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

Bakalářská práce

Nerigidní registrace povrchových svalových a šlachových snopců

Plzeň 2017 Martin Červenka

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: **2016/2017**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Martin ČERVENKA**
Osobní číslo:	**A14B0239P**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Informatika**
Název tématu:	**Nerigidní registrace povrchových svalových a šlachových snopců**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování:

1. Seznamte se s biomechanickou podstatou modelování architektury kosterních svalů.
2. Prozkoumejte stávající metody pro nerigidní registrace množin bodů.
3. Navrhněte automatickou příp. poloautomatickou metodu pro nerigidní registraci povrchových svalových a šlachových snopců reprezentovaných lomenými čárami s povrchovým modelem svalu reprezentovaným trojúhelníkovou sítí.
4. Navrženou metodu naimplementujte a otestujte na reálných datech z projektu LHDL (Living Human Digital Library). Dosažené výsledky zhodnoťte.

Rozsah grafických prací: **dle potřeby**

Rozsah kvalifikační práce: **doporuč. 30 s. původního textu**

Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

dodá vedoucí bakalářské práce

Vedoucí bakalářské práce: **Doc. Ing. Josef Kohout, Ph.D.**
Katedra informatiky a výpočetní techniky

Datum zadání bakalářské práce: **10. října 2016**

Termín odevzdání bakalářské práce: **4. května 2017**

Doc. RNDr. Miroslav Lávička, Ph.D.
děkan

L.S.

Doc. Ing. Přemysl Brada, MSc. PhD.
vedoucí katedry

V Plzni dne 13. října 2016

Prohlášení

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

V Plzni dne 2. května 2017 Martin Červenka

Poděkování

Tímto bych chtěl poděkovat panu Doc. Ing. Josefu Kohoutovi, Ph.D. za konstruktivní nápady, užitečné rady a věcné připomínky, které významnou měrou pozitivně přispěly ke vzniku této práce. Dále bych rád poděkoval své rodině za podporu poskytovanou během psaní tohoto dokumentu.

Abstract

This thesis targets non-rigid registration of superficial muscle tendon fascicles problem. The registration is required for existing method which solves muscle model decomposition into muscle fibres problem. Registration algorithm by Hao Li is chosen, which targets non-rigid registration of depth scans. Follows algorithm modifications to make algorithm able to register superficial muscle tendon fascicles represented by polylines on three-dimensional muscle surface instead of depth scans registering. Tests are executed on both artifical and real data and it is shown that designed algorithm can do registration in reasonable time, otherwise it is shown that the algorithm cannot be used fully automatic, because during registration there are few inaccuracies, which need to be subsequently fixed.

Abstrakt

Cílem této práce je provedení nerigidní registrace povrchových svalových a šlachových snopců, která je potřebná pro již existující řešení problému dekompozice svalu na svalová vlákna. Je zvolen registrační algoritmus navržený Hao Li, který provádí nerigidní registrace hloubkových skenů. Dále je algoritmus modifikován tak, aby dokázal registrovat místo hloubkových skenů povrchové svalové snopce reprezentované lomenými čárami na povrch trojrozměrného modelu svalu. Výsledky jsou otestovány na umělých i reálných datech a je ukázáno, že navržený algoritmus sice dokáže v rozumném čase provést registraci, také se ale zjistí, že navržený algoritmus nelze použít plně automaticky, protože během registrace vzniká několik málo nepřesností, které je třeba následně opravit.

Obsah

1. Úvod

Při léčbě různých onemocnění se lékaři bez základní znalosti anatomie těla neobejdou. Není tomu jinak ani v případě anatomie svalů a šlach. Většina pozorování je provedena na snímcích z ultrazvuku nebo dvojrozměrném modelu, je tedy zatížena určitou nepřesností oproti skutečnosti. Zmíněná nepřesnost může být snížena použitím 3D modelu, který zachycuje více souvislostí než 2D snímky či modely.

Zatímco 3D povrchový model svalu lze relativně snadno získat neinvazivní cestou z medicínských snímků, vnitřní struktura svalu na těchto snímcích není patrna[2].

Na Katedře informatiky a výpočetní techniky Západočeské univerzity v Plzni byla vyvinuta metoda[5], která dokáže aproximovat vnitřní architekturu svalu na základě znalostí 3D povrchového modelu svalu a modelu povrchových snopců, které lze získat neinvazivně např. z ultrazvukového vyšetření.

Poskytnutá data však pochází z různých měření, během kterých jsou svaly odlišně deformovány. Proto model povrchových snopců neleží přesně na modelu povrchu svalu a je tedy potřeba poskytnutá data registrovat. A právě problematikou nerigidní registrace povrchových svalových a šlachových snopců se zabývá tato práce.

V následující kapitole je ve stručnosti popsáno, jak sval pracuje a z čeho se skládá nebo například jaké důležité složky ovlivňuji sílu svalu. Také se zde píše, jakými způsoby a s jakými výsledky je v dnešní době možno provádět měření svalu. V následující kapitole 3 jsou popsány některé z dnes používaných algoritmů registrace množin bodů. V kapitole 4 je pak uveden návrh řešení problému registrace. Dále následují informace o implementaci programu v kapitole 5. Nakonec byla aplikace otestována na několika sadách dat a výsledky tohoto testování jsou popsány v kapitole 6.

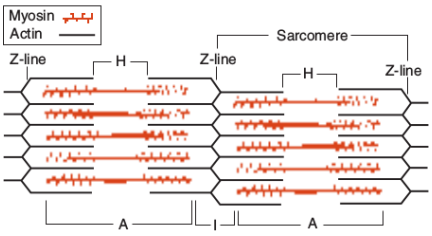
2. Anatomie a fyziologie svalu

Cílem této kapitoly je stručně nastínit anatomii a fyziologii svalu. Tyto informace jsou nezbytné k pochopení účelu provádění registrace povrchových snopců na povrch svalu.

2.1 Struktura vlákna

Sval je složen z vláken. Počet vláken a jejich směr se mění sval od svalu. Standardní způsob, jakým se určuje počet vláken, je dle plochy fyziologického příčného řezu (PCSA - physiological cross-sectional area) a je to plocha řezu, který prochází kolmo skrz všechna vlákna svalu. Vlákna nemusejí být ve svalu jen rovnoběžně se směrem působení výsledné síly, ale mohou být umístěna i šikmo. Sval se zpeřenými (šikmými) vlákny se vyznačuje tím, že dokáže vyvinout větší moment síly, naproti tomu má menší schopnost pohybu než sval s rovnoběžnými vlákny.

Základním stavebním kamenem každého svalového vlákna je sarkomera. Ta se skládá z proteinů aktinu a myosinu (viz obr. 2.1[1]).



Obrázek 2.1: Vnitřní struktura sarkomery. [1]

Základ pohybu svalového vlákna je tvořen posouváním řetězce myosinu po aktinu v celém svalovém vlákně, a to způsobuje prodlužování a zkracování vlákna, posléze i svalu. Samotný posun řetězce je inicializován nervovým impulzem.

2.2 Uchycení svalu

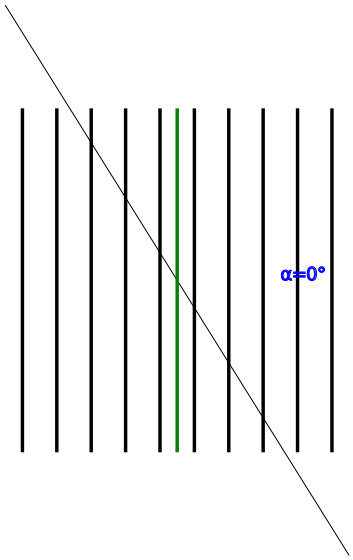
Svalová vlákna jsou na svých koncích přichycena k aponeuróze nebo šlaše, které jsou dále připojeny ke kostem. Zkracováním svalových vláken se přenáší působení síly ze svalu přes aponeurózy nebo šlachy na kosti a tím je tvořen vlastní pohyb kostí.

2.3 Zpeřený sval

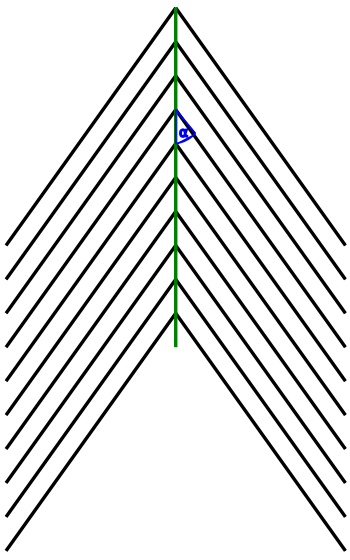
Pokud sval nemá vlákna rozložena rovnoběžně s působením síly svalu, pak se takovýto sval nazývá zpeřený. U zpeřeného svalu je nejdůležitějším parametrem úhel zpeření, to je úhel mezi směrem vláken a působením síly. Sílu celého svalu lze pak určit součtem rovnoběžných složek sil jednotlivých vláken. Tuto rovnoběžnou složku lze vypočítat funkcí kosinus z úhlu zpeření jednotlivých vláken. Tento výpočet je uveden v 2.1[2].

$$f_t = \sum_i f_m^i \cdot cos(\alpha_i) \tag{2.1}$$

Rozdíl vnitřní struktury mezi zpeřeným a paralelním svalem je ukázán na obrázcích 2.2 a 2.3. U paralelního svalu je úhel α roven nule, zatímco u zpeřeného svalu je větší než nula. Reálně tento úhel u zpeřených svalů zpravidla nepřekračuje hodnotu 35°[2].



Obrázek 2.2: Paralelní sval.



Obrázek 2.3: Zpeřený sval.

Protože je tento úhel jedním z klíčových parametrů ke stanovení celkové síly svalu, nestačí k determinaci celkové síly svalu jeho vnější tvar a rozměry, ale je potřeba znát i vnitřní strukturu svalu.

2.4 Síla svalu

Když je známa základní struktura svalu, lze z ní logicky odvodit základní faktory určující sílu svalu. Mezi hlavní faktory tedy patří:

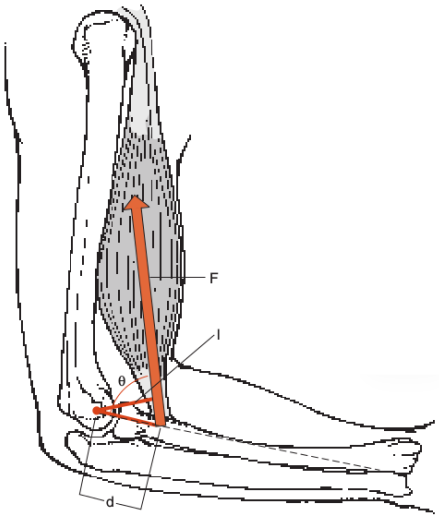
- Velikost svalu
- Délka ramena síly
- Zpeření svalu

2.4.1 Velikost svalu

Jedním z hlavních faktorů ovlivňujících sílu svalu je jeho velikost. Čím je sval větší, tím více obsahuje vazeb aktinu a myosinu a sval dokáže vyvinout větší sílu, protože síla kontrakce závisí na počtu vazeb mezi řetězci aktinu a myosinu.

2.4.2 Rameno a moment síly

Dalším faktorem, který ovlivňuje sílu svalu, je délka ramena síly (viz obr. 2.4[1]).



Obrázek 2.4: Délka ramena síly.[1]

Jak sval dokáže pohybovat kostmi je již zmíněno v sekci 2.2, není ale zmíněno, jak se mohou kosti pohybovat v těle.

Většina pohybů svalů a kostí v těle je převáděna na rotační pohyb se středem rotace v kloubu, kterým sval otáčí. Schopnost svalu pohybovat kloubem závisí na síle kontrakce svalu a na vzdálenosti vektoru síly od kloubu (jakožto středu rotace). Vzdálenost vektoru síly od středu rotace se nazývá délka ramena síly. Čím je rameno síly větší, tím větší moment může být vytvořen kontrakcí svalu, zároveň se tím ale negativně ovlivní rozsah pohybu kosti.

2.5 Měření svalů a povrchových snopců

Je tedy patrné, že určení působících sil ve svalu není vůbec jednoduchá záležitost. Navíc protože každý jedinec může mít odlišnou stavbu těla, není vhodné používat univerzální model, ale je potřeba provést konkrétní měření na daném jedinci.

Měření dat lze provádět dvěma základními postupy. Prvním je měření neinvazivní na živé osobě bez jakéhokoliv poškození tkáně, například ultrazvukem nebo magnetickou rezonancí, druhá možnost je měření invazivní například disekcí svalu mrtvého člověka. Měření se provádí také kvůli zjištění úhlů zpeření vláken ve svalu, pro tento parametr je ale potřeba znát vnitřní stavbu svalu. Jednotlivé měření tedy necílí jen na pouhé povrchové měření svalu, je potřeba prozkoumat sval i uvnitř.

