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

Bakalářská práce

Rychlá detekce kolizí v kontextu deformace svalů metodou Position Based Dynamics

Plzeň 2021 Ondřej Havlíček

ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta aplikovaných věd

Akademický rok: 2020/2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Ondřej HAVLÍČEK**
Osobní číslo:	**A17B0210P**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Informatika**
Téma práce:	**Rychlá detekce kolizí v kontextu deformace svalů metodou Position Based Dynamics**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se s výsledky diplomové práce „Deformace svalových vláken systémem vázaných částic“ z roku 2019.
2. Popište různé přístupy k detekci a řešení kolizí (CD & CR) deformovatelných objektů.
3. Prozkoumejte alespoň 5 metod pro rychlou detekci kolizí deformovatelných těles a proveďte jejich srovnání na teoretické úrovni. Zaměřte se zejména na metody, které mají veřejně dostupnou implementaci.
4. Po dohodě s vedoucím práce vyberte alespoň 2 dostupná řešení a proveďte jejich implementaci do stávajícího řešení.
5. Otestujte vytvořené řešení a výsledky důkladně zhodnoťte.

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

Seznam doporučené literatury:

Dodá vedoucí bakalářské práce.

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

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

L.S.

Doc. Dr. Ing. Vlasta Radová
děkanka

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

V Plzni dne 26. října 2020

Prohlášení

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

V Plzni dne 6. května 2021

Ondřej Havlíček

Poděkování

Vřele děkuji panu Doc. Ing. Josefovi Kohoutovi, Ph.D. za odborné vedení, věcné připomínky a velkou vstřícnost v průběhu zpracovávání této práce.

Abstract

This work focuses on finding the most fitting method for fast collision detection and resolution in the context of muscle deformation using a Position Based Dynamics Method. First this work describes the problem of collision detection and resolution in general, and then some methods to solve this problem are presented. Five various publicly available sources implementing some of these methods were compared. Based on the comparison, two of these methods were chosen and integrated into the current solution described in the work of M. Červenka from the year 2019 [4]. The resulting methods were tested using scenarios predefined in the Muscle Wrapping 2.0 project. One of the methods met both quality and time requirements and was found to be a suitable successor to the existing solution.

Abstrakt

Tato práce je zaměřena na nalezení nejvhodnější metody pro rychlou detekci a řešení kolizí v kontextu deformace svalů metodou Position Based Dynamics. Nejdříve je v práci popsána problematika řešení a detekce kolizí obecně a následně jsou představeny některé metody k řešení této problematiky. Došlo ke srovnání pěti různých veřejně dostupných implementací, využívajících některé tyto metody. Na základě srovnání pak byly dvě implementace vybrány a integrovány do stávajícího řešení práce M. Červenky z roku 2019 [4]. Nad výslednými metodami byly provedeny testy pomocí scénářů předdefinovaných v rámci projektu Muscle Wrapping 2.0. Jedna z metod vyhovovala jak kvalitativním tak časovým nárokům a byla shledána vhodným nástupcem stávajícího řešení.

Obsah

1 Úvod

Diplomová práce Ing. Martina Červenky (Deformace svalových vláken systémem vázaných částic z roku 2019 [4]), ze které tato bakalářská práce částečně vychází, řeší problém modelování dynamiky svalů.

Model dynamiky svalů se dá představit jako systém objektů, které se v prostoru pohybují v důsledku simulovaných fyzikálních sil. Při těchto pohybech ovšem může docházet k nerealistickým protnutím (kolizím) různých objektů, v tomto případě svalů a kostí. Je třeba protnutí v reálném čase simulace detekovat a řešit návrat objektů do realistického stavu. V diplomové práci M. Červenky z roku 2019 [4] řešení tohoto problému poskytuje nepřesné výsledky, detekce kolizí byla totiž zjednodušena pro urychlení simulace na úkor přesnosti [4]. Zejména v blízkosti kloubů se pak svaly *zasekávají* mezi kostmi a nepřirozeně deformují [4]. Stávající řešení, uvedené ve zdroji [4], tudíž nefunguje k plné spokojenosti.

Cílem této práce je zmíněné nedostatky analyzovat a reagovat na ně nalezením lepší metody pro rychlou detekci a řešení kolizí pro elastická tělesa. Nová metoda by měla být zejména přesnější a tím předcházet nerealistickým stavům simulace. Zároveň by měla být metoda dostatečně rychlá, protože stávající aplikace, pro kterou je tato metoda určena, probíhá v reálném čase. Konkrétně bude hledání metody v rámci bakalářské práce probíhat ve formě teoretického srovnání pěti dostupných metod, řešících tento problém. Z metod budou na základě srovnání dvě s existující veřejnou implementací integrovány do stávajícího řešení. Nad metodami budou provedeny testy a srovnání výsledků společně s původní metodou v práci M. Červenky [4].

Nejprve bude představena obecná problematika detekce a řešení kolizí v počítačové grafice. Dále budou popsány způsoby reprezentace simulovaných těles, mezi kterými detekce a řešení kolizí probíhá. Následně budou vylíčené různé druhy těles významné pro tuto práci a jejich odezvy na vnější síly. V další části práce budou do detailu popsány různé metody pro detekci kolizí. Pak bude představena metoda PBD použita v práci, do které budou zvolené metody integrovány. Nato bude popsán proces integrace a nakonec bude představeno testování integrovaných metod a vyhodnocení výsledků tohoto testování.

2 Problematika detekce a řešení kolizí

Kolize nastává, jestliže se dvě simulovaná tělesa pohybují a narazí do sebe. Detekce kolize (collision detection) je test, při kterém je zkoumáno, zda se tato dvě tělesa protnou/protínají. Jestliže test kolizi zaznamená, je třeba na ni patřičně reagovat. Tato reakce se označuje jako řešení kolize (collision resolution). Řešení kolize spočívá v aplikování kolizních sil na obě tělesa. K aplikaci sil může být žádoucí vědět jakým směrem a jakou rychlostí se objekt dostal do druhého či zjistit vnitřní, nebo povrchové tření obou z objektů. Tyto síly se chovají jako externí síly působící na tělesa podobně jako gravitace či odpor vzduchu. Na základě zjištěných parametrů je pak snaha uvést objekty do konzistentních stavů, ať už *uniknutím* z nastalé kolize či zabráněním budoucí kolize [3].

Existuje několik přístupů k problematice detekce kolizí obecných objektů. Konkrétně je rozlišována posteriorní neboli diskrétní detekce, kdy je kolize zaznamenána až po tom, co se stala a apriorní neboli kontinuální detekce, kdy se kolizi předchází. Posteriorní detekce hlásí vzniklou kolizi, a tak je následovně potřeba řešit navrácení střetlých objektů do konzistentního stavu. Oproti tomu apriorní detekce hlásí kolizi ještě předtím, než se stane, a tak nastává v simulaci *časové okénko* k uniknutí od kolize. Apriorní detekce má potenciál lépe odpovídat realitě (neboť nedojde k protnutí objektů) a často je obtížnější na implementaci.

Řešení kolizí pak může být různé v závislosti na požadovaných vlastnostech materiálu. Například modelování vln v moři do určité míry dovoluje, aby se *objekty vln* z části procházely. Při modelování dopadu činky na podlahu je naopak nežádoucí, aby na pár snímků činka procházela podlahou.

Pokud probíhá simulace v reálném čase, je vhodné, aby byl výpočet pro detekci kolizí časově efektivní. Taková detekce je označena jako rychlá. K urychlení výpočtů detekce se často zanedbává několik vlastností těles, konkrétně v kontextu této práce je to např. tření svalu o kost či tzv. sebeprotnutí. Požadovanými vlastnostmi jsou naopak tzv. anizotropie (svalový model má různé vlastnosti v různých směrech působících sil [4]), zachování vzdáleností mezi body, zachování objemu tělesa a zachování tvaru tělesa (po deformaci svalu a návratu modelu do původního stavu dojde k obnově původního tvaru svalu). K propagaci vlastností je možné použít různé reprezentace těles.

3 Reprezentace těles

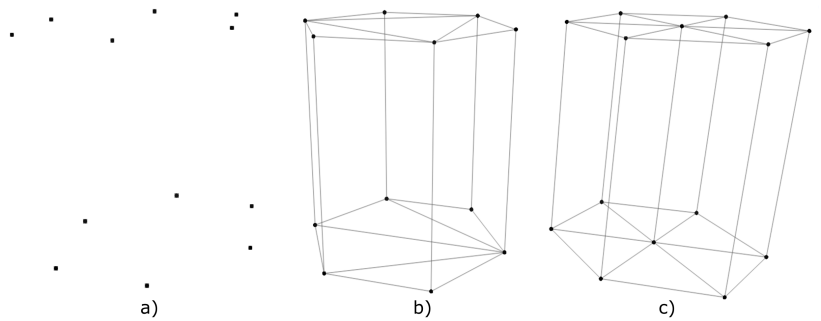
Objekty různých vlastností mohou být v kontextu počítačové grafiky rozčleněné do dvou hlavních kategorií. Těmi jsou tuhá tělesa (rigid bodies) a elastická tělesa (soft bodies). Dále jsou na pomezí těchto kategorií definovaná plastická tělesa (plastic bodies). Po tělesech z různých kategorií je požadováno různé chování, např. jiné odezvy na kolizi. V této práci budou popsány pouze typy těles významné pro modelování kostí (tuhá tělesa) a svalů (elastická tělesa).

Simulovaným tělesem je myšlen jakýkoliv objekt existující ve scéně, tedy např. model svalu či kosti. Základem simulovaného tělesa je množina vrcholů např. trojúhelníkové sítě. V simulovaných těles je často třeba zachytit různé fyzikální vlastnosti těles a jejich materiálů. Tyto vlastnosti bývají přiřazeny vrcholům trojúhelníkové sítě. Vlastnosti části objektu (zachycené v jednom vrcholu) ovšem v realitě často ovlivňují i určité okolí této části. Při simulaci proto dochází mezi vrcholy k jakési *propagaci* vlastností jednoho vrcholu do ostatních. Propagace vlastností způsobuje, že jsou vlastnosti jednotlivých vrcholů ovlivněny vlastnostmi vrcholů v určitém okolí.

V případě že je těleso konvexní, může ho být vhodné popsat pouze soustavou hmotných bodů (ve vrcholech) v prostoru bez struktury (meshless, obrázek 3.1, **a)**). Každý bod je v tomto případě ovlivněn všemi ostatními body tělesa, a tak je nutné vypočítat vliv pro všechny dvojice bodů, což ústí ve $\Theta(N^2)$ výpočetní složitost.

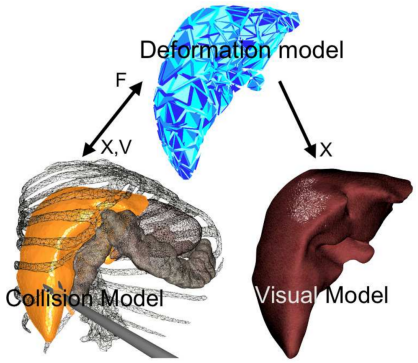
Pokud jsou ovšem modelována nekonvexní tělesa a je uváženo, že jsou materiály tělesa a scény nehomogenní, může být vhodné těleso reprezentovat povrchovou sítí (surface mesh, obrázek 3.1, **b)**), ve které jsou k dispozici navíc hrany spojující vrcholy tělesa, uvádající jeho tvar. Hrany tedy tvoří např. trojúhelníky či jiné n-úhelníky. Z porvchové sítě lze získat informaci, zdali je mezi dvojicí vrcholů těleso definováno, a tudíž jakými materiály s jakou mírou se má vlastnost šířit.

V posledním případě lze těleso reprezentovat objemovou sítí (volume mesh, obrázek 3.1, **c)**). Hranami jsou navíc spojeny nejen vrcholy na povrchu tělesa, ale i vrcholy uvnitř tělesa. Tato reprezentace může poskytovat za vyšší paměťovou náročnost lepší diskretizaci tělesa a tudíž je možné počítat vliv vrcholů pouze vzhledem k sousedním vrcholům. Navíc je možné jednotlivým částem objemu tělesa přiřazovat různé materiály, a tak realističtěji simulovat propagaci vlastností nehomogenních, nekonvexních těles.



