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

Diplomová práce

Vzájemná transformace 3D objektů reprezentovaných trojúhelníkovým povrchem

Plzeň, 2011 Bc. Ivo Zelený

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

V Plzni dne 23.*4.2011, Ivo Zelený,*

Abstract

The diploma thesis is focused on mesh transformation between two meshes in applications of virtual reality (VR). The work is divided into two parts. A survey of methods for transforming and deforming meshes as morphing, mass-spring system or mesh registration are situated in the first part. There is also discussed problem of muscle wrapping. The description of new mass-spring system method on GPU, used morphing approach and new muscle wrapping method and performed tests are described in the second part. The appendices of this thesis contain the user guide and slide show of proposed muscle wrapping algorithm. The proposed new system for muscle wrapping is based on the morphing template expressed as mass-spring system into mesh of muscle. During interpolation fibers, which are connected by springs, mapped into muscle mesh too. Although the system works well on generated data in the test application. The implementation in OpenMaf and testing on real data shows some problems, which was discussed in the end of work. Systems were implemented in C++ and OpenGL and was able to achieve result in seconds when running on Intel core 2 quad 2,4GHz, 4GB memory and graphics card NVidia geforce 9600 GT with 512 MB memory and Windows 7.

Obsah

Abstract....4
1 Úvod....6
2 Morphing....7
2.1 Transformace povrchu....7
2.1.1 Korespondence vrcholů....8
2.1.2 Interpolace....12
2.2 Registrace povrchů....13
2.3 Transformace objemu....14
3 Systémy pružin....15
3.1 Výpočty v systémech pružin....15
3.2 Princip výpočtu na GPU....18
3.3 Příklady aplikací systému pružin....19
4 Metoda RANSAC....22
5 Problém dekompozice svalů na svalová vlákna....24
5.1 Anatomie svalů....24
5.2 Svalová šablona....24
5.3 Stávající přístup pro dekompozici....25
5.4 Analýza problému dekompozice svalů na svalová vlákna....26
6 Návrh a implementace....29
6.1 Volba programovacího jazyka a prostředků....29
6.2 Implementace systému pružin....29
6.2.1 CPU řešení....29
6.2.2 GPU řešení....30
6.2.3 Fixace povrchových bodů....32
6.3 Implementace transformace povrchů....33
6.4 Testovací aplikace....37
6.5 Integrace do OpenMAF....38
7 Testy a porovnání....40
7.1 Testy systému pružin....40
7.1.1 Výpočetní a paměťové nároky....40
7.1.2 Porovnání modelu-6 a modelu-26....42
7.1.3 Nastavení parametrů systému....44
7.2 Testy morphingu....47
8 Závěr....50
Použité zdroje....52
Příloha 1 – průběh dekompozice na svalová vlákna....54
Příloha 2 – uživatelská příručka....55

1 Úvod

V reálném světě téměř neustále dochází ke změnám jako je posun objektů, jejich rotace a také deformace. Proto se i počítačové vědy a hlavně počítačová grafika zabývají jejich simulací a provedením. Deformace se od ostatních liší především změnou povrchu objektu. Lze si ji také představit jako dvě různá tělesa, mezi kterými je zapotřebí převést jejich reprezentaci. Z pohledu počítačové grafiky nebo výpočetní geometrie se jedná o vzájemnou transformaci objektů neboli morphing. V případě, že jsou obě tělesa reprezentována trojúhelníkovou sítí, jde o interpolaci vrcholů jednoho na povrch druhého. Základními kameny pak jsou: nalezení správné korespondence mezi vrcholy a následná interpolace.

První část práce obsahuje především seznámení s existujícími metodami pro morphing objektů reprezentovaných povrchem. Kromě těchto metod existují také metody pro transformaci objektů vyjádřených volumetrickými daty (objemem), které jsou taktéž v první části zmíněny. Je zde také nastíněn postup RANSAC pro transformaci objektů a popsán systém pružin, který se využívá při simulacích pro deformace objektů. V realizační části je pak navržen postup pro transformaci objektu reprezentovaného objemem (svalová šablona) do objektu reprezentovaného povrchem (sval), který se opírá o techniky morphingu povrchu a systému pružin. Následuje kapitola měření, kde jsou shrnuty veškeré naměřené výsledky a pozorování.

Důvodem vzniku práce bylo vytvořit nástroj pro dekompozici svalů reprezentovaných trojúhelníkovou sítí na svalová vlákna. Nástroj by pak sloužil pro tuto činnost v rámci projektu VPHOP (Virtual Physiological Human OsteoPorotic) pro výzkum osteoporózy. Osteoporóza se stává vážným problémem současné populace. Výzkumy ukazují, že 30 až 50 procent žen a 15 až 30 procent mužů se s touto nemocí během života setkají. Jen v Evropě dochází každoročně ke 4 000 000 zlomeninám. Po komplikacích v souvislosti s osteoporózou pak dochází i k úmrtí téměř 250 000 z nich. Předpovědi do budoucna jsou skeptické. Předpovídají až dvojnásobný nárůst do roku 2050. Z těchto důvodů vznikl projekt VPHOP pod záštitou Evropské unie, který má za cíl vytvořit prostředky pro rozpoznání rizika zlomeniny u pacienta v rámci prevence a předejít tak zlomeninám a následným komplikacím. Do projektu je zapojeno několik institucí z 5 států Evropské unie, jednou z nich je i Západočeská univerzita v Plzni. Vyvíjí se jak zařízení pro vyšetření (scannery a přístoje pro měření zátěže), tak i programové vybavení, na kterém se podílí univerzity v Plzni, Lutonu a Itálii. Programové vybavení pro konkrétního pacienta a jeho snímky ze scanneru (dva ortogonální snímky) přizpůsobí atlas (model člověka, obsahující kosti a svaly reprezentované povrchem, doplněný o sémantické informace jako spojení kostí, jejich pohybové omezení a podobně) a dynamická data pro pohyb uzlových bodů (kloubů a význačných bodů). Svaly pak budou dekomponovány na svalová vlákna, která budou vstupem řešiče pro výpočet namáhání kostí. Část projektu dekompozice svalů navazuje na předešlý projekt, ve kterém byla tato část řešena. Popis postupu a výsledky tohoto přístupu jsou uvedeny na závěr první (teoretické) části práce. Bohužel aplikace tohoto postupu na různé typy svalů ukázala jisté nedostatky, které má tato práce za cíl odstranit.

2 Morphing

První metody morphingu se zabývaly transformacemi digitálních obrázků. Převod spočívá ve třech krocích stanovení korespondence, transformace pixelů a převzorkování, interpolace barev (blending). V morphingu obrázků většinou stanovuje uživatel korepondenci (automatické metody mohou pouze rozpoznávat podobně barevné části nebo podobné hrany, což nemusí dávat očekávané výsledky). Výsledek pak závisí především na zkušenostech animátorů a uživatelů aplikace. Jednou z metod, kterou ve svém článku [BEI1992] popsali Beier a Neely, je použití uživatelsky definovaných úseček, které si odpovídají ve zdrojovém a cílovém obrázku. Pozice bodů korespondujících úseček pak definují transformaci (warping) složenou z translace (přesun počátečního bodu úsečky), rotace (otočení okolo počátečního bodu) a scalingu (poměr délek úseček ve zdrojovém a cílovém obrázku). Transformace se aplikuje na pixely a následně se výsledek převzorkuje, abychom dostali opět rastrový obrázek. Kromě transformace pozice pixelů se také musí provést blending barev mezi pixely původního a cílového obrázku na stejných pozicích. V článku je také popsán postup pro transformaci při použití více úseček, který je založen na definování vah. Kromě možnosti využití jako zajímavého efektu se morphing obrázků využívá například při generování videa jako spojitého přechodu mezi dvěma snímky (dva diskrétní stavy). Zásadním problémem morphingu obrázků je osvětlení, které je závislé na tvaru objektu. Bude-li docházet ke změně tvaru budou se měnit normály a zastínění, což není možné korektně zajistit pomocí 2D morphingu. Tyto důvody vedly k vývoji metod morphingu ve 3D, tedy již ve fázi těles a transformací jejich reprezentací. Kromě vytvoření zajímavých animací, jako vývoj jednoho tělesa v druhé (například vývoj živočišných druhů), se hojně využívá v generování počítačových animací obličejů. Velmi dobrý rozbor všech druhů morphingu spolu s rozpracováním otázky multimorphingu (kombinování více těles podle vah do jednoho tělesa) mohou čtenáři najít v disertační práci Jindřicha Paruse [PAR2009].

2.1 Transformace povrchu

Povrch je v aplikacích počítačové grafiky nejčastěji reprezentovaný trojúhelníkovou sítí, protože je tato reprezentace široce podporována grafickými kartami. Při transformaci povrchu tak hledáme transformaci vrcholů ležících na povrchu jednoho tělesa na povrch druhého. Také zde rozlišujeme tři základní kroky morphingu: definování stejných oblastí uživatelem (tedy kontrola nad transformací), hledání korepondence mezi vrcholy (nebo pozice bodů jednoho tělesa na povrchu druhého) a interpolace mezi původními pozicemi bodů a novými. Z pohledu prvního kroku lze metody dělit na uživatelem kontrolované a na ty, které interakci nevyžadují.

Základní triviální metodou pro vzájemný převod těles se stejným počtem vrcholů je interpoloce bodů se stejnými indexy. Metoda vlastně ignoruje část hledání korespondence vrcholů. Tento přístup je obvykle realizován v modelovacích nástrojích jako jeden z nástrojů. Velmi častým využitím je pro animaci obličejů. Animátor navrhne obličej modelu pro jeden klíčový snímek (významný snímek, ve kterém jsou definovány objekty, jejich pozice a transformace) animace a tentýž model pak upraví do jiné podoby v jiném klíčovém snímku. Například může animátor definovat obličej s úsměvem a obličej zamračený. Jelikož se jedná o tentýž model z hlediska počtu vrcholů, počtu hran, spojení vrcholů a indexů vrcholů, postačí pouze interpolovat. Morphingem se tak plynule vyplní volné snímky mezi oběma klíčovými snímky.

Pro transformaci odlišných těles s nedefinovanou korespondencí je zapotřebí sofistikovanější postup. Nejprve se vyvíjely metody pro triviální objekty typu konvexní polyhedron nebo starshape (obsahuje i konkávní oblasti, ale vždy platí, že uvnitř tělesa existuje jádro nebo-li oblast, ze které jsou všechny vrcholy tělesa viditelné). Jako nejlepší

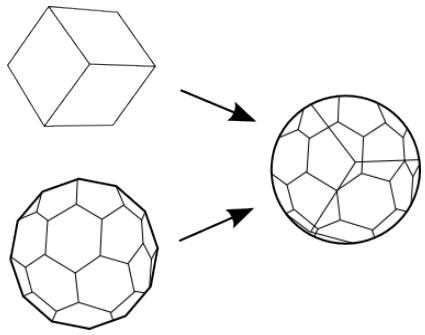
způsob se ukázala technika promítání a to buď jednoho tělesa na druhé a nebo pomocí průmětu na pomocné těleso. Teprve použitím této techniky nejprve Kent a spol. [KENT1992] a později Alexa [ALEX2000] představili metodu pro širokou skupinu objektů s genusem nula (uzavřené těleso bez děr). Teprve v nedávné době se objevila metody ([PRA2001] spolu s [HU2006]) pro tělesa s libovolným genusem využívající možnosti převodu tělesa do duálního parametrického prostoru a interpolace parametrů.

2.1.1 Korespondence vrcholů

Jak bylo uvedeno výše, ukázalo se jako vhodné použít techniku promítání. Celý problém korepondence vrcholů je řešen pomocí mezikroku a pomocného tělesa. Jako pomocné těleso se obvykle volí koule nebo disk, přičemž se opíráme o předpoklad, že všechna tělesa s genusem 0 jsou topologicky podobná kouli.

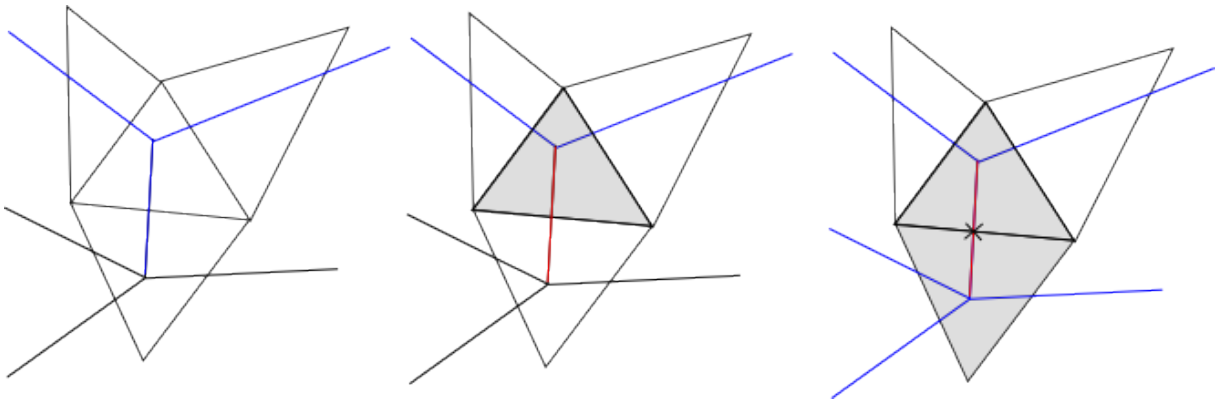
Kent a spol. [KENT1992] se nejprve zaměřili pouze na tělesa typu starshaped, kde, jak bylo uvedeno, lze vždy najít jádro tělesa a pak z libovolného bodu uvnitř jádra promítnou vrcholy povrchu na kouli. Dalším rozvíjením tohoto přístupu pak vyvinuli i metodu pro tělesa s genusem 0. Zejména se zaměřili na využití pružinových systémů (viz kapitola 3), přičemž techniky založené na takových systémech se podobají nafukování balónku. Tuhosti pružin se nastaví na malou hodnotu, aby těleso bylo pružné. Na body poté působí z vnitřku tělesa síly tlaku. Síly tlaku pomyslného vzduchu jsou aplikovány na středy jednotlivých trojúhelníků ve směru jejich normál, přičemž velikost síly proporcionálně roste vzhledem k obsahu plochy. Cílem je, aby došlo k převodu na těleso konvexní nebo typu starshaped. Problémy mohou nastat u cyklů krátkých hran, kde se při roztažení generují velké síly, které zabraňují dalšímu roztahování pružin. Navíc může při simulaci docházet ke stáčení, protínání, ke ztrátě informací ve smyslu zachování rozměrů trojúhelníků a poměrů mezi nimi. Tyto problémy vedly k vylepšení, které spočívá ve fixaci bodů na konvexní obálce. Ostatní body jsou připojeny pomocí pružin, kterým se vhodně nastaví počáteční délka tak, aby při simulaci došlo k vypnutí bodů na konvexní obálku. Ačkoli toto vylepšení minimalizuje problém stáčení a lépe zachovává poměry stran, může docházet ke kolapsům. Jeden typ simulace však dává permanentně dobré výsledky. Jedná se o konvexní oblast, do které se vypnou body vždy správně. Další vylepšení proto spočívá v dekompozici konvexní obálky na tyto oblasti a v nalezení neprotínajících se cest po povrchu modelu, které spojují dva body konvexní obálky, které jsou spojeny hranou v oblasti. Pokud lze takto model dekomponovat, lze ho i promítnout na jeho konvexní obálku a tedy i promítnout na kouli. Opět může dojít u komplexnějších modelů ke špatným výsledkům. Pro složité modely proto autoři navrhli uživatelem kontrolovaný morphing. Uživatel si sám zvolí konvexní oblasti tak, aby se dosáhlo nejlepšího možného výsledku. Například u postavy člověka by se jednalo o oblasti okolo ramen, okolo nohou u kyčlí a okolo krku.

Pokud dojde úspěšně k průmětu obou reprezentací povrchů na jednotkovou kouli se středem v počátku, následuje krok se sjednocením obou topologií. Sjednocení je důležité především, pokud požadujeme, aby se celý povrch (nejen body) přetransformoval přesně na povrch cílový. Význam je více zřejmý například při transformaci mezi velmi odlišnými tělesy o rozdílném počtu vrcholů jako krychle a mnohostěn na obrázku 2.1. Pokud bychom smíšení promítnutých reprezentací neprovedli a transformovali pouze vrcholy prvního tělesa, nedošlo by téměř k žádné změně tvaru tělesa (stále by se jednalo o šestistěn) během interpolace, ale pouze k přesunu osmi vrcholů. Během interpolace by tak nedošlo k přeměně jednoho tělesa v druhé.



Obrázek 2.1: Smíšení topologií krychle a mnohostěnu.

Problém smíšení topologií (znám také jako „map overlay“) je výpočetně nejnáročnější část morphingu povrchu. Myšlenka spočívá v ořezávání polygonu (většinou trojúhelníku) modelu *A* polygonem modelu *B*. Je proto nutné určit všechny průsečíky obou promítnutých topologií. V případě testování každých dvou hran složitost vyšplhá až na $O(N_A N_B)$, kde *N* značí počet hran modelu. Proto se autoři snažili optimalizovat především tuto část. Vylepšení spočívá ve vybudování datové struktury, kde hrana obsahuje seznam hran, které protíná, oba koncové body, oba trojúhelníky, které odděluje, a bod obsahuje původní pozici, promítnutou pozici a seznam hran, které z něj vedou. Algoritmus spočívá v nalezení trojúhelníku modelu *B*, který obsahuje první bod modelu *A* (v čase $O(N)$). K bodu uložíme index tohoto trojúhelníku. Hrany vedoucí z prvního bodu modelu *A* musí protnout hrany nalezeného trojúhelníku, pokud nějaké protínají, proto je přidáme do seznamu pracovních hran. Pro všechny pracovní hrany provádíme test s hranami trojúhelníku, který obsahuje počáteční bod hrany. Pokud je nějaká protnuta přidáme do seznamu kandidátů na protnutí další dvě hrany sousedícího trojúhelníku s protnutou hranou. Takto nalezneme i trojúhelník obsahující koncový bod hrany, který k bodu uložíme, a do seznamu pracovních hran přidáme nezpracované hrany vycházející z tohoto bodu. Obrázek 2.2 ukazuje průběh algoritmu. Složitost tohoto postupu je $O(N_A+I)$, kde *I* je počet průsečíků, přičemž v reálných aplikacích platí, že $I<<N_A N_B$. Po nalezení průsečíků je ještě nutné průsečíky pro každou hranu vhodně seřadit a pak provést triangularizaci.



Obrázek 2.2: Ukázka algoritmu pro hledání průsečíků. Modré hrany jsou v seznamu pracovních hran, červená je zpracovávaná hrana, označené trojúhelníky jsou takové, jejichž hrany jsou v seznamu kandidátů.

Situace při hledání průsečíků je navíc ztížena tím, že se nepohybujeme v rovině, ale na povrchu koule. Oproti rovině je komplikovanější například hledání průsečíků nebo test bod uvnitř polygonu. Hrana na povrchu koule je vlastně částí kružnice, jejíž středem je střed koule a která prochází oběma body hrany. Dále lze uvažovat, že kružnice leží v rovině. Průnikem obou rovin (pro obě kružnice obou hran) je přímka, která protíná kouli ve dvou bodech, které

jsou průsečíky obou kružnic. Výpočet těchto průsečíků pomocí vektorového součinu představuje vzorec 2.1, kde p_1 a p_2 jsou koncové body jedné testované hrany a q_1,q_2 jsou koncové body druhé hrany.

$$r = \pm(p_1 \times p_2) \times (q_1 \times q_2)$$

Vzorec 2.1: Nalezení průsečíku dvou kružnic na povrchu koule.

Průsečík leží na obou hranách, pokud s_p,s_q ve vzorci 2.2 leží v intervalu <0,1>.

$$t_p r = p_1 + s_p(p_2 - p_1)$$
$$t_q r = q_1 + s_q(q_2 - q_1)$$

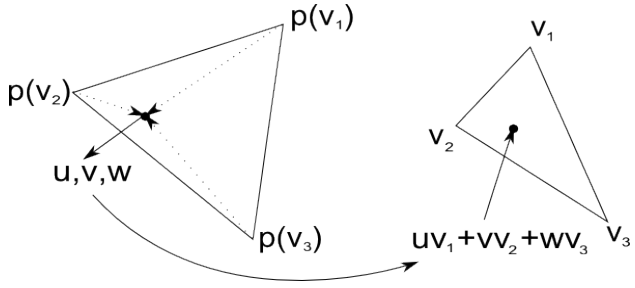
Vzorec 2.2: Test zda průsečík leží na obou hranách.

Druhým problémem je test bodu uvnitř polygonu, kterým je v našem případě trojúhelník, jenž je tvořen body *v1,v2,v3*. Aby platil následující vzorec 2.3, musí být body seřazeny proti směru hodinových ručiček (kvůli použití vektorového součinu, kde záleží na pořadí operandů).

$$((v_1 \times v_2)\, p \geq 0) \wedge ((v_2 \times v_3)\, p \geq 0) \wedge ((v_3 \times v_1)\, p \geq 0)$$

Vzorec 2.3: Test bod uvnitř trojúhelníku.

Doposud jsme získali potřebné body pro korespondenci na kouli. Pro převod souřadnic těchto bodů na povrch cílového tělesa slouží barycentrické souřadnice. Pomocí nich je možné vyjádřit libovolný bod uvnitř trojúhelníku jako lineární kombinaci bodů tvořící trojúhelník. Důvodem využití tohoto přístupu je nezávislost na transformaci tedy i projekci. Stejné barycentrické souřadnice reprezentují jak bod uvnitř trojúhelníku na povrchu koule, tak odpovídající bod uvnitř stejného trojúhelníku, který byl promítnut na povrch koule.



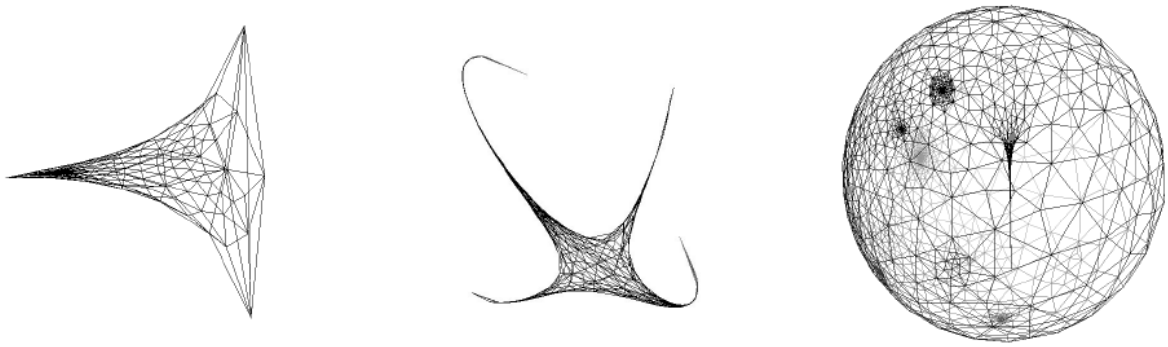
Obrázek 2.3: Barycentrické souřadnice.