2.5.1 Měření ultrazvukem

Neinvazivní měření lze provádět například ultrazvukem, výsledkem je pak 2D snímek svalu. Tato metoda měření je rozšířena hlavně kvůli své neinvazivnosti, jednoduchosti měření a možnosti měření přirozených pohybů svalu. Bohužel ale 2D snímek nedokáže dostatečně popsat interní strukturu svalu, proto je cílem dosáhnout maximální viditelnosti vláken na snímku.

2.5.2 Měření magnetickou rezonancí

Jako další neinvazivní formu měření lze použít magnetickou rezonanci. Pomocí tohoto měření se získá 3D matice hodnot, ze kterých se segmentací získá část matice reprezentující sval. Svalový segment se dále postoupí algoritmu extrakce povrchu, kterým se získá hrubý povrch svalu. Posledním krokem je vyhlazení, které může probíhat automatickým algoritmem nebo ručními úpravami. Výhodou tohoto přístupu je získání celého povrchu svalu, nevýhodou naopak je složitější získávání dat, protože je zapotřebí provést segmentaci a vyhlazení a na to je třeba použít složitější algoritmy nebo tyto akce musí provést člověk. Bohužel ale ani tato metoda nedokáže spolehlivě popsat vnitřní strukturu svalu.

2.5.3 Disekce

Invazivním měřením lze získat komplexnější data o svalu než z 2D snímku či 3D povrchu. Na vzorku tkáně lze přesněji měřit nežli na snímku, lze měřit úhly zpeření v jakékoliv rovině a k tomu je zajištěno, že se sval během měření nepohne. Nehybnost svalu je však i nevýhodou tohoto postupu. Není zde provedeno měření reálného stavu svalu (s reálným upevněním, v reálné zátěži atd.), pouze svalu ve stavu, ve kterém se v lidském těle většinou nenachází. Můžeme měřit dokonce sval, který již není upevněn, tato situace je většinou řešena simulovaným upevněním svalu, i tak ale dochází k jisté nepřesnosti měření oproti měření reálného svalu. Hlavní nevýhodou této metody je, že nemůžeme tímto způsobem měřit sval určitého jedince, jen můžeme vytvořit univerzální model. Jak je již zmíněno výše v této sekci 2.5, metoda aproximace univerzálním modelem není ve většině případů dostačující.

2.5.4 Naměřená data

Předem naměřená data k předzpracování byla naměřena dvěma ze tří výše zmíněných způsobů – magnetickou rezonancí a disekcí. Protože neexistuje neinvazivní metoda, která by spolehlivě určila vnitřní stavbu svalu, navrhl D. Cholt postup[5], jak vnitřní strukturu získat z povrchových dat. Tento postup však vyžaduje mít naměřená data o povrchu svalu a povrchová vlákna. Povrchový model svalu lze získat neinvazivně, měření povrchových vláken tímto způsobem však způsobuje nepřesnosti. Proto jsou povrchová vlákna naměřena disekcí a budou sloužit jako univerzální model vláken pro personalizovaný povrchový model svalu.

V této práci má každý sval naměřen svůj tvar (neinvazivně – magnetickou rezonancí) a pozice některých vláken na povrchu svalu (invazivně – disekcí). Z důvodů uvedených výše v předchozích sekcích naměřená vlákna neleží přesně na naměřeném povrchu svalu. Algoritmus pro aproximaci vnitřní struktury svalu od D. Cholta[5] však vyžaduje, aby vlákna na povrchu ležela, z čehož plyne potřeba provést registraci povrchových svalových snopců na povrch svalu. Následující kapitola ukáže možné postupy, jakým způsobem lze data registrovat.

3. Registrace množin bodů

V předchozí kapitole byly uvedeny možnosti měření. Tato práce se dále zaměří na registraci dvou množin bodů, kde jedna množina je naměřena magnetickou rezonancí (povrchový model svalu) a druhá množina je naměřena disekcí (vlákna na povrchu svalu). Registrace množin bodů je proces hledání prostorové transformace, která nejlépe srovná dvě množiny bodů[3]. Nejdůležitějším cílem registrace je přiblížit k sobě body obou množin, tím přiblížením je myšlena například minimalizace součtu vzdáleností dvojic nejbližších bodů.

3.1 Dělení algoritmů

Možnosti dělení algoritmu jsou na dělení dle pevnosti – rigidní (pevné) a nerigidní – a dělení na automatické, poloautomatické a ruční. Pokud zachováme tvar přesouvaného objektu, jedná se o pevnou (rigidní) transformaci, v opačném případě se jedná o transformaci nerigidní. V rigidních je na všechny body aplikována stejná transformace, zatímco v nerigidním případě tomu v obecném případě tak být nemusí, zde má každý bod svou vlastní transformaci. Většina efektivních algoritmů nerigidních registrací používá v počáteční fázi rigidní registraci, kdy se celý objekt registruje na vzor s velkou chybou a poté se aplikuje nerigidní transformace jednotlivých bodů, která provede nezávislou registraci jednotlivých bodů a výsledkem je menší chyba za cenu změny tvaru registrovaného objektu.

Výběr vhodného algoritmu také závisí na logických vlastnostech registrovaného objektu. Pokud nejsou předem známy vlastnosti registrovaného objektu, lze použít automatickou metodu nerigidní registrace. Naopak pokud zde jsou nějaké důležité vlastnosti, které je potřeba respektovat (např. část objektu je krychle), pak je vhodné zkombinovat s automatickou metodou i metodu ruční registrace – poloautomatická metoda.

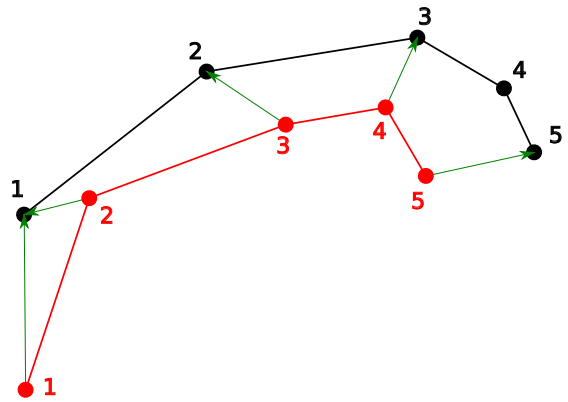
V následujících podsekcích budou popsány algoritmy jak rigidních, tak nerigidních transformací a jejich pozitivní a negativní vlastnosti.

3.1.1 Manuální registrace dat

U jednorázové transformace s malým počtem bodů uplatníme manuální registraci dat. Taková registrace se provádí ručním přesunem všech bodů dle daného vzoru na počítači v nějakém programovém vybavení. V případě registrace dat o svalech je potřeba, aby registraci prováděl odborník na anatomii těla a zároveň člověk s mírně pokročilou počítačovou gramotností, protože pro přesun bude muset používat daný software. Navíc každý odborník takovou registraci provede mírně odlišně. Tento postup je tedy nepřesný, záleží na subjektivním pohledu člověka, zbytečně plýtvá časem odborníků a nelze ho využít na větší sadu dat, kde se lépe uplatní automatizované nebo alespoň poloautomatizované algoritmy registrace dat. Tento přístup je však vhodný použít, pokud automatický algoritmus selže. Selháním automatického algoritmu může být myšlen například nějaký singulární případ, kdy nerigidní transformace nebude fungovat nebo algoritmus sice objekt registruje ve smyslu přiblížení dvou objektů, opomene však zachování některé vlastnosti registrovaného objektu, které musí být z logické podstaty registrovaného objektu zachovány.

3.1.2 Metoda nejbližšího souseda

Nejjednodušším algoritmem nerigidní transformace pro registraci množin bodů je metoda nejbližšího souseda. Algoritmus pro každý bod z množiny pohybujících se bodů najde jiný bod z množiny statických bodů a nastaví si jeho souřadnice. Bod, ze kterého si pohybovaný bod přebere souřadnice, si vybere jako první nejbližší dle použité metriky. Ilustrace jednoduché registrace pomocí této metody je na obrázku 3.1. Body označené červeně jsou body, které se registrují a budou se přesouvat na pozice nejbližších, černě označených, vzorových bodů ve směru zelených šipek.



Obrázek 3.1: Ilustrace metody nejbližšího souseda. Červené body se registrují na některé z černých dle zelených šipek.

Mezi výhody algoritmu patří jednoduchá implementace a fakt, že součet vzdáleností dvou nejbližších bodů bude po dokončení algoritmu nulový, dosáhneme tím tedy stoprocentní registrace.

Jednou z nevýhod metody je vlastnost, že ignoruje souvislost mezi body, protože bere každý bod nezávisle na okolí. Další nevýhodou je, že metoda nedokáže

odhalit nepřesnosti měření (tzv. outliers, česky „mimolehlé body“).

Metoda se tedy využije v případech, kdy spolu body nijak nesouvisí nebo například v koncové části registrace, kdy víme, že přesuny bodů budou jen nepatrné a zároveň je nutné dosáhnout stoprocentní registrace bodů.

Jak metoda funguje je popsáno v algoritmu 3.1. Jedná se o velmi jednoduchý postup, při kterém vnější cyklus prochází všemi pohybujícími se body a vnitřní cyklus hledá pro každý přesouvaný bod nejbližší statický bod. Proměnná M zastupuje množinu bodů, se kterými se bude hýbat, naopak proměnná S zastupuje množinu bodů, ke kterým se bude množina M registrovat.

```
Algoritmus NejblizsiSoused(M,S):
        for m_i ∈ M:
                X := 0
                for s_j ∈ S:
                        if ‖m_i, s_j‖ < ‖m_i, s_x‖
                                X=J;
                m_i := s_x
```

Listing 3.1: Algoritmus nejbližšího souseda

3.1.3 ICP

ICP (iterative closest point) je metoda rigidní transformace, která se snaží minimalizovat celkovou vzdálenost dvojic nejbližších bodů dvou objektů při zachování tvaru objektu. Protože na začátku není jasné, o které dvojice se bude jednat, metoda, jak již z jejího názvu vyplývá, řeší problém iteračně. Metoda si primárně neklade za cíl transformovat body, jejím hlavním cílem je nalézt optimální transformaci. Výstupem algoritmu pak tedy může být transformovaná množina bodů nebo transformace samotná. Transformace může být obecně jakákoliv afinní, dle konkrétních požadavků mohou být zakázány transformace typu zvětšování, zmenšování, zkosení objektu atd.

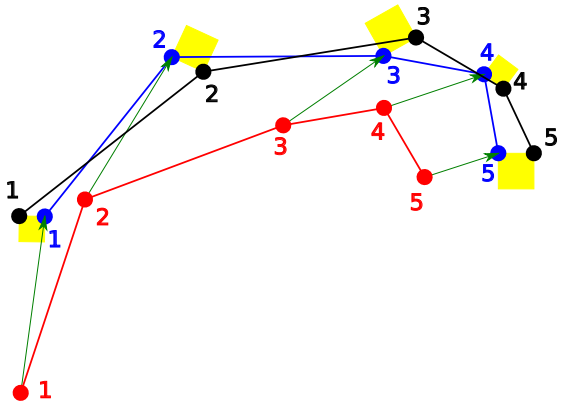
Jak metoda funguje je ilustrováno obrázkem 3.2. Vzorová množina bodů je označena černě, registrovaná červeně. Výsledkem (v optimálním případě, viz dále) bude přesun registrovaných bodů z červeně označených pozic po zelených šipkách na pozice značeny modře. V tomto ilustračním případě by se snažil algoritmus minimalizovat žlutou plochu, což je součet kvadrátů vzdáleností nejbližších bodů vždy jednoho vzorového a jednoho cílového bodu.

Výhoda metody oproti metodě nejbližšího souseda (viz předchozí sekce 3.1.2) je zachování tvaru objektu. S body se pracuje jako s jedním objektem a s ním lze pouze rotovat, přesouvat, kosit jej nebo měnit jeho měřítko.

Protože metoda řeší problém iteračně, lze konstatovat, že metoda přesné řešení najít nemusí, pouze se k němu může blížit, dokonce ani konvergovat do globálního minima nemusí. Metoda ve většině případů neregistruje body přesně – to se stane pouze v případě dvou stejných, různě afinně transformovaných objektů a pokud algoritmus dosáhne globálního maxima.

Z důvodu, že tato metoda je rigidní, je algoritmus vhodné použít v počáteční fázi registrace, kdy je požadovanou vlastností zmenšit vzdálenosti bodů a až následně použít nějakou z metod nerigidní registrace.

Jak tato metoda funguje je vidět v pseudokódu 3.2 [3]. Parametr S značí



Obrázek 3.2: Ilustrace metody ICP. Červená množina bodů se registruje na modrou dle černých (vzorových) bodů.

množinu bodů, ke kterým transformujeme body z množiny M, výsledná transformace je značena θ.

```
Algoritmus ICP(M,S):
        θ := θ_0
        while not registrovano:
                X := ∅
                for m_i ∈ T(M,θ):
                        s_j := nejblizsi bod k m_i z S
                        X := X + <m_i, s_j>
                θ := metoda nejmensich ctvercu(X)
        return θ
```

Listing 3.2: ICP algoritmus[3]

Transformační matice θ se v každém kroku spočítá metodou nejmenších čtverců tak, aby se minimalizovala vzdálenost mezi nejbližšími body a poté je použita jako transformace pro množinu M v dalším kroku. Metoda však počítá pouze pevnou transformaci celého objektu, takže objekt může pouze rotovat nebo se přesouvat – jedná se o tzv. afinní transformaci.