Obrázek 3.1: Různé reprezentace tělesa (zjednodušeně). Zleva je uvedena meshless reprezentace tělesa (**a)**), následně mesh reprezentace pomocí povrchové sítě (**b)**) a mesh reprezentace pomocí objemové sítě (**c)**).

Je možné těleso reprezentovat na různých úrovních pro různé účely (multi-model representation [6]). Tyto reprezentace mohou být propojené příslušným mapováním a mohou si vzájemně posílat data a tím propagovat např. deformaci jedné reprezentace do jiné. Navíc mohou být různé modely reprezentované jinými počty bodů a tím poskytovat požadovaný detail. Příklad více-úrovňové reprezentace je uveden na obrázku 3.2. Význam těchto reprezentací je převážně v urychlení výpočtů [6]. Například při simulaci deformace není třeba realistické vykreslování, a tak je těleso izolováno pro řešení aktuálního kroku.



Obrázek 3.2: Více-úrovňová reprezentace orgánu ledviny (obrázek je převzatý ze zdroje [6]). Mapování mezi modely znázorněné černými šipkami značí následující: z kolizního modelu se propagují síly **F** do deformačního modelu. Ten po aplikaci sil propaguje výsledek ve formě aktualizovaných pozic **X** a nových rychlostí **V** zpět do kolizního a vizualizačního modelu [6].

Informace o vrcholech trojúhelníkové sítě bývají uchávávány v datové struktuře tzv. částice. Částice může obsahovat kromě informace o své pozici v souřadném systému i různé fyzikální vlastnosti vrcholu, tedy hmotnost, směr

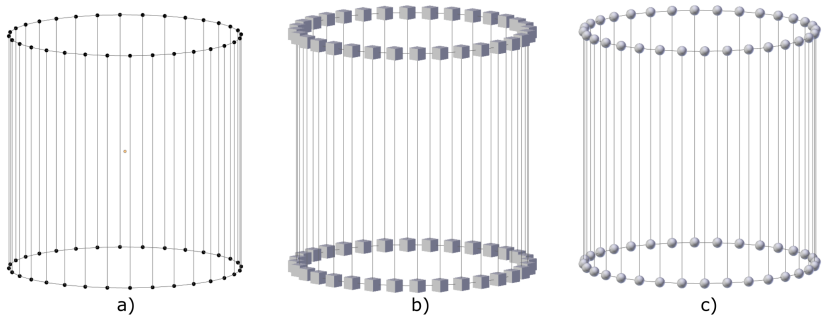
pohybu či vnitřní energii, potřebné pro výpočet propagovaných vlastností. Detekce kolizí je ovšem mezi dvěma částicemi bez rozměru téměř nemožná, neboť k takové kolizi dochází pouze v případě, že částice sdílí stejnou pozici v souřadném systému. Proto je často zaváděna úroveň reprezentace tzv. kolizní model tělesa.

3.1 Kolizní model tělesa

Neboť reprezentace bezrozměrnými částicemi není pro detekci kolizí vhodná, jsou částice nahrazovány rozměrovými primitivy. Pokud je těleso reprezentováno meshless, je nutné částice nahradit např. krychlemi či koulemi o příslušných rozměrech. V jiném případě, pokud je k dispozici reprezentace tělesa nějakou sítí, lze za primitiva považovat např. trojúhelníky, jejichž vrcholy jsou ony částice.

Reprezentace trojúhelníkem je vhodná, neboť je značná část simulovaných modelů popsána trojúhelníkovou sítí. Je ovšem obtížné předpovídat či detekovat již vzniklou kolizi, neboť při modelování ve 3D prostoru existuje mnoho vzájemných poloh těchto dvou primitiv [4]. Primitiva se totiž mohou lišit v prostoru nejen polohou, ale i orientací neboli úhlem mezi normálami.

Počet všech možných vzájemných poloh dvou primitiv klesá, jestliže jsou částice aproximovány např. čtyřstěny či koulemi. Oproti trojúhelníkům mají tato primitiva výhodu v tom, že není třeba vyšetřovat orientaci částic (krychle mohou být generovány v zájemné rovnoběžnosti), a tak zbývá otestovat vzdálenost dvou takových primitiv. V případě reprezentace koulemi s danými poloměry probíhá test následovně: jestliže je vzdálenost středů koulí menší nebo rovna součtu jejich poloměrů, nastává kolize. Ilustrační příklady přemodelování jsou uvedeny na obrázku 3.3.



Obrázek 3.3: Reprezentace částic primitivy: (a) reprezentace bez primitiv, (b) reprezentace částic krychlemi, (c) reprezentace částic koulemi.

4 Reakce těles na vnější síly

Na modely kostí (tuhá tělesa) a svalů (elastická tělesa) v průběhu simulace působí simulované vnější síly. Reakce tuhých a elastickcých těles na tyto síly se ovšem výrazně liší. Za tuhá tělesa můžeme považovat obecně objekty, jež mají velmi nízkou či zanedbatelnou pružnost materiálu (sklo, keramika, beton, ...). Jako elastická tělesa mohou být klasifikovány např. 1D objekty, jako jsou šaty, vlasy či tekutiny, nebo 3D objekty, jako jsou svaly nebo např. gumové objekty.

4.1 Tuhá tělesa

Tuhými tělesy jsou modelovány objekty, ve kterých vystupují natolik velké vnitřní síly, že je možné na objekty pohlížet jako na jakousi soustavu hmotných bodů [10]. Tyto body jsou od sebe po dobu celé simulace konstantně vzdáleny o iniciální vzdálenosti a obsahují informace o své poloze ve formě souřadnic.

Při kolizi tuhého tělesa v realitě dochází k deformaci např. ve formě fragmentace (prasklá kost) při překročení tzv. meze úměrnosti materiálu. Tuto možnost ovšem v kontextu deformace svalů neuvažujeme, a tak je řešení kolize tuhého tělesa značně jednodušší a k deformaci takového tělesa vlastně v simulaci nedochází. Z toho plyne, že reakce tuhého tělesa na externí síly je nakonec kombinace jednoduchého posunu ve směru aplikované síly a rotace tělesa. Invariance tvaru tuhého tělesa k vnějším silám je znázorněna na obrázku 4.1. Z obrázku je patrné, že tvar tělesa neovlivňuje počáteční síla F ani externí síly a dochází pouze k posunu.

4.2 Elastická tělesa

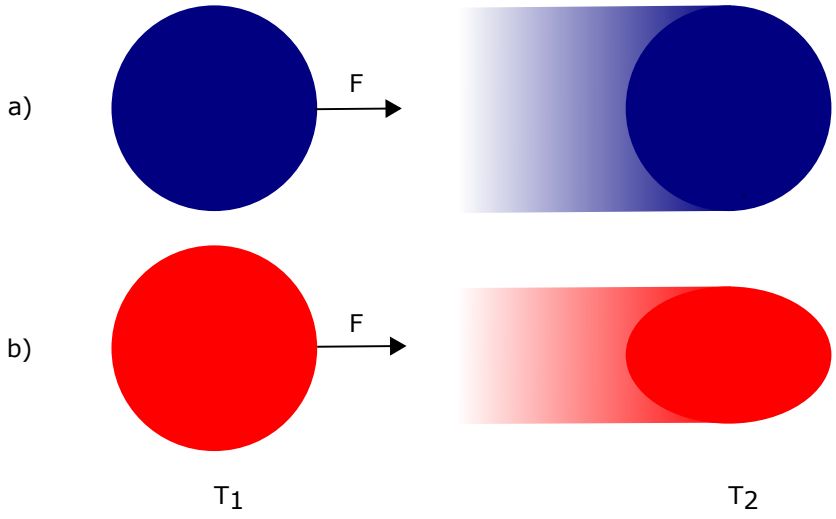
V elastických tělesech oproti tuhým vystupují vnitřní síly, které ovlivňují pohyb bodů. Soustava bodů tělesa je obohacena o dodatečné informace přiřazené každému bodu, a tak dokáže těleso reagovat na různé síly modelovaného prostředí. Těmito informacemi jsou např. rychlost a směr pohybu, zrychlení či hmotnost bodu, podle toho, jaké informace jsou potřeba pro metodu řešící pohyb těchto bodů. Kromě těchto základních informací o kinematice bodů se mohou brát v potaz další fyzikální vlastnosti, např. tření, hmotnost, tlak či vnitřní energie.

Body elastického tělesa reagují na kolize změnou polohy neboli posunem

o příslušný vektor. Směr i velikost tohoto vektoru jsou ovlivněny vlastnostmi materiálů obou kolidujících těles. Tyto vlastnosti jsou reprezentovány vnitřními silami a kinematickými vlastnostmi přiřazenými kolidujícím bodům. V rámci realistického zachování vlastností materiálů kolidujících těles (vzdáleností bodů, tvaru, objemu, ...) dochází navíc k ovlivnění polohy ostatních bodů.

V kontextu modelování svalů jsou požadovány vlastnosti zachování vzdáleností jednotlivých prvků modelu, zachování objemu celého modelu a zachování jeho původního tvaru [4]. K těm je nakonec možné přidávat další restrikce [4], viz následující kapitola.

Elastický model reflektuje při kolizi oproti tuhému tělesu i deformaci, avšak za cenu vyšší výpočetní složitosti. Oproti tuhému tělesu, kde všechny body při aplikaci externích sil vzájemně drží konstantní vzdálenosti, se zde pracuje s informacemi přiřazenými jednotlivým bodům, a tak se tyto body posouvají o různé vzdálenosti v různých směrech v závislosti na aktuálních kinematických a fyzikálních stavech bodů a vnitřích silách, působících na tyto body. Probíhá tedy elastická deformace. Po ukončení působení externích sil je předpokládáno, že se těleso navrátí do původního stavu (před deformací). Deformace elastického tělesa je znázorněna na obrázku 4.1, kde počáteční síla F a externí síly (zde odpor vzduchu) působí na těleso a vyvolávají v něm nenulové vnitřní síly.



Obrázek 4.1: Reakce tuhého (a) a elastického tělesa (b) na počáteční sílu F zaznamenána v časech T_1 a T_2.

4.3 Deformace elastických těles

Přístupy pro řešení kolizí 1D a 3D elastických těles se liší. Konkrétně nad 1D objekty oproti 3D objektům často není třeba konstruovat složité hierarchie ale stačí jakási struktura segmentů [13].

Elastické objekty kromě dimenzionality rozlišujeme podle očekávané reakce na kolizi. U objektů jako jsou právě šaty, vlasy či tekutiny je potřeba frekventovaně aktualizovat stav tělesa, neboť je zdánlivě v nepřetržitém pohybu. Tato aktualizace se může provádět např. výpočtem deformace jednoho bodu pouze vzhledem k několika okolním bodům. Těleso pak reaguje nepřetržitě na okolí, zatímco zachovává původní strukturu, neboť deformace jednoho bodu příliš neovlivňuje zbytek tělesa (tekutina zůstává v nádobě).

Elastická tělesa, kterými jsou v kontextu této práce především myšleny modely svalů, jsou při aplikaci externích sil deformována. Deformace probíhá již zmíněným propagováním vlastností a sil mezi body modelu. Propagaci deformace je možné implementovat několika standardními metodami. Rychlost konvergence metod je často závislá na velikosti a úrovni detailu vstupních dat. Často je potřeba pro specifický účel dodat k matematickému modelu modifikace, v tomto případě je potřebné zajistit, aby metoda obsahovala řešení restrikcí či aby bylo možné je pokud možno jednoduše dodat.

První standardní metodou je finite element method (dále jen FEM) neboli metoda konečných prvků. Metoda funguje na základě numerické aproximace pomocí bázových funkcí a provádí 3D rozdělení objemové sítě tělesa na co nejmenší části, kterým je možné přiřazovat vlastnosti a ty propagovat do sousedních částí. Podle počtu iterací může metoda poskytovat velmi přesné výsledky deformace (obecně přesnosti druhého až třetího řádu [7]), avšak na úkor vyšší výpočetní náročnosti. Použití této metody nemusí být vhodné pro simulaci deformace svalů v reálném čase, neboť vysoká výpočetní náročnost často neodpovídá nárokům na rychlost [10]. Navíc metoda konečných prvků sama o sobě neřeší konzervaci požadovaných atributů během deformace, a tak by byly potřebné další modifikace či nadstavby již tak složitého řešení [10].