Alexa ve svém článku také používá projekci povrchů obou těles na kouli, ale pomocí metody nazývané relaxace. Metoda je odvozena z teorie o grafech a hlavně rovinných grafů, což jsou takové grafy, které lze vždy zakreslit tak, aby se žádné dvě hrany nekřížily. Pro jejich překreslení bez křížení slouží právě relaxace, při které se musí zafixovat (jejich pozice se dále nemění) alespoň tři různé body, ostatní se vždy umisťují do středu jeho sousedů (vrcholy připojené hranou). V případě povrchu trojrozměrného tělesa se snažíme také překreslit jeho reprezentaci tak, aby se hrany nekřížily a aby vrcholy ležely na jednotkové kouli. Opět dochází k umístění bodu do středu jeho sousedů, spolu s tím je ale spojena normalizace na jednotkovou kouli. Označíme-li operaci normalizace vektoru v jako *(v)** můžeme relaxaci vyjádřit vzorcem 2.4.

$$\forall i, v_i = \left(\sum_{n \in sousedé\, v_i} v_n \right)^*$$

Vzorec 2.4: Výpočet relaxace.

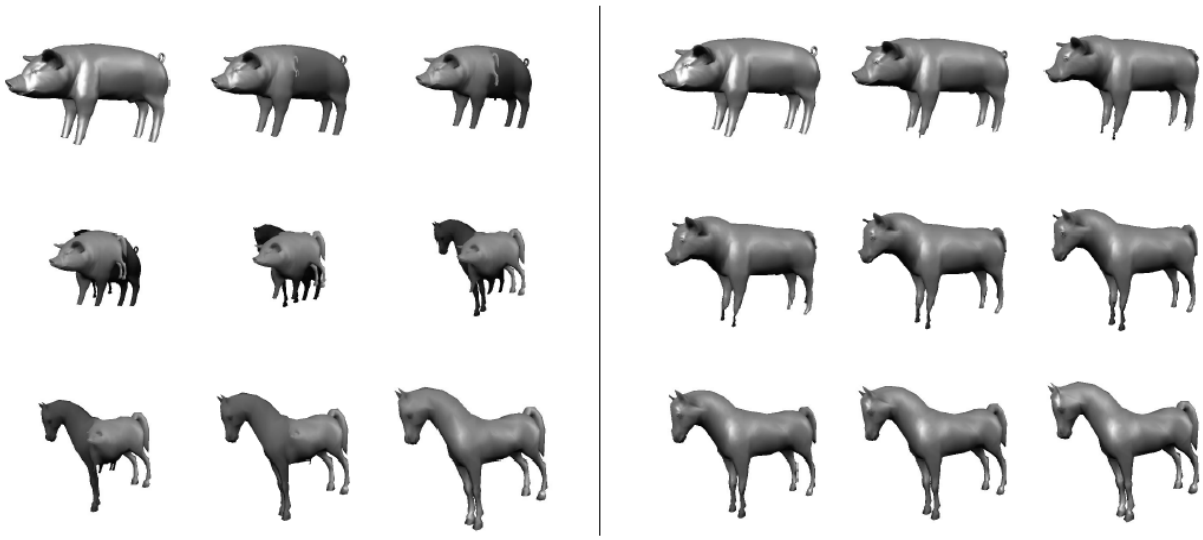
U prostorového případu je nutné stanovit alespoň čtyři vrcholy (nejlépe vrcholy pravidelného tetrahedronu) tak, aby neexistoval nezafixovaný bod v žádné polokouli pomocné jednotkové koule. Výběr fixovaných bodů u přístupu relaxace má zásadní vliv na výsledek, neboť může dojít ke kolapsu do nedefinovaného stavu. Kolaps je nutné detekovat a případně restartovat celý algoritmus. Ukázky kolapsu a zdařilé projekce jsou vyobrazeny na obrázku 2.4.



Obrázek 2.4: Projekce na kouli. Vlevo tři zafixované body, uprostřed čtyři zafixované body s kolapsem, vpravo korektní projekce. [ALEX2000]

Následujícím krokem je, stejně jako v předchozím přístupu podle Kent a spol., smíšení obou promítnutých topologií. Také zde se autor zaměřil na optimalizaci fáze hledání průsečíků. Přístup je obdobný, avšak s tím rozdílem, že je využita jiná datová struktura podobná okřídlené hraně, která následně umožňuje jednodušší sloučení topologií a následnou trinagularizaci. Opět i zde všechny body patřící prvnímu tělesu promítnuté na jednotkovou kouli vyjádříme jako barycentrické souřadnice.

Pro korektní transformaci mezi tělesy nesmí záviset na natočení a pozici význačných částí jako například hlava nebo končetiny. Bohužel automatické určení korespondence těchto částí je obtížné a v mnoha případech pouze uživatel zná vhodnou korespondenci pro konkrétní modely. Z těchto důvodů navrhl autor možnost ovlivnit průběh transformace pomocí určení klíčových bodů a algoritmus pro jejich zarovnání neboli feature aligning, který je zařazen po projekci obou těles na kouli. Spočívá v rotaci pomocné koule s jednou promítnutou topologií tak, aby se minimalizovala vzdálenost mezi odpovídajícími body. Poté dochází k fixaci těchto bodů a přesunu na jejich cílové pozice, přičemž při přesunu je nutné kontrolovat, zda není porušen požadavek na nepřekřížení hran a případně spustit relaxaci. Ukázka rozdílu mezi použitím (levá část obrázku) a nepoužitím (pravá část obrázku) feature aligning je vyobrazena na obrázku 2.5. V prvním případě, kde aplikace sice nevyžaduje žádnou interakci s uživatelem, působí interpolace (případně i animace) nerealisticky a dochází k řadě kolizí. Ve druhém získáme očekávaný průběh po specifikaci několika klíčových korespondujících částí (nohy, hlava, ocas).



Obrázek 2.5: Ukázka morphingu bez feature aligning (vlevo) a s ním. [ALEX2000]

Metoda s plnou kontrolou uživatele je popsána v článku [PRA2001]. Je založena na parametrizaci modelu. Uživatel specifikuje libovolný zjednodušený trojúhelníkový model s body a hranami spojujícími tyto vrcholy. Na modelech, které chceme transformovat, pak uživatel také provede výběr těchto klíčových bodů (nebo jsou nalezeny pomocí rozpoznávání hlavních komponent), čímž je stanovena korespondence, pro kterou zároveň platí, že se jedná o korespondenci bod k bodu. Pro každou hranu zjednodušeného modelu se nalezne odpovídající cesta po povrchu obou modelů. Tím jsou stanoveny takzvané kompatibilní reprezentace. Jednotlivé oblasti vymezené hranami lze chápat jako parametrické plochy, které můžeme transformovat. Metodu lze použít pro transformaci libovolných objektů s libovolným tvarem a genusem.

2.1.2 Interpolace

Interpolace je většinou realizována po přímce jako lineární interpolace mezi původní pozicí bodu na prvním tělese a pozicí odpovídajícího bodu na povrchu tělesa druhého. Je řízena parametrem z intervalu od nuly do jedné, kde nula znamená původní pozici a naopak jedna cílovou. V některých aplikacích není vhodné užít lineární interpolaci, proto přicházejí v úvahu buď parametricky vyjádřené křivky (například Hermitův spline) nebo nelineární krok parametru. Jedná se zejména o případy, kdy uživatel požaduje, aby během interpolace nedocházelo k sebeprotínání nebo kolizím mezi částmi tělesa. Protože je velmi obtížné volit cestu křivky automaticky tak, aby byly tyto podmínky splněny, jsou téměř vždy specifikovány uživatelem.

Kromě interpolace mezi pozicemi bodů můžeme požadovat i interpolaci dalších parametrů jako barvy, průhlednosti, texturovací souřadnice. V takovém případě se postupuje obdobně, ale stejně jako u určení cílové pozice, je nejprve nutné vhodně stanovit cílovou hodnotu těchto parametrů. V případě, že si počet vrcholů neodpovídá a korespondence vrcholů není vrchol k vrcholu, ale vrchol k pozici uvnitř trojúhelníku, musíme například barvu také zkombinovat z hodnot vrcholů trojúhelníku. K tomu je možné opět využít určené barycentrické souřadnice.

Existují také speciální typy interpolací zachovávající tvar či objem během převodu. Jedna z takových metod byla popsána v článku [HU2006] pány Hu, Liu a Wang. Metoda je z principu vhodná pro reprezentace bod k bodu, mezi kterými pomocí duálního laplacian prostoru provádí interpolaci. Sestává se ze čtyř kroků. Prvním je právě získání kompatibilní reprezentace (například použitím metody parametrizace modelů [PRA2001]). Druhým

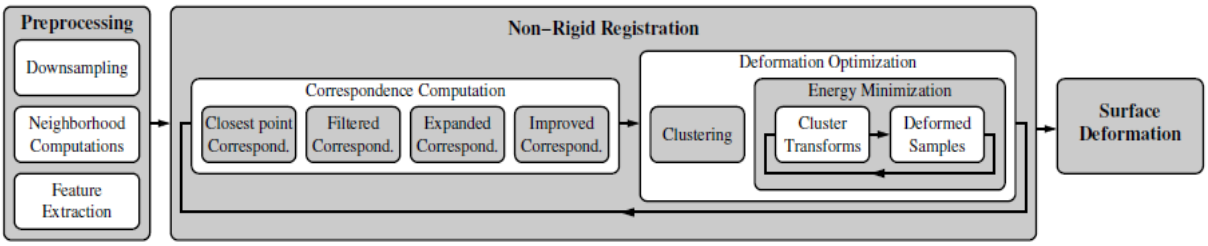
krokem je získání duálních reprezentací pomocí aplikace laplacova operátoru na množinu vrcholů. Aplikací operátoru získáme pro každý bod vektor odpovídající vážené sumě vektorů vedoucích z právě zpracovávaného bodu do bodů k němu připojených hranou. Tento vektor popisuje lokální geometrii. Pro získání vhodných vah jednotlivých hran existuje mnoho přístupů. Autoři zvolili součet cotangensů úhlů, které svírají zbývající hrany obou trojúhelníků obsahujících tuto hranu. Váhy hran zachycují tvar původního modelu. Třetím krokem je interpolace mezi duálními reprezentacemi a čtvrtým pak je rekonstrukce mezimodelu. V článku je ukázáno, že lze dosáhnout lepších výsledků v mezimodelech co se týče zachování tvaru a objemu. Autoři také uvádějí, že interpolací v duálním prostoru lze dosáhnout větší stability celého procesu.

2.2 Registrace povrchů

Mezi vzájemné transformace povrchů lze zahrnout kromě morphingu dvou těles i takzvanou registraci těles. V tomto případě se často jedná o velmi podobná tělesa (v některých případech totožná), mezi kterými je potřeba nalézt vhodnou transformaci. Metody řešící tento problém lze rozdělit do dvou základních kategorií a to registrace pevných těles (rigid) a registrace s možností deformace a ohybu (no rigid).

Reprezentantem první kategorie je technika iterative closest point (ICP), jejíž popis a rozbor jejích jednotlivých kroků lze najít například v článku [RU2001] od Rusinkiewicze a Levoye. ICP technika je široce používaná zejména pro registraci výstupů ze 3D scannerů. Skládá se ze šesti základních kroků: selekce, korespondence, vážení, eliminace, stanovení chyby a minimalizace chyby. Pro správný průběh algoritmu potřebujeme počáteční odhad transformace, kterou se budeme snažit vylepšit. Pro její stanovení můžeme například nalézt hlavní osy objektů, klasifikovat stejné oblasti nebo využít znalost o modelu (například směr scannování). Selekce spočívá v náhodném nebo pravidelném vzorkování povrchu jednoho či obou objektů. Korespondence pak sestává z využití odhadu transformace pro převod získaných vzorků blízko druhému povrchu a následně z nalezení korespondujícímu bodu (například nejbližší bod). Vážit body ve třetím kroku lze mnoho způsoby nejčastěji buď konstantními vahami nebo podle vzdálenosti mezi korespondujícími body. Krok eliminace slouží k odstranění chybných bodů (vzniklých například chybou měření), které by mohli způsobit chybný výpočet v následujících krocích. Nejčastěji se odstraňují všechny vzdálenější body než stanovená mez nebo určitý zlomek nejhorších bodů (například nejhorších deset procent). Stanovení chyby a její minimalizace se opírá hlavně o sumu čtverců vzdáleností a její minimalizaci opakováním celého procesu, kde jako odhad transformace používáme aktuální nalezenou transformaci, kterou tím neustále vylepšujeme.

Huang a spol pak ve svém článku [HUA2008] navrhli sofistikovanou metodu pro druhou kategorii tedy no rigid registrace. Ve své postatě se jedná o rozšíření metody pro rigid registraci o hledání oblastí a clusterizaci. Vlastní registraci předchází fáze předzpracování, která pro velmi objemné sítě sníží počet bodů na reprezentativní množinu, pro kterou nalezne aktuální sousednosti a detekuje významné oblasti. Následuje fáze výpočtu korespondence podobná rigid registraci. Dále pak hlavní rozdíl oproti rigid tedy určení deformace a nakonec transformace všech bodů originálního objektu. Pro určení deformace nejprve autoři provedou clusterizaci, což je určení reprezentativního bodu pro každou množinu bodů, pro kterou platí, že všechny její body lze transformovat stejnou cestou. Celý algoritmus je zachycen na obrázku 2.6.



Obrázek 2.6: Zobrazení algoritmu non-rigid transformace. [HUA2008]

2.3 Transformace objemu

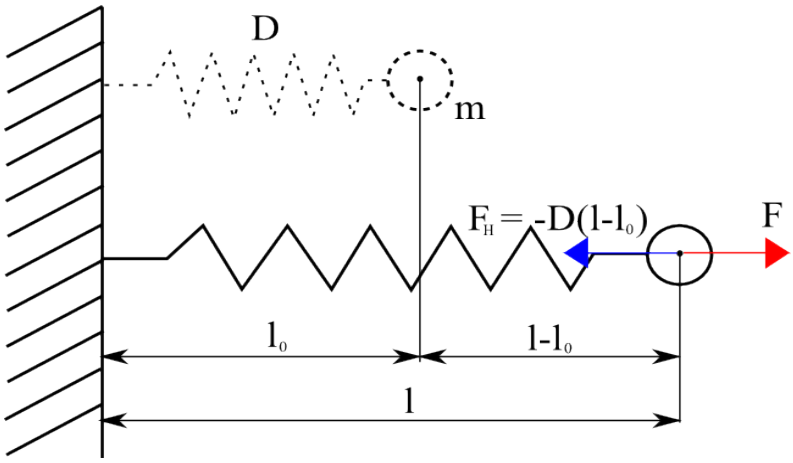
Jedná se o techniky zaměřené na objekty reprezentované volumetrickými daty. Většina postupů vychází z myšlenky, že jsou volumetrická data uspořádána do mřížky podobně jako rastrové obrázky. Jedná se tedy o diskrétní hodnoty v trojrozměrné mřížce, což si lze představit jako obrázek s dalším rozměrem navíc. Proto zde můžeme uplatnit techniky podobné 2D morphingu. Výhody oproti transformacím těles v prostoru reprezentovaných trojúhelníkovými sítěmi (viz výše) jsou hlavně v nezávislosti na tvaru objektu, jeho genusu či triangulaci povrchu (velikosti a tvary trojúhelníků). Autoři také uvádějí myšlenku provedení vzájemného převodu mezi volumetrickými daty a trojúhelníkovou sítí, a proto jsou tyto techniky obecnější. Také zde je možné rozlišovat mezi automatickými metodami, které vyhledávají podobné části, aby docházelo ke korektnímu morphování, a uživatelem ovládánými metodami. Oproti předchozím typům morphingu však nastává problém s interakcí s uživatelem a vizualizací volumetrických dat tak, aby uživatel mohl jednoduše specifikovat korespondující oblasti. Na základě definovaných oblastí se provede warping prvního modelu na druhý a opačně. Také zde se aplikuje zároveň s přesunem voxelů blending hodnot volumetrických dat.

3 Systémy pružin

Při transformacích povrchů může vzniknout požadavek na lokální deformaci. Například můžeme znát transformaci některých klíčových bodů nebo zjednodušeného modelu a potřebujeme deformovat či transformovat ostatní body na povrchu modelu. Právě pro takové deformace lze využít systémy pružin (v anglické literatuře Mass Spring Systems), které vycházejí z fyzikálního opodstatnění, neboť tělesa se skládají z atomů a molekul, které jsou vzájemně vázány silami. Nejčastějším využitím na poli počítačové grafiky však našly systémy pružin v realisticky vypadající fyzikální simulaci elastických těles jako látky (oblečení), kůže, svaly a také při simulaci různých jemných porostů jako travin či vlasů. Zásadní výhodou jejich aplikace je nízká výpočetní náročnost (v závislosti na počtu vrcholů a pružin). Nabízejí tedy možnost simulace v reálném čase, a proto i možnost využití na poli virtuální reality (VR).

3.1 Výpočty v systémech pružin

Systém se skládá z množiny bodů a množiny pružin, které spojují body a slouží tak k vzájemné interakci mezi body. Každý bod je definován svou pozicí, hmotností a útlumovým parametrem pro snížení setrvačnosti. Pružina je pak určena dvěma vrcholy, které spojuje (většinou indexy těchto bodů), počáteční délkou a tuhostí pružiny. Definujeme-li počáteční délku pružiny jako vzdálenost mezi oběma koncovými body, je takový stav stabilním a nedochází k žádným pohybům. Dojde-li k pohybu bodu (základy dynamiky hmotného bodu a pružin lze najít například na stránkách vysokého učení v brně [MAL2005]), způsobí tato změna změnu délky pružiny, čímž vznikne v pružině síla působící na její koncové body za účelem vrátit pružinu, respektive její délku, do původního stavu (počáteční délky), což je ilustrováno na následujícím obrázku 3.1.



Obrázek 3.1: Ilustrace pružiny, jejích základních parametrů a sil dle Hookova zákona.

Pro výpočet síly působící v pružině slouží Hookův zákon. Jeho vektorovou podobu popisuje následující vzorec 3.1.

$$F_{ij} = F_{ij}(x_i, x_j) = D_{ij} \frac{|l_{ij}| - |l_{ij}^0|}{|l_{ij}|} l_{ij}$$

Vzorec 3.1: Rovnice pro výpočet síly v pružině.

Kde dolní indexy *i* a *j* slouží pro označení koncových bodů (x_i,x_j) pružiny, *D* je tuhost

pružiny a l je rozdílový vektor mezi oběma koncovými body. Svislé závorky představují výpočet délky vektoru. Horní index 0, pak označuje původní délku pružiny.

Kromě síly působící v pružině na hmotný bod musíme uvažovat ještě pohybovou rovnici bodu obsahující aktuální rychlost a zrychlení bodu. Výslednou diferenciální rovnici druhého řádu pro jeden hmotný bod, kde m představuje hmotnost bodu, k tuhost pružiny a c tlumící koeficient, lze zapsat následovně (Vzorec 3.2).

$$m\ddot{x} + c\dot{x} + k\,x = 0$$

Vzorec 3.2: Diferenciální rovnice pro výpočet pohybu hmotného bodu.

Řešení této diferenciální rovnice ukazuje vzorec 3.3.

$$x = C_1 e^{\lambda_1 t} + C_2 e^{\lambda_2 t}$$

Vzorec 3.3: Řešení diferenciální rovnice.

Kde $\lambda_{1,2}$ je řešení charakteristické rovnice (viz vzorec 3.4), které lze určit následovně podle vzorce 3.4.

$$m\lambda^2 + c\lambda + k = 0, \lambda_{1,2} = \frac{-c \pm \sqrt{c^2 - 4\mathrm{km}}}{2\mathrm{m}}$$

Vzorec 3.4: Charakteristická rovnice a její kořeny.

Hodnota diskriminantu (výrazu pod odmocninou) určuje chování pružiny. Bude-li diskriminant kladný, bude mít rovnice reálné kořeny a pohyb nebude kmitavý. Naopak záporné znaménko bude znamenat výskyt imaginární části, kořeny budou komplexně sdružené a pohyb bude kmitavý. Položíme-li diskriminant rovný nule a vyjádříme-li tlumící koeficient, získáme takzvané kritické tlumení (Vzorec 3.5).

$$c = \sqrt{4\mathrm{km}}$$

Vzorec 3.5: Kritické tlumení.

Kritické tlumení způsobuje nekmitavý pohyb, přičemž platí, že k ustálení dojde v nejkratším možném čase.

Podle požadovaného chování systému lze kromě tlumení volit i další parametry systému, např. tuhost pružiny a hmotnost bodu. Nelze však zvolit parametry naprosto libovolně, neboť v závislosti na jejich volbě může systém buď konvergovat (reálná část kořenů charakteritické rovnice je záporná) do stabilního stavu či divergovat (reálná část je kladná). Divergencí v tomto případě rozumíme stav, kdy se body rozkmitají natolik, že je nelze utlumit. Jako tuhost pružiny se obvykle doporučuje zvolit převrácenou hodnotu její délky, protože nejlépe zachovává poměry mezi pružinami a také jejich původní délky (příklad jiného stanovení hodnoty tuhosti viz kapitola 3.3).

Celý systém s více hmotnými body spojenými pružinami popisuje níže uvedená soustava rovnic (Vzorec 3.6), kde m představuje hmotnost bodu a c tlumící koeficient. Při praktických aplikacích mohou navíc na celý systém působit vnější vlivy a tedy i další síly (gravitační síla, síla větru, kolize). Výslednice těchto sil pro každý bod je označena F^{ext} a představuje pravou stranu rovnice.

$$m_i \ddot{x}_i + c \dot{x}_i + \sum_{j \in N_i} F_{ij}(x_i, x_j) = F_i^{ext}$$

Vzorec 3.6: Soustava rovnic pro výpočet systému pružin.

Pro výpočet řešení diferenciální rovnice musíme dále aplikovat některou z numerických metod pro jejich řešení. Jednou z takových metod je Verletova integrace, která převádí diferenciální rovnice na rekurzivní tvar. Derivace se nahrazují jejich diferenčním zápisem (například použití centrální diference pro druhou derivaci a zpětné diference pro první derivaci viz vzorec 3.3), čímž dojde k eliminaci závislosti na rychlosti a zrychlení, ale za cenu horší přesnosti. Nová pozice bodu tak závisí na pozici ve dvou předchozích iteracích. Verletova integrace byla navržena pro výpočet na počítači, neboť se opírá o předpoklad, že probíhá stejně jako chod počítače v diskrétních krocích. Vhodné je však také zmínit, že touto úpravou dojde k zanesení dalšího parametru systému (*dt*), který má vliv nejen na přesnost výpočtu, ale také přímo na konvergenci. Volba příliš krátkých intervalů kroku času *dt* má za následek vysoké výpočetní nároky. Naopak příliš dlouhé intervaly způsobí nedostatečnou konzistenci mezi následujícími iteracemi a následně i divergenci.

Zápis zrychlení a rychlosti pomocí Verletovy integrace potažmo pomocí diferenciálního zápisu ukazuje vzorec 3.7.

$$\ddot{x}(t) = \frac{x(t-dt) - 2\mathrm{x}(t) + x(t+dt)}{dt^2}, \dot{x}(t) = \frac{x(t) - x(t-dt)}{dt}$$

Vzorec 3.7: Verletova integrace.

Po dosazení vzorce 3.7 do vzorce 3.6 bude výpočet nové pozice bodu vypadat následovně.

$$x_i(t+dt) = \frac{F_i^{tot}(t)}{m_i} dt^2 + 2x_i(t) - x_i(t-dt)$$
$$, kde\ F_i^{tot}(t) = F_i^{ext} - \sum_{j \in N_i} F_{ij} - c\frac{x_i(t) - x_i(t-dt)}{dt}$$

Vzorec 3.8: Výpočet nové pozice bodu po aplikaci Verletovy integrace.