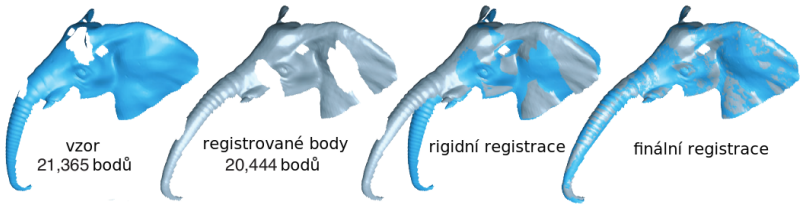
3.1.4 Hao Li algoritmus

Jedná se o komplexnější algoritmus nerigidní registrace množin bodů než ICP (viz sekce 3.1.3). Algoritmus pracuje stejně jako předchozí algoritmy na principu minimalizace, neminimalizuje však pouze vzdálenosti bodů, ale snaží se zároveň minimalizovat vzájemný pohyb bodů a dle autora není citlivý na mimolehlé body, tzv. outliers.

Základní princip algoritmu je současná minimalizace chyb čtyř proměnných – tuhosti, hladkosti, vhodnosti a důvěryhodnosti. Tuhost definuje, jak moc se každá individuální transformace liší od jednotkové matice, čímž penalizuje nerigidní transformace, hladkost penalizuje rozdílné sousední lokální transformace, a to způsobuje postupnou změnu tvaru napříč registrovaným objektem. Proměnná vhodnosti určuje, jak moc jsou registrované body aktuálně vzdáleny od vzorových bodů, a to dělá z této penalizace jedinou, která se zabývá i množinou vzorových bodů. Nakonec důvěryhodnost penalizuje mimolehlé body. Transformace se zde, jak již bylo naznačeno, dělí na dvě skupiny – transformace celého objektu a transformace jednotlivých bodů. Matice rotace celého objektu se značí R a je velikosti 3x3 prvky, vektor přesunu se značí t a obsahuje tři prvky. Podobně je tomu u transformace jednotlivých bodů, tam je značení matice A_i a vektor přesunu b_i. Jednotlivé body, které budou transformovány, jsou značeny x_i.

Algoritmus je původně navržen na registraci hloubkových skenů. V případě, že na vstupu bude jiný druh dat, algoritmus buď nebude fungovat, nebo bude potřebovat nějaké úpravy nebo předzpracování vstupních dat.

Vstup a výstup algoritmu Hao Li je naznačen na obrázku 3.3. V tomto případě jsou registrovány hloubkové skeny slůněte s různými pozicemi chobotu.



Obrázek 3.3: Vstup a výstup algoritmu Hao Li (převzato z [4], upraveno).

Tuhost

Chyba tuhosti se zvětšuje při nerigidních transformacích bodů. Penalizuje se zde jakákoliv nerigidní transformace a tato penalizace je dána vzorcem 3.1.

$$E_{rigid} = \sum_i Rot(A_i) \tag{3.1}$$

E_{rigid} je celková penalizace za nerigiditu a jedná se o součet funkcí *Rot* matic transformací (A_i) všech bodů z množiny pohybovaných bodů. Funkce *Rot* je dále definována vztahem 3.2.

$$Rot(A) = (a_1 \cdot a_2)^2 + (a_1 \cdot a_3)^2 + (a_2 \cdot a_3)^2 + (1 - a_1 \cdot a_1)^2 + (1 - a_2 \cdot a_2)^2 + (1 - a_3 \cdot a_3)^2 \tag{3.2}$$

Ve vzorci 3.2 je parametrem funkce *Rot* matice o rozměru 3x3 – transformační matice, a_i je $i$tý sloupec transformační matice. Je zřejmé, že nulového výsledku lze dosáhnout pouze jednotkovou maticí, jakákoliv jiná matice má hodnotu větší než nula – proto je chyba kladná při použití jakékoliv jiné transformace jednotlivých bodů s výjimkou identického zobrazení.

Hladkost

Hladkost udává rozdíl transformací v blízkém okolí jednoho bodu – při větším rozdílu transformací se zvětšuje chyba hladkosti. Vzorec pro výpočet je uveden v 3.3.

$$E_{smooth} = \sum_i \sum_{j \in N(i)} ||A_i(x_j - x_i) + x_i + b_i - (x_j + b_j)||^2 \tag{3.3}$$

$N(i)$ ve vzorci 3.3 značí množinu všech sousedních bodů bodu i. Nulovou hodnotu této penalizace získáme, pokud:

$$\forall i, j \in N(i) : A_i(x_j - x_i) + x_i + b_i - (x_j + b_j) = 0$$

Z toho lze usoudit, že získáme nulovou hodnotu, pokud je transformace všech sousedních bodů shodná, z čehož tranzitivně vyplývá, že všechny individuální transformace se musejí rovnat (v případě úplného vynulování chyby hladkosti), a v tom případě je výsledkem shodná transformace pro každý bod, kterou lze reprezentovat rigidní registrací. Důsledkem je, že chyba hladkosti penalizuje jakoukoliv změnu tvaru stejně jako chyba tuhosti, cílí však více lokálně na porovnání sousedních transformací.

Vhodnost

Chyba vhodnosti je v podstatě součet vzdáleností registrovaných bodů od bodů vzorových, kterou se předchozí algoritmy snaží minimalizovat, aby se registrovaný objekt co nejvíce přiblížil k vzorovému objektu, jenže v tomto případě je každá vzdálenost ještě vynásobena váhou, a to umožňuje různé vzdálenosti v různých místech s jinými penalizacemi. Vzorec pro výpočet chyby vhodnosti je uveden v rovnici 3.4.

$$E_{fit} = \sum_{i=1}^{n} \omega_i^2 ||\tilde{x}_i - c(\tilde{u}_i)||^2 \tag{3.4}$$

Parametr $\omega_i \in \langle 0; 1 \rangle$ je váha každého bodu, její význam je popsán v následující sekci 3.1.4, proměnná $\tilde{x}_i$ je transformovaný bod.

Proměnná $\tilde{u}_i$ je parametrickým vyjádřením cílové pozice registrovaného bodu. Parametrické vyjádření je zde použito k omezení schopnosti registrovaných objektů volně se hýbat. Konkrétně se jedná o dvojrozměrnou parametrizaci trojrozměrného prostoru, tzn. dvě souřadnice jsou libovolné a třetí se dopočítá.

Funkce $c(\tilde{u}_i)$ provádí převod z parametrické reprezentace bodu na reálné souřadnice. Základní myšlenkou algoritmu pracujícího s hloukovými skeny je, že lze provést parametrizaci z 2D prostoru do 3D prostoru. Protože původní data, se kterými se má pracovat, jsou hloubkové skeny, tak mají ve dvou ze tří dimenzí uniformní vzdálenosti (po zobrazení do 2D prostoru vznikne tvar mřížky) a lze je tedy reprezentovat dvojrozměrnou maticí, kde každý prvek matice obsahuje souřadnici třetí dimenze. Parametrizace z 2D do 3D se tedy provede jednoduchým zkopírováním dvou složek a třetí složka se získá z příslušného prvku matice hloubkového skenu.

Důvěryhodnost

Poslední chyba je chyba důvěryhodnosti. Důvěryhodností je zde myšleno, jak moc se u každého bodu nejedná o mimolehlý bod. Chyba spolupracuje s chybou vhodnosti a vynucuje zvyšování vah ω_i u všech bodů. Pokud je váha bodu nulová, vynuluje se chyba vhodnosti, ale chyba důvěryhodnosti bude 1. Parametr ω_i je tedy optimalizačním parametrem, který se za běhu algoritmu mění. Čím více se hodnota blíží jedné, tím klesá hodnota chyby důvěryhodnosti, ale může růst hodnota chyby vhodnosti. Pokud by hodnota chyby vhodnosti rostla více, než hodnota chyby důvěryhodnosti, ω_i se bude v dalším běhu algoritmu zmenšovat. Vzorec pro výpočet chyby důvěryhodnosti je uveden v 3.5.

$$E_{conf} = \sum_{i=1}^{n} (1 - \omega_i^2)^2 \tag{3.5}$$

Celková chyba

Celková chyba je váženým součtem všech čtyř předchozích chyb. Vzorec pro výpočet chyby je napsán v 3.6.

$$E = \alpha_{rigid}E_{rigid} + \alpha_{smooth}E_{smooth} + \alpha_{fit}E_{fit} + \alpha_{conf}E_{conf} \tag{3.6}$$

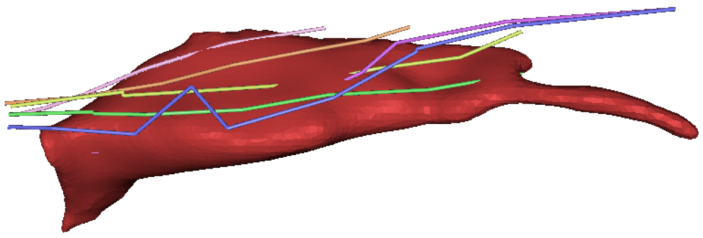
Proměnné α zastupují váhy jednotlivých chyb. Cílem algoritmu je minimalizace celkové chyby, tedy proměnné E. Řešením je po dosazení jednotlivých chyb soustava nelineárních rovnic, která (kromě singulárních případů) nemá řešení. Pak se použije Levenberg-Marquardův[6] algoritmus na řešení soustavy rovnic pomocí nejmenších čtverců pro minimalizaci proměnné E. Nejprve na začátku algoritmu určíme počáteční hodnoty vah (v originálním algoritmu[4] $\alpha_{rigid} = 1000$, $\alpha_{smooth} = \alpha_{confidence} = 100$ a $\alpha_{correspondence} = 0.1$). Algoritmus pak pracuje iteračně a v každém kroku se optimalizují transformace a proměnné ω_i pomocí metody nejmenších čtverců. Nedokáže-li již algoritmus efektivně optimalizovat (rozdíl chyb po sobě jdoucích iterací je menší než předem zvolený práh), rozpůlí se hodnoty α_{rigid}, α_{smooth} a $\alpha_{confidence}$ tak, aby na počátku převažovaly váhy pro zachování tvaru registrovaného objektu a tím proběhla rigidní registrace, zatímco v dalších iteracích začne mít větší váhu penalizace vhodnosti, takže bude docházet více k nerigidní registraci. Algoritmus končí, když $\alpha_{confidence}$ klesne pod hodnotu 1, takže proběhne sedmkrát. Výsledkem algoritmu jsou: matice R a A_i a vektory t a b_i určující všechny potřebné transformace.

4. Návrh řešení

V předchozích sekcích byl v obecnosti popsán problém a možnosti, jakými se dá řešit. Nyní se tato práce zaměří na konkrétní problém. To znamená určení dat a dle dat výběr vhodného algoritmu a případná úprava algoritmu, aby fungoval pro konkrétní poskytnutá data.

4.1 Vstupní data

Vstupní data jsou dvojího druhu – uzavřená trojúhelníková síť reprezentující povrchový model svalu a množina lomených čar reprezentující povrchová vlákna. Příklad lze vidět na obrázku 4.1.



Obrázek 4.1: Příklad vstupních dat[5].

Na tomto obrázku lze vidět i různé nepřesnosti měření. Opomene-li skutečnost, že reprezentace reálného vlákna lomenými čarami je jen velmi hrubou aproximací, tak vlákna ve většině případů neleží přímo na povrchu svalu, lomí se někdy výrazněji, než je realistické apod. Kvůli těmto nepřesnostem lze říci, že bude vhodné registrovat vlákna k modelu a ne opačně, protože povrchový model je přesnější, než-li model vláken.

Dalším důvodem je, že povrchová vlákna jsou naměřena univerzálně, protože je nelze dobře měřit neinvazivně (viz 2.5), tudíž nelze měřit na konkrétním jedinci. Zatímco povrchový model svalu už může být personalizovaný, měřený na konkrétním jedinci, tzn. pro konkrétní problém budou přesnější (další podrobnosti v sekci 2.5.4).

Na vstup může přijít povrchový model a vlákna jakéhokoliv svalu, proto nelze předpokládat nějakou vlastnost konkrétního svalu, protože svaly mají většinou velmi odlišné tvary. Povrchová vlákna jsou navíc měřena na povrchu jedné poloviny svalu. Nelze zde vůbec předjímat, o kterou polovinu se bude jednat.

4.2 Výběr algoritmu

V předchozí sekci o vstupních datech je uvedeno, že ve vstupních datech se nachází různé druhy svalů, pro které nejsou předem známy žádné vlastnosti. Proto (viz. sekce 3.1) bude vhodné použít automatický algoritmus registrace.