Další metodou je finite volume method (dále jen FVM) neboli metoda konečných objemů, která používá reprezentaci tělesa pomocí jednoduchých objemových primitiv podobně jako metoda konečných prvků. Rozdíl oproti předchozí metodě spočívá ve způsobu diskretizace výpočtu. Metoda konečných objemů vypočítává potřebné vlastnosti modelu pro každé primitivum individuálně a tyto mezivýsledky jsou v průběhu výpočtu používány pro výpočet vlastností sousedních primitiv napříč celým objemem modelu, což výpočet urychluje. Tento výpočet je realizován interpolací vlastností a pozic mezi centroidy primitiv. Metoda tedy dochází k méně přesnému výsledku (obecně

přesnosti prvního až druhého řádu) rychleji pomocí hrubější aproximace [7]. Metoda navíc dovoluje pohodlné zachování požadovaných vlastností tělesa během a po deformaci právě díky distribuci výpočtu do okolí jednotlivých primitiv [10]. Pro přesnější aproximaci je možné použít interpolaci vyššího řádu, tj. začlenit do výpočtu např. i centroidy sousedů druhého řádu [7].

Jednou z hojně používaných metod je mass spring system (dále jen MSS), která používá pro reprezentaci vrcholů vlastní primitiva částic. Částicí se zde rozumí koule, jejíž střed jsou souřadnice příšlušného vrcholu. Částice obyčejně obsahuje informace o své hmotnosti, poloměru, pozici, vektoru rychlosti, nebo např. vektoru působící síly. Metoda pracuje s virtuálními pružinami o daných tuhostech mezi libovolnými dvojicemi bodů, které vytváří jakousi pružinovou síť, díky níž pohyb částic tělesa ovlivňují impulzivní síly ostatních částic. Chování MSS je ovšem závislé jak na prostorovém návrhu sítě, tak na tuhosti pružin [1]. Tato konfigurace sítě pružin může vyžadovat pro specifické účely obtížné ladění. Nové pozice jednotlivých bodů jsou nakonec vypočítávány diferenciální rovnicí druhého řádu vzhledem ke každé z přilehlých pružin, což ústí v soustavu diferenciálních rovnic pro každý bod [4] a řešení nabývá na výpočetní složitosti. Definitní charakteristikou MSS, kvůli které může být v kontextu této práce metoda považována za nevhodnou, je absence vlastnosti zachování objemu tělesa. Tuto vlastnost je sice možné k metodě dodat, avšak tím je již tak složitý systém opět ještě náročnější na implementaci a výpočet [1]. Navíc taktéž MSS nezahrnuje zachování tvaru tělesa [4].

Jiná metoda tzv. position based dynamics (PBD) je robustní, stabilní a rychlá metoda pro simulaci pohybu elastického tělesa. Oproti ostatním metodám k simulaci deformace tato metoda nebere v potaz aplikované externí síly v čase, nýbrž nahlíží na výpočet nových pozic bodů modelovaného tělesa v jednom kroku simulace jako na tzv. kvazi-statickou úlohu [1] (podrobněji vysvětleno v další kapitole). Při použití PBD je možné nahlížet na těleso pouze jako na množinu bodů a množinu příslušných kinematických omezení [1]. Fakt, že není pro simulaci potřebný výpočet např. potenciální energie s využitím aplikovaných sil, činí metodu obzvlášť vhodnou pro interaktivní simulace běžící v reálném čase. Navíc je výpočet značně zjednodušen, neboť není potřeba v simulovaném kroce integrovat přes všechny aplikované síly a řešení omezení je přímo součástí metody.

Díky slibným vlastnostem PBD byla metoda použita i v práci M. Červenky [4] pro simulaci pohybu a kolizí již zmíněných modelů svalů, na kterou tato práce navazuje. V následující kapitole bude proto metoda popsána detailně včetně příšlušných algoritmů, vlastností a matematického pozadí.

5 Metody pro detekci kolizí

Detekce kolize mezi kterýmikoliv ze zmíněných primitiv a dalším primitivem či částí primitiva (přímkou) ovšem není tak přímočará, a tak se nad objekty staví různé hierarchické datové struktury pro urychlení výpočtu. V rámci PBD je možné předpovídat pohyb částic, a tak docílit kontinuální detekce kolizí, ovšem je možné pro urychlení či přesnost detekce použít navíc i hierarchickou datovou strukturu, ve které je možné dohledat již vzniklou kolizi.

Struktury virtuálně dělí prostor objektu na podprostory, z nichž se u některých dá jednoznačně určit, že ke kolizi rozhodně nedochází a efektivně vyhledat kolidující podprostory [4]. Běžně jsou používány struktury jako binary space partitioning trees (dále jen BSP stromy), bounding volume hierarchy trees (dále jen BVH stromy) či octrees, kdy konstrukce a aktualizace struktury probíhá rekurzivním dělením prostoru dle daného kritéria či heuristiky. Zmíněná konstrukce a aktualizace těchto stromů ovšem může být značně časově náročná, a proto jsou pro simulace v reálném čase vhodné uniformní struktury jako je jednoduchý grid. Aktualizace uniformní struktury sice má potenciál probíhat v konstantním čase, nevyhýbá se pak ovšem vysokým paměťovým nárokům [15].

Urychlení pomocí hierarchických struktur spočívá ve snaze o co nejrychlejší lokalizaci potencionálních kolizí. Kolize obecně nemají tendenci nastávat v jeden časový úsek pro všechna primitiva tělesa, či dokonce pro všechna primitiva všech těles ve scéně. Proto je za pomoci hierarchických struktur vhodné co největší části scény či části tělesa v kontextu potencionální kolize odmítnout a řešit kolize do detailu pouze v daných podprostorech z kolizí podezřených. Pro hierarchické struktury vybudované nad celou scénou jsou kolize testovány mezi primitivy nacházejícími se přímo uvnitř podezřelé buňky struktury. V druhém případě pro hierarchické struktury reprezentující jednotlivé objekty jsou kolize testovány mezi podezřelými uzly zvolených struktur. Díky urychlení lokalizace kolize je možné provést konkrétní detekci přesněji a tím popřípadě dosáhnout jak apriorní detekce tak následně realistického řešení kolize.

V práci M. Červenky [4] jsou využity hierarchické struktury pouze pro reprezentaci jednotlivých tuhých těles tedy kostí a přesné testy kolizí na úrovni podprostorů s modely svalů jsou vypuštěny úplně. Elastická tělesa tedy modely svalů jsou pro detekci a řešení kolizí reprezentována primitivy pouze ve formě trojúhelníků nad vrcholy povrchových sítí těchto těles. Detailní popis detekce a řešení kolizí této práce je k dispozici v sekci 6.5.

Struktura může být konstruována pomocí rekurze. Rekurzivní dělení prostoru probíhá např. konstrukcí objektů axis-aligned bounding box (dále jen AABB kvádry) neboli ohraničujících kvádrů (jenž jsou v některých případech efektivní na aktualizaci [15]) ortogonálních na osy souřadného systému, dokud neobsahují kvádry např. dvojice primitiv. Výsledná struktura má pak podobu takovou, že je každému např. listu stromu přiřazena dvojice potencionálně kolidujících primitiv. Efektivita použití struktury je závislá nejen na směru prohledávání stromů, ale i na způsobu aktualizace struktury. Ze způsobu vytváření struktur plyne, že pokud naznačují kolizi 2 rodičovské AABB kvádry, je vyšší pravděpodobnost lokalizace kolize v některých z jejich dětí (podstromů)[15]. Pro rychlé prohledávání struktur je proto doporučován přístup takový, kde je horní polovina např. octree prohledávána zdola a až v případě potencionální kolize je daná větev stromu prohledána shora. Pokud jsou navíc rodičovské podprostory dosti vzdálené a zkoumaný AABB kvádr mezi nimi se neposunul o větší vzdálenost, není třeba testovat celou takovou větev [15]. Metody prostorového dělení mohou být rozděleny podle přístupu k detekci kolizí. Metody budou podle přístupů v následujících sekcích rozděleny a popsány.

5.1 Metody pro posteriorní detekci kolizí

Hierarchické struktury jsou v metodách pro posteriorní detekci kolizí využívané k rychlému dohledání již vzniklých kolizí. Urychlení často spočívá v co nejrychlejším vyřazení co největších podprostorů hierarchických struktur. Často je cenou za přesnost detekce kolize paměťová náročnost struktury nebo výpočetní náročnost algoritmu [15].

5.1.1 BHV stromy

Hierarchické struktury jsou budovány nad jednotlivými simulovanými tělesy. Pro každé simulované těleso je pak hierarchická struktura realizována např. pomocí BSP stromů, BVH stromů, octrees či jiných m-árních stromů. Jedná se tedy o složitější struktury, kdy listy stromů reprezentují specifickým primitivem (např. AABB kvádrem) daný podprostor [15].

5.1.2 Prostorové hashování

Hierarchická struktura může být také zavedena nad celou simulovanou scénou bez rozlišování jednotlivých simulavých těles. Mezi tyto struktury se dají řadit uniformí struktury či tzv. prostorové hashování. Pro většinu zmíněných

metod (FEM, FVM, MSS) je možné k detekci kolizí metodu prostorového hashování použít [15]. Prostorové hashování je nezávislé na počtu a typech simulovaných objektů.

Prvním krokem metody je vytvoření 3D gridu s pozicemi vrcholů těles a jejich objemovými primitivy. Primitiva jsou v gridu popsána AABB kvádry. Tento grid je následně namapován hashovací funkcí do 3D hash mapy se zaokrouhlením na celá čísla. V této hash mapě lze pak efektivně vyhledávat pomocí hashovací funkce potencionální kolize vrcholů vzhledem k primitivům. Pokud se nachází v hash mapě vrchol na stejném indexu jako AABB ohraničení primitiva a toto primitivum nenáleží danému vrcholu, je třeba dvojici testovat na kolizi. Test průniku probíhá porovnáním barycentrických souřadnic vrcholu a primitiva. Tento test navíc vrací hloubku průniku, kterou je možné využít pro řešení kolize. Výhodou tohoto přístupu je, že je díky nezávislosti na jednotlivých objektech zároveň s kolizemi řešené i sebeprotínání. To nastává právě tehdy, když kolidující vrchol a primitivum náleží stejnému objektu. Další výhodou je paměťová náročnost. Nad prostorem se totiž nevytváří žádná konkrétní datová struktura a hash mapa je pouze virtuální. K metodě zbývá dodat dostatečně vhodnou hashovací funkci tak, aby rovnoměrně generovala indexy do mapy a určit velikost hrany krychle 3D gridu. Nevýhodou ovšem zůstává častá nutnost aktualizace tohoto gridu [15].

5.1.3 Average-case přístup

Tento přístup spočívá ve vyhodnocování počtu krychlí gridu mezi protínajícími se podprostory (např. AABB kvádry). V případě vysokého počtu těchto krychlí zaznamenává metoda vysokou pravděpodobnost kolize, a pak je možné vyhodnocovat kořeny polygonů konkrétních podezřelých primitiv [15].

Nad simulovanými tělesy jsou stavěny BVH stromy. Jednotlivé uzly a listy stromů popisují množiny polygonů v oblasti daného AABB kvádru. Pokud jsou testovány 2 uzly na kolizi, je nejdříve oblast průniku jejich AABB kvádru rozdělena do jednoduchého gridu. Přes jednotlivé buňky tohoto gridu je zjištěn obsah plochy, přítomného **a)** polygonu z první množiny AABB kvádru či **b)** polygonu z množiny druhého AABB kvádru. Z plochy obsažení polygonů z jednoho zkoumaného AABB kvádru v intersektujícím AABB kvádru je vyjádřena relativní plocha vůči celkové ploše polygonu z celého zkoumaného AABB kvádru. Dále je zjištěna plocha pro nějvětší polygon, který může být kompletně obsažen v jedné buňce intersektujícího AABB kvádru. Pokud je tato plocha menší, než relativní relativní plocha vzhledem k jednomu zkoumanému AABB kvádru, je daná buňka označena za

pravděpodobně kolidující. Pokud je buňka označena za pravděpodobně kolidující vzhledem k oboum zkoumaným AABB kvádrům, je označena za **kolidující**. Pro kolidující buňku je zvolena spodní hranice pravděpodobnosti, že se jedná o opravdovou kolizi. Nakonec je zaznamenána pravděpodobnost toho, že se v daném podprostoru průniku objevuje alespoň x kolidujících buněk [8].