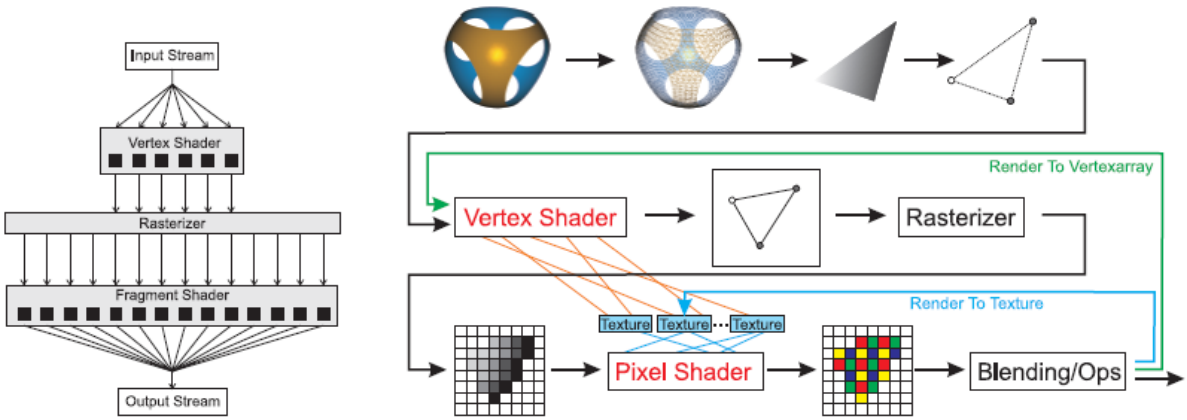
Ze vzorce 3.8 je vidět, že zvolíme-li při výpočtu konstantní krok (*dt*), je nutné pouze uchovávat aktuální pozici a pozici předchozí pro výpočet nové. Tato formulace je také ideální pro použití na GPU viz dále.

Posledním a neméně důležitým faktorem, který má vliv na výpočet, stabilitu, průběh a chování simulace, je diskretizace modelovaného objektu. Například použijeme-li stejné parametry časového kroku, hmotnosti, tlumení a tuhosti na dvě různé diskretizace téhož objektu, získáme odlišné chování, které dokonce může být i nestabilní. Proto je vhodné volit parametry pružin v závislosti na délce pružin (často se jedná o převrácenou hodnotu délky pružiny jako tuhost) a parametry bodů podle objemu nebo plochy, kterou reprezentují (viz kapitola 3.3).

3.2 Princip výpočtu na GPU

Pokud aplikace požaduje simulaci s velkým množství pružin a bodů (řádově desetitisíce vrcholů), nebude pravděpodobně možné ji v reálném čase počítat pouze na CPU. Tímto problémem se zabývali Georgii a Westermann [GEO2005] a navrhli dva přístupy, jak počítat systém pružin pomocí grafické karty.

Protože se architektury CPU a GPU významně odlišují, je nutné přizpůsobit výpočet a datové struktury. GPU architektura je typu SPMD, to znamená, že na základě stejného programu pracuje více procesorů nad různými daty a výpočet tak probíhá paralelně. Navíc instrukční sada GPU obsahuje i vektorové instrukce, protože se zaměřuje na vrcholy a práci s nimi v prostoru. Kvůli velkým přenosům dat má také GPU výhodu v efektivních přístupech do paměti a práci s ní (paměť grafické karty). Původně nebyla možnost grafickou pipeline programovat, ale postupem času se přistoupilo i k programovatelným jednotkám na grafických kartách. Nejprve bylo umožněno používat vertex shader (program pro zpracování vrcholů) a fragment shader (program pro zpracování fragmentu po rasterizaci primitiv). Kvůli dalším potřebám pak vznikaly i další programovatelné jednotky jako geometry shader (program pro zpracování primitiv), tesselation shader či compute shader. Průchod grafickou pipeline znázorňuje obrázek 3.2.



Obrázek 3.2: Grafická pipeline. [GEO2005]

Máme-li napsán program pro grafickou kartu (shader), který produkuje požadovaný výpočet například postprocessing obrázku, řešení soustavy rovnic nebo systému pružin, je nutné ještě výpočet spustit. Ke spuštění vždy dochází při existenci dat. Nejčastěji je jádro výpočtu situováno do fragment shaderu, kde dochází ke spuštění při přítomnosti fragmentů vzniklých rasterizací renderovaného tělesa, proto obvykle vykreslíme čtverec zarovnaný s obrazovkou, čímž dojde k existenci dat ve všech pixelech zdrojové textury s daty i cílové textury pro výsledek.

Jelikož vždy dochází ke zpracování pouze jednoho vrcholu (v případě vertex programu) či jednoho fragmentu (v případě fragment programu), nemá vrchol nebo fragment znalost o svých sousedech nebo svých stavech. Tyto dodatečné informace je možné uložit pouze do textur, ke kterým je pak možné přistupovat. V případě častých přístupů sehrává svou roli i cache paměť pro textury, ze které je čtení řádově rychlejší (i více jak 10x). Čili časté čtení z textury (například pro jeden fragment vícekrát) z různých míst (ne v blízkém okolí) má zásadní dopad na rychlost celého výpočtu.

Více než v jiných případech je důležité zvolit vhodnou strukturu pro popis systému. V článku jsou popsány dva přístupy: bodově orientovaný a hranově orientovaný. Oba postupy jsou porovnány z hlediska paměťových nároků, rychlosti a použití. První možností je bodově orientovaný přístup, kde vrcholy mají znalost o pružinách, které jsou k nim připojeny.

Výhodou přístupu je možnost jediné iterace pro výpočet, pokud počet texturovacích jednotek pro textury s uloženými pružinami nepřesahuje valenci vrcholů a jednodušší implementace. Základní zjevnou nevýhodou je dvojnásobný výpočet síly v pružině pro každý koncový bod. Pokud je různá valence vrcholů, je pak tento přístup extrémně paměťově neefektivní. Druhý přístup je hranově orientovaný, kde výpočet síly v pružině probíhá pouze jednou a výpočet je rozdělen do dvou základních iterací: výpočet síly a výpočet nové pozice bodů. Avšak je zde problém s mapováním síly pružin na body, neboť textury s pozicemi bodů a se silami mají naprosto odlišnou velikost (typicky 6x větší při valenci 6 pro trojúhelníkové povrchy). Tento problém je vyřešen vyrenderováním vrcholu pro každou pružinu na pozice obou jejích koncových bodů. Při této části výpočtu je povolen blending s funkcí jedna k jedné, aby se akumulovaly síly a výsledek uložen do textury akumulovaných sil. Při výpočtu nové pozice bodu se pak přistupuje pouze jedenkrát do každé ze tří textur: předchozí pozice, aktuální pozice a akumulované síly. Odstraňuje nevýhody prvního, ale urychlení, jak je vidět z výsledků v článku, není nikterak zásadní, neboť je zde náročná část přemapování sil, kde se využívá blendingu, což je ze své podstaty náročná funkce (musí se pokaždé číst již ukončený vyrenderovaný fragment).

3.3 Příklady aplikací systému pružin

Jedna z reálných aplikací je popsána v článku o využití systému pružin pro simulaci chování měkké tkáně dásní v kosmetické orthodontice [PHAN2008]. V závislosti na pohybech zubů (při rovnání a jiných kosmetických úpravách) dochází ke změnám i v dásňové tkáni a zubní lékař tak může s touto aplikací v reálném čase simulovat vývoj chrupu pacienta. Vstupem jejich aplikace je trojúhelníkový povrch reprezentující měkké tkáně a povrch přestavující zuby. Body představují hmotné body a každá hrana trojúhelníku představuje pružinu (pozn. každá pružina je obsažena ve dvou trojúhelnících). Důležitá je volba parametrů systému. Autoři zvolili nastavení hmotnosti bodu jako součin známé hustoty tkáně a součtu obsahů trojúhelníků, ve kterých je bod obsažen. Součin je navíc vydělen třemi, aby byla lépe aproximována hmotnost na jeden bod trojúhelníku. Tuhost pružiny je pak součin Youngova modulu (koeficient v závislosti na elastičnosti materiálu) a součtu obsahů trojúheníků obsahujících pružinu, vyděleno čtvercem délky pružiny. Tlumení bylo zvoleno na základě testování. Vzorce pro výpočet všech tří parametrů jsou shrnuty ve vzorci 3.9.

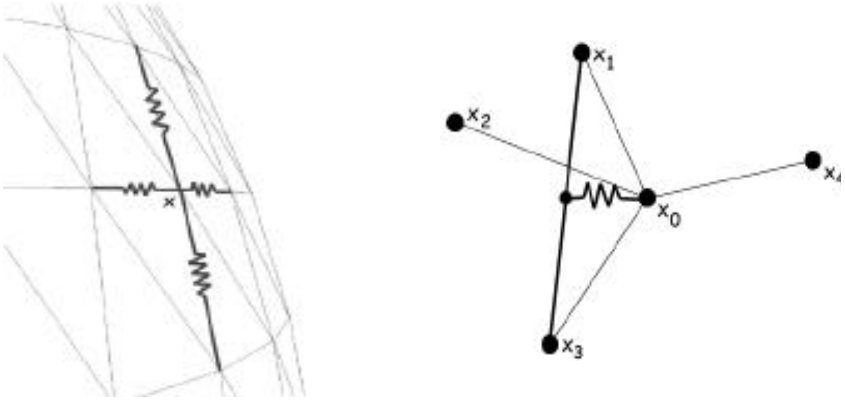
$$m_i = \frac{D\sum_j A_j}{3}, k = \frac{E\sum_j A_j}{L^2}, c = \frac{2\sqrt{kM}}{L}$$

Vzorec 3.9: Příklad určení tuhosti, hmotnosti a tlumení.

Hmotnost je označena *m*, *D* je hustota, *A* představuje obsah trojúhelníku, v němž je obsažen bod či pružina, *E* je Youngův modul, *L* délka pružiny, *M* je součet hmotností obou koncových bodů příslušné pružiny.

Systém pružin je také možné implementovat tak, aby při deformacích povrchu zachovával původní objem tělesa. Již v roce 1998 navrhli Nedel a Thalmann úpravy pro dosažení tohoto cíle [NED1998]. Cílem bylo vytvořit metodu pro realistický vzhled simulace pohybu svalů s požadavkem na simulaci v reálném čase. Princip úprav spočívá především ve správné volbě, které body spojit pružinou. Pro získání správné pružnosti tělesa (elastické síly) je bod spojen vždy se čtyřmi nejbližšími sousedy pružinou. Pro zachování objemu a správných ohybů povrchu (síly křivosti a síly zkrutu) navíc navrhli nový typ pružiny (nazývaná jako angulární) spojující bod se středem úsečky, která vede mezi dvěma proti sobě ležícími sousedními body. Pro názornost lze oba typy pružin vidět na obrázku 3.3. Dosažené výsledky byly v článku diskutovány zejména z pohledu vizuální stránky a možnosti simulace

v reálném čase.

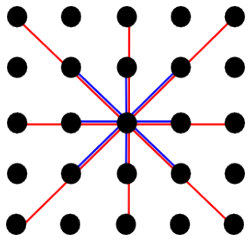


Obrázek 3.3: Vlevo - spojení bodu se 4 nejbližšími sousedy pružinami, vpravo příklad angulární pružiny. [NED1998]

Metodu s podobným zaměřením navrhli také Jung a spol. [JU2006]. Protože metoda konečných prvků má velké výpočetní nároky, zaměřili se autoři článku na použití metody pružinových systémů pro deformaci těles, která umožňuje i běh v reálném čase. Standartní systémy však objem nezachovávají, proto autoři přidali bodům systému pružin další omezující podmínky pro zachování objemu. Autoři rozlišují tři typy pružin, kterými jsou body provázány (viz obrázek 3.4 modré pružiny jsou strukturní a sheer, červené jsou ohybové). Strukturní pružiny jsou ty, které připojují čtyři nejbližší sousedy (levý, dolní, pravý a horní soused) ve vzdálenosti jedna (přímé sousedství, neboli nevede přes další bod) a které mají vliv především na elasticitu objektu. Sheer pružiny vedoucí do diagonálních sousedů zabraňují zkroucení objektu nebo jeho částí. Ohybové pružiny, vedoucí do všech sousedů do vzdálenosti dva, mají vliv na ohýbání objektu. I zde se síly z pružin aplikují v pohybové rovnici na jednotlivé body, ale také se přidají podmínky zachovávající globální objem. Podmínky vychází z toho, že objem lze vyjádřit jako integrál přes celý povrch tělesa, který je možný určit jako součet obsahů všech trojúhelníků. Z požadavku na rovnost objemu mezi dvěma iteracemi pak lze tyto dodatečné podmínky určit. Ne vždy je vhodné zachovávat globální objem (spíše homogenní tělesa, například balónek plněný plynem), proto autoři rozpracovali ještě systém vah. Každému bodu je stanovena váha v rozmezí nula až jedna. Nula znamená, že se podmínka pro zachování objemu neaplikuje a naopak jedna, že dojde k aplikace v plné výši. Pokud budou tedy všechny váhy nastaveny na jedna, bude chování odpovídat globálnímu zachování objemu. Váhy jsou nastavovány podle působiště síly, jejího směru, velikosti a délky trvání. Nejprve je zvolen radius zóny deformace a ten se proporcionálně s velikostí a délkou trvání síly zvyšuje. Výsledky ukazují, že metoda je jen o málo pomalejší než standartní systém pružin a tedy použitelná v aplikacích VR. Přesností téměř odpovídá použití metody konečných prvků.

Jak bylo zmíněno v úvodu této kapitoly, simulace tkanin a oblečení je jedním z hlavních použití. V aplikacích viruální reality je tělo postav většinou zakryto až z 90 procent, proto je žádoucí navrhnout systém realistického chování oblečení. Touto problematikou se zabývalo mnoho odborníků především v devadesátých letech. Řešení problému se dělilo na využití výpočetních metod jako metoda konečných prvků a použití fyzikálních metod jako systém pružin. V článku je diskutováno použití vhodného postupu a navržen přístup pomocí upraveného systému pružin. Problémy metody konečných prvků (i některých starších přístupů pomocí pružin) jsou především ve specifickém chování látek a to ve vyboulení, které se často jeví jako nerealistické a někdy dochází i k nestabilitě metody. Nový postup představený v využívá, jak standartních pružin se standartním výpočtem prezentovaným výše, tak i

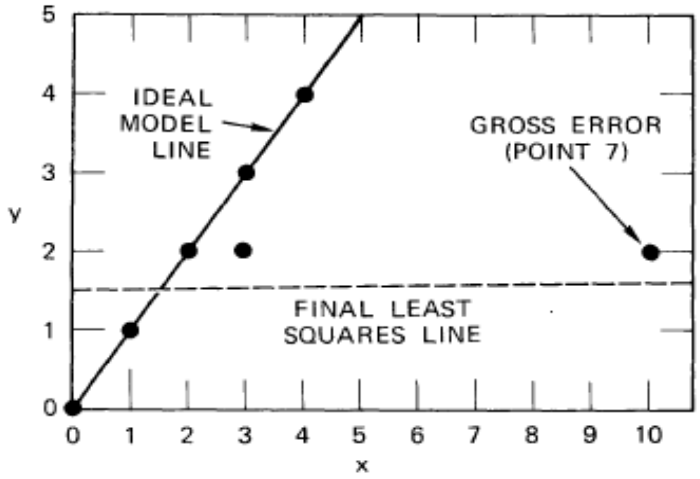
přidáních speciálních pružin k sousedům do vzdálenosti 2, jak je naznačeno obrázku 3.4, a také speciální výpočet sil v těchto pružinách tak, aby docházelo k realistickým vyboulením látky. Jako numerická integrace byla použita zpětná diference druhého řádu a jako tlumení jednoduše síla působící proti interakci se sousedními částicemi, která podporuje vyboulení, neboť nepůsobí v normále plochy, ve které částice leží.



Obrázek 3.4: Spojení bodu pomocí pružin se svými sousedy. Modré pružiny sousedé ve vzdálenosti 1 a červené pro vzdálenost 2.

4 Metoda RANSAC

Častým případem, který nastává při vzájemné transformaci těles, je nutnost nejprve nalézt vhodnou transformaci celého tělesa do druhého tak, abychom minimalizovali nutnost velkých přesunů a deformací, čímž bude výsledek při vizualizaci převodu těles lepší. Mezi takové transformace patří jak translace a rotace tak často i změna měřítka, proto není její stanovení triviální. Většinou se snažíme minimalizovat vzdálenosti, k čemuž se často využívá metoda nejmenších čtverců, která hledá vždy nejlepší průměrné řešení nad celou množinou a vlastně se tak snaží maximálně využít informace v datech. Lze však snadno ukázat (Obrázek 4.1), že i jeden chybný bod v celé jinak korektní vstupní množině může způsobit naprosto chybný výsledek. Samozřejmě existují heuristiky, které se snaží těmto problémům čelit. Většinou spočívají v nalezení prvního řešení, pak v odstranění bodů s největší odchylkou od tohoto řešení a opakovaně v nalezení nového řešení. Avšak i v případě heuristiky může metoda naprosto selhat. Z těchto důvodů vznikla metoda Random sample consensus zkráceně RANSAC [FIS1981] pro nalezení vhodného modelu nebo parametrů modelu nad množinou vstupních dat, která postupuje v podstatě opačně. Snaží se tedy z malého množství bodů určit řešení, které odpovídá co největšímu množství bodů z množiny a tím dojde k ignorování špatných bodů v řešení.



POINT	x	y
1	0	0
2	1	1
3	2	2
4	3	2
5	3	3
6	4	4
7	10	2

Obrázek 4.1: Ukázka kolapsu metody nejmenších čtverců jedním chybným bodem. [FIS1981]

Předpokládejme, že máme množinu bodů o velikosti *P*. K určení řešení postačuje alespoň *N* bodů, kde *N* je menší než *P*. Nejprve vybereme nejmenší potřebnou množinu bodů *S* (například pokud chceme určit nejlepší kružnici, která prochází naměřenými body, postačuje vybrat 3 body). Výběr může být realizován buď náhodně nebo deterministicky, pokud to má své odůvodnění. Z výběru se určí počáteční řešení *M*. Následně určíme množinu bodů *S**, pro jejíž body platí, že mají odchylku od řešení menší, než je maximální povolená odchylka *D*. Množina *S** se nazývá konsenzus. Pokud je počet bodů množiny *S** splňujících toto kritérium větší než zvolený práh *T*, pak použijeme nad touto množinou jednu z klasických metod (například metodu nejmenších čtverců) a její řešení *M** prohlásíme za výsledek (nebo lze ještě opakovaně přidávat další body splňující nové *M** a aktualizovat *S** i *M**). V opačném případě opakujeme výběr a celý proces, dokud nepřesáhneme povolený počet iterací. Pokud přesáhneme počet iterací, můžeme prohlásit za řešení nejlepší nalezené nebo skončit chybou.

Pro korektní výpočet je nutné správně zvolit parametry *D*, *T* a počet iterací. Odhad povolené chyby je obtížný, protože se může významně lišit pro různá vstupní data. Dokonce by se mohl korektní odhad lišit i pro jednotlivé hodnoty dat v rámci jednoho datového setu. Hodnotu je možné nastavit jako průměrnou odchylku od řešení metody nejmenších čtverců

nebo experimentálně. Maximální počet iterací lze stanovit na základě odhadu zastoupení chybných dat v setu a požadované přesnosti. Označíme-li *w* jako pravděpodobnost, že se vybraný bod nachází v intervalu definovaném povolenou odchylkou *D* a dále *k* jako počet pokusů potřebných k výběru počáteční nejmenší potřebné množiny, pak odhad počtu iterací *E(k)* lze zapsat dle vzorce 4.1.

$$E(k)=w^{-N}$$

Vzorec 4.1: Odhad počtu iterací.

Následující tabulka 4.1 shrnuje, jaký bude odhadovaný počet iterací *E(k)* v závislosti na *w* a *N*.

w	N = 1	N = 2	N = 3	N = 4	N = 5	N = 6
0,9	1,1	1,2	1,4	1,5	1,7	1,9
0,8	1,3	1,6	2	2,4	3	3,8
0,7	1,4	2	2,9	4,2	5,9	8,5
0,6	1,7	2,8	4,6	7,7	13	21
0,5	2	4	8	16	32	64
0,4	2,5	6,3	16	39	98	244

Tabulka 4.1: Odhadovaný počet iterací v závislosti na *w* a *N*.

Bude-li například pravděpodobnost chyby v rámci vstupních dat vzhledem k předpokládané odchylce rovna 0,5 a potřebujeme-li pro stanovení modelu alespoň 4 body (velikost nejmenší potřebné množiny), pak budeme potřebovat přibližně 16 iterací.

Odhad počtu iterací má však také svou odchylku *SD(k)*, kterou lze stanovit dle vzorce 4.2.

$$SD(k)=\sqrt{E(k^2)-E(k)^2},\ kde\ E(k^2)=\frac{2-w^N}{w^{2N}}$$

$$SD(k)=\frac{\sqrt{1-w^N}}{w^N}$$

Vzorec 4.2: Odchylka odhadu počtu iterací.

Odchylka *SD(k)* bude přibližně stejná jako odhad *E(k)*. Je proto vhodné volit dvou až třínásobek *E(k)* jako *k*. Stanovíme-li si navíc *z* jako pravděpodobnost, že počet iterací bude postačující, pak lze odhadnout *k* podle vzorce 4.3.

$$k\approx\log(1-z)E(k)$$

Vzorec 4.3: Stanovení počtu iterací *k* na základě odhadu počtu iterací *E(k)* a pravděpodobnosti, že tento odhad bude správný *z*.

Pořadujeme-li 90 procentní pravděpodobnost, že odhad bude postačující, pak počet iterací bude přibližně dvojnásobek odhadu *E(k)* tedy 36. S 95 procentní pravděpodobností bude výsledek přibližně trojnásobek tedy 48.

Práh *T* nelze opět přesně určit. Většinou se volí tak, aby počet bodů postačoval pro výpočet následné metody (například metoda nejmenších čtverců) s požadovanou přesností výpočtu.

5 Problém dekompozice svalů na svalová vlákna

Hlavním cílem této práce je vytvořit nástroj pro dekompozici svalů na svalová vlákna jako součást projektu EC FP7-ICT-223865 VPHOP : The Osteoporotic Virtual Physiological Human. Hlavní myšlenkou dekompozice je použití svalových šablon, které obsahují vlákna. Šablonu vhodně namapovat do svalu, který je reprezentován pouze povrchem.

5.1 Anatomie svalů

Popis funkcí a druhů svalů v lidském těle lze najít například v knize doktora Čiháka [ČIH2001]. Svaly jsou v knize definovány jako orgán umožňující pohyb. Tvoří je svalová tkáň, která se skládá z podlouhlých smrštění schopných elementů. Tyto elementy jsou seskupeny ve snopce a sítě. Podle typu elementů se rozlišují 3 typy svalstva: hladké svalstvo, příčně pruhované svalstvo a svalstvo příčně pruhované srdeční.

Hladké svalstvo tvoří protáhlé vřetenovité buňky délky řádově desítky mikrometrů. Je ovládáno autonomně, samo se pomalu smršťuje a uvolňuje a také téměř nepodléhá únavě. Nachází se například ve stěnách střev, průdušek, močovodů či ve stěně cév, kde má vliv na průtok krve. Příčně pruhované svalstvo tvoří z elementů funkční celky, které se vyskytují především ve spojení s kostrou. Oproti hladkému svalstvu je ovládáno nervy a bez jejich podnětů nefunguje. Vlákna jsou dlouhá řádově milimetry až centimetry a vyskytují se ve všech svalech pohybového ústrojí. Jediným příčně pruhovaným svalstvem neovládaným pomocí mysli je srdeční svalstvo, jehož vlákna nejsou podlouhlá, ale mají nepravidelný tvar a mají četné výběžky. Také nejsou seskupována do snopců, ale do sítí.

