podotknout, že na stejných platformách, konkrétně *Windows 10*. Testování probíhalo následujícím způsobem. Tester dostal aplikaci společně s daty a návodem, který je popsán v příloze A. Poté co pomocí návodu uvedl aplikaci do provozuschopného stavu, zkusil vytvořit různé konfigurační soubory z dodaných dat. Bylo vytvořeno celkem 16 různých konfiguračních souborů, z čehož 4 z nich nedokázal plug-in pro automatické generování svalových vláken načíst. Důvodem byla v některých případech špatná konfigurace ze strany testera, v jiných zase špatná data, která plug-in nezvládl zpracovat.

9 Závěr

Cílem této práce bylo analyzovat strukturu softwarové architektury open-source systému OpenSim za účelem jeho rozšíření o grafické rozhraní pro přídavný modul, neboli plug-in, Muscle Wrapping 2.0. Společně s ním analyzovat i samotný plug-in a navrhnout vhodné grafické rozhraní zastupující konfigurační modul. Dále navržené grafické rozhraní implementovat a vytvořenou implementaci otestovat na úloze flexe kyčelního kloubu pro vybrané svaly v oblasti stehenní kosti datového setu LHLD.

Zadání práce bylo splněno v celém rozsahu. V kapitole 2 byla představena základní anatomická část za účelem lepšího porozumění tématu a následné snadnější orientaci v pozdějších kapitolách či testování. Kapitola 3 poté představila reprezentace muskuloskeletálního systému, se kterými plug-in pracuje (konkrétně s reprezentací pomocí Line of action) a jak probíhá jejich simulace. V kapitole 4 byla provedena analýza systému OpenSim. Analýza systému byla rozdělena do tří podkapitol (Core, GUI a Plug-ins), kde každá detailněji zkoumá danou část. Analýza plug-inu pro automatické generování svalových vláken byla provedena v kapitole 5. Kapitola se podrobně zaměřuje na onen proces kvůli lepšímu pochopení konfiguračních dat, které práce vytváří a modifikuje. Kapitola 6 nabízí možná řešení grafického rozhraní doplněné o názory tázané skupiny, využití možných knihoven společně s požadavky na grafické prvky. Samotná implementace konečného návrhu je poté popsána v kapitole 7. Samotné testování konfigurace je popsáno v kapitole 8 společně s porovnáním různých konfigurací mezi sebou.

Navázáním na tuto práci by mohlo být přidání dalších konfiguračních nastavení, která jsou momentálně ignorována. Jedná se například o konfiguraci pro vyhlazování nebo o zvolení šablony svalu, která byla zmíněna v podkapitole 5.1. Dále by bylo možné aplikaci rozšířit o funkci, která by dokázala v průzkumníku souborů vybraný sval rovnou zobrazit, aby se uživatel mohl předem podívat jak sval zhruba vypadá než si jej vybere.

Je zde i prostor pro další zlepšení samotného grafického rozhraní. Rozhraní by mohlo být obohaceno a nový kabátek v podobě lepších fontů, nových barev či po stylistické stránce lépe vypadajících tlačítek a jiných prvků.

Literatura

[1] Skeletal system, Nov 2019. Dostupné z: https://my.clevelandclinic.org/health/body/21048-skeletal-system.

[2] OpenSim API, Aug 2021. Dostupné z: https://simtk.org/api_docs/opensim/api_docs/index.html.

[3] Biologie člověka Základní Orientace na Lidském těle Pohyby V Kloubech. Dostupné z: https://eluc.kr-olomoucky.cz/verejne/lekce/177.

[4] OpenSim documentation. Dostupné z: https://simtk-confluence.stanford.edu:8443/display/OpenSim/OpenSim+Documentation.

[5] Angles, B. et al. Viper: Volume invariant position-based elastic rods. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*. 2019, 2, 2, s. 1–26.

[6] Delp, S. L. et al. OpenSim: open-source software to create and analyze dynamic simulations of movement. *IEEE transactions on biomedical engineering*. 2007, 54, 11, s. 1940–1950.

[7] IBM, Sep 2019. Dostupné z: https://www.ibm.com/docs/en/i/7.3?topic=introduction-advantages-xml.

[8] Kohout, J. – Kukačka, M. Real-Time Modelling of Fibrous Muscle. In *Computer Graphics Forum*, 33, s. 1–15. Wiley Online Library, 2014.