Kvůli složitosti výpočtu v průběhu simulace je proces zjednodušen na případ, kdy jsou jednotlivé AABB kvádry rozděleny do jednoduchých gridů již na začátku simulace. Pro jednotlivé buňky gridu je sečten počet pravděpodobně kolidujících buněk. Tento počet je uložen do každého uzlu BVH stromu pro dané těleso. Pro každou dvojici uzlů je pak na základě počtu pravděpodobně kolidujících buněk vypočtena celková pravděpodobnost kolize. Tyto hodnoty jsou uloženy do *look-up* tabulky, kterou není v průběhu simulace (pokud nejsou uvažovány změny topologie těles) aktualizovat. Celý proces přiřazování pravděpodobností kolizí tudíž může proběhnout pouze jednou v pre-processingové části simulace [8].

K vypočtení celkové pravděpodobnosti kolize dvou uzlů v časovém kroku simulace je nutné zjistit 3 informace: **a)** počet buněk podprostoru průniku uzlů s, **b)** počet pravděpodobně kolidujících buněk jednoho uzlu v podprostoru průniku uzlů s_A, **c)** počet pravděpodobně kolidujících buněk z druhého uzlu v podprostoru průniku uzlů s_B. Pokud jsou AABB kvádry uzlů různých rozměrů, je třeba najít jiné uzly v jednom z těles tak, aby byly rozměry alespoň podobné [8].

Pro dva zkoumané AABB kvádry A a B, kdy kvádr A je menších rozměrů, než kvádr B je možné vypočítat nutné 3 informace následovně [8]:

$$s = \frac{Vol(A \cap B)}{Vol(A)} \cdot c(A) \tag{5.1}$$

$$s_A = \frac{Vol(A \cap B)}{Vol(A)} \cdot c_p(A) \tag{5.2}$$

$$s_B = \sum_{i=1}^{n} s_{B_i} \tag{5.3}$$

$$s_{B_i} = \frac{Vol(B_i \cap A)}{Vol(B_i)} \cdot c_p(B_i) \tag{5.4}$$

Kde $c(A)$ značí počet buněk v kvádru A, $c_p(A)$ počet pravděpodobně kolidujících buněk v kvádru A, $c_p(B_i)$ počet pravděpodobně kolidujících buněk v i-tém dítěti kvádru B tělesa, jež tento kvádr obsahuje. Pro kvádr B je počet pravděpodobně kolidujících buněk obsažených v intersektujícím kvádru

sečítán přes uzly n dětí, dokud není velikost dětského uzlu menší, nebo rovna velikosti uzlu A kvůli urychlení celkového kolizního testu [8]. Pokud ovšem ani při dosažení listu tělesa, jež obsahuje kvádr B není objem i-tého kvádru menší, nebo rovno kvádru A, je nutné nahradit v rovnici (5.3) členy s_{B_i} menším inkrementem $s^-_{B_i}$ [8]:

$$s^-_{B_i} = s_{B_i} \cdot \tau_{AB_i} \cdot \frac{1}{\sqrt[3]{\tau_{AB_i}}} \tag{5.5}$$

Kde τ_{AB_i} značí poměr objemu i-tého dítěte kvádru B a objemu kvádru A.

Pro výpočet pravděpodobnosti $P_{collCell \geq x}$, že v průnikovém podprostoru $A \cap B$ existuje alespoň x praděpobodně kolidujících buněk, použijeme následující výpočet:

$$P_{collCell \geq x} = 1 - \sum_{t=0}^{x-1} \frac{\binom{s_B}{t} \cdot \binom{s-s_B}{s_A-t}}{\binom{s}{s_A}} \tag{5.6}$$

Kde t je počet sečtených pravděpodobně kolidujících buněk kvádru A v prostoru $A \cap B$.

Konečně pro pravděpodobnost, zdali mezi uzly s kvádry A a B dochází ke kolizi, použijeme nerovnici (5.7) [8]:

$$P_{AcollB} \geq \max_{x \leq min\{s_A, s_B\}} \{P_{collCell \geq x} \cdot (1 - (1 - LB(collCell))^x)\} \tag{5.7}$$

Kde člen $(1 - (1 - LB(collCell))^x$ značí spodní hranici pravděpodobnosti, že alespoň jedna z x kolizních buněk detekuje pravou kolizi. Hodnota $LB(collCell)$ je aproximována rovnicí (5.8) [8]:

$$LB(collCell) \approx \frac{d_A + d_B}{d_{max_A} + d_{max_B}} \tag{5.8}$$

Kde d_A značí hloubku uzlu s kvádrem A, d_B v prvním tělese, značí hloubku uzlu s kvádrem B v druhém tělese a hodnoty d_{max_A} a d_{max_B} značí maximální hloubky těles s kvádry A a B [8].

Pro výpočet rovnic (5.1) - (5.4) je nutný výpočet objemů intersektujících kvádrů. Pokud jsou středy kvádrů (z nichž jeden má velikosti hran a, b, c) vzdálené o vzdálenost d, lze aproximovat objem jejich průniku nalezením maxima $max\{V_1, V_2, V_3\}$, [8] kde

$$V_1 = (a - d) \cdot b \cdot c$$
$$V_2 = a \cdot b \cdot c \cdot (1 - \frac{d}{\sqrt{a^2 + b^2}})^2$$
$$V_2 = a \cdot b \cdot c \cdot (1 - \frac{d}{\sqrt{a^2 + b^2 + c^2}})^3$$

Vypočtené pravděpodobnosti kolizí nad AABB kvádry jsou v algoritmu používány v kombinaci s abstraktní datovou strukturou prioritní frontou. Jelikož algoritmus pracuje na základě pravděpodobností, poskytuje rychlé výsledky za cenu přesnosti řešení. Tato přesnost se dá modifikovat parametrem, který určuje, od jaké hodnoty vykazuje pravděpodobnost kolize opravdovou kolizi. Metoda může být aplikována na jakékoliv BHV stuktury, ovšem BHV stromy s AABB kvádry se zdají jako nejlepší volba pro účely této práce z hlediska rychlosti [8].

5.1.4 Distance fields

Metoda distance fields neboli pole vzdáleností uchovává pro vrcholy nejmenší vzdálenosti k povrchu tělesa. Součástí těchto vzdáleností může být i informace o tom, zdali je daný vrchol uvnitř, či vně tělesa. Taková metoda se nazývá signed distance fields (dále jen SDF) neboli pole vzdáleností se znaménky. Výhoda tohoto přístupu tkví v nezávislosti na tvaru modelovaného tělesa. Protože je pro každý bod jasně definovaný vztah k povrchu tělesa, stačí na otestování kolize porovnat body na povrchu jednoho tělesa s polem vzdáleností jiného tělesa a naopak. K zamezení zbytečnému protínání je navíc k povrchu tělesa přidána virtuální vrstva o volitelné hloubce ε. Metoda distance fields navíc při kolizi zaznamenává hloubku proniknutí objektů, a tím je ulehčeno řešení nastané kolize. Pro zlepšení přesnosti detekce je možné přidat virtuální vrcholy např. do středů všech hran těles a počítat vzdálenostní pole i pro ně. Aktualizace tohoto pole je opět úskalím, k jehož řešení je možné použít například image-space přístup, kdy jsou vzdálenosti aproximovány pomocí 2D projekcí objektů [15].

V kontextu modelování kostí tedy tuhých těles, lze ovšem metodu SDF použít bez nutnosti pole aktualizovat. Díky tomu, že jsou na kosti aplikované pouze afinní transformace (posun, rotace), lze tyto samé transformace aplikovat na hodnoty, pro které je hodnota SDF zjišťována tedy pozice částic modelů svalů, a tím dostat validní informaci o tom, zdali se částice modelu

svalu nachází uvnitř, či vně kosti. K metodě je možné navíc přidat rozšíření, které má potenciál předcházet většímu počtu kolizí, a tím je již zmíněné přidání virtuální vrstvy.

5.1.5 Image-space techniky

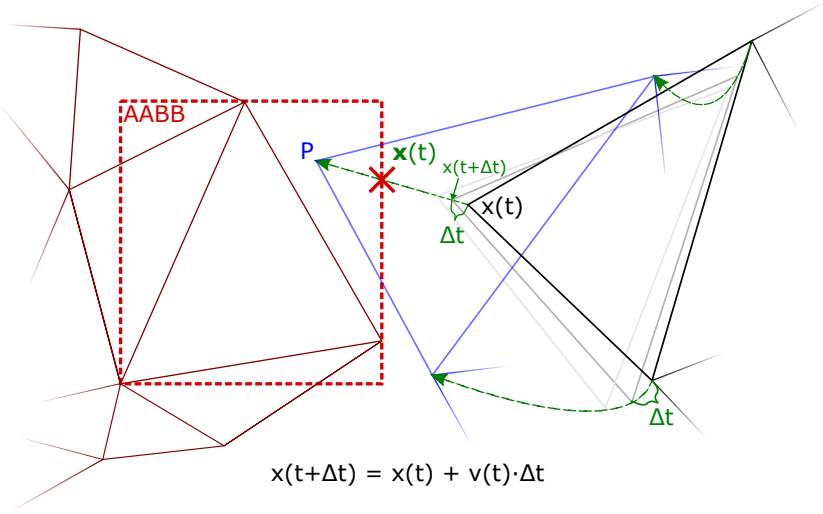
Tyto techniky využívají 2D projekci tělesa pro výpočet hodnoty layered depth images (dále jen LDI) neboli obrazy hloubky ve vrstvách. Pomocí LDI je pak 3D průnik těles po vrstvách aproximován např. trojúhelníkovou sítí. Pro detekci kolize je nejprve vypočítána oblast průniku v podobě ohraničujícího kvádru. Pokud je tato oblast neprázdná, je oblast průniku aproximována pomocí LDI. Následovně jsou vyhledány kolidující části objemů těles a kolidující vrcholy či jiná kolidující primitiva těchto těles. Řešení sebeprotínání je možné s dodatečnou informací o orientaci povrchů kolidujících těles. Přesnost metody závisí na detailu rozlišení používaných 2D projekcí [15].

5.2 Metody pro apriorní detekci kolizí

V kombinaci s PBD může být vhodné držet dvě hierarchické struktury najednou, ve dvou po sobě jdoucích časových úsecích (t, $t + \Delta t$, kde Δt je zvolený konstantní časový úsek mezi kroky simulace) a tím dosáhnout apriorní detekce kolizí. V závislosti na velikosti časového úseku Δt může ovšem docházet k tzv. tunelovému efektu, kdy primitivum projde jiným, neboť má natolik vysokou rychlost, že mezi začátkem a koncem časového úseku vůbec ke kolizi nedojde. K řešení tohoto problému je popisována dráha primitiva v tomto časovém úseku funkcí 3.1. Tato funkce popisuje pohyb částice s aktuální rychlostí $v(t)$ po úsečce mezi časy t a $(t + \Delta t)$:

$$\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + v(t) \cdot \Delta t \tag{5.9}$$

Situace detekce kolize apriorní metodou může potom vypadat následovně (obrázek 5.1):



Obrázek 5.1: Trojúhelník pohybujícího se tělesa (vpravo) má pro každou částici (zde vrcholy) známou funkci $\mathbf{x}(t)$, která aproximuje v daném časovém kroku dráhu bodu úsečkou. Intersekce této úsečky s daným primitivem může být realizována např. Eulerovo metodou je vždy proveden test na průnik úsečky s primitivem (zde AABB obdélník) podezřelého tělesa (vlevo). Pokud částice, jejíž dráhu úsečka popisuje, náleží podezřelému tělesu, je nalezením průniku úsečky a primitiva řešené i sebeprotínání. Pokud není kořen soustavy nalezen, tak kolize nenastává a částice X je transformován na novou pozici P. V opačném případě je kolize detekována a je možnost např. omezit další pohyb částice, vrátit na původní pozici x, nebo řešit složitější odezvu s využitím nalezeného předpovídaného času kolize na dané dráze pohybu.