Na svalu rozeznáváme funkční a tvarové úseky, kterými jsou:

- začátek - origo (tato oblast svalu bude dále nazývána jako origin), což je část, kterou se sval upíná pomocí šlachy ke kosti
- hlava svalu - caput musculi je masitá část svalu, kterou začáteční úsek pokračuje
- svalové bříško - venter musculi je nejširší úsek svalu, který pokračuje ve zúženou část
- úpon - insertio (insertion oblast) je připojení svalu ke kosti pomocí šlachy

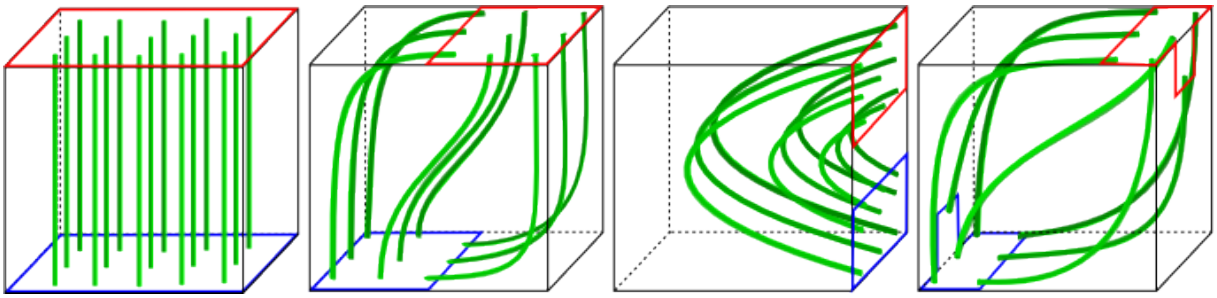
Rozdíl mezi úponem a začátkem je, že za začátek se obvykle považuje méně pohyblivé místo a za úpon pohyblivější.

Růst svalu probíhá jednak do délky přibýváním svalových vláken na koncích, jednak do šířky například tréninkem pouze ztlušťováním vláken. Svalová tkáň nepatří mezi tkáně, jejíž buňky se obnovují, proto lze říct, že počet svalových elementů se od narození prakticky nemění. Vzhledem k zaměření práce se budeme dále zabývat svaly dolních končetin.

5.2 Svalová šablona

Šablonu tvoří krychle, na jejímž povrchu jsou označeny origin a insertion oblasti. Mezi oblastmi jsou definovány nejméně čtyři základní křivky vedoucí z origin do insertion oblasti, jejichž kombinací lze získat libovolné vlákno uvnitř šablony. Podle typu svalu, na který mají být použity, rozlišujeme čtyři základní druhy. Prvním druhem je šablona parallel s paralelními vlákny vhodná pro podlouhlé svaly jako například Sartorius. Druhým typem je pennate šablona s šikmo umístěnými oblastmi, které se ale nacházejí na protějších stranách. Příkladem takového svalu je Vastus Medialis. Třetím typem je curved šablona, která má obě úponové oblasti na stejné stěně vymezující krychle. Tento typ je vhodný například pro sval Iliacus. Poslední typ je zvláště navržen pro sval Rectus Femoris, který byl definován kvůli

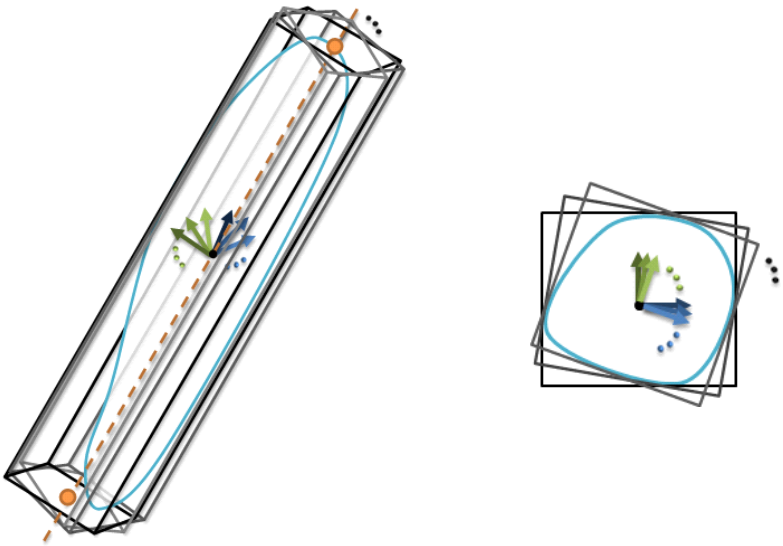
atypickým tvarům úponových svalů pouze pro tento sval. Druhy jsou také znázorněny na obrázku 5.1, kde červenou oblastí chápeme origin oblast a modrou insertion oblast. Body na jednotlivých křivkách reprezentující vlákna můžeme ze šablony získat zadáním tří parametrů, které můžeme označit jako *r, s, t*. Parametry *r, s* představují zvolenou křivku (vlákno) a *t* parametr křivky.



Obrázek 5.1: Ukázky šablon - zleva postupně parallel, pennate, curved a rectus.

5.3 Stávající přístup pro dekompozici

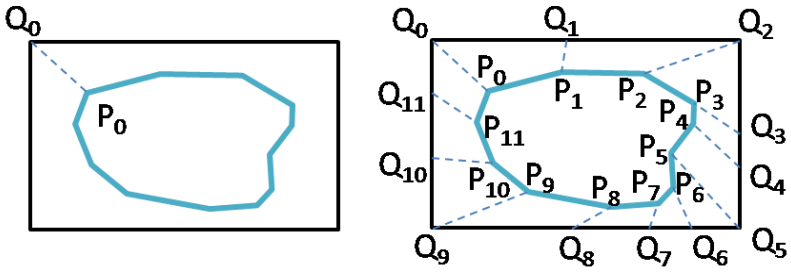
Tento přístup [KOH2010] byl vyvinut v předchozím projektu, na který VPHOP navazuje. Vychází také z použití uvedených svalových šablon. Nejprve dojde k nalezení hlavních os pro šablonu i povrch svalu. Následuje transformace šablony tak, aby se obě hlavní osy sjednotily. Poté následuje transformace scaling (změna měřítka) tedy roztažení svalové šablony tak, aby se celý povrch svalu nacházel uvnitř stěn šestiúhelníku ohraničujících šablonu. Metoda pak používá rotaci šablony kolem její hlavní osy tak, aby došlo k nejlepšímu možnému zarovnání obou úponových oblastí svalu i šablony. Průběh a výsledek tohoto procesu představuje obrázek 5.2.



Obrázek 5.2: Nalezení vhodné transformace šestiúhelníku tak, aby došlo k zarovnání hlavních os a sval se nacházel uvnitř. Rotací kolem osy lze najít nejlepší natočení vzledem k minimalizaci vzdálenosti mezi origin a insertion oblastmi. Vpravo je pak ukázka řezu a kontur šablony i svalu. [KOH2010]

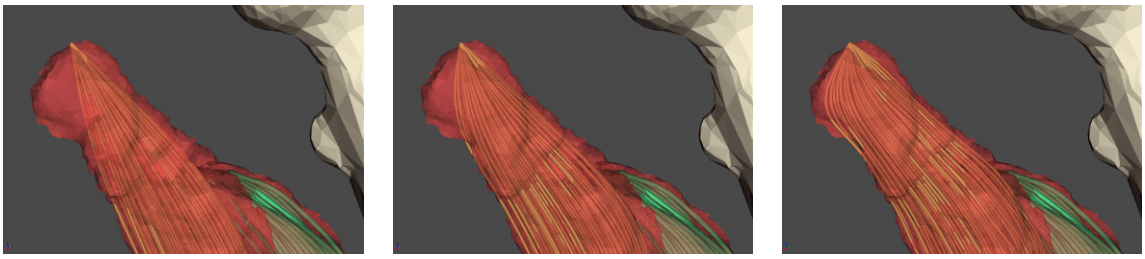
Následně dojde k vytvoření paralelních řezů (kvůli využití řezů bude dále metoda označována jako „metoda řezů“) šablony podél hlavní osy spojující obě úponové oblasti. Řezem získáme dvě kontury. V případě svalové šablony se jedná o konturu čtyřúhelníku a v případě svalu o n-úhelník. Dále je zapotřebí určit pozice vrcholů n-úhelníku na čtyřúhelníku.

Toho lze docílit výpočtem obvodu a vyjádřením bodu pomocí poměru vzdálenosti po obvodu od prvního bodu ku celkovému obvodu. Tento poměr pak naneseme podél obvodu čtyřúhelníkového půdorysu, jak je to znázorněno na obrázku 5.3.



Obrázek 5.3: Nalezení vhodné transformace šestiúhelníku tak, aby došlo k zarovnání. [KOH2010]

Posledním krokem je přemapování bodů vyjádřením těchto bodů pomocí zobecněných barycentrických souřadnic jako kombinace bodů kontury svalové šablony a následně jako kombinace bodů kontury svalu. Metoda pracuje dobře pro podlouhlé typy svalů s malými úponovými oblastmi. Zásadní problém, jak je vidět na následujícím obrázku 5.4, je upínání vláken do jednoho bodu, což je způsobeno tím, že oba koncové řezy podél hlavní osy obsahují jako konturu svalu pouze jediný bod. V závislosti na volbě počtu vláken a jemnosti řezů dochází k lepšímu kopírování povrchu u křivek jednotlivých vláken. Metoda také naráží na problémy u komplikovaných svalů, kde dochází k nerealistickým kolapsům a protínání vláken.



Obrázek 5.4: Vliv rozlišení na dekompozici na svalová vlákna. Zleva doprava je postupně voleno rozlišení 50, 100, 200 bodů na vlákně. [KOH2010]

5.4 Analýza problému dekompozice svalů na svalová vlákna

Problém dekompozice na svalová vlákna si lze představit jako speciální druh morphingu. Svalovou šablonu můžeme reprezentovat jako množinu bodů uvnitř hexahedronu, přičemž body ležící na křivkách představují vlákna. Jedná se tedy o částečně uspořádaná (po směru vláken) volumetrická data. Naproti tomu je sval vyjádřen pouze svým povrchem, který je reprezentován trojúhelníkovou sítí. Cílem je vlastně morphing volumetrických dat do tělesa s povrchovou reprezentací. Tento problém bohužel není standartním typem morphingu, a proto pro něj neexistuje známé rozšířené řešení, kromě výše popsané metody dekompozice pomocí řezů a přemapování bodů dovnitř kontur svalu.

Jednou z možností, která se nabízí, je převést sval reprezentovaný povrchem na volumetrická data a provést morphing objemových dat. Jedním z problémů tohoto postupu ale může být to, že se data svalové šablony nenacházejí v pravidelné mřížce. Řešení dekompozice také vyžaduje minimum interakce s uživatelem, přičemž jedinou známou korespondencí jsou

origin a insertion oblasti. Hraniční body oblastí na šabloně lze získat zadáním vhodných parametrů při generování (např. jeden z rohů origin oblasti jako r=0, s=0, t=0). Oblasti pro svaly jsou definovány množinou bodů uspořádaných kolem oblasti s nedefinovanou korespondencí mezi oblastmi. Autodetekce podobných tvarů použitelná v morphingu objemových dat však nebude funkční, protože sval se pravděpodobně nikdy nebude podobat šestistěnu. Morphing objemu nám také pravděpodobně nezaručí minimum kolizí vláken, což je z povahy problému žádoucí.

Základní myšlenkou této práce je vyjádřit objem svalové šablony jako systém pružin. To nám zaručí minimum kolizí mezi vlákny, jejichž body budou vázány na sousední. Nemělo by docházet k nežádoucímu protínání vláken. Po převodu šablony na systém pružin se snažíme deformovat celý systém tak, aby se nacházel uvnitř povrchu svalu. Můžeme vycházet z toho, že krajní vlákna mohou definovat povrch svalové šablony (například šestistěn v případě šablony paralelních vláken), ke kterému je možné body vláken připojit také pomocí pružin. Tento povrch můžeme následně transformovat do povrchu svalu pomocí známých metod morphingu povrchů.

Primitivní metodu interpolace mezi vrcholy se stejnými indexy nelze využít, protože má pro naše účely podstatné omezení, neboť nesplňuje kritérium v zadání na tělesa, která nemusejí být obecně konvexní a mohou mít různý tvar i počet vrcholů. Musíme tedy použít jednu ze sofistikovanějších metod, například metodu představenou v článku od Alexa [ALEX2000]. Metoda by měla být postačující, neboť funguje pro libovolné objekty s genusem 0, což lze u svalů předpokládat. Interpolace mezi povrchem svalové šablony a povrchem svalu definuje pozice povrchových bodů systémů pružin a také počáteční podmínky simulace v jednotlivých krocích interpolace. Přesun bodů pak bude mít za následek změny pozic bodů uvnitř šablony vyjádřené systémem pružin. Protože by i na povrchové body působily síly od připojených bodů snažící se vrátit body zpět, je nutné tyto body nějak zafixovat nebo tyto síly kompenzovat. I zde vzniká problém se zarovnáním origin a insertion oblastí, který však již můžeme řešit pomocí feature aligning postupu popsaného také v Alex. V našem případě můžeme odlehčit metodu o fázi mísení topologií, neboť postačí hledat jen pozici bodů z povrchu šablony na povrchu svalu. Požadovanou přesnost aproximace povrchu bude možné volit parametrem počtu bodů podél vlákna.

Při použití systému pružin vzniká otázka, jak nastavit parametry systému. K pohybu bodů systému bude docházet hlavně uvnitř (přesun bodů na povrchu zajistíme morphingem a následnou fixací). Protože nemáme dodatečné informace o vlastnostech svalové tkáně a nejedná se o povrch (pro něž jsou definovány výpočty jednotlivých parametrů na základě trojúhelníků sdílejících hranu), nastavíme tuhost pružiny jako převrácenou hodnotu její počáteční délky. Podle informace uvedené v jednom z článků vykazuje tato volba vhodné zachování poměrů, což požadujeme. Z pohledu využití aplikace jako interaktivní je důležitá doba výpočtu, na což mají parametry vliv. Žádoucí je dosáhnout cílového stavu v nejkratším možném čase, což v případě jedné pružiny zajistí kritické tlumení. Protože každý bod v systému je připojen k více pružinám, není možné přesně stanovit toto tlumení, které bude pro různé pružiny rozličné. Vhodná volba tlumení a hmotnosti bodů bude proto nalezena experimentálně viz kapitola 7.

Celkový počet bodů získaných vzorkováním šablony vláken se může při cílovém použití vyšplhat k počtu, který nebude možné simulovat pomocí CPU v uspokojivém čase, proto je vhodné implementovat i postup využívající GPU a porovnat obě metody, abychom zjistili, kdy je využití GPU přístupu adekvátní a jaké jsou jeho omezení. V případě potřeby simulace velkého množství bodů pak můžeme přistoupit k použití GPU řešení, kde lze předpokládat urychlení díky paralelnímu zpracování.

Metodu RANSAC navzdory původním předpokladům není nutné použít. Její využití se předpokládalo pro nalezení vhodné transformace šestistěnu obsahujícího šablonu tak, aby

došlo k přiblížení origin a insertion oblastí a aby se povrch svalu nacházel uvnitř transformovaného šestistěnu. Cílem tedy bylo vylepšit fázi hledání vhodného natočení u stávající metody pomocí řezů. Protože však toto zajistíme pomocí morphingu, nemusíme k využití RANSAC přistoupit.

6 Návrh a implementace

Z analýzy problému vyplývá, že pro dekompozici svalu na vlákna musíme transfomovat nejen povrch, ale i objem svalové šablony. Navrhované řešení se proto skládá ze dvou základních kamenů a to systému pružin a morphingu povrchu. Na svalové šabloně musíme znát nebo určit body na povrchu, na které se bude aplikovat morphing povrchu. Pozice povrchových bodů musí být pevně zafixována a určuje výchozí stav pro simulaci systému pružin. Přesunem povrchových bodů se bude díky simulaci transformovat i objem, tedy body připojené k těmto poverchovým bodům pomocí pružin. Postup implementace obou klíčových částí a návrh řešení dalších sekundárních problémů jsou podrobně popsány v této kapitole.

6.1 Volba programovacího jazyka a prostředků

Jako programovací jazyk jsem zvolil, vzhledem k integraci do systému OpenMAF, jazyk C++. Rozhraní pro komunikaci s grafickou kartou jsem pak zvolil OpenGL. Pro výpočty na GPU je kromě rozhraní OpenGL nutné použít i GLSL [KES2010], což je jazyk pro programování shaderů.

6.2 Implementace systému pružin

V předchozích pracích pro předměty OPPG (oborový projekt) a GRG (grafická rozhraní) byly vyvinuty obě implementace systému pružin jak s pomocí CPU tak GPU. V případě varianty GPU byla pro pozdější účely v rámci projektu VPHOP vyvinuta nová metoda výpočtu systému pružin GPU založeného na postupu popsaného v kapitole 3.2. Postup byl rozšířen o využití geometry shader kvůli snížení výpočetních a zejména paměťových nároků. V následující kapitole 7 je popsáno porovnání nové metody, hranově orientovaného GPU přístupu a CPU řešení. Všechny postupy využívají Verletovu integraci z důvodu oddělení jednotlivých kroků a vhodnosti i k implementaci na GPU.

6.2.1 CPU řešení

Výpočet systému pružin se provádí podle vzorce 3.8. Protože používáme Verletovu integraci, bude výpočet postupně probíhat v diskrétních krocích, které navíc můžeme díky tomuto způsobu integrace dále rozdělit na výpočet síly v pružinách spolu s aplikací sil na oba koncové hmotné body pružin a výpočet Verletovy integrace pro pohyb bodů s aktualizací pozic bodů pro další iteraci. Protože nejprve proběhne výpočet síly z aktuálních pozic a výpočet nové pozice je uložen do jiného pole, je zajištěna konzistence dat. Pokud by výpočet pro jeden bod probíhal v celku (například jiným způsobem integrace) a výsledek by se zachovával v jednom poli, docházelo by k závislosti celé simulace na způsobu procházení polem.

Pro výpočet síly v pružině (viz vzorec 3.1) musíme znát vzdálenost mezi koncovými body, tuhost pružiny a její počáteční délku. Obvykle se volí tuhost pružiny jako převrácená hodnota počáteční délky, což dobře zachovává původní poměry. Pokud budeme tuhost takto volit, musíme pro každou pružinu uložit dvě hodnoty typu integer (jako indexy koncových bodů) a jednu hodnotu typu float (jako počáteční délku pružiny). Hodnoty pružin se uchovávají v poli o velikosti počtu pružin. Dále potřebujeme určit kumulované síly pro každý jeden bod, které uložíme do jednoho pole o velikosti počtu vrcholů. Síla je v případě prostorového řešení vektor pozice o třech složkách. Před provedením kroku kumulace se musí pole nastavit na nulové vektory. Kumulace se provádí jedním průchodem pole přes všechny pružiny. Vypočte se velikost a směr síly. Vektor síly se přičte do pole kumulovaných sil k

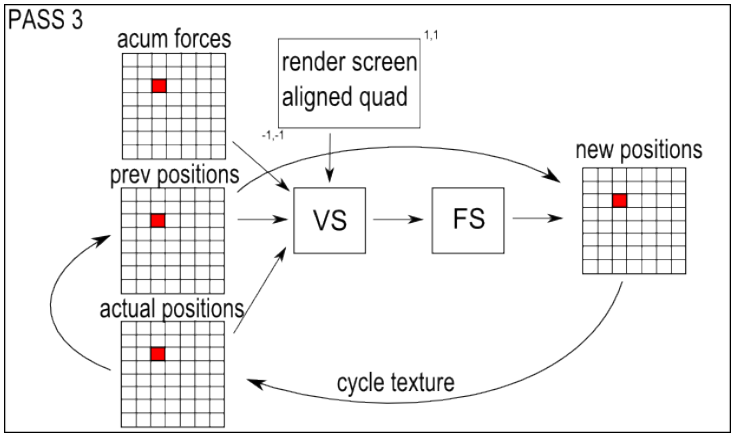
vektoru na indexu odpovídajícímu počátečnímu bodu pružiny a zároveň se přičte síla opačného směru k vektoru uloženému na indexu koncového bodu pružiny.

Na výpočet diferenciální rovnice pohybu bodu (viz vzorec 3.8) pomocí Verletovy integrace potřebujeme také znát, kromě výslednice sil z připojených pružin, i velikost tlumení, hmotnost bodu a jeho aktuální pozici spolu s předchozí pozicí. Pro uložení potřebných pozic postačí dvě pole o velikosti počtu vrcholů. Protože pole aktualizujeme postupně po vrcholech a není zde již závislost na okolí, můžeme získat pozici z pole předchozích pozic a pozici z pole aktuálních pozic a novou pozici uložit do pole předchozích pozic. Po ukončení výpočtu nových pozic vyměníme ukazatele na pole předchozích pozic a aktuálních pozic, čímž je systém připraven na další iteraci.

6.2.2 GPU řešení

Jak bylo uvedeno výše, potřebujeme v případě výpočtu pomocí GPU uložit veškerá data do textur. To je umožněno pomocí kombinace funkcí OpenGL glGenTextures pro vygenerování identifikačního čísla textury, glBindTexture pro připojení textury do stanoveného slotu (je používáno vždy pokud chceme používat texturu v shaderech na příslušném slotu) a hlavně pomocí glTexImage2D. Poslední jmenovaná, jak název napovídá, slouží pro přenos obrázků a tedy i dat do paměti GPU. Historicky uměly grafické karty pracovat hlavně se čtvercovými texturami, které by měly mít nejlépe délku strany definovanou jako mocnina dvou (např. 1024 x 1024). Moderní grafické karty umožňují pracovat s texturami libovolných rozměrů (i obdélníkového tvaru) bez větších časových sankcí. Protože se však snažíme o co nejefektivnější implementaci, je i v případě moderní grafické karty lepší stanovit rozměry jako mocniny dvou.

Jádro algoritmu tvoří výpočet pozice již výše zmiňovanou Verletovou integrací, pro kterou je zapotřebí znát celkovou sílu na bod, aktuální pozice a předchozí pozice bodů. Budou tedy zapotřebí 4 textury o velikosti počtu bodů. Každá pro jeden uvedený parametr a čtvrtá pro zapsání výsledků (rendering do textury), protože není možné zapisovat a zároveň číst z jedné textury. Při běhu programu je pak nutné cyklicky obměňovat texturu pro rendering a textury s aktuální pozicí a předchozí pozicí. Průběh této fáze výpočtu ilustruje obrázek 6.1. Hlavním problémem pak je výpočet celkové síly na hmotný bod, pro který je důležité zvolit vhodnou datovou strukturu.

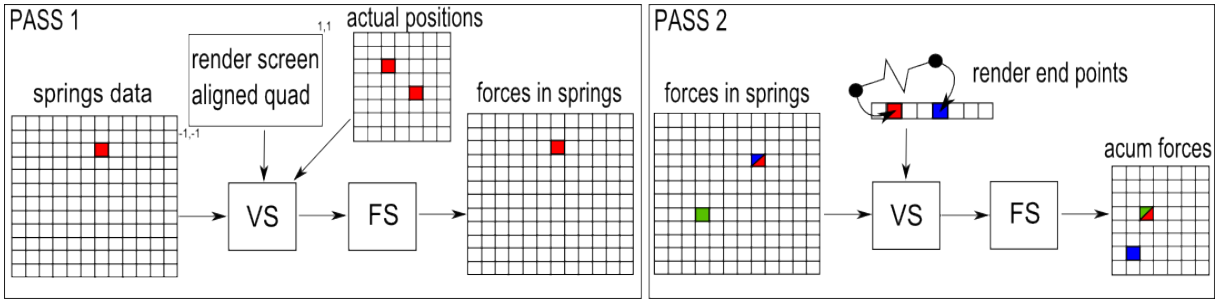


Obrázek 6.1: Výpočet nové pozice.