[9] 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.

[10] Oracle. Lesson: Document Object Model, . Dostupné z: https://docs.oracle.com/javase/tutorial/jaxp/dom/index.html.

[11] Oracle. Lesson: Simple API for XML, . Dostupné z: https://docs.oracle.com/javase/tutorial/jaxp/sax/index.html.

[12] Palmer, B. Using JavaFX with Java 11 or higher, Feb 2022. Dostupné z: https://blog.idrsolutions.com/using-javafx-with-java-11/.

[13] Petříček, R. *Výtvarná anatomie*. Univerzita Pardubice, 2020. ISBN 9788075602824.

[14] Rahman, M. et al. Musculoskeletal model development of the elbow joint with an experimental evaluation. *Bioengineering*. 2018, 5, 2, s. 31.

[15] Sendic, G. Musculoskeletal system, Oct 2021. Dostupné z: `https://www.kenhub.com/en/library/anatomy/the-musculoskeletal-system`.

[16] Thelen, D. G. – Anderson, F. C. Using computed muscle control to generate forward dynamic simulations of human walking from experimental data. *Journal of biomechanics*. 2006, 39, 6, s. 1107–1115.

[17] 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*. 2012, 226, 2, s. 161–169.

Seznam zkratek

GUI Grafical User Interface - Grafické rozhraní

LHDL Living Human Digital Library - množina dat reprezentující digitální atlas člověka.

MSM Musculoskeletal Models - Muskuloskeletální modely

IOR Instituto Ortopedico Rizzoli - Ortopedický institut Rizzoli

VIPER Volume Invariant Position-based Elastic Rods - Objemově invariantní elastické tyče založené na pozici

PBP Position-based Dynamics - Algoritmy dynamiky založené na pohybu

RRA Residual Reduction Algorithm - Redukční reziduální algoritmus

CMC Computed Muscle Control - Vypočtená svalová kontrola

AWT Abstract Window Toolkit - Knihovna grafických uživatelských prvků pro platformu Java

SWIG Simplified Wrapper and Interface Generator - Softwarový nástroj používaný k propojení počítačových programů nebo knihoven napsaných v C nebo C ++ se skriptovacími jazyky

API Application Programming Interface - Rozhraní pro programování aplikací

XML Extensible Markup Language - Značkovací jazyk

SAX Simple API for XML - Algoritmus pro práci s XML dokumenty řízený událostmi

DOM Document Object Model - Algoritmus pro objektově orientovanou reprezentaci XML dokumentu

Seznam obrázků

A Uživatelská dokumentace

Instalace systému OpenSim společně s přidaným modulem grafického rozhraní je velice jednoduchá. Součástí odevzdaného archívu je instalátor OpenSim, `opensim-4.4-win64.exe` průvodce nastavením. Tento instalátor nainstaluje celý systém OpenSim na uživatelem zvolené místo společně s přidaným grafickým rozhraním a vším potřebným. Stačí pouze následovat kroky průvodce. Aplikace se následně spustí buď vytvořenou ikonou na ploše nebo ve složce `bin` pomocí `OpenSim64.exe`.

A.1 Ovládání programu

Po prvotním načtení systému se zobrazí tlačítko k nahrání dodatečných zdrojů pro OpenSim. Tento krok můžeme přeskočit, jelikož jsou již součástí instalátoru. Nyní, když máme otevřené grafické rozhraní, můžeme otevřít grafické rozhraní modulu. To otevřeme vybráním `Tools` z horní lišty nabídek a kliknutím na možnost `Muscle Wrapping GUI`. Kliknutím by se měl otevřít panel nástroje, stejný jako na Obrázku 6.12. Při prvním spuštění se ve složce `OpenSim 4.4-` vytvoří textový soubor `properties.txt`. Ten může uživatel editovat a měnit tím vlastnosti aplikace. Soubor nyní obsahuje tři vlastnosti a to URL adresu na tlačítko `Help`, zapisování cest do konfiguračních souborů (`relative` nebo `absolute`) a cestu k domovskou složce pro usnadnění práce.