Konkrétním problémem je pak hledání průniku části polynomu této dráhy (aproximované úsečkou) s jiným primitivem, čili hledání kořenů soustavy jejich rovnic. To ale může být značně výpočetně náročné, a tak je navíc testování detekcí převedeno na jednodušší případ, kdy jsou podle různých metod přiřazovány např. pravděpodobnosti průniků zkoumaných podprostorů. Tyto metody často za cenu ztráty přesnosti provádí detekci kolizí rychleji, avšak je pak nutné implementovat řešení kolizí a sebeprotínání [15].

6 Dostupné implementace detekce kolizí

V této sekci budou popsány metody pro detekci kolizí, pro které existuje veřejně dostupná implementace. Zásadní rozdíly mezi těmito metodami budou v použitých hierarchických strukturách, kterými metody detekci kolizí urychlují, či způsobem reprezentace těles. Neboť je stávající řešení M. Červenky [4] založeno na metodě PBD a projekt je implementován programovacím jazykem C++, jsou upřednostňovány implementace právě v programovacím jazyce C++, nejlépe rovnou v kontextu metody PBD.

Navíc protože je stávající řešení v práci M. Červenky [4] implementováno v rámci projektu, který velkou mírou využívá open-source vizualizační software zvaný The Visualization Toolkit (dále jen VTK), je vhodné zmínit implementaci detekce kolizí tímto nástrojem.

6.1 VTK

Součástí VTK je třída `vtkCollisionDetectionFilter` napsaná v programovacím jazyce C++. Třída využívá k detekci kolizí hierarchickou strukturu oriented bounding box tree strom (dále jen OBB strom). OBB stromy využívají pro rozdělení tělesa datovou strukturu oriented bounding box, jež je podobna AABB kvádrům s rozdílem, že tyto kvádry nemusí být v rovnoběžnosti s osami souřadného systému. Třída poskytuje detekci kolizí pouze pro dvě tělesa reprezentovaná polygonem. Test kolize je proveden na každou hranu jednoho polygonu vzhledem k druhému. Následně je test proveden i pro každou hranu druhého polynomu vzhledem k prvnímu [14]. Jedná se tedy o algoritmus hrubou silou, který je pro simulaci elastických těles v reálném čase nevhodný především pro vysokou časovou náročnost, jak je popsáno v sekci 3.3. Třída je součástí vizualizačního balíku VTK, jenž je open-source software pod licencí **BSD License** (Bližší informace jsou k dispozici na oficiálních stránkách VTK `https://vtk.org/about`.)

6.2 Discregrid

D. Koschier et al. nabízí statickou knihovnu zvanou Discregrid, implementovanou v programovacím jazyce C++. **Použití** i **implementace** celé knihovny

jsou dostupné na stránkách GitHub repozitáře projektu. Projekt, ve kterém je knihovna použita, je jinou knihovnou, poskytující implementaci simulace látek, tuhých i elastických těles metodou PBD. To činí implementaci Discregrid obvzlášť vhodným kandidátem pro integraci do stávajícího řešení.

Knihovna umožňuje generaci signed distance function neboli funkce oznaménkovaných vzdáleností, která pro zadaný bod s pozicí $\{x, y, z\}$ vrací skalární hodnotu, reprezentující nejkratší vzdálenost k povrchu tělesa. Pokud má nejkratší vzdálenost k povrchu pro daný bod kladné znaménko, znamená to, že se bod nachází vně tělesa a přibližuje se k jeho povrchu povrchu zevně. Naopak v případě záporné návratové hodnoty funkce se zkoumaný bod vzhledem k tělesu nachází uvnitř, a tedy se k povrchu blíží opačným směrem. Nalezení analytického předpisu této funkce je ovšem pro složitější útvary (zde povrchové sítě těles) velmi výpočetně náročné. Prostor tělesa je proto diskretizován pomocí voxelizace a reprezentován pomocí bounding sphere hierarchy (dále jen BSH), tedy hierarchické struktury BSH stromu, kdy ohraničujícím primitivem pro středy voxelů je koule. Pro středy jednotlivých koulí, tedy středy voxelů, je pak nalezen trojúhelník na povrchu tělesa, který je ke středu nejblíže. Hodnota signed distance funkce je pak vzdálenost tohoto trojúhelníku od středu koule. Tím je vytvořena hierarchická struktura SDF.

Pro jakoukoliv 3D souřadnici v prostoru kvádru, kde je voxelizace provedena, pak SDF vrací hodnotu signed distance funkce pro střed voxelu, ve kterém se souřadnice nachází či je provedena interpolace signed distance function hodnot ve vrcholech voxelů v určitém okolí zkoumané souřadnice. Konkrétně Discregrid provádí interpolaci Lagrangeovými polynomy. Těmito polynomy je inkrementálně určován i gradient v libovolném vrcholu voxelizovaného prostoru. Gradient poskytuje informace o hloubce a směru průniku. Souřadnice může být pozicí bodu, který je vrcholem jiného tělesa či pozicí nějakého primitiva tohoto tělesa. Hodnota SDF nakonec poskytuje úplnou informaci existence kolize, navíc obsahuje užitečné informace pro její řešení.

Knihovnu je možné použít pod licencí **Massachusetts Institute of Technology License** (Bližší informace jsou k dispozici na oficiálních stránkách repozitáře knihovny `https://github.com/InteractiveComputerGraphics/Discregrid`.)

6.3 NVIDIA FleX

Společnost NVIDIA poskytuje proprietární knihovnu pro simulaci plynů, tekutin, elastických a tuhých těles a látek zvanou NVIDIA FleX. Knihovna je

implementována v programovacím jazyce C++. Simulace interakcí těles jsou implementovány metodou PBD, což z knihovny činí stejně jako knihovnu Discregrid, vhodného kandidáta pro integraci do stávajícího řešení. Knihovna za použití PBD přistupuje k simulaci všech druhů těles (plyny, tekutiny, elastická a tuhá tělesa, ...) jednotným způsobem [9], kdy je každé těleso voxelizováno a v celém objemu tělesa jsou generovány částice o stejném rozměru. Pro každý druh tělesa knihovna definuje v rámci PBD různá omezení (viz sekce 3.2.1).

Knihovna používá pro reprezentaci většiny simulovaných objektů částice. Částice je zde definovaná aktuální pozicí a rychlostí vrcholu, kterému náleží, objemem a nakonec indexem udávajícím, v jaké fázi simulace se částice nachází [9]. Modifikací indexu fáze lze např. určitou částici vynechat z procesu detekce kolizí. Pro všechny částice je pak volitelný tzv. interaction radius neboli poloměr interakce, který pro elastická tělesa udává poloměr koule kolem částice, ve které se nachází ostatní částice, které mají na ni vliv. Pro tuhá tělesa je to délka hrany jednoho voxelu gridu. Nad částicemi je konstruována hierarchická struktura uniformního prostorového hash gridu pomocí voxelizace. V této struktuře jsou nejprve pro každou částici nalezeny všechny sousední částice. S každým sousedem je pak proveden diskrétní test, zdali se částice překrývají. K předcházení tunelovému efektu jsou v této fázi algoritmu poloměry částic zvětšeny o volitelné procento. Toto procento ovšem musí být dosti malé, aby nedocházelo k detekci neexistující kolize [9]. Temporální zvětšení poloměrů částic je pouze virtuální pro konkrétní test kolize a poloměry všech částic simulované scény jsou konstantní a stejné pro všechny částice v každém časovém kroku simulace. Díky tomu je pak detekce kolizí, založená na uniformních gridech, obzlášť efektivní na výpočet [9].

Rozdíl oproti reprezentaci těles pomocí částic v práci M. Červenky [4] spočívá v tom, že knihovna NVIDIA FleX provádí uniformní voxelizaci nad všemi tělesy v rozsahu celého objemu tělesa do částic, kdežto v implementaci, uvedené v sekci 6.5, jsou částicemi reprezentované pouze vrcholy povrchových sítí simulovaných modelů svalů. Knihovna navíc simulaci všech částic řeší naráz. K tomu používá formát tzv. structure of arrays neboli struktura polí, ve kterém očekává všechna modelovaná tělesa na vstupu. Z toho plynou pro knihovnu např. nevýhody vyšších paměťových nároků pro samotnou reprezentaci scény nebo vyšší výpočetní náročnost při detekci a řešení kolizí. Výhodou je naopak např. existence detekce a řešení kolizí sebeprotínání.

Pro reprezentaci v nějakém smyslu velkých a jednoduchých těles, jako jsou simulované zdi či podlahy, knihovna nepoužívá částice z důvodu nízké časové efektivity detekce kolizí těchto těles. K jejich reprezentaci jsou místo částic používána primitiva, jako jsou konvexní obálky, SDF či trojúhelníkové

sítě [9].

Detekce a řešení kolizí v celé scéně jsou prováděny ve čtyřech vrstvách podle priority:

1. Trojúhelníkové sítě
2. Statické SDF
3. Tuhé konvexní meshe
4. Elastické konvexní meshe

Priorita je zavedena pro předcházení tunelovému efektu a zabránění tomu, aby částice elastického tělesa neprocházela tuhými tělesy.

Speciálně pro tuhá tělesa je použita reprezentace částicemi v kombinaci s hierarchickou strukturou SDF. V SDF polích je uchovávána pro každou částici informace o vzdálenosti částice od povrchu tělesa, neboli velikosti gradientu z pozice částice, a směru gradientu z pozice částice. Reprezentace částicemi ovšem může být příliš hrubá, a tak knihovna dovoluje míru překrývání částic, zatímco je těleso nehybné. Velikost i směr gradientu jednotlivých částic jsou v případě kolize (částice jiného tělesa je uvnitř SDF) využity pro realistické řešení kolize [9].

Částice elastických těles jsou navíc shlukovány do shluků, které definují pro tuto skupinu částic omezující PBD funkci zachování tvaru. Shluky je vhodné pro realistickou deformaci tělesa překrývat. Pokud dochází v elastickém tělese navíc k nepřirozenému ohybu a kroucení, je možné spojit jednotlivé částice virtuální vazbou, která přidává mezi částice další omezení, např. lepší zachování tvaru [9].

Knihovna je proprietární (vyjímaje rozšíření pro snažší integraci) pod licencí **Nvidia Source Code License** (Bližší informace jsou k dispozici na oficiálních stránkách GitHub repozitáře knihovny `https://github.com/NVIDIAGameWorks/FleX/blob/master/LICENSE.txt`.)

Knihovna funguje vzhledem k jednotnému přístupu k tělesům nejlépe tak, že je jí implementována celá simulace, a tak by její integrace do stávajícího řešení mohla znamenat náročné *osekávání* nepotřebných částí, konvertování vstupních i výstupních dat (ve formátu structure of arrays) a celkově degradaci efektivity knihovny i obtížnou implementaci. Další nevýhodou je nutnost zakoupení knihovny.

6.4 Flexible Collision Library

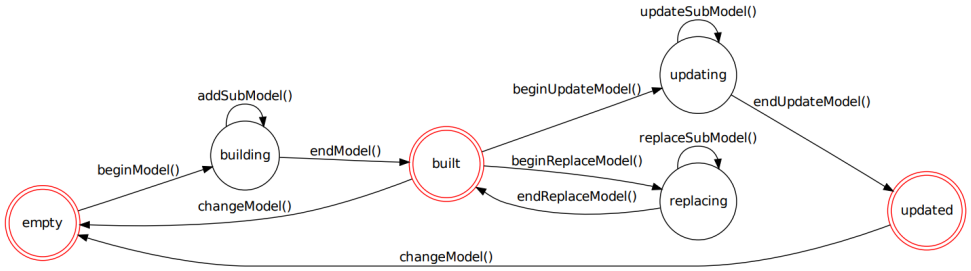
Flexible Collision Library (dále jen FCL) [11] je knihovna implementována v jazyce C++ poskytující mnoho možností detekce kolizí nad tuhými i deformovatelnými tělesy, danými trojúhelníkovou sítí. Možnostmi realizace detekce jsou jak různé hierarchické struktury, tak i způsoby, kterými je možné struktury prohledávat a různé druhy primitiv, kterými lze podprostory tělesa reprezentovat. Urychlení nad obecnou hierarchickou strukturou je provedeno metodou Sweep and Prune (dále jen SoP) [11] neboli prořezávání stromů, která řadí jednotlivé podprostory struktury podle pozic na třech hlavních osách souřadného systému (x, y, z), což značně urychluje proces detekce kolize na úrovni celých těles i podprostorů. Detailnější detekce kolize je pak řešena pro specifickou reprezentaci podprostoru primitivem a specifický typ hierarchické struktury. Z toho vyplývá, že není nutné používat pro všechna simulovaná tělesa jednotné struktury ani primitiva.