Ve výše uvedeném článku jsou popsány 2 přístupy. První je bodově orientovaný, který je vhodný zejména, pokud je počet pružin pro každý bod konstantní (nebo menší roven přijatelné hodnotě, nejlépe počtu texturovacích jednotek). Definice datové struktury vychází z toho, že každý bod zná své pružiny a má je tedy někde vhodně uloženy. Jedinou možností, jak je uložit, jsou opět textury, kde v každé bude uložen jeden nebo více indexů připojených

pružin. Pro výpočet akumulované síly na bod projdeme všechny připojené pružiny a určíme v nich síly, které sečteme. Je zřejmé, že mezi základní nevýhodu přístupu patří velký počet náhodných přístupů do textur, což může být velmi pomalé, neboť nedochází k využití cache pro textury (rozdíl čtení z cache a z paměti může být i desetinásobný). Další nevýhodou je dvojnásobný počet výpočtů sil v pružinách, neboť se síla počítá pro každý koncový bod, tedy dvakrát. Z pohledu využití paměti dochází k ukládání velkého počtu duplicit a navíc při rozdílných valencích vrcholů dochází i k nevyužití rezervovaného místa pro indexy připojených pružin. Výhodami jsou naopak jednoduchost implementace a nutnost pouze 2 průchodů.

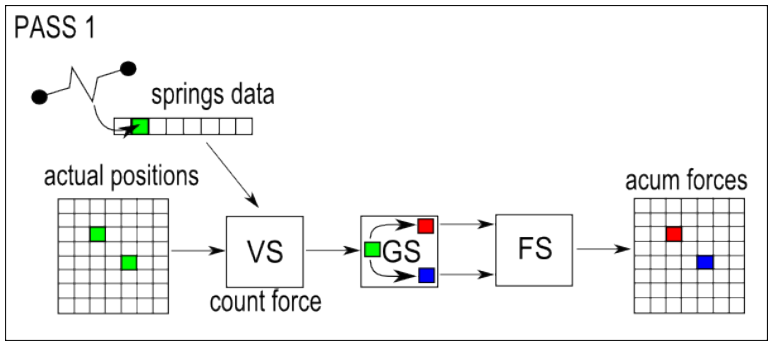
Druhý přístup je pružinově orientovaný. Pružiny mají v tomto případě uloženy indexy obou koncových bodů, body naopak nemají informaci o připojených pružinách. K tomu potřebujeme texturu o velikosti počtu pružin, která bude několikanásobně větší (podle valence vrcholů) než textura bodů, a způsob jakým namapujeme sílu na bod. Mapování síly lze pak provést tak, že pro každou pružinu vyrenderujeme dva body (nebo jeden dvakrát) na pozice umístění jeho koncových bodů v textuře bodů (akumulovaných sil) a tím aplikujeme výpočet síly, která je následně ve fragment shaderu uložena na správnou pozici do textury akumulovaných sil. Zapnutím správné blend funkce (one, one) během renderování pružin pak získáme akumulovanou sílu. Průběh obou průchodů pro získání akumulovaných sil v jednotlivých bodech znázorňuje obrázek 6.2. Pružinově orientovaný postup potřebuje oproti předchozímu jeden průchod navíc. Celý výpočet tedy vyžaduje tři průchody (čtyři pokud renderujeme bod pro pružinu dvakrát), dvě textury o velikosti počtu pružin (v jedné budou uloženy parametry pružin jako tuhost a indexy obou koncových bodů, v druhé pak síly) a čtyři textury o velikosti počtu vrcholů. Sílu v pružině tedy vypočteme pouze jednou. Přístup zajišťuje minimalizaci přístupů do textur pro výpočet síly a nedochází k ukládání duplicit. Vzniká zde další režie s přepínáním textur. Hlavním problémem metody jsou velké textury o velikosti počtu pružin. Při použití modelu s 26 sousedy je počet pružin až 13 krát větší než počet bodů.



Obrázek 6.2: První dva průchody pro pružinově orientovaný přístup. Vlevo výpočet síly, vpravo získání akumulované síly.

Vzhledem k nevhodnosti prvního postupu (velké plýtvání místem a navíc není známo jaká bude požadovaná valence vrcholů (možná i 26 sousedů)) a velkým nárokům na velikost textury s uloženými silami a pružinami u druhého přístupu (viz kapitola 7.1.1) jsem navrhl vlastní přístup, který vychází z druhého pružinově orientovaného přístupu a využití další programovatelné jednotky geometry shaderu. I zde pro každou pružinu vyrenderujeme bod (GL_POINTS) na pozici odpovídající jednomu z koncových bodů pružiny v textuře akumulovaných sil. Tento bod má oproti předchozímu uloženy hodnoty pružiny (tuhost, indexy koncových bodů) a umožňuje tak přistupovat rovnou do textury s pozicemi vrcholů. Můžeme určit sílu v pružině v tomto kroku a vyrenderovat ji. Každé geometrické primitivum tedy i GL_POINTS prochází geometry shaderem, který umožňuje generovat nová primitiva. Můžeme tedy předat s bodem do geometry shaderu (pomocí varying proměnné) i vypočítanou sílu a každý bod rozdvojit tak, aby se síla aplikovala na oba koncové body. Díky tomu dochází k výpočtu síly v pružině pouze jednou a to ve vertex shaderu. Nový způsob

akumulace sil je pro názornost vyobrazen na obrázku 6.3. Změnou této části výpočtu tak pro celé řešení problému potřebujeme: 2 průchody (druhý průchod s výpočtem nové pozice je shodný s předchozími viz obrázek 6.1) a 4 textury o velikosti počtu pružin. Dochází proto k eliminaci velmi paměťově náročných textur pružin, ale je zde nárůst dat v bodech renderovaných jako pružiny (nárůst není vysoký, neboť předtím se nevyužila plně data bodu - byl tam jen jeden index). Také je zde menší režie s přepínáním kontextu, protože ubyde průchod s renderováním sil do textury sil. Největší část dat je tak uložena v bodech array bufferu, který umí grafická karta spravovat velmi dobře a pravděpodobně lépe než obrovské textury pružin, neboť například umožňuje, aby byla část dat uložena v paměti počítače.



Obrázek 6.3: Výpočet akumulované síly s použitím geometry shaderu.

Složitost jedné iterace všech uvedených postupů je závislá především na počtu pružin (*S*), neboť vždy musíme určit sílu v každé pružině. Lze tedy psát, že složitost iterace je *O(S)*. Otázkou však je, kolik iterací je potřeba pro celou simulaci. Odpověď není jednoznačná a závisí na nastavení parametrů systému a volbě propojení bodů se svými sousedy. Tato problematika je blíže prozkoumána v kapitole 7.

Protože data jsou po skončení výpočtu stále v paměti GPU, je nutné ještě provést jejich přečtení neboli readback. Protože je sběrnice pro komunikaci s GPU navržena zejména pro přenos dat do paměti GPU, je tato operace poměrně náročná a měla by se provádět co možná nejméně, nejlépe po skončení výpočtu. Data jsou i tentokrát uložena v textuře, do které se provedl poslední render. Nejprve připojíme frame buffer, který využíváme pro render do textury, pomocí glBindFrameBuffer. Aplikací glViewPort se nastavuje rozměr textury a glFrameBufferTexture2D se nastavuje textura, ze které požadujeme čtení. Vlastní čtení se pak provádí voláním glReadPixels, která čte data z textury. Od generace grafických karet podporující OpenGL 2.1 existuje možnost asynchronního čtení dat z textury a dalších objektů nazývaných pixel buffer object (dále PBO) umožňujících rychlejší zpracování a čtení. Také zde musí dojít k připojení frame buffer avšak s volbou GL_PIXEL_PACK_BUFFER a ukazatel na PBO, čímž dojde k napojení PBO, ze kterého je pak možné číst stejnou metodou glReadPixels. Volání funkce nezpůsobuje žádnou prodlevu a čtení se provádí asynchronně do PBO. Později je pak možné namapovat část paměti grafické karty vymezenou pro PBO pomocí glMapBuffer do adresového prostoru procesu a přečíst libovolná data. Na závěr je nutné ještě paměť odpojit díky funkci glUnmapBuffer. Porovnání obou variant je provedeno v kapitole 7.

6.2.3 Fixace povrchových bodů

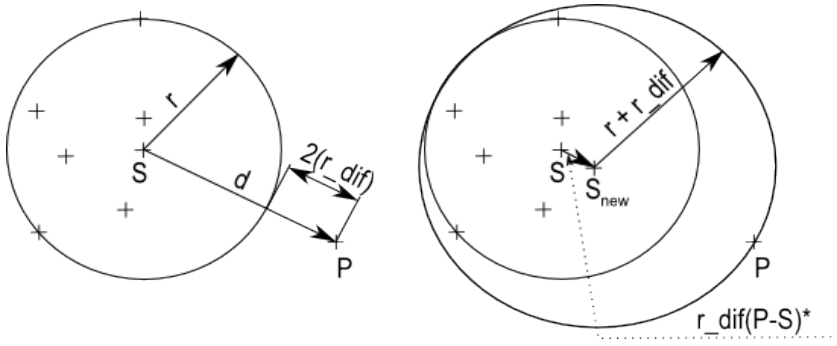
Protože je zapotřebí některé body, v našem případě povrchové body šablony, přesunout na cílové pozice na povrchu modelu svalu a pak je po zbytek simulace udržovat na těchto pozicích, musíme pro tyto body výpočet působení sil z připojených pružin přizpůsobit neboli je zafixovat. Fixování bodů je možné řešit pomocí kompenzace sil působících na bod tak, aby výslednice sil byla nulová. Stejného efektu je možné dosáhnout i nepřičtením síly do pole akumulovaných sil. V případě GPU řešení tímto postupem dochází k šetření výpočetního

času, neboť se síla v těchto bodech nemusí kumulovat a nedochází tak k náročnému výpočtu blend. Přesun bodu je možné také realizovat pomocí působení sil, ale problémem mohou být síly působící v pružinách, které se budou neustále měnit v závislosti na pohybu bodů. Fixací těchto bodů se těchto závislostí zbavíme a body pak můžeme přesunout buďto silami, které by musely být určeny pro každý bod, nebo můžeme interpolovat pozici bodů mezi původní a cílovou pozicí, což je přístup odpovídající morphingu. Při přímém nastavení pozice je vhodné zmínit, že se nově nastavená pozice musí nastavit do obou polí aktuálních pozic i předchozích pozic, aby došlo také k vynulování rychlosti a zrychlení. Jinak by docházelo k pohybu bodu díky pohybové rovnici (nenulová rychlost a zrychlení).

6.3 Implementace transformace povrchů

Jak již bylo uvedeno v pasáži o analýze problému dekompozice svalů na svalová vlákna, vychází postup pro transformování povrchu z práce Alexa [ALEX2000]. Tento postup lze rozdělit na tři části. První je promítnutí obou těles na pomocné těleso a to kouli. Druhou fází je hledání společné topologie na povrchu koule a třetí pak převod souřadnic z pomocného tělesa zpět pomocí barycentrických souřadnic. Protože pro naše účely (dekompozice svalů na svalová vlákna) požadujeme především transformaci vnitřku svalové šablony a ne, aby se povrch šablony a jeho hrany přesně během interpolace měnil na povrch svalu, nemusíme vyhledávat průsečíky promítnutých trojúhelníků, mísit topologie a tento krok přeskočit. Můžeme pak již navázat převodem pozic bodů zpět z pomocné koule vyjádřením bodů uvnitř trojúhelníků cílového povrchu (svalu) jako barycentrické souřadnice. V zadání je také požadavek na transformaci s omezeními, přičemž omezení jsou v tomto případě myšlena jako definované úponové oblasti, které se mají po převodu shodovat. V případě svalových šablon se jedná o označené origin a insertion oblasti (červené a modré oblasti viz obrázek 5.1) a v případě svalů jde o posloupnosti bodů vymezujících úponové oblasti na povrchu svalu (nebo poblíž jeho povrchu). Podobná záležitost je také v článku [ALEX2000] řešena pod názvem feature aligning. I proto bylo přistoupeno k použití tohoto postupu.

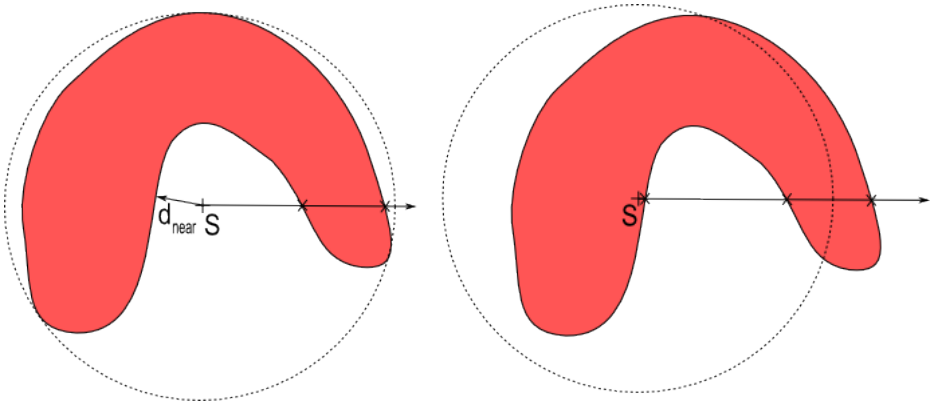
Fázi relaxace a promítání na pomocnou kouli předchází nalezení minimální obalové koule. Pro tyto účely je možné použít například rychlý postup pro její odhad [RIT1990]. Během jednoho průchodu polem vrcholů najdeme minimální a maximální hodnoty ve všech třech osách. Hodnoty uprostřed minim a maxim nalezených hodnot v jednotlivých osách definují první odhad středu obalové koule a největší rozdíl mezi tímto středem a nalezenými extrémy v osách definuje první odhad poloměru. Je zřejmé, že se jedná pouze o první přiblížení a nalezené řešení nedefinuje kouli uvnitř, ve které by se celé těleso nacházelo. Proto následuje další průchod všemi vrcholy, kde se odhad aktualizuje. Pokud nalezneme vrchol, který se nachází mimo aktuální obalovou kouli, označíme *r_dif* jako polovinu rozdílu mezi stávajícím poloměrem a vzdáleností zpracovávaného bodu od stávajícího středu. Nový poloměr získáme přičtením *r_dif* ke stávajícímu poloměru. Nový střed dostaneme, když přičteme ke stávajícímu středu vektor, který vznikne vynásobením *r_dif* a normalizovaného vektoru rozdílu pozice bodu a stávajícího středu.



Obrázek 6.4: Algoritmus nalezení obalové koule v rovinném případě.

Uvedený obrázek 6.4 zobrazuje průběh algoritmu v rovinném případě. Tato metoda nalezne, jak autor uvádí, přibližné řešení obalové koule s chybou přibližně deset procent, což je pro naše potřeby dostačující.

Obalová koule je pomocným tělesem, na které se bude provádět projekce. Pokud se její střed nebude nacházet uvnitř tělesa, které budeme promítat, mohou později během relaxace vznikat problémy. Proto je vhodné střed přesunout do nejbližšího bodu uvnitř tělesa. Obecně je těleso nekonvexní, proto musíme zvolit vhodný test bod uvnitř mnohostěnu. Rovinnou obdobou je test bod uvnitř nekonvexního polygonu popsaný například v knize Moderní počítačová grafika [ŽBSF2004]. Spočívá ve vržení paprsku (polopřímky) libovolným směrem z testovaného bodu. Posléze se otestují všechny hrany polygonu zda jsou protnuty paprskem. Pokud ano, zvýšíme čítač počtu průsečíků. Pokud je počet průsečíků roven lichému číslu, nachází se bod uvnitř polygonu. V opačném případě (sudý počet) leží bod vně (viz obrázek 6.5). Princip algoritmu můžeme po úpravách aplikovat i v prostoru. Místo hledání průsečíku dvou přímek (polopřímky a úsečky) použijeme hledání průsečíku přímky a roviny (polopřímky a trojúhelníku). V tomto případě je vhodné vyjádřit přímku jako průsečík dvou rovin a navíc definovat rovinu kolmou na polopřímku procházející jejím počátkem pro rozdělení prostoru na dvě části, abychom mohli testovat trojúhelníky pouze v jedné části (obdobně jako v případě polopřímky). Protože můžeme volit polopřímku libovolně, je vhodné pro zjednodušení výpočtu zvolit roviny ležících v souřadnicových osách kartézské soustavy souřadnic x, y a z. Průsečík rovin xy a xz posunutých do testovaného bodu definuje přímku a rovina yz oddělovací rovinu. Procházíme skrz pole všech trojúhelníků a každý trojúhelník nejprve otestujeme vůči oddělovací rovině. Pokud alespoň jeden z vrcholů trojúhelníku leží ve zvoleném poloprostoru (leží nad rovinou, tedy ve směru normály dělící roviny), otestujeme, zda roviny definující přímku oddělují vrcholy trojúhelníku. Pokud se jeden z vrcholů nalézá v opačném poloprostoru než ostatní dva pro obě roviny definující přímku, může přímka procházet skrz trojúhelník. V této fázi je již velmi pravděpodobné, že je trojúhelník protnut a proto spustíme výpočet průsečíku přímky s rovinou trojúhelníku a následně otestujeme, zda se bod nachází uvnitř trojúhelníku (viz vzorec 2.3). Lichý počet průsečíků opět znamená, že se bod nachází uvnitř tělesa. Singulární případy mohou nastat jak v rovinném tak v prostorovém provedení. Prvním je přímka procházející skrz bod, kde se průsečík může nasčítat vícekrát podle počtu trojúhelníků sdílejících bod, a druhým je, že přímka leží v rovině trojúhelníku, kde může nebo nemusí dojít k započtení průsečíku. Tyto případy je vhodné detekovat a případně spustit výpočet znovu pro jiný směr polopřímky. Nachází-li se bod vně tělesa, je nutné transformovat střed koule dovnitř tělesa, například nalezením nejbližšího bodu a posunem středu těsně za tento bod. Následující ilustrace 6.5 prezentuje test zda bod leží uvnitř tělesa a případné přesunutí středu.



Obrázek 6.5: Ilustrace zjištění, zda střed leží uvnitř svalu a případného přesunu bodu těsně za nejbližší bod.

Máme-li obalovou kouli se středem nacházejícím se uvnitř tělesa, přesuneme kouli i

těleso tak, aby se střed koule nacházel v počátku souřadnicového systému. Poté ještě provedeme scaling (změnu měřítka) koule i tělesa tak, aby došlo k převedení koule na jednotkovou kouli. Tyto operace jsou zařazeny zejména kvůli zjednodušení a zefektivnění následujících výpočtů. Také můžeme jednoduše promítnout všechny body tělesa na jednotkovou kouli jen pomocí operace normalizace (vydělení všech prvků vektoru pozice délkou vektoru pozice) aplikovanou na tyto body. Díky normalizaci se budou všechny body nacházet na povrchu koule, ale trojúhelníky tvořené těmito body se mohou protínat a mohou být převrácené.

Po těchto krocích následuje volba náhodného pravidelného tetrahedronu, jehož vrcholy budou určovat kotvící body neboli fixované body pro následnou relaxaci. V inicializaci definujeme tetrahedron a následně můžeme použít transformaci jeho vrcholů. Pro tuto funkci jsem využil rotaci okolo libovolné osy, jejíž jednotkový směrový vektor určíme náhodně stejně jako úhel rotace. Nejbližší body na povrchu koule k vrcholům tetrahedronu zafixujeme pro relaxaci, ke které pak můžeme přistoupit.

Relaxace je klíčovou částí zvoleného postupu pro transformaci povrchů. Jak je popsáno v kapitole 2.1.1, vychází relaxace z teorie o grafech, hlavně rovinných grafech, které lze překreslit tak, že se žádné dvě hrany nebudou křížit. V případě uzavřeného objektu bez děr (s genusem 0) reprezentovaného trojúhelníkovou sítí můžeme překreslit celou síť na povrch koule tak, aby se ani hrany této reprezentace nekřížily. Tímto překreslením dosáhneme toho, že každý trojúhelník bude mít jednoznačně definovaný odpovídající sférický trojúhelník na kouli. Myšlenka algoritmu pro překreslení rovinného grafu spočívá v přesunu každého bodu do středu jeho sousedních bodů. Tento přístup rozšíříme v případě prostoru a průmětu na kouli o normalizaci bodů po každém přesunu. Přesuny opakujeme, dokud existuje přesun o větší délce než stanovená mez. Nyní je nutné zkontrolovat, zda nedošlo ke kolapsu relaxace. To lze provést například tak, že nalezneme nejbližší body pro body, které jsou symetrické podle počátku k zafixovaným bodům tetrahedronu. Pokud jsou tyto vzdálenosti větší než hrana fixačního tetrahedronu, je nutné opětovně spustit relaxaci s novou náhodnou volbou fixovaných bodů. V opačném případě zafixujeme symetrické body podle počátku pro fixované body a spustíme druhotnou relaxaci, která probíhá opět, dokud není splněna podmínka, že největší přesun je menší než stanovená mez. V případě splnění této podmínky musíme ještě ověřit, zda mají všechny trojúhelníky na povrchu koule stejnou orientaci (viz vzorec 6.1). Tento test se opírá o předpoklad, že všechny trojúhelníky originálního tělesa mají stejnou orientaci, což je v aplikacích počítačové grafiky častým případem, neboť při vykreslování tělesa často stanovujeme orientaci přilehlých trojúhelníků pro snížení množství zpracovávaných trojúhelníků v grafické kartě. Pokud ovšem není předpoklad zajištěn, je nutné je zorientovat, jinak nedojde k zastavení výpočtu.

$$sgn((v_0 \times v_1) \cdot v_2)$$

Vzorec 6.1: Test orientace trojúhelníku.

Pokud splníme i tento test, signalizuje to úspěšný průmět a rozložení na povrchu koule. Pokud není tato podmínka splněna, je nutné snížit stanovenou mez a opakovat druhotnou relaxaci, dokud ji nesplníme. Je zjevné, že složitost jedné iterace relaxace je *O(E)*, kde *E* představuje počet hran. Složitost celé relaxace je obtížně stanovitelná a závisí především na počtu vrcholů a geometrii objektu. Tento aspekt bude vzhledem k důležitosti znalosti o výpočetních nárocích ještě diskutován v kapitole 7.