Vzhledem k tomu, že požadavek na registraci je, aby povrchová vlákna přesně ležela na povrchu svalu[5], nelze použít samotnou rigidní metodu registrace. Z kapitoly 2.5 (kde je popsáno, jakým způsobem byla pořízena vstupní data) lze usoudit, že v datech bude mnoho mimolehlých bodů a vzdálenosti korespondujících bodů budou velké. Z toho důvodu lze vyloučit použití metody nejbližšího souseda k řešení daného problému. Nabízí se tedy řešení problému algoritmem navrženým Hao Li[4], který kombinuje oba přístupy.

4.3 Úpravy algoritmu

V sekci 4.1 jsou popsány vlastnosti vstupních dat. Lze si všimnout, že na vstupu algoritmu Hao Li jsou data reprezentována jinak, než je potřeba. Použití existující Hao Li metody registrace tedy bude vyžadovat několik zásadních změn. Místo toho, aby jako vstupní data byly použity snímky hloubkového skenu, použije se jako vzor trojúhelníková síť a jako transformovaný objekt množina lomených čar. Je tedy potřeba projít nejprve všechny vzorce algoritmu a určit problémy, ke kterým může u té které rovnice dojít.

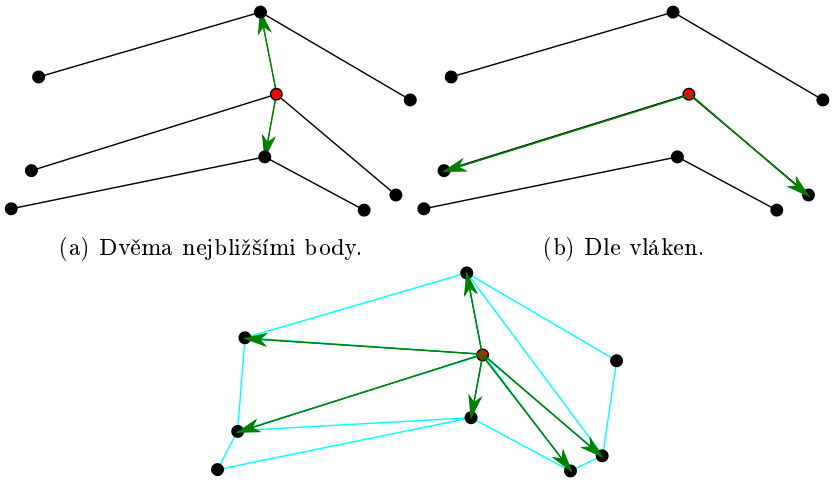
4.3.1 Chyba hladkosti

Prvním problémem se zdá být výpočet hladkosti ve vzorci 3.3. Zde se pro každý bod prochází seznam sousedících bodů. V původním algoritmu se řešila sousednost jako sousední body mřížky hloubkového skenu, tento přístup však zde nepůjde použít.

Řešením může být ignorování vazeb vláken a použití množiny vláken jen jako množiny nezávislých bodů, stejně jako je tomu v původním algoritmu, a pro každý bod vybrat jako sousedící body n nejbližších bodů, jenže tím můžeme příliš poškodit tvar jednotlivých vláken.

Další možností je použití množiny lomených čar, protože každý bod v lomené čáře sousedí se dvěma (uprostřed vlákna), jedním (okraj vlákna) nebo žádným (extrémní případ jednobodového „vlákna“, nelze ale říci, že ve vstupních datech se nevyskytne) bodem. Vlákna by pak ale na sobě byla nezávislá.

Lepším řešením je provést triangulaci vláken se zachováním již existujících hran (vláken) – to umožní zvýšení počtu sousedů a zároveň se pohyb vláken bude hodnotit v závislosti na tom, jak se pohybují vlákna přímo sousedící – tímto způsobem byla chyba hladkosti původně navržena, aby respektovala přímo sousedící entity, pouze v původním algoritmu je entitou bod, zatímco zde je entitou celé vlákno.



(a) Dvěma nejbližšími body.

(b) Dle vláken.

(c) Triangulací.

Obrázek 4.2: Různé možnosti definice sousednosti. S červeným bodem sousedí všechny body, do kterých vede zelená šipka.

Obrázek 4.2 ukazuje tyto různé přístupy k determinaci sousedů. Nejlepší definici sousednosti slibuje triangulace, proto bude vhodné ji použít.

4.3.2 Chyba vhodnosti

Dalším problémem je výpočet vhodnosti. Zde byl předpoklad, že dokážeme provést mapování z 2D prostoru do 3D prostoru. Toto mapování, jak již bylo uvedeno v 3.1.4, lze na hloubkových skenech snadno provést. V případě 3D modelu svalu bude potřeba použít jinou formu parametrizace.

Parametrizace indexací

Nejjednodušší parametrizací je mapování z 1D prostoru do 3D. Pokud jsou body očíslovány, lze mapovat pořadovým číslem. Každý bod bude u sebe uchovávat místo informace o 2D souřadnici korespondujícího bodu jen jeho pořadové číslo. Postup pak bude popsán v bodech:

- určení vzorového bodu u_i dle pořadového čísla patřící bodu x_i.
- provedení rigidní transformace u_i a získání bodu $\tilde{u}_i$.
- nalezení nejbližšího bodu $\tilde{u}_j$ k bodu $\tilde{u}_i$.
- index j je nové pořadové číslo korespondujícího bodu.

Pokud nastane situace, že v té jedné iteraci se stane, že se i nebude rovnat j, znamená to, že se mění nejbližší bod vzoru pro daný transformovaný bod.

Toto řešení však není vhodné, protože parametrizace by nebyla spojitá a nebylo by možné použít jiné body, než body ze vzorové množiny bodů. V následující sekci tedy navrhuji lepší řešení.

Parametrizace sférickými souřadnicemi

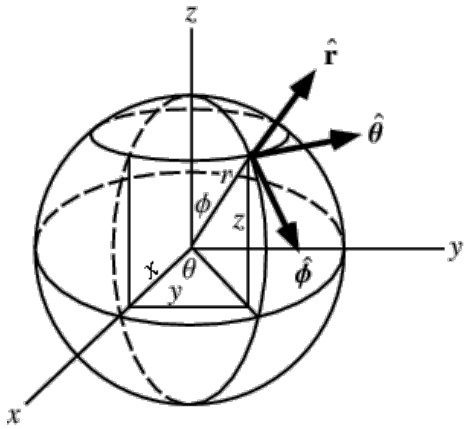
Sférické souřadnice jsou systémem křivočarých souřadnic, které jsou přirozené pro popsání pozice na povrchu koule[9]. Skládají se ze dvou úhlů a poloměru. Obrázek 4.3 ilustruje tyto souřadnice.

Úhel θ (azimut) definuje úhel definovaný na ploše $z = 0$, s počátkem na kladné části osy x ($\theta = 0 \Rightarrow x \geq 0$), pokračujícím do kladných hodnot na ose y ($\theta = \frac{\pi}{2} \Rightarrow y \geq 0$).

Úhel ϕ (zenit) definuje úhel od plochy z v kladném směru ($\phi = \frac{\pi}{2} \Rightarrow z \geq 0$).

Poloměr r pak definuje vzdálenost zobrazovaného bodu od počátku soustavy souřadnic.

Příkladem z praxe je souřadný systém používaný na Zemi. Úhel θ definuje zeměpisnou šířku, úhel ϕ je zeměpisnou délkou a poloměr r je přibližně 6 378km[1]. Například souřadnice centra města Plzně by pak byly vyjádřeny takto: $\theta \doteq 49°74'74"$, $\phi \doteq 13°37'76"$, $r \doteq 6.378 \cdot 10^6 m$.



Obrázek 4.3: Sférické souřadnice $(\theta; \phi; r)$ [9].

Vždy je ale třeba určit počátek soustavy souřadnic. V případě registrace objektů lze použít třeba těžiště buď bodů vzorových nebo bodů transformovaných. Dále z této soustavy souřadnic lze využít úhly θ a ϕ jako parametry a hodnota r bude závislá proměnná – bude se dopočítávat například tak, že se použije vzdálenost nejbližšího bodu (dle směrů obou úhlů) od zvoleného počátku.

Tato parametrizace se zdá vhodná pro registraci na povrch svalu, protože se dá předpokládat, že tato parametrizace bude lépe fungovat při registraci k objektům blížících se tvarem ke kouli, než u rovinných objektů, a to odpovídá tvarům svalů, tuto domněnku však potvrdí nebo vyvrátí až testování.

Zbytek původního algoritmu od Hao Li[4] může být pro tento problém použit beze změny, protože je nezávislý na typu vstupních dat. Nyní lze přejít k implementaci nerigidní registrace algoritmem od Hao Li.

[1] V ideálním případě, když by byla zeměkoule absolutně kulatá.

5. Implementace

Po teoretickém popsání problému již nezbývá než provést implementaci. Pro implementaci jsem zvolil programovací jazyk *C++* a to proto, že ve stejném programovacím jazyce byla napsána i práce[5], na kterou tato práce navazuje. Ze stejného důvodu byla zvolena i knihovna *VTK*[7] pro práci a vizualizaci 3D objektů.

Dále bylo vhodné použít knihovnu pro řešení soustavy nelineárních rovnic. Za tímto účelem byla zvolena knihovna *Ceres Solver*[8]. Následují informace o těchto knihovnách.

5.1 VTK

Knihovna *VTK* je volně dostupný software s otevřeným zdrojovým kódem pro 3D počítačovou grafiku, zpracování obrazu a vizualizaci[7]. Knihovna je napsána v *C++*, ale lze ji použít i v programovacích jazycích *Python* nebo *Java*. Tato knihovna je licencována novou *BSD* licencí.

VTK následuje architekturu toku dat (anglicky data-flow). Data, jako je například běžné v objektově orientovném programování, nepatří jednomu objektu, ale jednotlivé algoritmy (filtry) si je předávají pomocí vstupních a výstupních filtrů. To znamená, že nad jedněmi daty lze sestavit sekvenci filtrů elementárních operací a tím řešit složitější problémy. Protože vstupních filtrů může být libovolný počet, mohou se filtry spojit do jakéhokoliv grafu (příklad dále v 5.1).

V této práci má *VTK* široké využití, od různých filtrů upravujících vstupní data (spojování dat, odstranění bodů, ...[1]) až po vizualizaci výsledku.

5.2 Ceres Solver

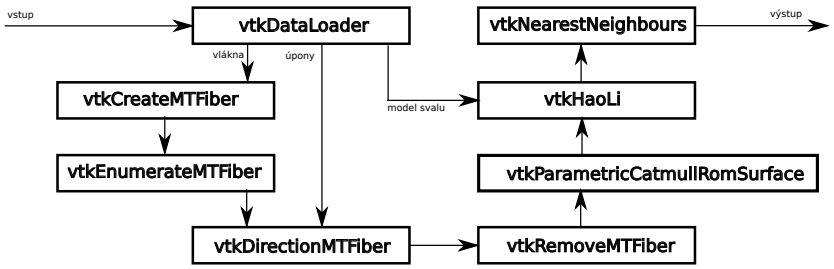
Ceres Solver (dále též „optimalizér“) je *C++* knihovna s otevřeným zdrojovým kódem pro modelování a řešení velkých nebo komplikovaných optimalizačních problémů[8]. Optimalizér je stejně jako knihovna *VTK* licencována novou *BSD* licencí. V této práci jsem tuto knihovnu použil konkrétně pro řešení nelineární soustavy rovnic k nalezení optimálních globálních i lokálních transformací.

Knihovnu *Ceres Solver* jsem zvolil z důvodu poměrně svobodné licence a jednoduchého použití. Dalšími důvody jsou, že knihovna *Ceres Solver* je stále udržována a dobře dokumentována.

[1] Použité filtry budou popsány v sekci 5.3

5.3 Moduly a filtry

Implementační část práce se skládá z knihovny Hao Li algoritmu (viz následující sekce 5.3.1) a z dalších modulů a *VTK* filtrů, které předem připraví vstupní data pro registraci. Tok dat filtry je znázorněn diagramem 5.1.



Obrázek 5.1: Diagram toku dat mezi filtry.