FCL dokáže pracovat s osmi druhy tvarů těles. Těmi jsou obecné trojúhelníkové sítě, koule, AABB kvádry, kužely, válce, kapsle, konvexní meshe a elipsoidy. Tato tělesa jsou označena jako *kolizní objekty* [11]. Detekce kolizí je možná mezi jakoukoliv dvojicí kolizních objektů. V rámci knihovny je k dispozici i implementace sebeprotínání.

Nad kolizními objekty mohou být konstruovány BHV stromy. Knihovna podporuje čtyři reprezentace podprostorů daného BHV stromu. Těmi jsou AABB kvádry, OBB kvádry, RSS primitiva (rectangle swept sphere neboli kvádry s kulatými hranami) vhodné pro výpočet vzdáleností dvou těles a discrete oriented polytope primitiva (dále jen kDOP) neboli diskrétní orientované polytopy, kdy je těleso rozděleno určitým počtem rovin na intervaly. Primitiva OBB a RSS jsou autory doporučována především pro apriorní detekci kolizí tuhých těles, zatímco AABB kvádry a kDOP primitiva jsou doporučena pro apriorní detekci pro elastická tělesa [11]. Primitivum je vždy přiřazeno právě jednomu uzlu stromu daného BHV. Uzly drží detaily o BVH, ve které se nacházejí (kompletní informace k procházení BHV z daného uzlu, jakého typu je BHV, jakým primitivem jsou podprostory reprezentovány, ...), kde je v případě apriorní detekce taktéž uchována informace o stavu tělesa v předchozím časovém úseku. Přímo v BHV strukturách jsou navíc vždy přítomny informace o všech vrcholech a trojúhelnících daného tělesa [11].

V knihovně je zaveden pro konstrukci, aktualizaci a výměnu stromu jiným typem BHV stavový automat vyobrazen na obrázku 3.3. Snahou tohoto automatu je předcházet např. nevhodné volbě geometrické reprezentace tělesa či druhu BHV. Automat taktéž popisuje postup, kterým jsou tělesa

aktualizována v případě apriorní detekce. Konstrukce BHV stromu a jeho aktualizace je v rámci FCL prováděna rekurzivním dělením primitiv na dvě části, z nichž každá je ohraničena AABB kvádrem a značí podprostor hlubší úrovně [11].



Obrázek 6.1: Stavový automat konstrukce, aktualizace a výměna BHV stromu (převzato z [11]). Konstrukce BHV stromu: `empty` → `building` → `built`, výměna BHV stromu: `built` → `replacing` → `built`, aktualizace BHV stromu: `built` → `updating` → `updated`.

Pro řešení apriorní detekce kolizí je rovněž v knihovně k dispozici implementován algoritmus tzv. conservative advancement (CA), který inkrementálně postupuje s tělesy v malých časových intervalech, zatímco předchází akutní kolizi [11].

Kolize dvou těles, reprezentovaných BHV je prováděna průchodem tzv. bounding volume test tree neboli strom testů ohraničujících objemů, což je BHV strom vygenerovaný z dvou testovaných BHV. Kromě SoP metody knihovna používá pro urychlení hledání kolidujících podprostorů v tomto stromu testů tzv. front list. V seznamu je uchovávána informace o předchozím dotazu na detekci. Využití seznamu je obzvlášť efektivní, nastává-li kolize v nějakém blízkém uzlu předchozího uzlu, kde kolize probíhala [11].

Knihovna poskytuje rozhraní pro kolize prostřednictvým dotazů. Dotaz může být podán pouze na existenci kolize či na přesné informace o průniku a směru této kolize. Jsou taktéž k dispozici dotazy na apriorní detekci kolizí, a to jak na existenci předvídané kolize v daném časovém intervalu, tak i na čas průniku [11]. To, jakým způsobem je dotaz vykonán, závisí na zvolené strategii průchodu hierarchií, jejím typu i druhu primitiva. Mimo poskytovanou strategii průchodu hierarchií je v knihovně totiž možné implementovat strategii vlastní.

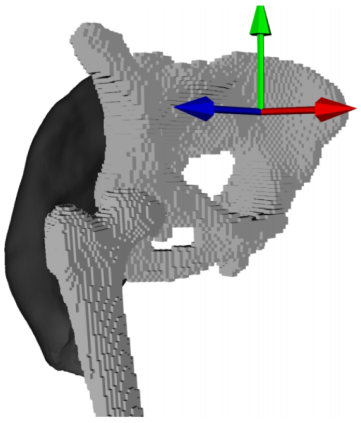
Knihovna FCL může být pro integraci vhodná nejen díky implementaci a rozhraní pro programovací jazyk C++, ve kterém je implementováno i stávající řešení M. Červenky [4], ale i pro širokou nabídku možností, jak detekci kolizí provádět. Knihovnu je možné použít pod licencí **Software**

License Agreement (BSD License) (Bližší informace jsou k dispozici na oficiálních stránkách GitHub repozitáře knihovny `https://github.com/flexible-collision-library/fcl/blob/master/LICENSE`.)

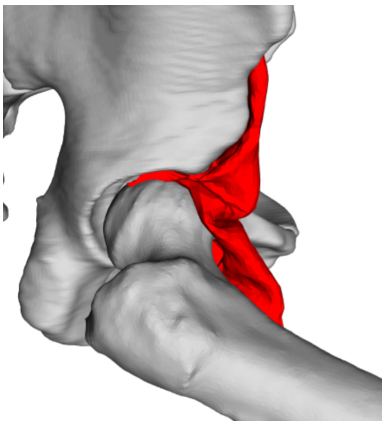
6.5 Voxelizace

V práci M. Červenky z roku 2019 [4] je detekce kolizí zjednodušena z důvodu urychlení výpočtu [4] za použití metody voxelizace. Voxelizace je metoda dělení prostoru (rozdělení prostoru do mřížky), kdy je v každém voxelu (buňce mřížky) zaznamenána informace o tom, jestli voxelem nějaká částice prochází, či nikoliv. Voxelizace byla provedena pouze na objekty kostí. Částice je v této práci primitivem koule o daném poloměru, hmotnosti, pozici a rychlosti. Těmito koulemi jsou reprezentovány vrcholy povrchových trojúhelníkových sítí simulovaných těles. Pokud pak zkoumaná úsečka dráhy částice prochází voxelem, kterým prochází jiný objekt, dochází ke kolizi.

Voxelizace je provedena uniformě metodou *flood-fill* [4] zvlášť nad každou kostí, kdy je počet voxelů odvozen od velikosti ohraničujícího kvádru. Výsledný voxelizovaný model kosti je z pohledu detekce a řešení kolizí ovšem stále dosti hrubý, a tak dochází např. k nerealistickým uvíznutím částic mezi kostmi. Hrubost voxelizace ovšem dává vzhledem ke kubické paměťové náročnosti reprezentace smysl [4]. Příklad voxelizace je uveden na následujícím obrázku 6.2 a jeden z nedostatků pak na obrázku 6.3.

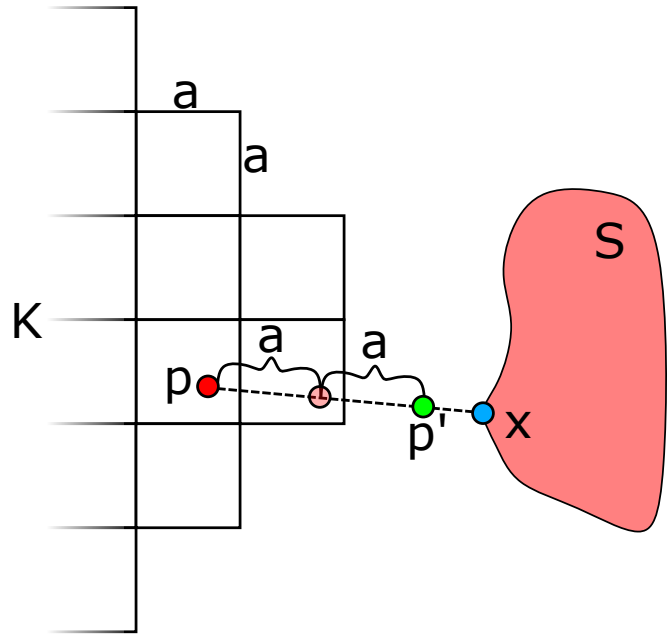


Obrázek 6.2: Ukázka voxelizace kosti pánve (*pelvis*) a kosti stehenní (*femur*), převzato z práce Kohout, J. a Červenka, M. 2020 [5].



Obrázek 6.3: Ukázka nerealistické simulace chybou voxelizace, převzato z práce Červenka, M. 2019 [4].

Pokud je pomocí PBD odhadu částice svalu **p** zaznamenána uvnitř voxelu nějaké kosti, začíná proces řešení kolize. Z pozice **p** je částice přesouvána krokem 1 voxelu zpět, směrem k původní pozici **x**, dokud se nenachází mimo voxely kostí. Pokud ovšem pozice částice dojde zpět k původní pozici **x** a stále se nachází uvnitř kosti, znamená to, že částici svírají 2 tělesa kostí a řešení selhává, či v lepším případě je kvůli hrubé voxelizaci částice svalu mimo vizuální reprezentaci kosti, a tak je situace v pořádku. Proces detekce a řešení kolize s využitím voxelizace je znázorněn na obrázku 6.4.



Obrázek 6.4: Detekce a řešení kolize pomocí voxelizace. Bod **x** svalu **S** je v aktuálním simlačním kroce vytlačován silami a omezeními na pozici **p**. Ta se ovšem nachází uvnitř voxelové struktury kosti **K** s hranou voxelu o velikosti **a**. Je provedeno řešení kolize iterativním posouváním z pozice **p** zpět k bodu **x** s velikostí posunu **a**, dokuď není nalezena nekolidující pozice **p'**.

V další kapitole bude popsána metoda PBD, jakožto metoda použitá v práci M. Červenky [4] a tudíž metoda, v jejímž kontextu bude tato práce prováděna.

7 Position Based Dynamics

Pozice částic jsou při použití PBD určovány v každém kroku simluace vyřešením tzv. kvazi-statické úlohy. Kvazi-statická úloha lze v daný moment považovat za statickou. Statická úloha dovoluje zanedbat vektory rychlostí jednotlivých částic a pozice bodů jsou zjišťovány přímo za běhu simulace bez působení impulzních sil. Díky tomuto přístupu lze během kroku simulace vypočítat numerickou aproximaci časového integrálu předpovídajícího následující pozici částice a tím dosáhnout apriorní aplikace požadovaných omezení [1], jimiž jsou zachování tvaru, objemu, vzdáleností, realizace anizotropie, navíc je v kontextu PBD za omezení považována i detekce kolizí. To je realizováno opravou předpokládaných pozic vzhledem k daným omezením [1].

Díky přímočarosti aplikace restriktivních omezení je možné metodou PBD simulovat materiály různých charakterů najednou. K tomu je nutné nějakým způsobem (hierarchické rozdělení či obarvení částic) rozdělit např. síť tělesa na požadované oblasti a těm je pak možné definovat rozdílná omezení [2].

7.1 Algoritmus PBD

Metoda pracuje s množinou částic (např. vrcholy trojúhelníkové sítě) N a množinou omezení M. Každá částice $i \in [1,\ldots,N]$ nese informaci o své hmotnosti m_i, pozici $\mathbf{x}_i$ a rychlosti $\mathbf{v}_i$ [12]. Vliv jednotlivých omezení je popsán parametrem tuhosti $k \in [0,1]$.