Další částí, která je ale z našeho pohledu velmi důležitá, je zarovnání úponových oblastí. Pro zarovnání odpovídajících si bodů existuje v článku [ALEX2000] postup zvaný feature aligning. Uživatel pak může stanovit na obou tělesech body, které se na pomocném tělese sjednotí, čímž dojde během interpolace k přímému převodu odpovídajícího bodu na

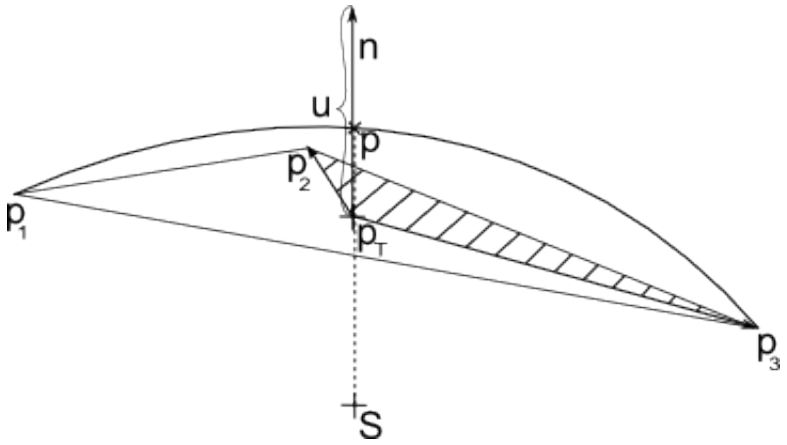
povrchu jednoho tělesa na bod na povrchu druhého tělesa. Tím je například zajištěno, že se nohy jednoho zvířete převedou správně na nohy druhého zvířete (jak je tomu na obrázku 2.5). Nelze samozřejmě přesunout pouze tyto body, ale také jejich okolí tak, aby nedošlo k nežádoucím kolapsům již nalezených reprezentací na povrchu koule a následně nežádoucím deformacím během interpolace. Nejprve je vhodné rotovat koulí (respektive jednou reprezentací na povrchu koule) tak, aby se k sobě odpovídající zarovnávané body co nejvíce přiblížily. Tento krok se provádí kvůli možnosti, že jednotlivé body od sebe mohou být velmi vzdálené a dokonce na protějších stranách, což by mohlo v následujících krocích působit problémy. Také by mohla být tělesa oproti sobě značně otočená (jejich významné oblasti definované zarovnávanými body), což tímto krokem také napravíme. V článku je použito zarovnávání, abychom minimalizovali vzdálenosti k bodům, tedy aby suma čtverců vzdáleností mezi korespondujícími body byla minimální. Protože pro naše účely je komplikované stanovit, které body se mají k sobě zarovnat, zejména kvůli odlišnému počtu bodů u oblastí definovaných na šabloně a na svalu, musíme si situaci v této fázi zjednodušit. Stanovíme proto pro každou oblast její střed, který se budeme snažit zarovnávat. Protože si tím problém zjednodušíme na dva body, lze zarovnat jeden bod (středy origin oblastí) přesně a následně rotovat kolem vektoru pozice tohoto bodu tak, abychom minimalizovali vzdálenost druhého bodu (středy insertion oblastí). Přesun insertion bodu na přesnou cílovou pozici stejně jako přesun jednotlivých bodů oblastí na své cílové pozice (viz níže) je zajištěn iterativním postupným přesouváním podle formule uvedené ve vzorci 6.2.

$$\forall i \forall j : v_j = \begin{cases} (v_j + t(d - \|v_j - v_{f_i}\|))^* & if \ \|v_j - v_{f_i}\| < d \\ v_j & if \ \|v_j - v_{f_i}\| \geq d \end{cases}$$

Vzorec 6.2: Přesun bodu na povrchu pomocné promítací koule pro zarovnání oblastí.

Ve vzorci figurují: v_j označující pozici aktuálního bod povrchu, t jako vektor rozdílu cílové pozice a aktuální pozice zarovnávaného bodu, v_{fi} jako aktuální pozice zarovnávaného bodu a d jako zvolený práh. Složitost fáze zarovnávání je $O(NP)$, kde N je počet bodů a P je počet zarovnávaných bodů. Postup lze opakovat a zpřesňovat snižováním d, dokud bude větší než 0.

Posledním krokem, protože vynecháváme krok se slučováním reprezentací (viz kapitola 5.4 o analýze problému dekompozice na svalová vlákna), je převedení pozic z pomocné koule zpět. K tomu je použito barycentrických souřadnic. Pro zjištění, zda se bod nachází uvnitř sférického trojúhelníku, respektive, zda se bod nachází v části prostoru, který je vymezen třemi rovinami spujující vždy dva body trojúhelníku a počátek, použijeme výpočet podle vzorec 2.3. I zde je nutné uvést, že se použití opírá o předpoklad, že jsou všechny trojúhelníky zorientovány stejným směrem. Pokud se bod nachází uvnitř, vypočteme průsečík přímky procházející bodem a počátkem a pro tento průsečík jeho barycentrické souřadnice, tedy vyjádříme průsečík jako kombinaci vrcholů trojúhelníku. Koeficienty lze určit jako obsah protilehlého trojúhelníku tvořeného ze zpracovávaného bodu a ostatních vrcholů trojúhelníku, kde obsah lze určit jako délku vektoru vzniklého vektorovým součinem vektorů mezi vrcholy trojúhelníku a nalezeným průsečíkem. Výpočet také prezentuje obrázek 6.6.



Obrázek 6.6: Výpočet barycentické souřadnice *u* pro bod p_1.

Protože byla vynechána část se smíšením topologií, nebyla implementována datová struktura provazující vrcholy, hrany a trojúhelníky u trojúhelníkové sítě svalu. Bylo proto použito pouze triviální vyhledávání trojúhelníku, což vede na složitost *O(NT)*, kde *N* je počet vrcholů na povrchu svalové šablony a *T* počet trojúhelníků povrchové reprezentace svalu. Částečného urychlení bylo dosaženo přidáním testu, zda se bod nachází nad trojúhelníkem. Tento dodatečný test byl inspirován myšlenkou, že aby se bod na kouli nacházel v trojúhelníku, jehož body také leží na kouli, musí ležet ve směru normály roviny procházející tímto trojúhelníkem.

6.4 Testovací aplikace

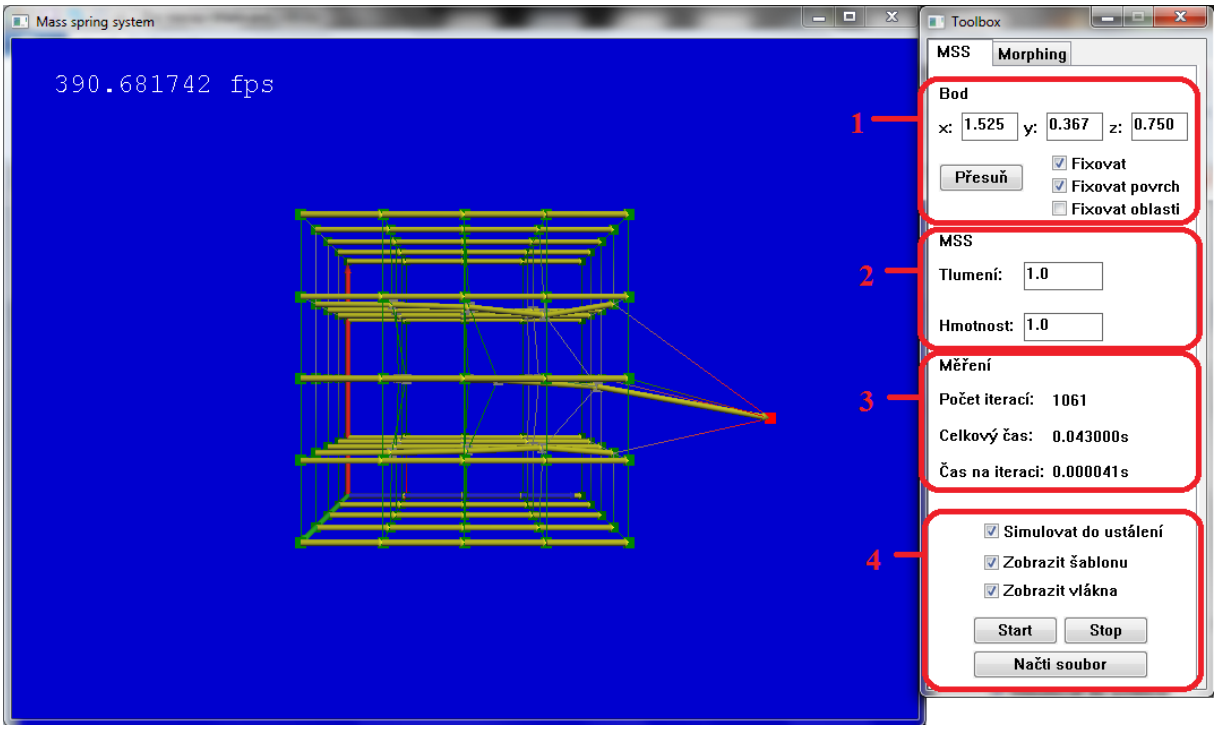
Z důvodu vizualizace průběhu celého procesu, provedení testů ohledně pružinových systémů a zjištění jejich chování pro různé typy provázání a nastavení parametrů byla vyvinuta kromě cílové implementace do prostředí OpenMaf i testovací aplikace. Protože je cílová aplikace orientována na rozhraní OpenGL je i testovací aplikace postavena na tomto rozhraní. Protože nebyl stanoven požadavek na cílovou platformu, byla zvolena jako cílová platforma windows a tedy pro vlastní okenní aplikaci prostředek WinApi. Ačkoli je použití OpenGL na platformě nezávislé, byly užity některé závislé části jako vytvoření renderovacího okna či vykreslení textu (zobrazení počtu snímků). Základ tvoří renderovací okno a okno s ovládacími prvky dále nazývané jako toolbox. Toolbox obsahuje dvě záložky, první s ovládacími prvky systému pružin (záložka MSS) a druhou s prvky pro morphing (záložka Morphing).

Testovací aplikace umožňuje načítání dat pro pružinové systémy (struktura viz přílohy) a nastavení parametrů systému pružin (oblast 2 na obrázku 6.7) a pozice bodů. Pro interaktivní práci bylo implementováno označování a přesouvání bodů pomocí myši (další ovládání viz příloha 2). Této funkce je dosaženo vyrenderováním vrcholů pružinového modelu při stisku tlačítka myši, přičemž do barvy vrcholu zakódujeme jeho index (index = $r*256^2 + g*256 + b$). Barva ostatních pixelů je nastavena na černou (0,0,0). Následným sejmutím barvy pixelu pomocí glReadPixels získáme index. Poté je scéna smazána a vyrenderována standartním způsobem s obarvením vrcholů podle vzdálenosti od pozorovatele a s označeným vrcholem (červená barva). Přesun bodu je umožněn dvěma způsoby. Prvním je možnost zadat pro označený bod do textových polí nové souřadnice bodu a poté použít tlačítko pro přesunutí bodu (oblast 1 na obrázku 6.7). Druhou možností je přesunout vrchol pomocí myši. Nová pozice se určuje v rovině kolmé k pozorovateli na základě množství pixelů mezi pozicemi a úhlem (úhel určený jako poměr k nastavenému field of view úhlu renderevacího okna typicky $\pi/4$). Dále uživatelské prostředí obsahuje tlačítka start a stop (pro spuštění a zastavení výpočtu MSS) a několik prvků typu checkbox (pro zafixování vrcholu či

skupiny vrcholů).

V rámci ovládání morphingu je možné načítat modely svalů jako x-file. Potřeba korektní trojúhelníkové sítě pro morphing povrchu a dosavadní zkušenosti s některými exportéry do x-file souboru vedla k nutnosti zavést test na duplicitní body (každý trojúhelník může mít vlastní vrcholy s unikátními indexy tak, že se indexy vrcholů pro sousední trojúhelníky neshodují). Ovládání je shrnuto do tlačítek pro zarovnání oblastí a výpočet morphingu a slider pro postupnou interpolaci mezi oběma povrchy.

Snímek obrazovky testovací aplikace spolu s označenými oblastmi ovládacích prvků shrnuje obrázek 6.7.



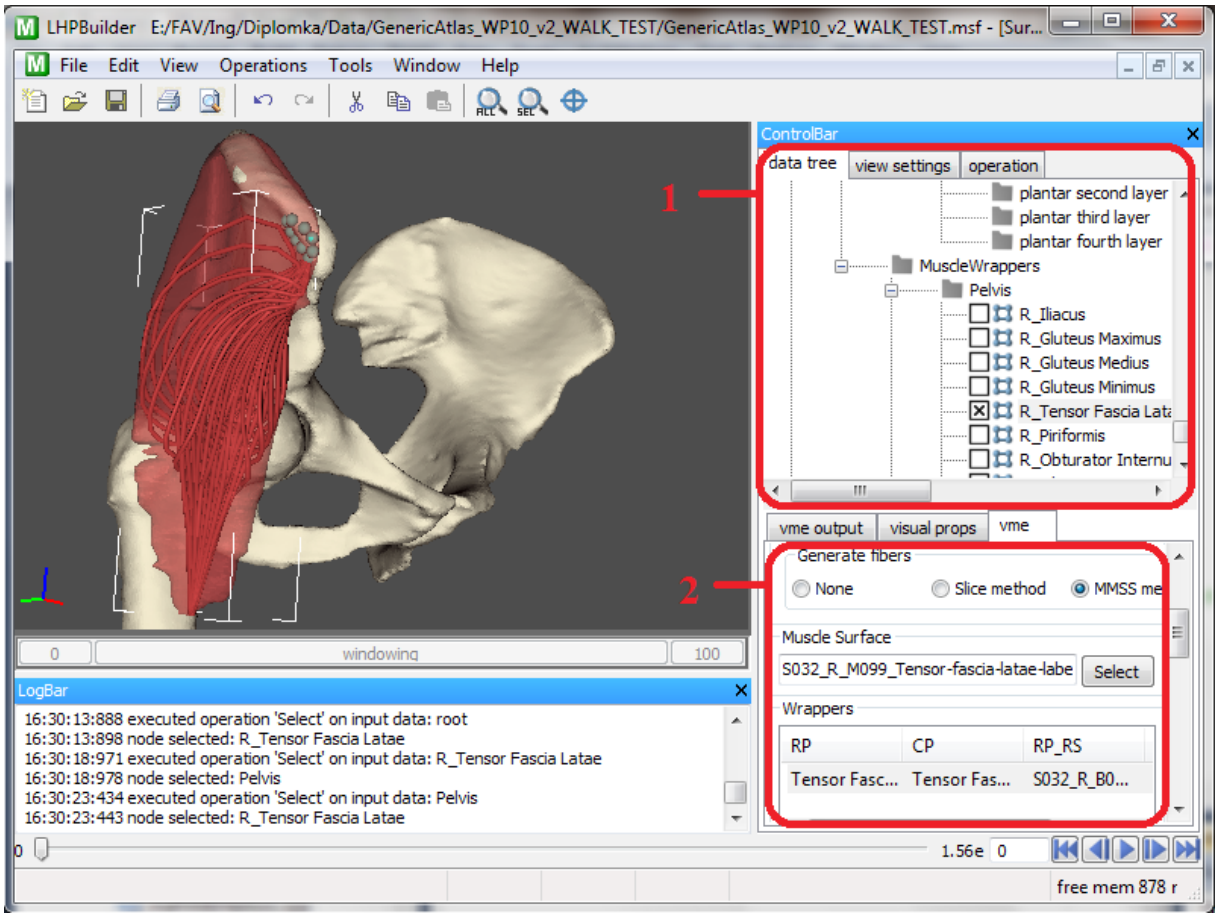
Obrázek 6.7: Ukázka testovací aplikace. Oblast 1 shrnuje ovládací prvky pro označený bod, oblast 2 nastavení parametrů systému pružin, oblast 3 zobrazení měření, oblast 4 ovládání vizualizace a systému pružin.

6.5 Integrace do OpenMAF

Pro integraci do systému OpenMAF je nutné implemenovat převod svalové šablony (jejího objemu tedy vláken) na systém pružin. Svalová šablona je vyjádřena jako kombinace čtyř křivek spojující obě úponové oblasti. Pro jednodušší spojení vláken a jejich bodů pomocí pružin jsem zvolil pravidelné vzorkování (metoda řezů využívá pseudonáhodné rozložení). Pružiny pak lze jednoduše definovat mezi sousedními body.

Struktura OpenMAF je založena na modulovém systému pro vizualizaci dat VTK (visual toolkit). Lze využít toho, že je již zde zaintegrována metoda řezů pro dekompozici na svalová vlákna. Uživatelské rozhraní, které je potřeba rozšířit kvůli ovládacím prvkům pro změnu typu dekompozice, se nachází v podprojektu lhpVME v souboru medVMEMuscleWrapping. Samotná dekompozice je pak zahrnuta do knihovny vtkLHP jako vtkMAFMuscleDecompositionMMSS. Snímek obrazovky aplikace openMaf spolu s označenými oblastmi ovládacích prvků shrnuje obrázek 6.8.

Použitá data svalů dolních končetin jsou volně dostupná po registraci na adrese https://www.biomedtown.org/ jako msf files.



Obrázek 6.8: Ukázka aplikace OpenMaf. Oblast 1 obsahuje ovládací prvky pro zobrazování svalů, kostí a úponových oblastí. Oblast 2 na kartě vme obsahuje ovládání daného modulu.

7 Testy a porovnání

Následující kapitola obsahuje testy a měření systémů pružin a morphingu. Také jsou zde shrnuty poznatky o obou stěžejních částech, stejně jako jejich porovnání se stávajícími metodami. Veškeré níže uvedené naměřené výsledky byly získány na počítači s procesorem intel core 2 quad 2,4GHz, pamětí o velikosti 4GB a grafickou kartou NVidia geforce 9600 GT, 512 MB.

7.1 Testy systému pružin

Kromě nastavení parametrů systému má vliv na simulaci i to, s kolika sousedy spojíme bod. V následujících testech a pozorováních jsou zavedeny dva modely propojení bodů v systémech pružin. V prvním je bod spojen se šesti nejbližšími sousedy (dále označován jako model-6). Ve druhém je bod vázán silněji, tedy se všemi přímými dvaceti šesti sousedy (dále označován jako model-26).

7.1.1 Výpočetní a paměťové nároky

Níže uvedená tabulka 7.1 zachycuje výsledky měření z pohledu časové náročnosti na jednu iteraci pro model-6. Porovnávány byly tři varianty řešení tedy CPU, GPU s pružinově orientovaným přístupem a GPU řešení s navrženým vylepšením za použití geometry shaderu. Z tabulky mimo jiné vyplývá, že pro velmi malé sítě (do tisíce bodů) se přístup pomocí GPU nevyplatí. Výsledek je patrně ovlivněn velkou režií pro přepínání kontextu pro vykreslení a nastavování grafické pipeline. Pohybujeme-li se ve vyšších řádech dochází již ke znatelnému zrychlení a to až k desetinásobnému. Navrhované vylepšení za použití geometry shaderu je jen o málo horší než původní řešení pro velké sítě. Tento výsledek je pravděpodobně zapříčeněn tím, že grafické karty mají největší výpočetní prostředky rezervované pro fragment programy. Umístění výpočtu síly do fragment shaderu je z pohledu rychlosti výpočtu pro model-6 výhodnější, což se zde projevilo. Naopak pro malé sítě je rychlejší navrhovaný způsob s geometry shaderem, neboť zde došlo k avizovanému snížení režie (viz kapitola 6.2.2).

Velikost sítě	CPU [ms]	GPU pružinově orientovaný [ms]	GPU s GS [ms]
10 x 10 x 10	0,25	0,61	0,35
20 x 20 x 20	2,13	0,73	0,57
30 x 30 x 30	7,58	1,1	1,12
40 x 40 x 40	17,64	1,77	2,35
50 x 50 x 50	34,47	3,6	4,33
60 x 60 x 60	59,42	5,31	7,04
70 x 70 x 70	94,61	11,32	13,8
80 x 80 x 80	141,34	18,44	19,65
90 x 90 x 90	203,69	25,65	27,32
100 x 100 x 100	274,12	31,57	33,66

Tabulka 7.1: Porovnání naměřených časů pro model s vrcholy mající valenci 6 na jednu iteraci.

Následující tabulka 7.2 shrnuje provedená měření pro model-26. I zde jsou porovnány všechny tři implementované varianty. Obě GPU řešení jsou výrazně rychlejší než CPU a v případě metody s navrženými úpravami pro GPU i pro velmi malé sítě. Avšak již zde dochází k problémům s pamětí, neboť model-26 má přibližně čtyřikrát více pružin. Problémy nenastávají u CPU varianty, která zvládla vypočítat všechny testované sítě. Obě varianty pro GPU již nikoli. Inovativní přístup však dokázal vypočítat i simulaci na síti 70x70x70 bodů, což v počtu bodů znamená až dvojnásobek oproti porovnávanému GPU způsobu.

Velikost sítě	CPU [ms]	GPU pružinově orientovaný [ms]	GPU s GS [ms]
10 x 10 x 10	0,83	0,95	0,28
20 x 20 x 20	7,52	1,7	1,22
30 x 30 x 30	25,61	3,82	3,84
40 x 40 x 40	61,94	8,46	9,39
50 x 50 x 50	124,36	19,22	19,1
60 x 60 x 60	215,06	34,02	33,35
70 x 70 x 70	344,3	-	53,21
80 x 80 x 80	493,94	-	-
90 x 90 x 90	683,57	-	-
100 x 100 x 100	907,32	-	-

Tabulka 7.2: Porovnání naměřených časů pro model s vrcholy mající valenci 26 na jednu iteraci.

Protože nelze u velkých sítí zanedbat dobu potřebnou pro tzv. readback (stažení výsledků z paměti GPU do paměti CPU), byly provedeny i testy pro její měření, jejichž výsledky jsou obsaženy v tabulce 7.3. Použitím PBO a možnosti asynchronního čtení dojde k urychlení okolo dvaceti pěti procent. Větší odstupy v po sobě následujících hodnotách (například mezi 40x40x40 a 50x50x50) jsou dány odstupňováním velikosti textury na velikost 2^N.

Velikost sítě	Read pixels [ms]	PBO + asynchronní read pixels [ms]
10 x 10 x 10	1,5	1
20 x 20 x 20	1,4	1,1
30 x 30 x 30	2	1,5
40 x 40 x 40	1,8	1,7
50 x 50 x 50	6,3	4,7
60 x 60 x 60	6,9	5,4
70 x 70 x 70	23,2	17,7
80 x 80 x 80	24,8	19,3
90 x 90 x 90	26,3	20,4
100 x 100 x 100	28,1	22,6

Tabulka 7.3: Porovnání rychlosti čtení dat z paměti GPU pro klasické read pixels a asynchronní read pixels s použitím PBO.

Paměťová náročnost pro výpočet systému pružin bude také především záviset na počtu pružin. Označíme-li valenci vrcholu jako V a N jako počet vrcholů, lze zapsat počet pružin podle vzorce 7.1.

$$počet\ pružin \approx \frac{V}{2} * N$$

Vzorec 7.1: Odhad počtu pružin.

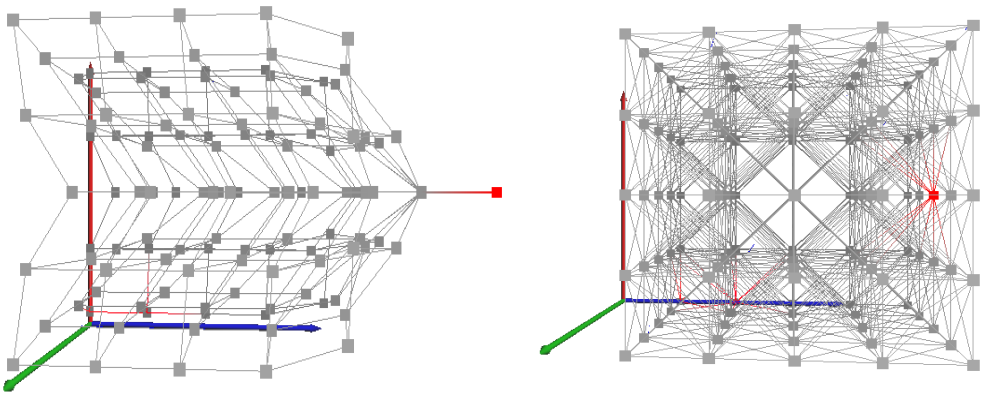
V případě modelu-6 pak podle vzorce 7.1 bude počet pružin přibližně třikrát větší než počet vrcholů. Síť o velikosti 100x100x100 obsahuje jeden milión bodů a počet pružin se tak vyšplhá přibližně k hodnotě tří miliónů pružin. Pro každou pružinu potřebujeme uložit 16B (pro uložení indexů koncových vrcholů, tuhosti a počáteční délky pružiny), což znamená vymezit paměť 48MB pro pružiny a 48MB pro uložení vrcholů (čtyři textury o velikosti počtu bodů, kde pro bod opět potřebujeme alespoň 12B pro uložení vektoru pozice se třemi float hodnotami).