5.3.1 Knihovna Hao Li

Nyní popíši v této sekci jednotlivé moduly, ze kterých se skládá knihovna algoritmu, které jsem implementoval dle návrhu Hao Li. Základem algoritmu je modul *vtkHaoLi*, který slouží spolu s modulem *haoLiConfiguration* jako vnější rozhraní této knihovny.

node

Modul *node* slouží jako přepravka pro data o jednotlivých bodech. Pro každý transformovaný bod se mimo jiné uchovávají během výpočtu neměnná data (údaj mezi lomítky udává počet prvků v poli):

- původní pozice bodu /3/
- pole referencí na sousedící body /$\leq$6, záleží na poloze jednotlivých bodů/
- pole referencí na nejbližší body /4 dle originálního návrhu, lze změnit/
- předem spočítané váhy pro nejbližší body /4, shodně jako v předchozím bodě/

a dále data, která se po každé iteraci mění:

- matice afinní transformace (A_i) /9/
- vektor posunu (b_i) /3/
- korespondující bod parametrizovaný pomocí dvou sférických souřadnic ($u_i = (\theta; \phi)$) /2/
- hodnota důvěryhodnosti ($\omega \in \langle 0; 1 \rangle$) /1/

haoLiConfiguration

Tento modul uchovává konfiguraci Hao Li algoritmu, kterou lze předat pomocí parametrů z příkazové řádky (parametry viz příloha A.2). Jedná se o parametry převážně určené optimalizéru, jakou měrou má penalizovat jednotlivé chyby, ale také kde má optimalizace začít, skončit a jaký má být krok mezi jednotlivými optimalizacemi.

tables

Tento modul slouží k rychlejšímu výpočtu výpočetně složitých funkcí sinus, kosinus a arkus kosinus. Modul *tables* si předem spočítá hodnoty funkcí do pole a když je pak třeba spočítat hodnotu některé z předem spočítaných hodnot, použije se hodnota z tabulky. Tím se sice sníží přesnost výpočtu, ale výpočet se urychlí. Během jednotlivých iterací optimalizéru není potřeba počítat přesné[2] hodnoty funkcí.

Předem se počítají následující funkce v následujících intervalech:

- sinus v intervalu $(0; 2\pi)$, kvůli periodičnosti lze použít pro jakékoliv reálné číslo
- cosinus v intervalu $(0; 2\pi)$, lze použít pro jakékoliv číslo
- arkus kosinus v intervalu $\langle -1; 1 \rangle$

updater

Updater se spouští před každou iterací optimalizéru. Je zodpovědný za přepočítání korespondujících bodů před každou iterací – hledá pro každý optimalizovaný bod nejbližší vzorový bod.

util

Modul *util* obsahuje pomocné funkce. Funkce většinou pracují s obecnými trojrozměrnými poli (vektory) a provádí různé operace od základních binárních operací (sčítání, odčítání, násobení vektorů atd.) až po složitější operace, například transformace z kartézských souřadnic do sférických (viz „Parametrizace sférickými souřadnicemi“ v sekci 4.3.2) souřadnic apod.

vtkAnimation

Filtr *vtkAnimation*, jak již název napovídá, provádí animaci. Vstupem do filtru jsou dvoje data – zdrojová a cílová. Filtr pak provádí lineární interpolaci mezi zdrojovými a cílovými pozicemi bodů. Předpokladem zde je, že indexy bodů zdrojových a cílových dat sobě odpovídají (provádí se např. interpolace pátého bodu zdrojových dat s pátým bodem cílových dat). Filtr jsem použil k animování registrace, animace však jen lineárně interpoluje mezi předzpracovanými daty a výslednými daty registrace, nesimuluje průběh registrace.

[2] V rámci přesnosti datového typu a v rámci přesnosti knihovních funkcí

vtkHaoLi

Tento filtr nejprve transformuje všechna potřebná vstupní data do interní struktury algoritmu (modul *node* viz výše), poté definuje a sestaví problém pro optimalizér, spustí optimalizér a nakonec transformuje všechny body z interních struktur zpět do struktur knihovny *VTK* a nakonec vrátí výsledek.

Filtr uchovává následující optimalizační proměnné:

- matice globální rigidní rotace (R) /3/[3]
- globální vektor posunu (t) /3/

a dále uchovává vektor těžiště /3/ bodů z množiny transformovaných bodů. Těžiště je spočítáno jako průměrná hodnota každé souřadnice všech bodů, které se budou registrovat. V průběhu algoritmu se již těžiště dále nemění.

Moduly cost_*

Skupina modulů, která počítá chybu všech čtyř parametrů, je implementována v knihovně Hao Li konkrétně v modulech `cost_rigid`, `cost_smooth`, `cost_corresponence` a `cost_confidence`. Tyto moduly využívá optimalizér, aby získal hodnoty všech chyb při každé jeho iteraci. Moduly obsahují převážně šablonované funkce. Důvod pro šablonování je ten, že optimalizér volá výpočty chyb se dvěma různými datovými typy, v jednom případě pro výpočet reálného čísla, vlastní hodnoty chyby a v druhém případě výpočet derivace, aby optimalizér věděl, který parametr má optimalizovat jakým směrem. Pro výpočet derivace musí využívat jiný, komplexnější datový typ než pro výpočet hodnoty chyby, proto i sami autoři optimalizéru doporučují použití šablon (dle oficiálních stránek optimalizéru[8] v sekci „Modelling Non-linear Least Squares“).

5.3.2 Knihovna FibreLibrary

Poté, co je implementována knihovna Hao Li, je nutná implementace filtrů které jsou nezbytné pro předzpracování vstupních dat a následné zpracování pro výslednou vizualizaci. Následuje seznam filtrů, které jsem použil. Filtry *vtkCreateMTFiber*, *vtkDataLoader*, *vtkDirectionMTFiber* a *vtkSortMTFiber* jsem implementoval v rámci předchozího studia[4], takže jejich implementaci nelze považovat za součást této práce, ale bylo potřeba tyto filtry použít v této práci, proto je zde uvádím.

vtkDataLoader

Modul načte vstupní data pro celý algoritmus. Vstupní data jsou uložena v jedné složce a názvy jednotlivých souborů mají přesně definovanou strukturu. Názvy souborů jsou tedy v následujícím formátu uvedeným regulárním výrazem:

```
([a|b])?(f|m|a|b)(\d+)\.vtk
```

První část výrazu (\1) definuje stranu uchycení, protože sval je většinou uchycen na dvou stranách k různým kostem. Druhá část (\2) definuje typ dat. Znak *f*

[3] Parametrizováno třemi souřadnicemi, aby byla umožněna pouze rotace.

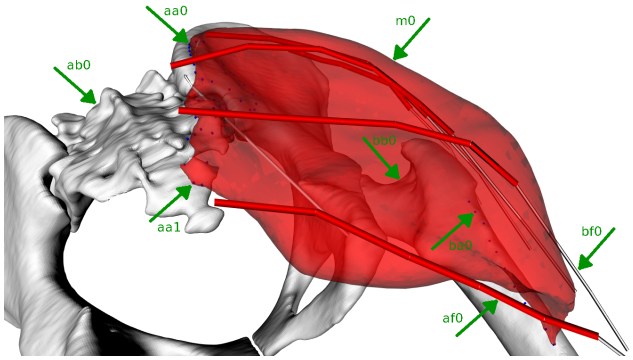
[4] v rámci předmětu *KIV/PRJ3*

definuje vlákna, *m* model svalu, *a* body, kde jsou vlákna uchycena ke kostem a znak *b* definuje modely kostí. Následuje třetí část (\3), která slouží jako pořadové číslo, protože například souborů dat uchycení vláken na stejné straně může být více, pak se rozliší tímto číslem.

První část je nepovinná, protože u modelu svalu nelze stranu definovat. Další výjimka je, pokud se jedná o vlákno, pak se první část názvu používá k rozlišení, zda-li se jedná o vlákna svalu nebo šlachy (znak *a* znamená, že se jedná o svalová vlákna, *b* šlachová). Pokud tedy zahrneme do regulárního výrazu skutečnost, že pokud je znak druhé části *m*, pak první část neexistuje, jinak je vyžadována, pak bude regulární výraz následující:

```
(m|[a|b](f|a|b))\d+\.vtk
```

Pro spuštění demonstrační aplikace je dostačující, aby existovaly v jedné složce soubory `m0.vtk`, `af0.vtk` a `bf0.vtk`, modul je ale připraven pro načítání dalších druhů dat, které mohou být použity například k vizualizaci okolních souvislostí, jakými jsou například modely kostí. Veškerá data, která modul načte, jsou ilustrována na obrázku 5.2 (na obrázku je sval *gluteus maximus* a jeho okolí). Poloprůhledný červený objekt zde značí povrch svalu, bílé objekty kosti, červené lomené čáry značí naměřená svalová povrchová vlákna, bílé lomené čáry pak značí šlachová vlákna. Konečně modré „kostičky“ ohraničují oblast, kde jsou svaly uchyceny ke kostem.



Obrázek 5.2: Model *gluteus maximus*, kompletní data.

vtkCreateMTFiber

Nejprve je potřeba uvést, jak jsou uložena data o lomených čarách v datech knihovny *VTK*. Jedná se pouze o pole, kde jsou lomené čáry uloženy za sebou a pro každou je uložen nejdříve počet prvků dané lomené čáry následovaný indexy bodů, ze kterých se lomená čára skládá. Příkladem uložení lomených čar může být obrázek 5.3, který reprezentuje uložení lomených čar z obrázku 5.5.

```
5 2 9 16 0 13 5 3 6 12 1 4 5 10 17 18 14 5 5 11 7 19 8 15
```

Obrázek 5.3: Příklad uložení lomených čar v knihovně *VTK*.

Bohužel v reálných vstupních datech jsou lomené čáry reprezentovány jako množina úseček a že se jedná o jednu lomenou čáru značí opakující se index bodu v poli přesně tak, jak je to uvedeno na obrázku 5.4, navíc jednotlivé úsečky příslušné jedné lomené čáře mohou být libovolně promíchané napříč celým polem.

Data na obrázcích 5.3 a 5.4 definují vizuálně identické lomené čáry, pouze jiným způsobem a proto hlavní funkcí filtru *vtkCreateMTFiber* je právě převod množin úseček (např. na obrázku 5.4) na reprezentaci s co nejmenším počtem lomených čar (např. na obrázku 5.3). Převod je výhodný nejen k ušetření malé části paměti, ale s daty se pak i snáze pracuje, navíc převedená data vyžaduje také filtr *vtkParametricCatmullRomSurface* (viz dále).

```
2 2 9   2 9 16   2 0 16 2 13 0
2 6 3   2 7 11   2 6 12 2 10 17
2 19 7 2 18 17 2 12 1 2 14 18
2 14 5 2 19 8   2 4 1   2 8 15
```

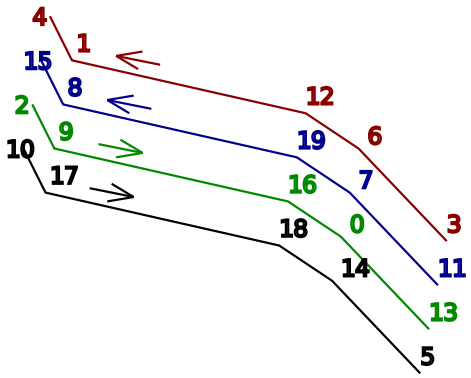
Obrázek 5.4: Uložení lomených čar v reálných vstupních datech.

Tento filtr nejen upraví pole lomených čar na nejkratší možné, dokáže dokonce spojovat i pole napříč jednotlivými daty (svalovými a šlachovými vlákny). Do filtru vstupuje různý počet různých (svalových nebo šlachových) vláken. Výstupem jsou jen jedna data s nejkratším možným polem lomených čar a s uloženou informací o každém bodu, jestli reprezentuje sval nebo šlachu.

vtkSortMTFiber

Filtr seřadí vlákna tak, aby v datech o lomených čarách po sobě vždy následovala každá dvě nejbližší vlákna. Tato filtrace je nezbytná pro vytvoření vyhlazené plochy, protože modul vytvářející plochu vyžaduje správné pořadí vláken.

Pokud bude do tohoto filtru vstupovat pole lomených čar z obrázku 5.3 a pokud budou vlákna fyzicky ležet například jak je uvedeno na obrázku 5.5, pak výsledkem filtru bude pole 5.6.



Obrázek 5.5: Pozice vláken na vstupu filtru *vtkSortMTFiber*.

```
5 3 6 12 1 4 5 11 7 19 8 15 5 2 9 16 0 13 5 10 17 18 14 5
```

Obrázek 5.6: Výsledek algoritmu *vtkSortMTFiber*.

Prostorové řazení vláken probíhá ve dvou krocích. V prvním kroku se identifikuje krajní vlákno. Takové vlákno je identifikováno nejdelší vzdáleností ke všem ostatním vláknům[5], následně je toto vlákno označeno jako první vlákno. Druhým krokem je seřazení ostatních vláken. Každé následující vlákno je takové, které má nejbližší vzdálenost k aktuálnímu vláknu při ignorování již použitých vláken. Algoritmus funguje dobře v případě, kdy vlákna se příliš nekříží. Pokud by se vlákna výrazně křížila, pak vlákna není možné prostorově seřadit.

Z obrázku 5.5 je zřejmé, že všechna vlákna nesměřují stejným směrem a indexy bodů ve vláknech jsou neuspořádané. A právě tím se bude zabývat další filtr *vtkDirectionMTFiber*.

[5]Vzdáleností dvou vláken je myšlena vzdálenost mezi dvojicí nejbližších bodů dvou vláken.