Jelikož metoda pracuje iterativně, je nutné na řádcích (1)-(3) nejprve inicializovat hodnoty částic. Pro hmotnost je volena inverzní hodnota w_i z důvodu efektivity výpočtu [4]. Na řádce (5) jsou aktualizovány rychlosti částic vzhledem k vlivu externích sil $\mathbf{f}_{ext}(\mathbf{x})$ za daný časový úsek Δt a na řádku (6) jsou na základě těchto rychlostí předpovězeny nové pozice částic $\mathbf{p}_i$ za časový úsek Δt. Dohromady řádky (5)-(6) provádí jednoduchou implicitní Eulerovu metodu vycházející z Newtonova druhého pohybového zákona. Kód na řádce (7) testuje, zdali pohyb z aktuální pozice $\mathbf{x}_i$ na předpovídanou pozici $\mathbf{p}_i$ částice i způsobí kolizi, jde tedy o apriorní detekci kolize. Pokud kolizi způsobí, je předpovídaná pozice označena kolizním omezením [4]. Dále na řádcích (8) až (10) jsou předpovídané pozice $\mathbf{p}_1, \ldots, \mathbf{p}_N$ opraveny vzhledem k omezením $C_1, \ldots, C_{M+M_{Coll}}$, kde M značí obecné vnitřní omezení vygenerované při spuštění simulace [1] a M_{Coll} značí vnější omezení, tedy kolizní. Na řádcích (11)-(14) jsou vypočteny nové rychlosti částic (12) a jejich pozice (13). Na konec na řádku (15) je na rychlosti částic aplikované tření [4] [1].

Algoritmus 1 PBD [1]

1: **for all** vertices i **do**
2: initialize $\mathbf{x}_i = \mathbf{x}_i^0, \mathbf{v}_i = \mathbf{v}_i^0, w_i = 1/m_i$
3: **end for**
4: **loop**
5: **for all** vertices i **do** $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t w_i \mathbf{f}_{ext}(\mathbf{x}_i)$
6: **for all** vertices i **do** $\mathbf{p}_i \leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
7: **for all** vertices i **do** generateCollisionConstraints($\mathbf{x}_i \rightarrow \mathbf{p}_i$)
8: **loop** solverIterations **times**
9: projectConstraints($C_1, \ldots, C_{M+M_{Coll}}, \mathbf{p}_1, \ldots, \mathbf{p}_N$)
10: **endloop**
11: **for all** vertices i **do**
12: $\mathbf{v}_i \leftarrow (\mathbf{p}_i - \mathbf{x}_i)/\Delta t$
13: $\mathbf{x}_i \leftarrow \mathbf{p}_i$
14: **end for**
15: velocityUpdate($\mathbf{v}_i, \ldots, \mathbf{v}_N$)
16: **end loop**

7.1.1 Aplikace obecných omezení

Mezi řádky (8)-(10) v algoritmu 1 dochází k aplikaci všech omezení. Pro každé omezení je definována tzv. penalizační funkce $C(\mathbf{p}_1, \ldots, \mathbf{p}_N)$, kterou je snaha minimalizovat [4]. Pro nalezení lokálního minima penalizačních funkcí PBD používá iterativní optimalizační algoritmus, tzv. gradientní sestup. K tomu je potřeba, aby penalizační funkce byla v okolí opravovaného bodu diferencovatelná. Výstupem gradientního sestupu dané penalizační funkce je nakonec gradient, který udává směr, kterým se penalizace blíží k nule [4]. Směr korekce vede proti směru tohoto získaného gradientu. Po normalizaci tohoto protijdoucího gradientu je pak o hodnotu penalizační funkce předpovídaný bod $\mathbf{p}_i$ posunutý. Tím je oprava provedena. Posun předpovídaného bodu $\Delta\mathbf{p}_i$ lze popsat následujícími rovnicemi [1]:

$$\Delta\mathbf{p}_i = \frac{-w_i \nabla_{p_i} C(\mathbf{p}_1, \ldots, \mathbf{p}_N)}{w_j \sum_j \mid \nabla_{\mathbf{p}_j} C(\mathbf{p}_1, \ldots, \mathbf{p}_N) \mid^2} \cdot C(\mathbf{p}_1, \ldots, \mathbf{p}_N) \tag{7.1}$$

Kde je v čitateli vypočítán záporný směr gradientu působení omezení, ve jmenovateli je tento směr znormován na hodnotu z intervalu [0,1] a za zlomkem je znormovaný směr znásoben velikostí omezení [1]. Hmotnosti w_i a w_j reprezentují vážený průměr a v případě neuvážení hmotností bodů lze za tyto proměnné dosadit hodnotu 1 [4].

Pro jednotlivá omezení je nutné definovat penalizační funkci $C(\mathbf{p}_1, \ldots, \mathbf{p}_N)$, která by měla jako návratovou hodnotu vracet skalár symbolizující jakousi sílu omezení. Obecně je možné za penalizační funkci považovat funkci pracující s pozicemi, orientacemi, lineární a úhlovou rychlostí či jejich derivacemi [1]. V rámci PBD ovšem tyto funkce závisí převážně na pozicích a orientacích bodů a okolních primitiv v čase [1].

Penalizační funkce nejsou obecně lineární [4]. Počet penalizačních funkcí je identický s počtem požadovaných omezení, a k uspokojení všech omezení najednou je tudíž třeba vyřešit soustavu nelineárních rovnic. K tomu je v PBD metodách používána Gauss-Seidlova iterativní metoda pro řešení soustav nelineárních rovnic [1].

Vliv penalizačních funkcí je popsán parametrem tuhosti $k \, \epsilon \, [\,0{,}1\,]$. Ten je aplikován na příslušnou funkci iterativně, takže má tuhost na konečnou pozici nelineární vliv. K vyjádření vzniklého vztahu tuhosti a nové pozice je možné použít následující rovnici, která vztah linearizuje vzhledem k parametru k:

$$\mathbf{p}_i = \mathbf{p}_i \cdot (1 - (1 - k)^{\frac{1}{n_s}}) \tag{7.2}$$

Kde proměnná n_s značí aktuální počet iterací. Chyba této funkce je nezávislá na počtu iterací n_s, ovšem výsledná tuhost objektu stále závisí na času simulace. Tato závislost zapříčiňuje, že se limitně tuhost k blíží k hodnotě 1 (těleso se limitně stává dokonale tuhým) s nekonečně narůstajícím počtem iterací. Na tento problém reaguje rožšíření základní PBD metody, tzv. extended PBD.

7.1.2 Extended PBD

Alternativní formou rovnice je tvar (7.3) s použitím Lagrangeových multiplikátorů λ (7.4):

$$\Delta\mathbf{p}_i = -\lambda w_i \nabla_{\mathbf{p}_i} C(\mathbf{p}_1, \ldots, \mathbf{p}_N) \tag{7.3}$$

$$\lambda = \frac{C(\mathbf{p}_1, \ldots, \mathbf{p}_N)}{w_j \sum_j \mid \nabla_{\mathbf{p_j}} C(\mathbf{p}_1, \ldots, \mathbf{p}_N) \mid^2} \tag{7.4}$$

V metodě extended PBD neboli rozšířeném PBD je zavedena inverzní tuhost $\alpha = \frac{1}{k}$ pro každé omezení a její podoba vzhledem k časovému úseku $\widetilde{\alpha} = \frac{\alpha}{\Delta t^2}$.

Langrangeův multiplikátor pro specifické omezení lze vyjádřit pomocí matice hmotností bodů $\mathbf{M} = diag(m_1, m_2, \ldots, m_N)$ takto:

$$\lambda = \frac{C(\mathbf{p}_1, \ldots, \mathbf{p}_N)}{\nabla C(\mathbf{p}_1, \ldots, \mathbf{p}_N)\mathbf{M}^{-1}\nabla C(\mathbf{p}_1, \ldots, \mathbf{p}_N)^T} \tag{7.5}$$

Zavedením inverzní tuhosti $\widetilde{\alpha}$ do rovnosti 7.5 vzniká vztah pro inkrementální přírustek multiplikátoru $\Delta\lambda$:

$$\Delta\lambda = \frac{-C(\mathbf{p}_1, \ldots, \mathbf{p}_N) - \widetilde{\alpha}\lambda}{\nabla C(\mathbf{p}_1, \ldots, \mathbf{p}_N)\mathbf{M}^{-1}\nabla C(\mathbf{p}_1, \ldots, \mathbf{p}_N)^T + \widetilde{\alpha}} \tag{7.6}$$

Na závěr každé aplikace omezení jsou při použití extended PBD aktualizovány nejen pozice, ale i Lagrangeovy multiplikátory za použití vztahu 7.7:

$$\lambda = \lambda + \Delta\lambda \tag{7.7}$$

Použití extended PBD přináší za malou cenu (jediná hodnota k uchování a jediný výpočet na omezení navíc) znamenitý výsledek ve formě konzistentních řešení [1]. Tato řešení nejsou závislá na čase simulace.

7.1.3 Tlumení

Tlumení simulace je používáno pro zvýšení stability a zlepšení rychlosti. Je to snaha o minimalizaci lokálních oscilací bodů, které vychází z vlivu rychlostí (řádky (5)-(6) v algoritmu 1) a chyby numerické aproximace, popř. chyby softwarové reprezentace čísel a matematických operací. Nevhodná implementace tlumení ovšem může ústit v nežádanou změnu momenta celého simulavéno objektu [1].

7.2 Detekce kolizí v PBD

Detekce kolizí v celé scéně, kde je použita pouze základní reprezentace těles pomocí vrcholů a trojúhelníků sítí je velmi časově náročná. Musí dojít k porovnání hrubou silou všech dvojic částic (trojúhelníky a vrcholy sítí modelů) nacházejících se ve scéně [4]. Při počtu n částic nabývá problém detekce kolizí časové náročnosti $O(N^2)$. Tento přístup je pro simulace v reálném čase tohoto rozsahu zcela nedostačující z hlediska časové náročnosti.

Proto je vhodné počítat jednotlivé kroky na GPU a použít pro reprezentaci částic hierarchickou strukturu, která rapidně sníží počet porovnávání na relevantní částice pro potencionálně kolidující podprostory. Těmito strukturami mohou být SDF pro kosti v kombinaci s využitím např. AABB kvádrů

ohraničujících jednotlivé trojúhelníky s vrcholy či z nich vhodně sestavené n-tice v rámci těles svalů. Tyto AABB primitiva navíc může být vhodné udržovat např. ve strukturách BVH stromů a ty efektivně prohledávat a aktualizovat. Další přístupy pro urychlení detekce kolizí jsou uvedeny v kapitole 5. Zvolené metody budou popsány v následující kapitole.

8 Zvolené řešení

Využití knihovny VTK pro detekci kolizí se nezdá být vhodnou variantou, neboť třída navíc k časově neefektivnímu testu na kolizi vrací v případě kolize pouze pozice dvojic bodů v dané kolizi. Bylo by tudíž nutné implementovat vlastní reakci na kolizi dvojice bodů pouze z těchto dvou pozic.

První knihovna zvolená pro integraci detekce kolizí je knihovna Discregrid. Knihovna totiž navíc k detekci kolizí poskytuje i úplnou informaci (hloubku průniku a gradient) potřebnou pro řešení dané kolize. Vygenerovaná pole gradientů (SDF) navíc není nutné v průběhu simulace aktualizovat. Oproti jiným knihovnám Discregrid nabízí velmi přímočarý přístup k získání těchto informací o kolizi, a tak je knihovna zvolena nejen kvůli nízké výpočetní náročnosti, ale i kvůli zdánlivě jednoduché integraci.

Dalším slibným kandidátem se zdá být knihovna NVIDIA FleX, která je ovšem značně robustnější než knihovna Discregrid, a tudíž by její integrace mohla s sebou přinést mnoho implementačních problémů. Knihovna navíc taktéž implementuje pro detekci kolizí strukturu SDF, a tak je v kontextu této práce na knihovnu nahlíženo spíše jako na složitější alternativu ke knihovně Discregrid.

Druhou volbou je knihovna FCL. Knihovna jako jediná z analyzovaných nabízí detekci kolizí pomocí urychlení nad hierarchickými strukturami množiny objektů, reprezentovaných různými ohraničujícími primitivy. Konkrétně pro deformovatelná tělesa jsou to primitiva AABB a OBB. Autoři dokonce uvádí rychlou aktualizaci BVH stromů těchto těles. Při nalezení kolidujícího páru bodů je pak k dispozici dostatečná informace o kolizi tj. vektor od jednoho primitiva, ve kterém se nachází jeden z bodů, ke druhému primitivu, ve kterém se nachází druhý bod, a hloubka průniku těchto primitiv.