Pro model-26 již však bude počet pružin přibližně třináctkrát větší. To znamená pro stejnou síť třináct miliónů pružin a 208MB pro jejich uložení. Pro uložení vrcholů je zapotřebí stejná velikost jako v případě modelu-6. Tyto výpočty platí pro nový přístup s využitím geometry shaderu. V případě pružinově orientovaného přístupu je nutné ještě ukládat síly v pružinách, což znamená nutnost vymezit další paměť o velikosti tři čtvrtiny uvedených velikostí pro uložení pružin.

Z uvedených skutečností vyplývá, že pokud potřebujeme ještě další paměť grafické karty pro vykreslování, nelze takto velké sítě se systémy pružin pro model-26 simulovat na GPU (myšleno GPU použité při těchto testech s pamětí o velikosti 512MB), což také vyplývá z tabulky 7.2.

7.1.2 Porovnání modelu-6 a modelu-26

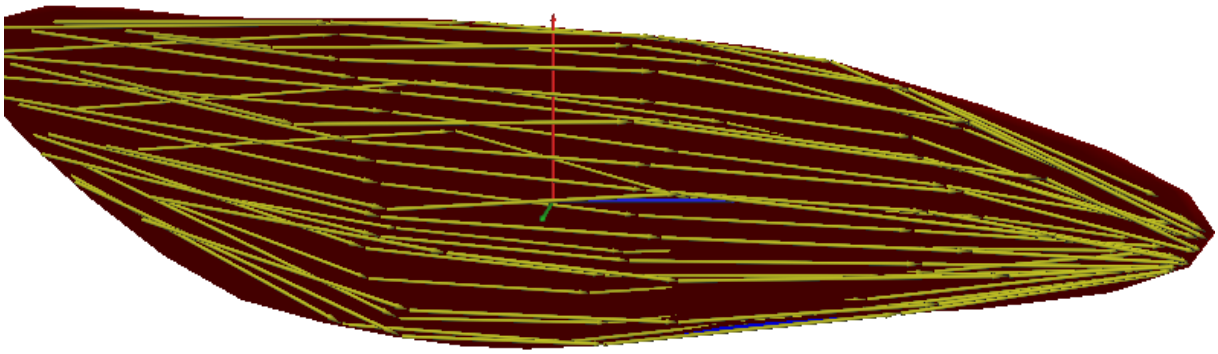
Pro oba modely byla proveden kontrola správnosti simulace z pohledu ustáleného cílového stavu. Bylo tedy otestováno, zda v ustáleném stavu při nezafixování žádného bodu jsou délky pružin shodné s jejich původními délkami. Zde je vhodné uvést, že nezafixujeme-li žádný bod pro model-26 a přesuneme-li body libovolným způsobem, dojde u tohoto modelu vždy k navrácení bodů zpět tak, že je dosaženo identických vzdáleností mezi libovolnými body. Celé těleso se tak navrátí do původního tvaru. U modelu-6 je situace odlišná. Dojde samozřejmě k tomu, že vzdálenosti mezi body spojenými pružinami jsou shodné s původními, ale ostatní vzdálenosti mohou být naprosto odlišné a celý tvar také. Rozdíl se snaží demonstrovat následující obrázek 7.1.



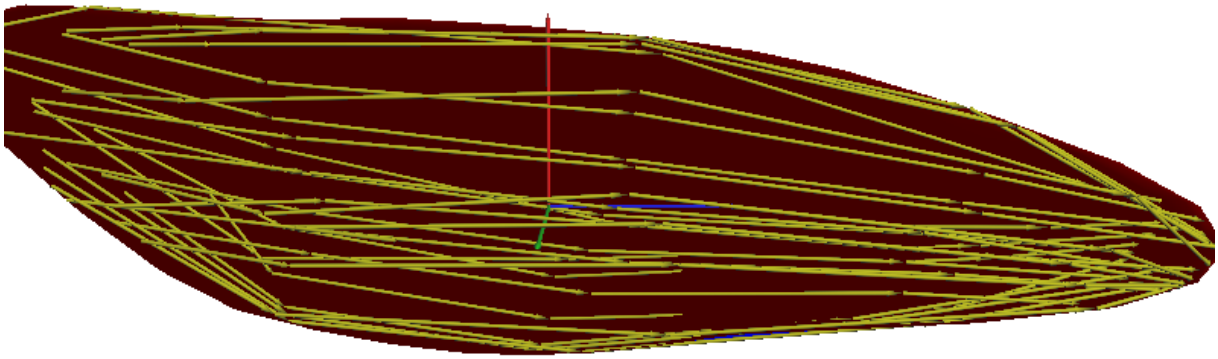
Obrázek 7.1: Možný rozdíl v ustálených stavech nezafixovaných modelu-6 a modelu-26.

Obrázek 7.1 vznikl opakovaným přesunem označeného bodu (červený bod). Bod byl vždy přesunut, poté došlo ke spuštění simulace a po chvíli k zastavení. Tento postup byl několikrát opakován a nakonec spuštěna simulace do ustálení. V případě modelu-6 se simulace zastaví ve vyváženém stavu, kde tvar objektu neodpovídá původnímu tvaru. V případě modelu-26 je původní tvar zachován a dojde pouze k posunu ve směru posouvaného bodu. Obrázek mimo jiné demonstruje, že přesuny bodů působí na další body (i nepřipojené), hlavně ve směrech bodů připojených (v případě modelu-6 frontálně, nahoru, dolů, vlevo, vpravo a u modelu-26 ve všech směrech).

Model-26 se jevil během simulací více stabilněji, ale méně tvárně v tom smyslu, že až příliš držel původní tvar. Posun vrcholu sice ovlivní více vrcholů, ale menším způsobem, neboť je změna distrubuována do více sousedů, kteří jsou navíc silně vázáni ve svých pozicích. Následující dva obrázky reprezentují pozorování ohledně vhodnosti obou modelů pro použití v algoritmu dekompozice na svalová vlákna. Obrázky byly získány z testovací aplikace. První (obrázek 7.2) je výsledný stav algoritmu pro model-6 obsahující body v pravidelné síti 5x5x5 bodů. Druhý (obrázek 7.3) pak pro stejnou síť zachycuje výsledný stav pro model-26.



Obrázek 7.2: Výsledek dekompozice na svalová vlákna testovací aplikace pro model-6.



Obrázek 7.3: Výsledek dekompozice na svalová vlákna testovací aplikace pro model-26.

Z porovnání obrázků 7.2 a 7.3 vyplývá, že použitím modelu-6 získáme znatelně lepší vyplnění prostoru svalu, z čehož lze usuzovat, že pro deformace uvnitř systému pružin bude model-6 postačující. Rozdíl mezi modely je patrně způsoben avizovanými silnějšími vazbami vláken u modelu-26 než v případě modelu-6, což zabraňuje roztažení směrem k povrchu. Problém by šlo patrně vyřešit snížením tuhostí křižných pružin, čímž bychom se ale jen více

přibližovali k modelu-6, který, jak bylo uvedeno výše, má oproti modelu-26 znatelně menší paměťové nároky.

7.1.3 Nastavení parametrů systému

Nastavení parametrů systému je pro naše potřeby důležité zejména z pohledu stability a rychlosti konvergence do ustáleného stavu. Aby platil vzorec pro výpočet systémů pružin (vzorec 3.8) musí být všechny parametry kladné nenulové hodnoty. Hodnota tuhosti byla zvolena jako převrácená hodnota původní délky pružiny z důvodu dobrých vlastností pro naše potřeby jako zachovávání původních poměrů a také tvarů. Tvrzení, že volba parametrů má zásadní vliv na dobu ustálení deklaruje tabulka 7.4.

c	m=8 [kg]	m=4 [kg]	m=2 [kg]	m=1 [kg]	m=0,5 [kg]	m=0,25 [kg]	m=0,125 [kg]
1	6052	3865	2079	1126	579	328	183
2	3190	2018	1007	589	308	158	117
3	2556	1371	681	378	210	179	202
4	1676	962	521	295	220	256	267
5	1355	753	450	243	298	317	326
6	1352	688	390	324	361	375	380
7	1022	577	313	394	419	429	434
8	960	542	390	453	471	479	483
9	730	501	471	506	519	528	532
10	742	438	528	554	568	578	578
11	750	501	577	602	616	622	624
12	712	577	623	651	662	667	669
13	584	632	675	726	707	710	712
14	590	676	726	742	750	753	755
15	638	729	771	786	792	795	796

Tabulka 7.4: Shrnutí měření pro model-6 a síť 5x5x5.

Z tabulky lze usuzovat, že špatnou volbou můžeme dosáhnout i minimálně třicetinásobku doby ustálení (teoreticky i mnohem více) než v případě vhodné volby. Dále z ní plyne, že pro naše účely je vhodné volit nízkou hmotnost, čímž můžeme dosáhnout výrazně kratší délky simulace. Při experimentování lze však také pozorovat, že není vhodné snižovat hmotnost příliš, neboť pak může docházet k nekorektní simulaci či dokonce k numerické nestabilitě, protože hmotností při výpočtech dělíme. Jako vhodnou hodnotu lze doporučit hmotnost řádově v setinách až desetinách při použití šablony o jednotkových rozměrech (hodnoty parametru *r, s, t* od 0 do 1). Tvrzení se opírá i o fyzikální podstatu, neboť hustota svalové tkáně je přibližně 1,1 kg/m^3 a jednotková šablona znamená objem o velikosti 1m^3, tedy hmotnost jednoho bodu pro síť 5x5x5 bude 0,0088 kg (vypočteno jako 1,1 kg/125 bodů).

Následující tabulka 7.5 zobrazuje nejlepší nalezené hodnoty tlumení pro zvolené hmotnosti tak, aby byla doba ustálení minimalizována. Tabulka potvrzuje domněnku, že nižší hodnota hmotnosti má za následek kratší dobu ustálení i pro model-26 a různé velikosti sítí. Doba do ustálení je u modelu-26 přibližně dvounásobně kratší než pro model-6.

	m [kg]	8	4	2	1	0,5	0,25	0,125
5x5x5 model-6	c	13,15	9,35	6,75	4,85	3,45	2,35	1,75
	Počet iterací	578	421	307	231	173	141	91
10x10x10 model-6	c	12,4	8,75	6,15	4,25	3,15	2,2	1,5
	Počet iterací	583	448	320	240	192	142	106
20x20x20 model-6	c	11,8	8,35	5,9	4,1	2,95	1,95	1,35
	Počet iterací	610	461	352	272	203	147	107
5x5x5 model-26	c	26,75	18,85	13,3	9,55	6,7	4,65	3,3
	Počet iterací	266	192	137	105	74	56	42
10x10x10 model-26	c	25,95	18,4	13,1	9,2	6,5	4,6	3,2
	Počet iterací	242	175	138	100	72	54	40
20x20x20 model-26	c	25,15	17,65	12,5	8,85	6,25	4,45	3,1
	Počet iterací	397	224	156	113	90	57	37

Tabulka 7.5: Experimentálně nalezené nejlepší hodnoty tlumení a počet iterací pro danou hmotnost. Porovnání je pro model-6 i model-26 a sítě o 5x5x5, 10x10x10 a 20x20x20 bodů.

Jak bylo uvedeno v kapitole 3.1, je nejlepším možným způsobem tlumení nastavit tzv. kritické tlumení. V tomto tvaru (vzorec 3.5) je však definované pouze pro jeden hmotný bod na pružině, která je navíc na svém druhém konci pevně ukotvena. V našem případě se jedná o podstatně složitější systém, kde hodnotu tlumení nelze jednoduše stanovit. Uvedené poznatky vedly k určení výpočtu hodnoty tlumení experimentálně v závislosti na zvolené hodnoty hmotnosti.

Z výše uvedených důvodů se pokusy o nalezení vztahu mezi parametry týkaly především modelu-6, který bude pravděpodobně v cílové implementaci zvolen jako vhodný. Určení vztahu se opíralo zejména o provedené měření, jehož výsledky jsou uvedeny v tabulce 7.5, a vzorec kritického tlumení (vzorec 3.5). Protože je ke hmotnému bodu připojeno více pružin, budeme při výpočtech uvažovat hodnotu tuhosti jako průměr z tuhostí pružin připojených k bodu. Pružina spojuje dva hmotné body, proto by měly být také zohledněny hmotnosti obou bodů, které jsou shodné. Z tohoto důvodu ve výpočtu figuruje dvojnásobek hmotnosti. Z těchto poznatků byl stanoven vzorec 7.2, který se od vzorce kritického tlumení odlišuje také ve stanovení koeficientu tuhosti nastaveném jako k_6, kde L je průměrná délka pružiny tedy převrácená hodnota průměrné tuhosti.

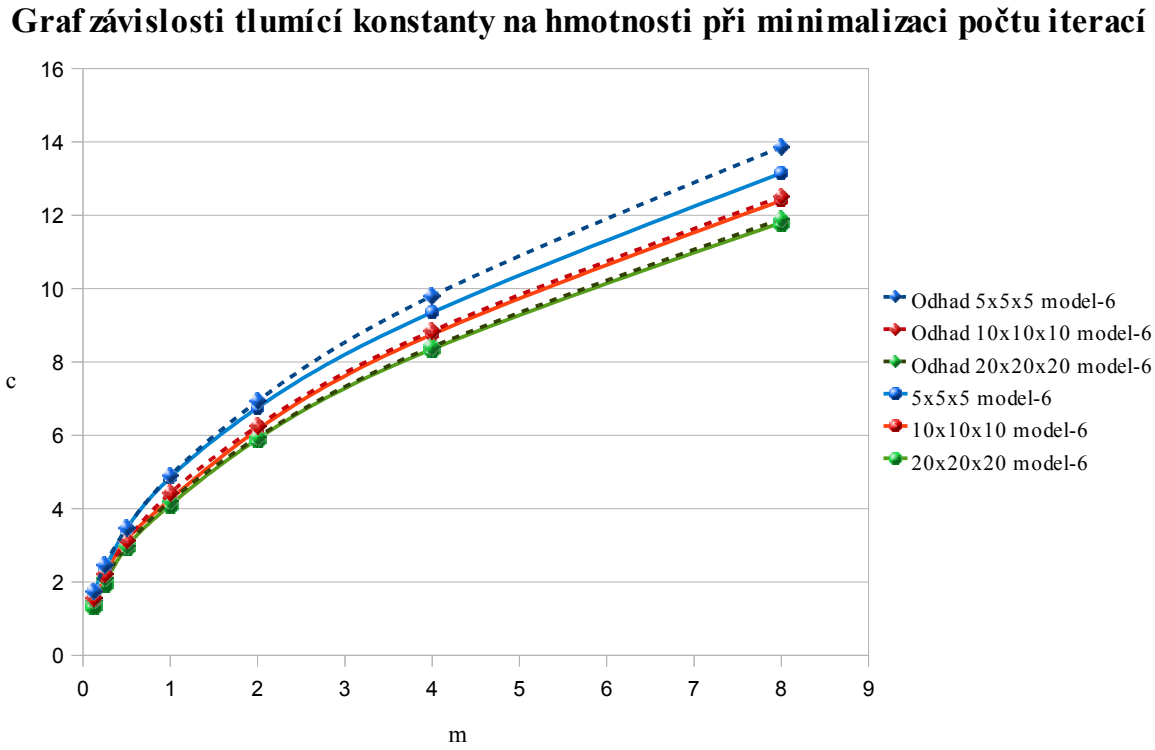
$$c = \sqrt{4\mathrm{k}_6 2\mathrm{m}},\ k_6 = 2 + 4\mathrm{L}$$

Vzorec 7.2: Odhad tlumícího koeficientu.

Tabulka 7.6 ukazuje hodnoty tlumení stanovené podle vzorce 7.2 a doby ustálení. Porovnání vypočtených hodnot tlumícího koeficientu a nejlepších naměřených koeficientů pak zobrazuje graf 7.1.

	m [kg]	8	4	2	1	0,5	0,25	0,125
5x5x5 model-6	c	13,86	9,8	6,93	4,9	3,46	2,45	1,73
	Počet iterací	588	430	312	234	173	125	90
10x10x10 model-6	c	12,51	8,84	6,25	4,42	3,13	2,21	1,56
	Počet iterací	583	446	323	252	192	142	107
20x20x20 model-6	c	11,89	8,41	5,95	4,21	2,97	2,1	1,49
	Počet iterací	610	460	372	271	204	154	120

Tabulka 7.6: Vypočítané hodnoty tlumení pro model-6 a sítě 5x5x5, 10x10x10 a 20x20x20 spolu s počty iterací.



Graf 7.1: Závislost tlumícího koeficientu na hmotnosti při minimalizaci počtu iterací. Čárkované závislosti jsou odhady a spojité jsou naměřené hodnoty.

Tabulky a graf ukazují, že odhad tlumení na základě vzorce 7.2 vytváří poměrně dobré přiblížení k nejlepším naměřeným hodnotám. Z porovnání hodnot počtu iterací v tabulce 7.5 a tabulce 7.6 vyplývá, že se odhad neliší o více než patnáct procent oproti naměřeným hodnotám. Z grafu 7.1 je možné vidět, že se odhad odchýlil především pro síť 5x5x5, což ale nezpůsobuje velký nárůst iterací a v některých bodech dokonce nižší počet iterací. Navíc z měření lze usuzovat, že rozsah přijatelných hodnot je u menších sítí širší než u sítí větších, které odhad stanovuje přesněji. Je vhodné také upozornit, že odhad generuje spíše větší hodnoty tlumení, což je z našeho pohledu lepší, neboť přetlumený systém se při předčasném zastavení více přibližuje cílovým hodnotám a je tedy vhodnější než podtlumený, kde dochází k překmitům.

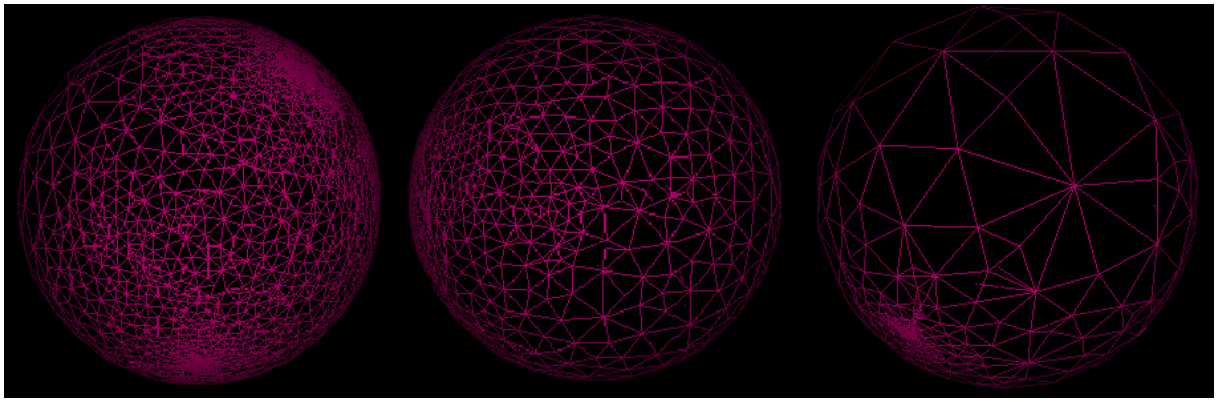
7.2 Testy morphingu

Testování morphingu a dekompozice bylo kromě testovací aplikace (viz obrázky 7.2 a 7.3) provedeno na reálných datech v aplikaci OpenMaf, kde byla provedena integrace. Bohužel extrakce povrchu z volumetrických dat s sebou přináší problémy, které se projeví tím, že většina trojúhelníkových sítí nejsou tělesa typu manifold (neprotínající se a bez děr) a některá dokonce nejsou jediným tělesem. To však porušuje předpoklad, že budeme mít tělesa s genusem 0 a metoda morphingu s tím má značné problémy, neboť dochází k pádům relaxace. Z dostupných dat bylo možné metodu aplikovat na tři možné svaly: tensor fascia latea, gluteus maximus a membranosus. Počet bodů těchto svalů, jejich počet trojúhelníků, čas výpočtu morphingu a počet iterací jsou uvedeny v tabulce 7.7.

	Počet iterací	Čas výpočtu [s]	Počet trojúhelníků	Počet bodů
Tensor fascia latea	1055	6,38	14998	7502
Gluteus maximus	1185	6,44	15000	7502
Membranosus	> 10000	2,37	6512	3258

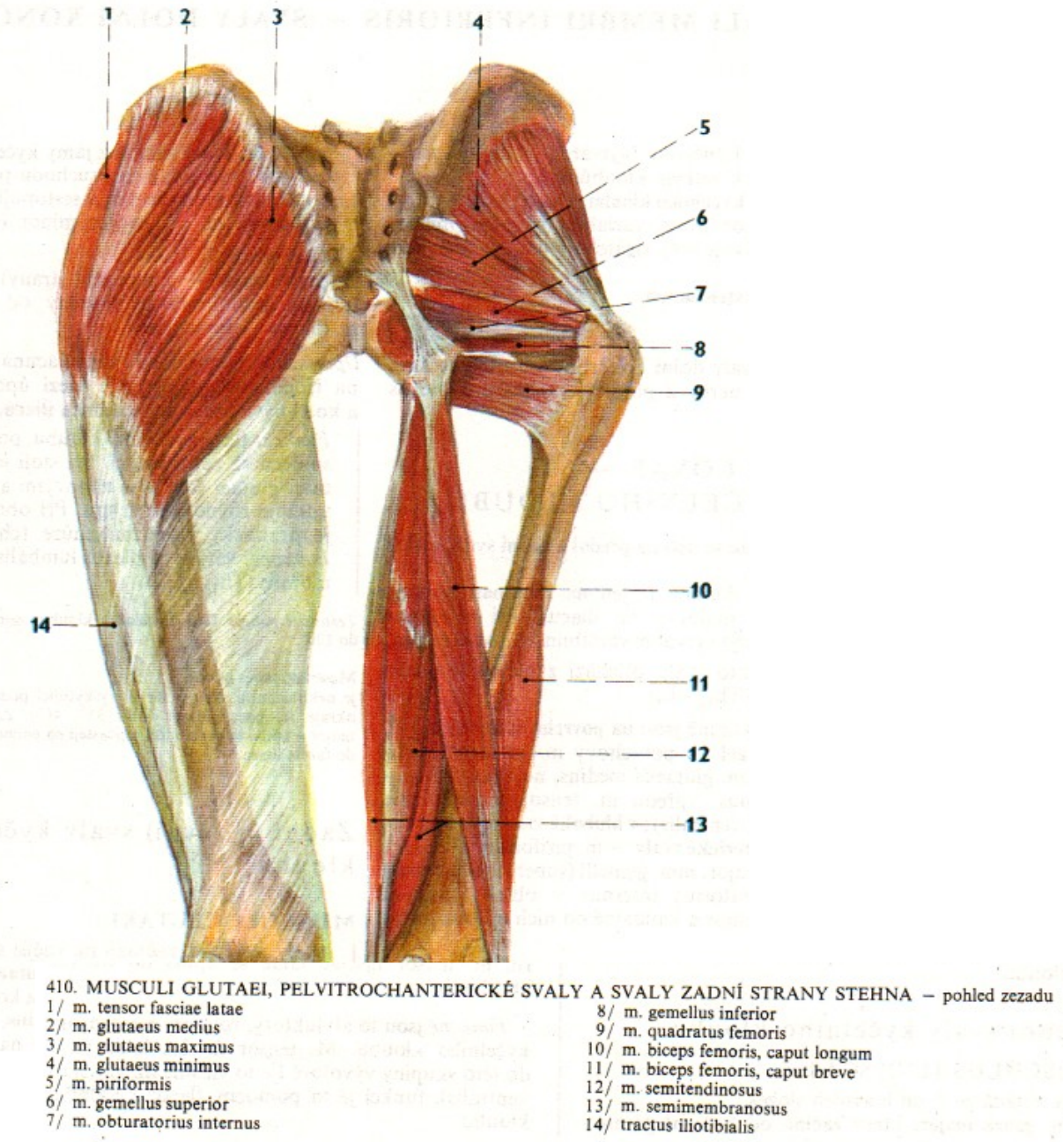
Tabulka 7.7: Souhrn vlastností a měření pro svaly Tensor fascia latea, Gluteus maximus a Membranosus

Tabulka naznačuje problém u svalu membranosus, kde je symbolicky uvedeno, že počet iterací je větší než deset tisíc. Iterace se totiž zastaví až na dodatečné podmínce na počet iterací. Při jejím navýšení však také nedojde k zastavení. Problém je v geometrii svalu, kterou použitá metoda morphingu neumí zpracovat tak, aby měly všechny promítnuté trojúhelníky na pomocné kouli stejnou orientaci. Různá orientace pak působí další problémy při hledání trojúhelníku obsahující bod a brání správné relaxaci. Relaxaci na povrchu koule zobrazuje následující obrázek 7.4. Zatímco pro svaly gluteus maximus a tensor fascia latea, které se více podobají svým tvarem kouli, je převod rovnoměrný, pro sval membranosus, kvůli jeho podlouhlému tvaru, dojde k výskytu dvou oblastí, kde jsou koncentrovány body a zbývající body (poblíž středu svalu) jsou rovnoměrně rozloženy a zabírají nepoměrně více povrchu koule. To způsobuje problémy při mapování šablony, kde dojde k namapování většiny šablony do středu svalu a pouze definované oblasti se upnou na svá místa (viz obrázek 7.7). Navíc zde nejsou, jak bylo uvedeno, správné orientace všech trojúhelníků, což může způsobit, že se nenalezne trojúhelník a vlákna pak kolabují do počátku.