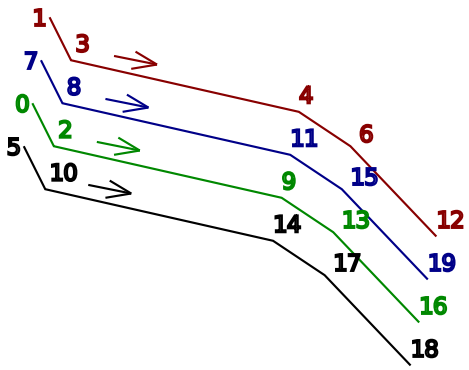
vtkDirectionMTFiber

Modul *vtkDirectionMTFiber* seřadí vlákna tak, aby všechna vlákna směřovala stejným směrem a aby se indexy bodů ve vlákně zvětšovaly v každém vlákně taktéž stejným směrem. Tato akce je nezbytná pro vytvoření vyhlazené plochy modulem *vtkParametricCatmullRomSurface*, který vyžaduje, aby body byly ve vlákně seřazeny. Filtr pro svoji funkci potřebuje znát body, kde je sval uchycen ke kosti, a tím definuje shodný směr pro všechna vlákna. Pokud tato data filtr nedostane, potom neprovede žádnou akci.

Do algoritmu vstupují vlákna z obrázku 5.5 a výstupem jsou vlákna, kde jsou indexy seřazeny. I pokud by se provedla libovolná permutace indexů vstupních bodů v rámci vlákna, výstup algoritmu bude shodný s výstupem na obrázku 5.8. V interní struktuře *VTK* bude pole definující lomené čáry seřazeno podle indexů (viz obrázek 5.7).

```
5 0 2 9 13 16 5 1 3 4 6 12 5 5 10 14 17 18 5 7 8 11 15 19
```

Obrázek 5.7: Uložení lomených čar po použití filtru *vtkDirectionMTFiber*.



Obrázek 5.8: Výstup algoritmu *vtkDirectionMTFiber*.

vtkLinesRestorer

vtkLinesRestorer je filtr, který dokáže obnovit data o lomených čárách, které se z dat odstraní po použití filtru *vtkParametricCatmullRomSurface*. Filtr předpokládá, že všechna výsledná vlákna jsou stejně dlouhá (ve smyslu „skládající se ze stejného počtu bodů“); takový výsledek poskytuje již zmíněný *vtkParametricCatmullRomSurface*.

Výsledek tohoto filtru je použit pouze pro vizualizaci výsledné registrace a nemá vliv na výpočet registrace.

vtkNearestNeighbours

Filtr provede registraci algoritmem nejbližšího souseda. Filtr vyžaduje dvě množiny vstupních bodů. Jednu množinu, která se bude transformovat, a jednu vzorovou. Body z transformované množiny se poté každý samostatně přemístí na

souřadnice nejbližšího bodu z množiny vzorových bodů. Tato registrace se využije v konečné fázi registrace množin bodů, aby registrovaná množina přesně ležela na vzorové množině.

vtkRemovePoints

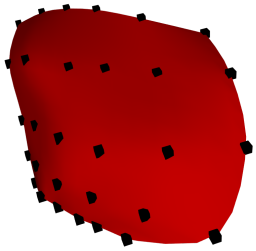
Dalším užitečným filtrem je filtr k odstranění nepotřebných bodů a právě k tomu slouží *vtkRemovePoints*. Protože se registruje na povrch svalu, není třeba mít v datech body reprezentující šlachy a tím se tyto body stávají oněmi nepotřebnými. Modul *vtkRemovePoints* odstraní body ze vstupní množiny dle zvoleného módu. Módy odstranění jsou:

1. seznamem bodů k odstranění
2. podmínkou – algoritmu se předá ukazatel na funkci nebo lambda výraz, který pro každý bod vyhodnotí, jestli má bod zůstat v množině či nikoliv
3. počtem zbývajících bodů – body budou odstraněny uniformně podle označení bodů

Pokud se odstranění provádí dle bodu 1 nebo 2, pak algoritmus vyžaduje, aby vstupní data obsahovala i informace o lomených čárách, ze kterých se zvolené body také následně vyřadí.

vtkParametricCatmullRomSurface

Tento modul vytvoří vyhlazenou plochu. Filtr předpokládá, že vstupní data obsahují informace o lomených čárách, tyto čáry směřují stejným směrem, indexy bodů rostou stejným směrem (zajišťuje *vtkDirectionMTFiber*) a že dvě sousedící vlákna v datech následují (zajišťuje filtr *vtkSortMTFiber*). U tohoto filtru se určí počty bodů ve dvou směrech, filtr poté vytvoří nové body převzorkováním vstupních dat podélně i příčně Catmull-Rom[11] křivkou a nakonec provede triangulaci, čímž vznikne plocha. Plocha, která vznikla z černých bodů, je zobrazena na obrázku 5.9.



Obrázek 5.9: Plocha vytvořená algoritmem *vtkParametricCatmullRomSurface*.

vtkParametricCatmullRomSurface filtr jsem nepoužil primárně k vytvoření plochy, ale jeho hlavní využití v této práci je ke zvýšení počtu bodů převzorkováním vstupní množiny bodů. Ve vstupních datech se nachází málo vláken, která se skládají většinou z nízkého počtu bodů. Během implementace Hao Li algoritmu se ukázalo, že algoritmus je citlivý na nízký počet vstupních bodů.

Tento filtr jsem neimplementoval, poskytl mi ho vedoucí této práce.

5.3.3 Demonstrační aplikace

Zdrojové kódy aplikace se nacházejí ve složce `src/FibreOperation`. Jedná se o dva moduly *demoApp* a *vtkDataDisplay*, kterými se spouští program a vizualizuje výsledek. Dále se v této složce nachází zdrojový soubor *vtkTestDataCreator*, který vytvoří testovací umělá data využitá v následující kapitole 6.

vtkDataDisplay

Tento modul vytváří posloupnost *VTK* filtrů skládající se ze všech potřebných filtrů pro předzpracování dat a samotnou registraci (viz obr. 5.1). Nakonec modul provede vykreslení výsledků. Modul nejprve spojí všechna vlákna do jedné sady dat pomocí filtru *vtkCreateMTFiber* (viz výše), dále seřadí vlákna podélně i příčně za pomocí filtrů *vtkSortMTFiber* a *vtkDirectionMTFiber*. Poté se vyřadí z dat všechny body, které patří šlachám, a provede se převzorkování na konstantní počet bodů za pomocí filtru *vtkParametricCatmullRomSurface*, čímž se data připraví na registraci. Dále se provede registrace Hao Li (*vtkHaoLi*) algoritmem a nakonec se data registrují metodou nejbližšího souseda (*vtkNearestNeighbours*). Následuje poslední část – vykreslení. Vytvoří se filtr pro zobrazení vláken (knihovní *VTK* filtr *vtkTubeFilter*), do okna se vloží výsledná vlákna, vzorový model svalu a animace registrace (podrobnosti viz *vtkAnimation*) a nakonec se okno s výsledkem vykreslí.

Program ještě obsahuje modul *vtkTestDataCreator* (viz dále 6.1) a některé další moduly, které pouze slouží jako podpůrné, a proto zde nejsou uvedeny.

6. Dosažené výsledky

Registrační program je implementován a nyní je třeba ho otestovat. Program jsem testoval na počítači s procesorem Intel Core i3-5005U, 2GHz a pamětí 8GB RAM na operačním systému OS Linux ve verzi jádra 4.10.0-19.

Jako vstupní data jsem použil reálná data svalů *gluteus maximus*, *gluteus medius* a *iliacus* získaná z evropského projektu LHDL (Living Human Digital Library)[10], který byl aktivní v letech 2006-2009 a jehož cílem bylo vytvořit umělý model lidského těla. Dále jsem vytvořil testovací umělá data, kde se registrují vlákna na povrch polokoule vytvořená modulem *vtkTestDataCreator*.

Oproti originálnímu algoritmu, kde byly parametry registrace konstantní, je možné tento algoritmus spustit i s jinými parametry a tím docílit různých výsledků. Volitelné parametry byly původně zamýšleny proto, že originální návrh měl pevně dané rozměry vstupních dat. V případě svalů se jejich rozměry liší sval od svalu, proto jsou parametry nastavitelné. Dalším pozitivem je možnost plynulého nastavení nerigidity dat, tzn. jak moc bude objekt schopen měnit svůj tvar.

6.1 vtkTestDataCreator

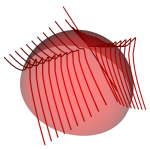
Modul *vtkTestDataCreator* vytvoří testovací data, která budou použita v této kapitole. Modul vytvoří dvě sady dat, jedna data jsou data vzorových bodů a jedná se o body ležící na povrchu polokoule. Druhou sadou dat je množina bodů k registraci. Body leží na deformované ploše a její povrch je vidět z vláken na obrázku 6.1a. Tvar vláken byl navržen tak, aby ho bylo možno snadno matematicky definovat dle parametrických souřadnic 6.1, které jsou zobrazením $\mathbb{R}^2 \to \mathbb{R}^3$, ale aby zároveň částečně procházel polokoulí.

$$\begin{aligned} \tilde{x} &= x \\ \tilde{y} &= 200 - \sqrt{|x * z|} \\ \tilde{z} &= z \end{aligned} \tag{6.1}$$

Na obrázku 6.1a jsou vyobrazena vstupní data svalu *gluteus maximus* před předzpracováním. Fialově a poloprůhledně je vykreslen tvar svalu, vlákna jsou vykreslena čárami, a to červeně pro svalové vlákno a bíle pro šlachové vlákno. Algoritmus pak z těchto dat provede předzpracování, které obnáší filtraci bodů reprezentující šlachová vlákna, seřazení bodů, ze kterých se vlákna skládají, seřazení samotných vláken a vytvoření vyhlazené plochy převzorkováním vláken.

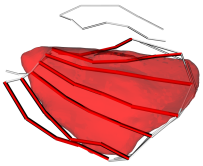
Za předzpracovaná data se pak považuje vyhlazená plocha tvořená registrovanými body a model svalu, na který se budou tyto body registrovat. Na obrázku 6.2 je červeně vyznačena tato vyhlazená plocha. Obdobně tomu tak je v případě dalších dvou svalů a umělých dat, vstupní data jsou ilustrována na obrázcích 6.1b, 6.1c a 6.1d a po předzpracování vznikne červená plocha na obrázcích 6.4, 6.10 a 6.14.

Protože následující obrázky nemusí být příliš průkazné, lze si program nainstalovat a spustit dle přílohy A a pak lze objekty prohlížet z různých úhlů, popřípadě na přiloženém DVD lze nalézt demonstrační videa některých z následujících testů, kde jsou objekty vidět z více úhlů.

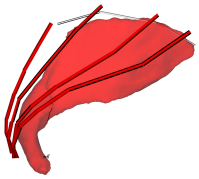


(a) Polokoule.

(b) *Gluteus maximus*



(c) *Gluteus medius*

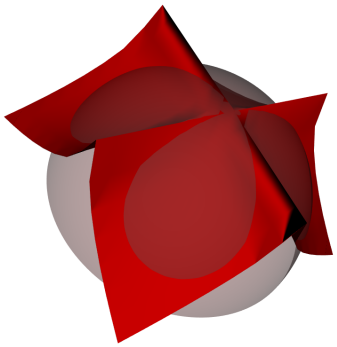


(d) *Iliacus*

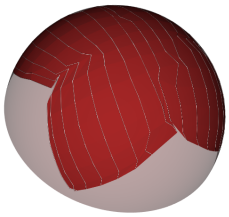
Obrázek 6.1: Vstupní testovaná data.

6.2 Umělá data

První test, který provedu, je spuštění programu v základním nastavení bez nepovinných parametrů nad vstupními umělými daty polokoule. Základním nastavením je myšleno, že optimalizér bude spuštěn desetkrát, v prvních iteracích se bude snažit zachovávat tvar objektu, takže bude docházet k rigidní registraci, naopak v posledních iteracích bude dovoleno tvar objektu měnit, a tím dojde k registraci nerigidní. Jak spustit program v základním nastavení nebo s parametry je uvedeno v uživatelské příručce v příloze A.2. Předzpracovaná data jsou vyobrazena na obrázku 6.2, výstup programu na obrázku 6.3.



Obrázek 6.2: Model umělých dat, předzpracovaná data.



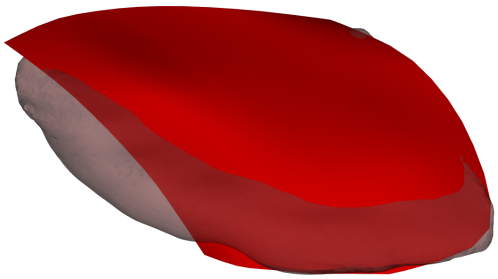
Obrázek 6.3: Model umělých dat, výsledná registrace.

K drobným nepřesnostem dochází v místech, kde se původní registrovaný povrch prudce lámal. Zde vznikají zlomy i po registraci, což je ale vlastnost algoritmu, který se snaží zachovat tvar původního objektu. Lze říci, že na těchto umělých datech funguje algoritmus dobře, což bylo očekáváno kvůli zvolené parametrizaci korespondujících bodů sférickými souřadnicemi (viz 4.3.2).