Pomocí tlačítek `Select` zvolíme správné soubory (model - OSIM, motion - MOT, output - vytvořit nový OSIM) a pro algoritmus zvolíme buď *Luca2018* nebo *Cervenka2019*. Pomocí tlačítka plus nebo mínus manipulujeme s generátory svalu a pomocí tlačítka `Edit` modifikujeme vybraný generátor svalu. U generátoru tlačítkem `Select` zvolíme soubor svalu (OBJ nebo VTK). Referenční systém získáme z již načteného souboru s modelem. Pomocí `Scale factor` měníme jednotky míry. Tabulka funguje na stejném principu jako předešlá, akorát manipuluje s úponovými oblastmi generátoru svalu. *Fiber arrangement* určuje uspořádání svalů (náhodné - random nebo jednotné - uniform). `Numer of lines` určuje počet počet vláken (automaticky nebo libovolné číslo, v případě uniformního uspořádání se musí jednat o druhou mocninu čísla) a `Line Resolution` určuje rozlišení vláken. Pomocí `Edit` se dostaneme tentokrát na rozhraní úponové oblasti. Zde pomocí tlačítka `Select` můžeme vybrat soubor s body (značený I jako insertion nebo O jako origin formátu OBJ a VTK) nebo ručně přidat pomocí tabulky. `Load from`

určuje, zda má konfigurační soubor brát data z tabulky nebo ze souboru. `Type` může být buďto již zmínění počátek (origin) nebo oblast úponu (insertion). `Body`, stejně jako referenční systém z generátoru svalů přebírá data z souboru s modelem. Stejnou funkci jako u generátoru zde zastává i `Scale factor`. Poté co dokončíme nastavení, lze ho uložit pomocí tlačítka `Save` jakožto XML dokument. Ten můžeme pomocí `Load` i načíst. Tlačítkem `Run` spustíme plug-in automatického generování svalových vláken. Aplikace zobrazí výpis plug-inu a stav, ve kterém se momentálně nachází. Po úspěšném generování zobrazí hlášku a výpis schová, při neúspěšném vrátí návratovou hodnotu a nechá výpis zobrazený.

A.2 Limitace

Během práce jsem objevil několik limitací či nepříjemností, které jsou spojené jak se systémem OpenSim tak se samotným plug-inem. Jsou zde zmíněny z toho důvodu, aby uživatel věděl, že mohou nastat a jak se jich případně zbavit.

Při opakovaném spuštění OpenSim se může stát, že zmizí zabudované panely systému, jako je například `Navigator`, `Properties` nebo `Messages`. Vývojáři o problému ví a snaží se jej opravit. Dočasná pomoc je smazání složky `.OpenSim` v `C:\Users\Jmeno`. Tím donutíme systém vygenerovat rozhraní znovu.

Pokud uživatel spouští algoritmus *Cervenka2019*, může se stát, že plug-in vyhodí chybovou hlášku kvůli vytvořenému souboru `Cache.txt`, který se vytvoří ve složce jako je spuštěný konfigurační soubor. Jedná se o chybu plug-inu, stačí soubor smazat a plug-in pomocí grafického rozhraní spustit ještě jednou.

Další limitace je v umístění datových souborů. Konfigurační modul, který jsem vytvořil, podporuje jak absolutní, tak i relativní cesty. Bohužel, plug-in takovou funkci zatím neobsahuje a očekává, že data budou ve stejné složce jako je konfigurační soubor. Je tedy potřeba, aby se konfigurační soubor nacházel ve **stejné** složce jako data.

Nakonec je zde jedno upozornění. V některých případech běhu algoritmu může nastat, že vyskočí upozorňovací hláška. Tato hláška nabízí tři možnosti, přerušit, opakovat a ignorovat. Přerušením zrušíme generování, opakováním

ho naopak shodíme. Jelikož se jedná pouze o upozornění při ignorovaní, pokračuje algoritmus dál a vrátí validní výsledek.

B Popis adresářové struktury

- **Aplikace_a _knihovny** - obsahuje tři podsložky: `Installer`, `Source` a `Javadoc` společně s Readme návodem pro spuštění. `Installer` obsahuje instalátor systému OpenSim společně s konfiguračním modulem. Složka `Source` obsahuje všechny .java soubory. Dále se dělí na jednotlivé balíky (`actions`, `code`, `test`, `windows`). `Javadoc` obsahuje vygenerovanou dokumentaci.
- **Text_prace** - obsahuje všechny zdrojové soubory dokumentace
- **Vstupni_data** - obsahuje data potřebná k vytvoření konfiguračního XML dokumentu
- **Výsledky** - obsahuje konfigurační XML dokumenty vytvořené pomocí konfiguračního modulu.