Obrázek 7.4: Promítnutá topologie reprezentace svalů na pomocnou kouli. Zleva pro gluteus maximus, tensor fascea latea, membranosus.

Je také vhodné alespoň podle atlasu lidského těla zkontrolovat, zda má průběh vláken fyziologický smysl.

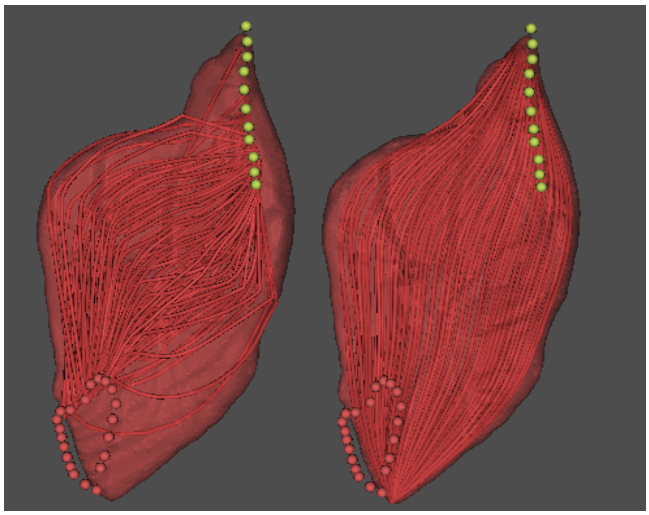


410. MUSCULI GLUTAEI, PELVITROCHANTERICKÉ SVALY A SVALY ZADNÍ STRANY STEHNA – pohled zezadu

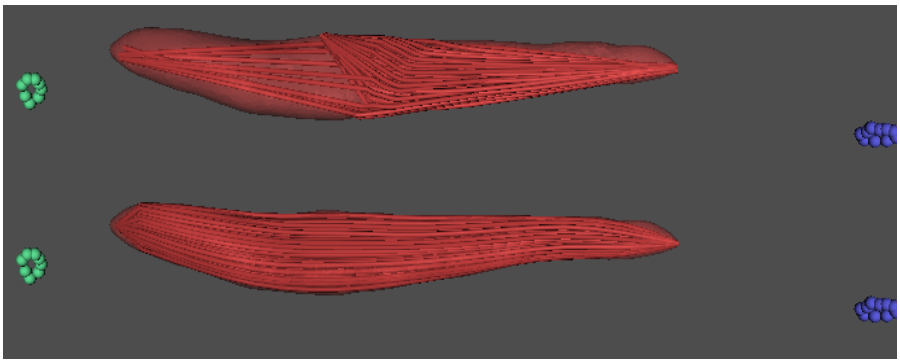
1/ m. tensor fasciae latae
2/ m. glutaeus medius
3/ m. glutaeus maximus
4/ m. glutaeus minimus
5/ m. piriformis
6/ m. gemellus superior
7/ m. obturatorius internus
8/ m. gemellus inferior
9/ m. quadratus femoris
10/ m. biceps femoris, caput longum
11/ m. biceps femoris, caput breve
12/ m. semitendinosus
13/ m. semimembranosus
14/ tractus iliotibialis

Obrázek 7.5: Ilustrace svalů zadní stěny stehna. [ČIH2001]

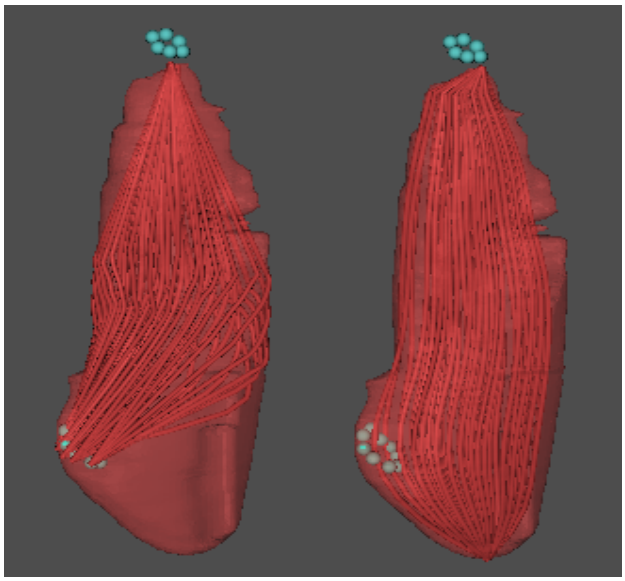
Z porovnání níže uvedených obrázků oproti atlasu je vidět, že vlákna nejsou nějakým zásadním způsobem v rozporu s anatomií. Z uvedených obrázků je také zřejmé, že díky nové metodě dochází k upínání do správných oblastí, což probíhá automaticky bez zásahů uživatele. Metoda má však zjevné problémy na reálných datech. Hlavní problém, jak již bylo uvedeno, jsou podlouhlé svaly, kde relaxace má znatelné potíže se s geometrií vypořádat a nemusí proběhnout dokonce. Mapování tak většinou končí v nežádoucích stavech. U ostatních dvou svalů sice relaxace potíže nemá, i když je zde dvojnásobek bodů i trojúhelníků. Ukazuje se ale další problém a tím je ignorování tvaru poblíž úponových oblastí. To je zapříčiněno tím, že úponové oblasti na šabloně jsou nepoměrně větší než na svalu (tvoří až třetinu povrchu koule). Smrštěním oblastí však není dosaženo korektní roztažení ostatních bodů po povrchu tak, aby lépe vyplňovaly prostor okolo oblastí na svalu.



Obrázek 7.6: Gluteus maximus. Vlevo metoda pomocí pružinových systémů a morphingu, vpravo stávající metoda řezů.



Obrázek 7.7: Membranosus. Nahoře metoda se systémy pružin a morphingem, dole stávající metoda řezů.



Obrázek 7.8: Tensor fascia latea. Vlevo metoda se systémy pružin a morphingem, vpravo stávající metoda řezů.

8 Závěr

V první části práce byl proveden průzkum různých metod pro transformaci povrchů, jako jsou pružinové systémy, morphing a registrace povrchů. Také jsem se seznámil s projektem VPHOP a hlavně s částí pro dekompozici svalů na svalová vlákna. Pro řešení této problematiky byla především provedena analýza možných řešení. Jako vhodný inovativní přístup byl navrhnut postup, který obsahuje jak systém pružin tak techniku morphingu povrchů. Princip spočívá v převodu svalové šablony na pružinový systém a v zafixování povrchových bodů, které jsou následně pomocí techniky morphingu namapovány na povrch svalu. Jejich přesunem a následným spuštěním systému pružin dojde také k namapování vláken šablony do svalu.

Implementovaný systém pružin využívá možnosti výpočtu na GPU. Pro výpočet byla použita inovace dělící nároky na paměť mezi textury a vertex buffer objekt s použitím geometry shaderu, přičemž práce vycházela především z článku o výpočtech pružinových systémů na GPU [GEO2005]. Nová metoda, metoda uvedená v článku a metoda s výpočtem na CPU byly porovnány a otestovány z pohledu výpočetních nároků i vhodnosti pro dekompozici na svalová vlákna.

Jako vhodná technika morphingu byla zvolena metoda s převodem povrchů na pomocný objekt [ALEX2000], kde se zjišťuje korespondence mezi vrcholy a trojúhelníky. Vyjádření bodů lze provést pomocí barycentrických souřadnic a díky tomu převést souřadnice zpět na povrch svalů. Zvolená metoda také umožňuje zarovnání význačných oblastí na správná místa, což bylo požadováno v zadání, neboť potřebujeme svalová vlákna upínat do úponových oblastí.

Nová metoda pro dekompozici svalů na svalová vlákna provádí smyslupnou (ve smyslu anatomie) dekompozici a navíc pracuje automaticky při zarovnávání, díky čemuž se vlákna upínají do oblastí. Nepotřebuje tedy interakci s uživatelem. Během testování v testovací aplikaci na umělých datech vytvořených v modelovacím nástroji probíhala dekompozice podle teoretických předpokladů. Testování na reálných datech vzniklých extrakcí trojúhleníkových sítí z volumentrických dat však byly zjištěny závažné nedostatky. V prvé řadě nejsou vstupní data, až na několik výjimek, typu manifold, ale dochází k protínání, dírám v sítích či dokonce k oddělení na více těles. Tato data nesplňují předpoklad metody na tělesa s genusem 0. Dekompozici bylo možné provést pouze pro tři svaly a to membranosus, gluteus maximus a tensor fascia latea. I na těchto svalech byly zjištěny nedostatky zvoleného přístupu pro morphing povrchů. Hlavním problémem jsou podlouhlé a zakroucené svaly jako membranosus, kde se metoda velmi špatně vyrovnává s geometrií nepodobající se kouli a samotná relaxace nemusí končit v předpokládaném cílovém stavu. U ostatních svalů se ukazuje další problém a to, že úponové oblasti na šabloně a na svalu jsou nepoměrně velikostně odlišné. Smrštění velkých oblastí na povrchu šablony do malých oblastí způsobuje špatné rozlišení svalových vláken u úponových oblastí, čímž dojde k ignorování geometrie poblíž oblastí. Stávající metoda pro dekompozici pomocí řezů [KOH2010] naopak pracuje pro všechny svaly (i s těmi sítěmi co nejsou typu manifold) a hlavně velmi dobře pro podlouhlé svaly, kde je upínání do jednoho bodu korektní. Upínání do jednoho bodu u ostatních svalů je však zároveň hlavním nedostatkem metody. Předpokládané výsledky na syntetických datech v testovací aplikaci naznačují, že použití této metody má smysl, ale bude pravděpodobně zapotřebí přidat spolupráci uživatele a možná změnit metodu morphingu na stabilnější variantu i pro komplexnější geometrie svalů.

Byla naimplementována testovací aplikace pro vývoj systému pružin a ověření jeho vlastností stejně jako vlastností morphingu. Aplikace obsahuje základní funkcionalitu: propojení s OS (windows - winApi), vytvoření okna, zobrazení vrcholů a pružin (pomocí OpenGL), interaktivní ovládání pomocí myši. Pro kontrolu efektů a systému pružin bylo přidáno okno s panelem nástrojů. Kromě vývoje testovací aplikace byla provedena i integrace

do systému OpenMaf, kde se předpokládá využití pro dekompozici svalů na vlákna a kde již je implementována stávající metoda pomocí řezů.

Použité zdroje

[ALEX2000] Alexa, M. *Merging Polyhedral Shapes with Scattered Features.* The Visual Computer. Springer-Verlag. Vol.16, s. 26-37, 2000.

[BEI1992] Beier, T., Neely, S. *Feature-Based Image Metamorphosis.* SIGGRAPH '92: Proceedings of the 19th annual conference on computer graphics and interactive techniques, s. 35-42, 1992.

[ČIH2001] Čihák, R. *Anatomie 1*. 2. vydání. Praha : Grada. 497 s. 2001. ISBN 80-7169-970-5.

[FIS1981] Fischler, M.A., Bolles, R.C. *Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography.* Communications of the ACM, s. 381–395, 1981.

[GEO2005] Georgii, J., Westermann, R. *Mass-Spring Systems on the GPU.* Elsevier Science, 2005.

[HUA2008] Huang, Q-X., Adams, B., Wicke, M., Guibas, L. J. *Non-rigid registration under isometric deformations.* Computer graphics forum, s. 1449-1457, 2008.

[JU2006] Jung, S., Choi, M.-H., Welch, S. W. J. *Fast Volume Preservation for a Mass-Spring System.* IEEE computer graphics and applications, s. 83-90, 2006.

[KENT1992] Kent, J., Parent, R. E., Carlson, W. E. *Shape Transformation for Polyhedral Objects*, Proceedings of SIGGRAPH 92, s. 47-54, 1992.

[KES2010] Kessenich, J., Baldwin, D., Rost, R. *OpenGL Shading Language* [online]. 2010 [cit. 4.2011]. http://www.opengl.org/registry/doc/GLSLangSpec.3.30.6.pdf.

[KOH2010] Kohout, J., Gordon, J. C., Wei, H., Martelli, S., Viceconti, M., Agrawal, A. *Fast muscle wrapping*. Submitted to Computer Methods and Programs in Biomedicine, 2010.

[LER1995] Lerios, A., Garfinkle, Ch. D., Levoy, M. *Feature-based volume metamorphosis.* In Proceedings of the 22nd annual conference on Computer graphics and interactive techniques (SIGGRAPH '95), s. 449-456, 2005.

[MAL2005] Malenovský, E. *Základy lineární teorie kmitání s jedním stupněm volnosti* [online]. 2005 [cit. 4.2011]. http://www.umt.fme.vutbr.cz/~pkrejci/opory/dynamika/.

[NED1998] Nedel, L. P., Thalmann, D. *Real Time Muscle Deformations using Mass-Spring Systems*. In Proceedings of the Computer Graphics International 1998, Washington, USA, 1998.

[PAR2009] Parus, J. *Morphing of Geometrical Objects in Boundary Representation.* University of West Bohemia, PhD thesis, 2009

[PHAN2008] Phannurat, P., Tharanon, W., Sinthanayothin,Ch. *Simulation of elastic soft*

tissue deformation in orthodontics by mass-spring system.The 3rd International Symposium on Biomedical Engineering, 2008.

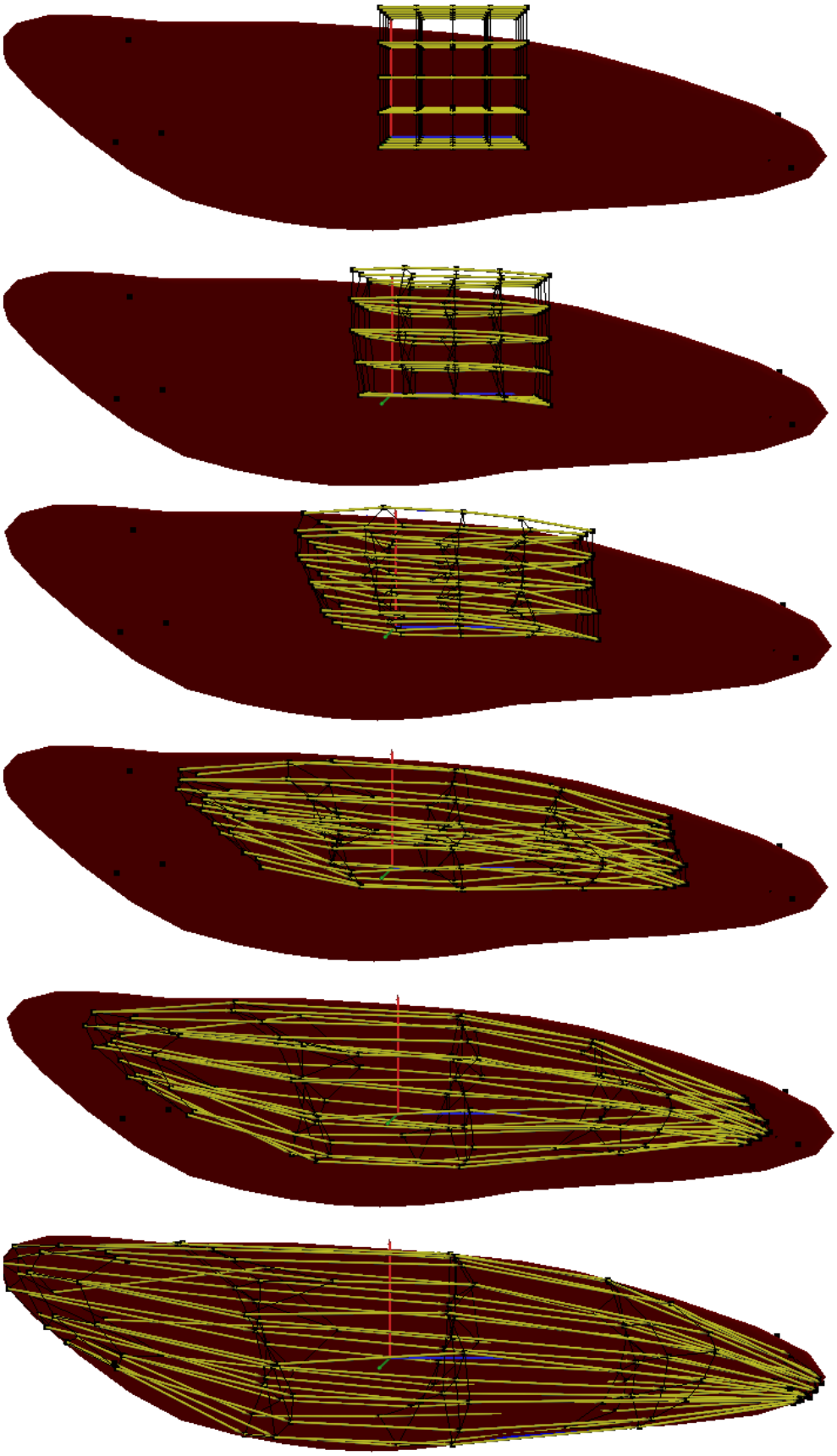
[PRA2001] Praun, E., Sweldens, W., Schröder, P. *Consistent mesh parameterizations*. In proceedings of ACM SIGGRAPH 2001, s. 179-184, 2001.

[RIT1990] Ritter, J. *An efficient bounding sphere.* Graphics Gems. s. 301-303, 1990.

[RU2001] Rusinkiewicz, S., Levoy, M. *Efficient variants of the ICP algorithm.* In proceedings of 3-D Digital imaging and modeling. s. 145-152, 2001.

[ŽBSF2004] Žára Jiří, Beneš Bedřich, Sochor Jiří, Felkel Petr. *Moderní počítačová grafika*. 2.Vydání. Brno: Computer Press, 612 s., 2004, ISBN 80-251-0454-0.

Příloha 1 – průběh dekompozice na svalová vlákna



Obrázek 8.1: Slideshow průběhu dekompozice.

Příloha 2 – uživatelská příručka

a) použití knihovny

Knihovna je vytvořena jako VTK modul a má stejný způsob použití. Vytvoření instance, nastavení vstupů, nastavení výstupu, update, úklid.

```
vtkMAFMuscleFibers* pFibres = NULL;
//create new instance
vtkMAFMuscleDecompositionMMSS* pMD = vtkMAFMuscleDecompositionMMSS::New();
//set inputs
pMD->SetInput(m_pDeformedMuscle);//set input mesh as vtkPolyData
pMD->SetFibersTemplate(pFibres);//set type of fiber template (parallel, pennate,
curved, rectus)
pMD->SetNumberOfFibres(m_FbNumFib);//number of fibers
pMD->SetResolution(m_FbResolution);//points per fiber
pMD->SetOriginArea(ori_points);//origin area as set of points
pMD->SetInsertionArea(ins_points);//insertion area as set of points
pMD->SetSmoothFibers(m_FbSmooth);//set smoothing points along fiber
pMD->SetSmoothSteps(m_FbSmoothSteps);//smoothing resolution along fiber
pMD->SetSmoothFactor(m_FbSmoothWeight);//weights of points along fiber for
smoothing

pMD->SetOutput(m_pDecomposedMuscle); //set output
pMD->Update(); //start counting, result will be in output

pMD->SetOutput(NULL); //disconnect output

pMD->Delete();//clean modul
```

Výstupem modulu jsou také vtkPolyData jako jednotlivá vlákna.

b) ovládání testovací aplikace

Základní ovládání je inspirováno modelovacími nástroji. Veškeré pohyby (posun a rotace) jsou prováděny pomocí myši v kombinaci se stiskem tlačítka (viz tabulka). Pro jemnější rotaci je pak možné použít šipek klávesnice.

Pohyby (rotace, translace)	
Pohyb myši + střední tlačítko myši	Rotace objektu
Scroll myši	Zoom (pohyb v ose z)
Scroll myši + CTRL	Posun vlevo, vpravo
Scroll myši + SHIFT	Posun nahorů, dolů
Šipky klávesnice	Rotace
Selekce, efekty	
Levé tlačítko myši	Výběr vrcholu
Pravé tlačítko myši	Přesun vrcholu

Testovací aplikace předpokládá načtení dat z textového souboru. Data slouží pro načtení pružin, bodů a povrchových trojúhelníků. Struktura souboru je následující:

N O

v1.x v1.y v1.z 'm'/'i'

...

vN.x vNy vNz 'm'/'i'

T

i11 i12 i13

...

iT1 iT2 iT3

S

i11 i12 s1

...

iS1 iS2 sS

F

i11 i12

...

iF1 iF2

OA

i1

...

iOA

IA

i1

...

iIA

Kde N je počet vrcholů a O je počet bodů na povrchu. Vrchol je popsán souřadnicemi v.x, v.y, v.z a znakem m (povrchový bod) nebo i (vnitřní bod). T je počet trojúhelníků. Indexy vrcholů trojúhelníku jsou i1, i2, i3. S je počet pružin. Pružina je definována indexy koncových bodů i1, i2 a tuhostí pružiny s. F je počet vláken, která lze definovat mezi indexy vrcholů (i1, i2). OA je počet bodů tvořící origin úponovou oblast a IA naopak insertion úponovou oblast. Obě jsou pak definovány indexy vrcholů, které je tvoří.