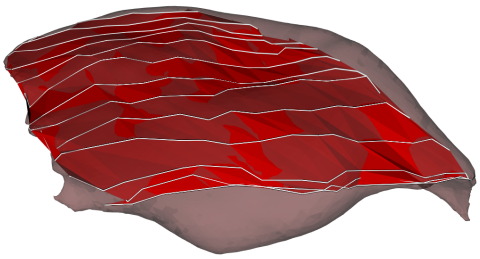
6.3 Gluteus maximus

Na datech svalu *gluteus maximus* jsem provedl nejprve test programu bez nepovinných parametrů. Předzpracovaná data jsou znázorněna obrázkem 6.4 a výsledek registrace je vyobrazen na obrázku 6.5.

Protože se jedná o bodovou registraci a nikoliv registraci povrchovou, tak na sobě plochy přesně neleží, což způsobuje jejich vzájemné pronikání a na výstupních obrázcích (například na 6.5) to tvoří tzv. „mapy“. To ale není chyba algoritmu, to právě naopak dokazuje, že body obou množin jsou natolik blízko, že trojúhelníky, ze kterých jsou povrchy modelů tvořeny, se prolínají.



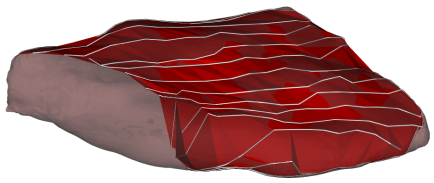
Obrázek 6.4: Model *gluteus maximus*, předzpracovaná data.



Obrázek 6.5: Model *gluteus maximus*, výsledek algoritmu.

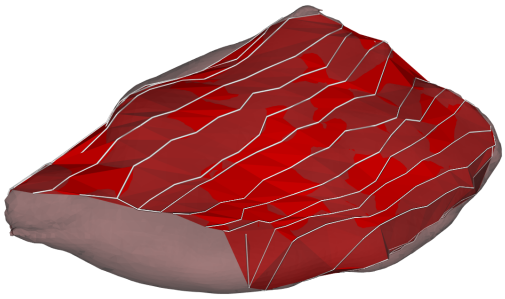
Zdá se v tomto případě, že registrace proběhla úspěšně. Bohužel ale při pohledu na druhou stranu modelu vznikly dva mimolehlé body (obrázek 6.6), které se transformovaly až na druhou stranu povrchu svalu. Tyto body vznikly v závěrečné fázi registrace, kdy byly body registrovány algoritmem nejbližšího souseda a tyto body byly blíže k druhé straně povrchu než k té, ke které by se měly registrovat. Dále je vidět, že ve střední části modelu proběhla registrace poměrně bez problémů, na okrajích se vlákna nahustila blíže k sobě a překrývají se.

Lze se tedy pokusit o odstranění mimolehlých bodů úpravou parametrů algoritmu. V tomto konkrétním případě, kdy bod prošel povrchem, je potřeba



Obrázek 6.6: Model *gluteus maximus*, mimolehlé body.

více penalizovat hladkost, aby nedošlo k neúměrné transformaci jednoho bodu vzhledem k okolí, tím se zachová tvar a při finální registraci algoritmem nejbližšího souseda se mimolehlé body budou raději registrovat jen na jednu stranu povrchu svalu. Dále je třeba provést jen první iteraci, protože při první iteraci je nejvíce zachován tvar registrovaného objektu. Nastavíme tedy konec iterací z originální hodnoty 2000 na hodnotu 1.5 a vyšší penalizaci hladkosti z hodnoty 100 na hodnotu 1000000[1]. Výsledek je vyobrazen na obrázku 6.7.



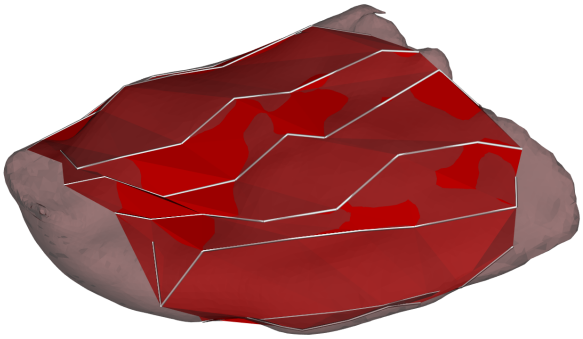
Obrázek 6.7: Model *gluteus maximus*, pokus o opravu mimolehlých bodů.

Ani v tomto případě ale registrace tohoto bodu není dokonalá, původně mimolehlý bod sice neprochází na druhou stranu svalu, stále ale nekoresponduje s ostatními body ve stejném vlákně. Lepšího výsledku se mi již změnou parametrů dosáhnout v případě modelu svalu *gluteus maximus* nepodařilo, protože při ještě větším zvýšení penalizace hladkosti již nedochází k výrazným změnám, naopak při snižování vzniká více mimolehlých bodů.

Mohu ale ukázat, jak dopadne optimalizace, když zvýším nebo snížím počet registrovaných bodů. Nejprve snížím počet bodů čtyřikrát z 256 na 64 bodů. Výsledek registrace je ukázán na obrázku 6.8.

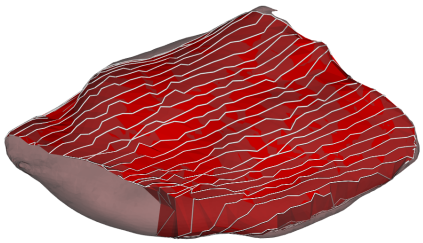
Registrace sice proběhne velmi rychle (viz sekce „Doba výpočtů“), ale registrace dopadne velmi špatně. To jen potvrzuje, že algoritmus je závislý na

[1] Jak nastavit parametry programu viz příloha A.2



Obrázek 6.8: Model *gluteus maximus*, čtyřikrát menší počet bodů.

dostatečně velkém počtu vstupních bodů. Zvýší-li se naopak počet bodů čtyřikrát (z původních 256 na 1024), registrace proběhne jako na obrázku 6.9

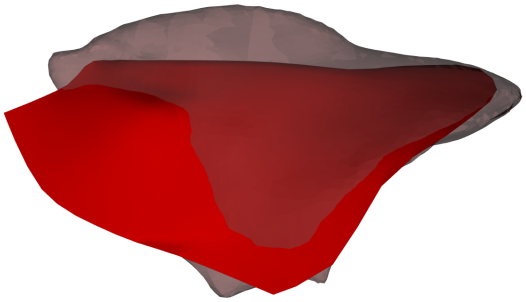


Obrázek 6.9: Model *gluteus maximus*, čtyřnásobný počet bodů.

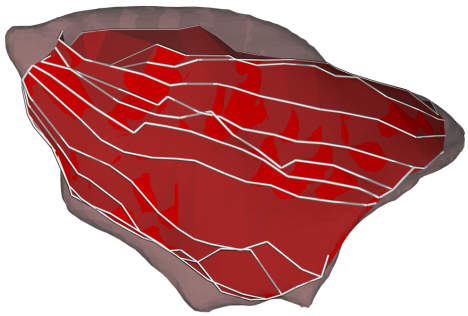
V takovém případě vznikají nějaké mimolehlé body (viz 6.9), ale poměr mimolehlých bodů ku všem bodům je přibližně zachován. Celkově registrace dopadla podobně jako v případě s 256ti body, výhodou však je, že vlákna jsou přesněji aproximována více body, naopak nevýhodou je delší doba běhu algoritmu.

6.4 Gluteus medius

Dalším testem je spuštění programu bez nepovinných parametrů se vstupními daty svalu *gluteus medius*. Tento sval má složitější tvar na registrované straně svalu než v případě svalu *gluteus maximus* v předchozím případě, protože je ve střední části registrované poloroviny více vypouklý, než je tomu v předchozím případě. Předzpracování vstupních dat lze vidět na obrázku 6.10, výsledná registrace je ilustrována obrázkem 6.11.



Obrázek 6.10: Model svalu *gluteus medius*, předzpracovaná data.

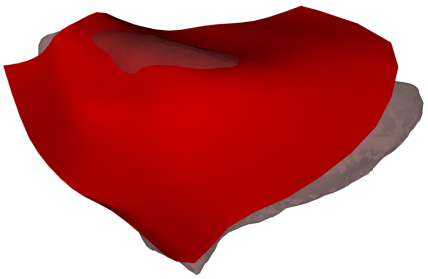


Obrázek 6.11: Model svalu *gluteus medius*, výsledná registrace.

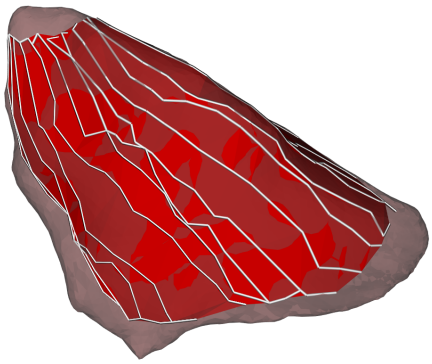
V případě svalu *gluteus medius* dopadla registrace mnohem hůře než v případě předchozího svalu *gluteus maximus*. Vzniklo zde několik mimolehlých bodů. Na místě, kde je povrch vypouklý, neleží žádná vlákno, což způsobuje, že výsledná plocha prochází svalem. Opakuje se zde i stejný problém, že na krajích jsou vlákna hustší než ve střední části registrovaných vláken. Po důkladné analýze problému jsem zjistil, že optimalizér sice konverguje, ale konverguje do místa, odkud je registrace započatá, tzn. registrace se vlastně provádí pouze algoritmem nejbližšího souseda jako konečná fáze. Nastavení parametrů registrace tuto skutečnost nijak nezměnilo.

Problémem v tomto případě je, že registrovaná plocha proniká příliš svalem a optimalizér se nedokáže rozhodnout, na kterou stranu svalu provede registraci.

Jedním řešením by mohlo být ruční vytažení plochy více od středu svalu a následné provedení registrace. To je provedeno na obrázcích 6.12 (ručně posunutá předzpracovaná data) a 6.13 (výsledná registrace).



Obrázek 6.12: Model svalu *gluteus medius*, předzpracovaná ručně posunutá data.



Obrázek 6.13: Model svalu *gluteus medius*, výsledná registrace posunuté plochy.

V tomto případě registrace proběhla mnohem lépe než bez posunu, hlavně nevznikají body, které by pronikaly na druhou stranu svalu. Na druhou stranu vzniká křížení vláken. Navíc posun byl proveden ručně, a tím mohlo dojít k nepřesnostem, protože výsledná registrace (protože je nerigidní) závisí na směru posunu.

6.5 Iliacus

Dalším testovaným modelem svalu je *iliacus*. Jedná se o podlouhlý, na jedné straně tenký a na druhé straně tlustší, podlouhle zkřivený sval, jak je vidět na obrázcích 6.1d, 6.14 nebo 6.15.

Obrázek 6.14: Model svalu *iliacus*, předzpracovaná data.

Obrázek 6.15: Model svalu *iliacus*, výsledná registrace.

Z výsledku registrace (obr. 6.15) je zřejmé, že vlákna nepokrývají celou délku svalu. To však nepovažuji za chybu registrace, ale spíše za chybu vstupních dat (obr. 6.1d), která taktéž nepokrývají celou délku, takže v tomto případě popisují tvar vláken nedostatečně. Jako v předchozích případech i zde vznikl mimolehlý bod, jinak registrace proběhla poměrně úspěšně. Ani v tomto případě zvolení jiných parametrů nepřineslo lepší výsledek.

6.5.1 Doba výpočtů

Za účelem měření času jsem do funkce `main(int, char**)` na její začátek a konec vložil měření času. Dále jsem vypnul jakékoliv výpisy ať na standardní výstup nebo na standardní chybový výstup a nakonec jsem odstranil vykreslování okna, protože v tom případě by program čekal na akci uživatele, než zavře vizualizační okno. Měření jsem provedl pro všechna testovaná data desetkrát a do tabulky jsem zapsal průměrnou dobu běhů. Tabulka 6.1 ukazuje doby běhu při běhu bez parametrů.

Model	Vzorových bodů	Registrovaných bodů	Čas [s]
gluteus maximus	9 878	64	10.8
gluteus maximus	9 878	256	251.8
gluteus maximus	9 878	1024	841.9
gluteus medius	5 313	64	4.8
gluteus medius	5 313	256	71.2
gluteus medius	5 313	1024	556.4
iliacus	6 931	64	6.7
iliacus	6 931	256	152.6
iliacus	6 931	1024	973.2
umělá data	129 240	64	90
umělá data	129 240	256	1279.2
umělá data	129 240	1024	1661.5

Tabulka 6.1: Počty bodů modelů a doby běhu algoritmu.

Dále jsem testoval dobu běhu při běhu pouze první iterace algoritmu Hao Li, která ve většině případů trvá nejdéle. V ostatních směrech probíhalo měření za stejných podmínek jako měření předchozí. Výsledné časy běhu programu při jedné iteraci algoritmu jsou v tabulce 6.2, pro porovnání uvádím i čas původního algorimu dle [4].

Model	Vzorových b.	Registrovaných b.	Iterací	Čas [s]
gluteus maximus	9 878	64	79	3.7
gluteus maximus	9 878	256	77	24.1
gluteus maximus	9 878	1024	84	261.9
gluteus medius	5 313	64	15	1.5
gluteus medius	5 313	256	15	3.4
gluteus medius	5 313	1024	15	50.7
iliacus	6 931	64	35	2.3
iliacus	6 931	256	55	9
iliacus	6 931	1024	415	685.6
umělá data	129 240	64	80	30.4
umělá data	129 240	256	108	191.4
umělá data	129 240	1024	77	582.7
Originální Hao Li	119 518	336	219	199

Tabulka 6.2: Počty bodů modelů a doby běhu jedné iterace algoritmu.

Z časů běhů je vidět, že více závisí na počtu registrovaných bodů, než na počtu bodů vzorových. Pokud je počet registrovaných bodů 1024, celková doba běhu se pohybuje na testované platformě mezi osmi až třiceti minutami, což je značná doba. Při průměrném počtu bodů (256) se čas pohyboval přibližně mezi jednou až dvaceti minutami. To je uspokojivé zrychlení vzhledem k relativně malé ztrátě přesnosti. Při malém počtu bodů (64) se čas pohyboval do dvou minut, bohužel výsledná registrace nebyla uspokojivá, proto takto nízké počty bodů nejsou použitelné.

Dle analýzy algoritmu vychází asymptotická složitost $O(M \cdot N \cdot I)$, kde M je počet vzorových bodů, N je počet registrovaných bodů a I je počet iterací. To proto, že v každé iteraci z I iterací se ke každému registrovanému bodu z množiny N bodů hledá jeden nejbližší vzorový bod z mnonžiny M bodů. Testy takový výsledek sice nevyvrací, nelze ale ani s jistotou říci, že analiticky určenou asymptotickou složitost potvrzují. Pokud porovnám běh programu s originálním algoritmem od Hao Li, časy běhů tohoto programu jsou srovnatelné, a to i přes to, že zde navržený algoritmus je výpočetně složitější a byl testován na méně výkonném[2] počítači. Proto lze říci, že časy běhu algoritmu jsou i v rámci minut uspokojivé. Jistého urychlení by se dalo docílit paralením zpracováním, ale vzhledem k provázanosti výpočtů není paralelizace algoritmu snadno řešitelný problém.

Z těchto konkrétních testů je dále zřejmé, že ve všech reálných datech vzniká několik mimolehlých bodů, které by bylo nutné ručně nebo nějakým jiným algoritmem opravit. Ve všech testech reálných dat dochází k problému, kdy vlákna jsou na krajích hustší. Pozitivní na algoritmu je možnost, byť malá, opravení mimolehlých bodů změnou parametrů. To se však podařilo jen jednou ze tří případů a ani oprava nebyla dokonalá.

[2] Originální algoritmus byl testován na počítači s 3GHz oproti 2GHz.

7. Závěr

V rámci této práce byl navržen a následně implementován algoritmus nerigidní registrace povrchových svalových a šlachových snopců. Byl vybrán algoritmus založený na algoritmu Hao Li, který na počátku sliboval dobré výsledky. Navržený algoritmus byl implementován v jazyce *C++* s podporou knihoven *VTK* a *Ceres Solver*, což díky velmi svobodné licenci obou zmíněných knihoven (nová *BSD* licence) umožňuje použití téměř kdekoliv.

Díky obecnému přístupu k implementaci algoritmu odvozeného od algoritmu Hao Li si lze také snadno představit i jiné využití samotné registrace nejen v oblasti registrace svalových a šlachových snopců. Díky knihovně *VTK* lze výsledky poměrně snadno zobrazit, cílový uživatel je pak schopen objektem pohybovat, rotovat, přibližovat jej a tím je schopen lépe určit, zda-li registrace proběhla úspěšně či nikoliv.

Algoritmus ve všech testovaných scénářích fungoval přinejmenším stejně dobře jako registrace nejjednodušším algoritmem nejbližších sousedů. Na testovaných datech sice vzniká několik mimolehlých bodů, ale není jich mnoho, tvar vláken je v rámci možností zachováván a schopnost zachovávání tvaru lze i nastavovat.

Mezi vlastnosti, které nelze zařadit ani do kladných ani do záporných, patří možnost/nutnost zadání parametrů registrace. Ne vždy algoritmus funguje správně v základní konfiguraci, proto je někdy nutné parametry upravit, což vyžaduje zásah uživatele, ale dobré je, že algoritmus je schopen produkovat různé výsledky v závislosti na vstupních parametrech.

Na druhou stranu má algoritmus několik nedostatků. Algoritmus je závislý na velkém počtu vstupních bodů, s malým počtem sice funguje, ale neprodukuje výsledky, které jsou očekávány. Další vlastnost tohoto algoritmu je, že úspěšnost rigidní registrace se snižuje se zvětšující se vzdáleností transformovaných bodů. Problémem jsou také hustší a křížící se krajní vlákna oproti středním vláknům, která nejsou sice vyhlazená, ale nepřesnosti jsou mnohem menší, než na krajích. Dalším negativem je neochota algoritmu registrovaný objekt zvětšovat. Testování také ukázalo, že ve většině případů vznikají mimolehlé body, které nejsou žádoucí, naštěstí je jich mnohem méně, než jich je na vstupních datech.

Celkově tedy výsledky tohoto algoritmu musím hodnotit jako průměrné, protože algoritmus dle testování trpí některými nežádoucími vlastnostmi a obávám se, že ve většině případů nebude fungovat plně automaticky.

Práci lze uzavřít tím, že algoritmus je schopen provést registraci průměrně, ve většině případů je však nutné data mírně upravit. Algoritmus lze tedy použít, ale ne zcela automaticky.

Literatura

[1] OATIS, Carol A. *Kinesiology: The Mechanics and Pathomechanics of Human Movement. Lippincott Williams & Wilkins*,2009.Recall Series.[cit. 2017-01-10].ISBN 97807817742222.

[2] LEE, Dongwoon a Zhi LI a Qazi Z. SOHAIL a Ken JACKSON a Eugene FIUME a Anne AGUR. *A three-dimensional approach to pennation angle estamination for human sceletal muscle. Computer Methods in Biomechanics and Biomedical Engineering.*2015.vol. 18, s. 1474-1484 [cit. 2017-01-12].

[3] *Point set registration — Wikipedia: The Free Encyclopedia* [online]. Wikipedia Foundation, 2016 [cit. 2017-01-30], Dostupné z `https://en.wikipedia.org/w/index.php?title=Point_set_registration&oldid=732521446`

[4] HAO, Li a Robert W. Sumner a Mark Pauly. *Global Correspondence Optimalization for Non-Rigid Registration of Depth Scans. Proceedings of the Symposium on Geometry Processing.*2008

[5] CHOLT, David. *Dekompozice modelu svalu na svalová vlákna.* Plzeň. 2013. 71 str. Diplomová práce na Fakultě aplikovaných věd Západočeské univerzity v Plzni. Vedoucí diplomové práce Doc. Ing. Josef Kohout, Ph.D.

[6] MADSEN, Kaj a Hans B. NIELSEN a Ole TINGLEFF. *Methods for Non-Linear Least Squares Problems Informatics and Mathematical Modelling, Technical University of Denmark.*2004

[7] Kitware Inc. *VTK: The Visualization Toolkit.* [cit. 2017-04-17], dostupné z `www.vtk.org`

[8] AGARWAL, Sameer a Keir MIERLE *Ceres Solver* [cit. 2017-04-04], Dostupné z `http://ceres-solver.org`

[9] WEISSTEIN, Eric W. *Spherical Coordinates. MathWorld – A Wolfram Web Resource* [cit. 2017-04-10], Dostupné z `http://mathworld.wolfram.com/SphericalCoordinates.html`

[10] *Living Human Digital Library*, Dostupné z `http://cordis.europa.eu/project/rcn/79478_en.html`

[11] CATMULL, Edwin a Rapharel ROM. *A class of local interpolating splines Computer Aided Geometric Design.*1974. s. 317-326.

[12] CEDILNIK, Andy a Bill HOFFMAN a Brad KING a Ken MARTIN a Alexander NEUNDORF. *CMake: Cross Platform Make.* [cit. 2017-04-17], Dostupné z `https://cmake.org`

A. Použití programu

Zde se nacházejí stručné informace, jak se program používá. Program je nutné před spuštěním nejprve zkompilovat. Následující informace mohou být platformně závislé, proto následující postupy nemusejí vždy fungovat, v takovém případě mohou pomoci návody na oficiálních stránkách knihoven [7][8] nebo programu CMake[12].

A.1 Kompilace programu

V této sekci stručně popíši, jak zkompilovat a spustit program přiložený k této práci. Jak jsem již uvedl v sekci 5, je potřeba mít nainstalované knihovny *VTK* a *Ceres Solver*. Dále je vhodné k překladu použít program *CMake*[12].

Knihovnu *VTK* lze stáhnout z oficiálních stránek knihovny *VTK*[7] v sekci „Download“, kde lze stáhnout instalátor pro operační systémy *Windows*, *Linux* nebo *Darwin*, popřípadě zkušenější uživatel může instalovat knihovnu po překladu ze zdrojových kódů.

V případě instalace knihovny *Ceres Solver* neexistují oficiální instalátory pro běžné operační systémy, na druhou stranu ale na oficiálních stránkách knihovny[8] lze v sekci „Installation“ nalézt rozsáhlý návod, jak knihovnu nainstalovat na různých platformách.

Po úspěšné instalaci obou knihoven lze zkompilovat samotný program. Kompilaci vřele doporučuji provést programem *CMake*, protože v tomto případě je instalace velmi jednoduchá. Instalaci lze provést buď pomocí grafického uživatelského rozhraní nebo pomocí příkazové řádky. V případě kompilace přes příkazovou řádku stačí být ve složce se zdrojovými kódy `src` a odtud zavolat následující tři příkazy:

```
mkdir ../build
cd ../build
cmake ../src
```

Pokud se překlad zdaří, vzniknou ve složce `build/FibreOperation` spustitelné soubory `FibreOperation` a `Generator`. Spuštění těchto programů je popsáno v následující sekci A.2

A.2 Spuštění programu

V předchozí sekci je uveden postup, jak program přeložit, a kde se nachází spustitelné soubory; nyní je lze spustit.

Prvním programem je program `Generator`, který se spouští bez jakýchkoliv povinných i nepovinných parametrů. Jeho během vzniknou v pracovním adresáři soubory `af0.vtk`, `bf0.vtk` a `m0.vtk`, které lze použít jako testovací data.

Druhý program `FibreOperation` je program samotné registrace. Vyžaduje jeden povinný parametr, cestu do složky s validními daty. Tato složka musí obsahovat přinejmenším soubory `af0.vtk`, `bf0.vtk` a `m0.vtk`. Pokud spustíme program s tímto parametrem, program začne pracovat, což lze vidět na standardním výstupu programu.

Na přiloženém DVD ve složce `Data` se nachází testovací data, která lze použít pro spuštění programu, stačí jen zadat cestu do této složky a vybrat jednu z příslušných podsložek. Tento program má další nepovinné parametry, kterými lze měnit konfiguraci registrace. Parametry se zadávají ve formátu `--x=y`, kde `x` je název parametru a `y` je numerická hodnota. Dostupné parametry a jejich přednastavené hodnoty jsou uvedené v tabulce A.1.

Název parametru (`x`)	Přednastavená hodnota (`y`)
`rigid`	1000
`smooth`	100
`confidence`	100
`correrspondence`	0.1
`begin`	1
`end`	2000
`step`	2
`resolution`	16

Tabulka A.1: Parametry programu a jejich hodnoty.

Nastavením parametrů `rigid`, `smooth`, `confidence` a `correspodence` je umožněna změna penalizací. Čím je hodnota parametrů větší, tím větší je penalizace. Například pokud zvýším hodnotu `rigid` z původní hodnoty 1000 na hodnotu 10000, bude se více penalizovat, když transformace nebude rigidní, tím se tedy docílí, že algoritmus bude preferovat zachování tvaru na úkor jiných vlastností.

Parametry `begin`, `end` a `step` ovlivňují počátek, konec a počet iterací. Během každé iterace vzniká hodnota, která je v první iteraci rovna hodnotě `begin` a násobí se po každé iteraci hodnotou `step`, dokud tato hodnota nepřesáhne hodnotu `end`. Hodnota, která se během iterací mění, dělí penalizační konstanty `rigid`, `smooth` a `confidence` tak, aby bylo registrovanému objektu umožněno se postupně více a více deformovat, což způsobí nejprve rigidní registraci a následnou nerigidní transformaci. Čím tedy bude hodnota iterace větší, tím více bude umožněno registrovanému objektu se deformovat. V původním algoritmu bylo navrženo sedm iterací, tato hodnota musela být upravena na deset, aby lépe vyhovovala vstupním datům.

Hodnota `resolution` ovlivňuje hladkost registrovaného objektu. Čím je parametr `resolution` větší, tím je registrace přesnější, ale je výrazně pomalejší.