Knihovny Discregrid a FCL jsou vhodné nejen proto, že je jejich implementace realizována v programovacím jazyce C++ a v případě knihovny Discregrid i v kontextu PBD (v jazyce C++ je implementováno stávající řešení, založené na metodě PBD), ale navíc používají teoreticky časově efektivní a vhodné techninky pro zrychlení detekce kolizí v simulaci tuhých a elastických těles v reálném čase. Podle autorů vykazují realistické výsledky dokonce i pro složité scény s velkým množstvím simulovaných těles, a tudíž vysokým požadavkem na rychlost detekce. V Následující kapitole proto bude popsán návrh jejich integrace do stávajícího řešení.

9 Návrh integrace knihoven

Stávající řešení je implementováno v programovacím jazyce C++ v kontextu vizualizačního open-source softwaru VTK. VTK poskytuje více-úrovňovou vizualizaci s nastavitelným mapováním. Do VTK jsou v rámci simulace periodicky posílána aktualizovaná data a ta vykreslována na výstup. K tomu je potřeba při každém kroku simulace konvertovat interní datové struktury zvoleného algoritmu pro určování nových stavů objektů ve scéně na datové typy VTK. Základní datovou strukturou PBD je v práci M. Červenky [4] struktura grafu, nad kterou jsou vykonávány operace nad uzly, hranami a trojúhelníky všech objektů ve scéně.

Nyní bude schématicky popsán algoritmus detekce a řešení kolizí, který s touto strukturou pracuje.

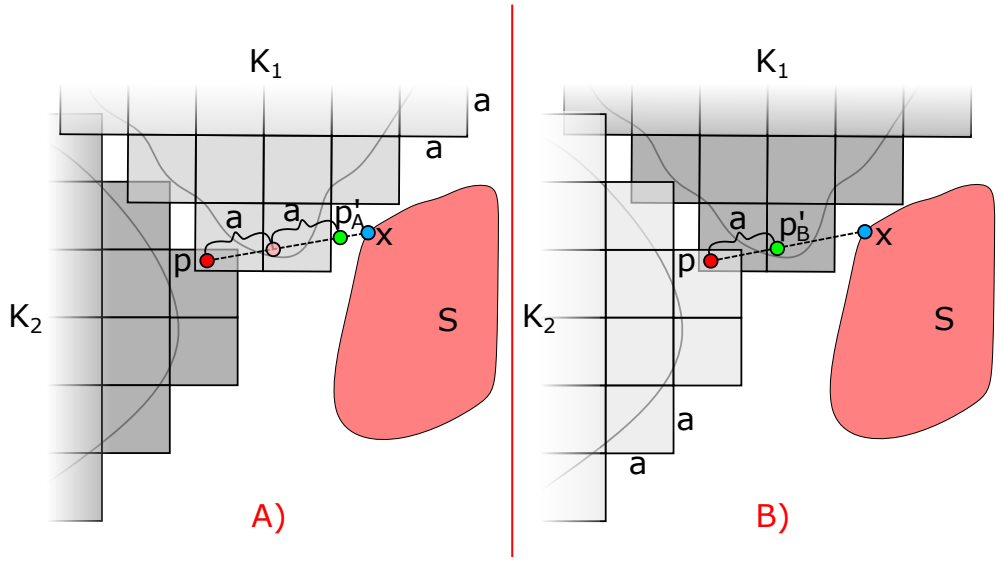
9.1 Stávající detekce a řešení kolizí

V inicializaci první iterace simulace dochází ke generaci detekce kolizí tj. voxelizaci pro každou kost ve scéně. Následovně je odstartován vnější cyklus simulace. Nejprve dojde k aplikaci externích sil na body ve scéně a je odstartován vnitřní cyklus metody, kdy dochází k aplikaci omezení zachování vzdálenosti, zachování objemu a tvaru, přičemž poslední aplikované omezení je omezení kolizní.

Nejdříve je provedena detekce kolizí pro všechny svaly přes všechny kosti. Tedy pro každý bod daného svalu je položen dotaz na detekci kolizí dané kosti, reprezentované voxelizovaným prostorem, zdali ke kolizi dochází. Součástí dotazu je originální pozice bodu $\mathbf{x}$ a pozice tohoto bodu po aplikaci externích sil a předchozích omezeních $\mathbf{p}$. Dále je v případě kolize očekávaná i pozice bodu po řešení dané kolize. Je nutné uvažovat, že jsou testy prováděny nad neaktualizovaným voxelizovaným prostorem kostí, které ovšem v simulaci podléhají transformaci. Je proto nutné na dotazované pozice bodu $\mathbf{x}$ a $\mathbf{p}$ aplikovat nejdříve inverzní transformaci dané kosti a na výslednou pozici v případě kolize zase aplikovat aktuální transformaci této kosti. Detekce a řešení kolizí tedy ve skutečnosti probíhá neustále v oblasti původních transformací kostí ve scéně.

Pokud se bod $\mathbf{p}$ nachází v kolizi, je danou metodou nalezena nová, vyhovující pozice bodu, kdy už se tento bod v kolizi nenachází a tento bod je přidán mezi kolidující. Kolidující body jsou po detekci přes všechny kosti opraveny z pozic $\mathbf{p}$ na nalezené vyhovující pozice, ovšem na daný bod je

aplikována pouze poslední korekce. V případě kolize bodu svalů s více kostmi je tedy opravena pouze kolize s poslední kostí. Kolize bodu svalu s více kostmi je možná kvůli hrubé voxelizaci kostí. Příklad takové kolize je vyobrazen na obrázku 9.1.

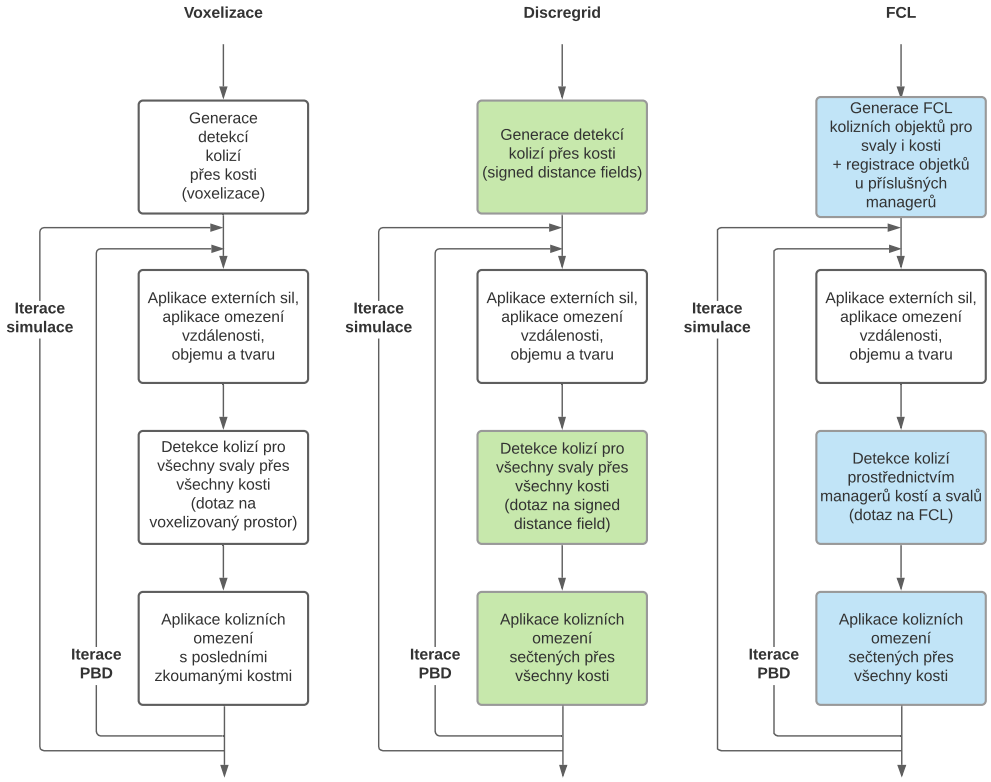


Obrázek 9.1: Ilustrační příklad kolize bodu svalu s více kostmi v jednom kroku simulace, zjednodušeno na 2D. Ve scéně se nachází sval **S** s bodem **x** a kosti $\mathbf{K_1}$, $\mathbf{K_2}$ s provedenou voxelizací o rozměru voxelu **a**×**a**. V obou případech A) a B) je bod svalu **x** vytlačován vnějšími silami a ostatními omezeními na pozici **p**. V případě A) je nejprve detekována kolize s kostí $\mathbf{K_1}$. Nová pozice $\mathbf{P'_A}$ pro bod **x** je nalezena iterativním posunem z pozice **p** po vektoru $\overrightarrow{\boldsymbol{px}}$ o velikosti **a**. Stejným způsobem je v případě B) nalezena nová pozice $\mathbf{P'_B}$, ovšem vzhledem ke kosti $\mathbf{K_2}$. Bod **x** je posunut nejdříve na pozici $\mathbf{P'_A}$ a pak na pozici $\mathbf{P'_B}$. Tím končí aplikace kolizních omezení na tento bod a přesto zůstává v kolizi s kostí $\mathbf{K_1}$, první řešení kolize s kostí $\mathbf{K_1}$ je tak ignorováno.

Pokud se v bod **p** v kolizi nenachází, není přidán do kolizního seznamu a v následujícím kroku simulace se opravdu bod **x** pohne na pozici bodu **p**.

9.1.1 Návrh úprav algoritmu

Na obrázku 9.2 je znázorněno blokové schéma návrhu úpravy popsaného stávajícího algoritmu voxelizace pro knihovny Discregrid a FCL. V následujících sekcích budou návrhy pro obě knihovny rozebrány do detailu.



Obrázek 9.2: Blokové schéma návrhu úprav stávajícího algoritmu pro knihovny Discregrid a FCL. V levém sloupci je znázorněn stávající algoritmus, v prostředním sloupci je znázorněn upravený algoritmus pro knihovnu Discregrid (úpravy zelenou barvou) a v pravém sloupci pro knihovnu FCL (úpravy modrou barvou).

9.2 Návrh integrace knihovny Discregrid

Knihovna Discregrid umožňuje generování SDF pole nad každou kostí. Tato pole není třeba aktualizovat, neboť nejsou kosti deformovány, nýbrž pouze transformovány. V kontextu stávajícího řešení se tudíž nabízí možnost jednoduše vyměnit generování struktur detekce kolizí pomocí voxelizace v první iteraci simulace za generování SDF polí pro každou kost.

Dále by nemělo být obtížné položit dotaz na toto gradientové pole dané kosti pro daný bod svalu stejně jako u voxelizace. V tomto případě ovšem není potřeba v případě kolize např. postupně vracet bod $\mathbf{p}$ na originální pozici $\mathbf{x}$, ale stačí posunout bod po opačném směru gradientu o hloubku penetrace s kostí sečtenou s hloubkou jakéhosi vnějšího ohraničení kosti (aby algoritmus do jisté míry dokonce kolizi předcházel).

Nakonec je možné sečíst všechny posuny pro daný bod vzhledem k více kostem a tím se vyhnout tomu, že by byla aplikovaná pouze poslední korekce a bod by zůstal v kolizi s jinou kostí.

9.3 Návrh integrace knihovny FCL

Pro detekci kolizí za použití knihovny FCL je třeba vygenerovat kolizní objekty nejen nad objekty kostí, ale i svalů, neboť knihovna neposkytuje přímočaré řešení pro detekci kolize bodu vůči koliznímu objektu. To s sebou ovšem přináší možnost detekce kolizí nejen svalů s kostmi, ale i svalů s ostatními svaly. FCL poskytuje pro komplikované scény s mnoha objekty tzv. kolizní manažery. Manažeři spravují množiny tzv. FLC kolizních objektů, které jsou prezentovány danými BVH stromy. V první iteraci je tedy podobně jako u předchozích případů vhodné provést generaci těchto FCL kolizních objektů nad všemi kostmi a registrovat je u manažera kostí, dále také nad všemi svaly a ty zase registrovat u manažera svalů.

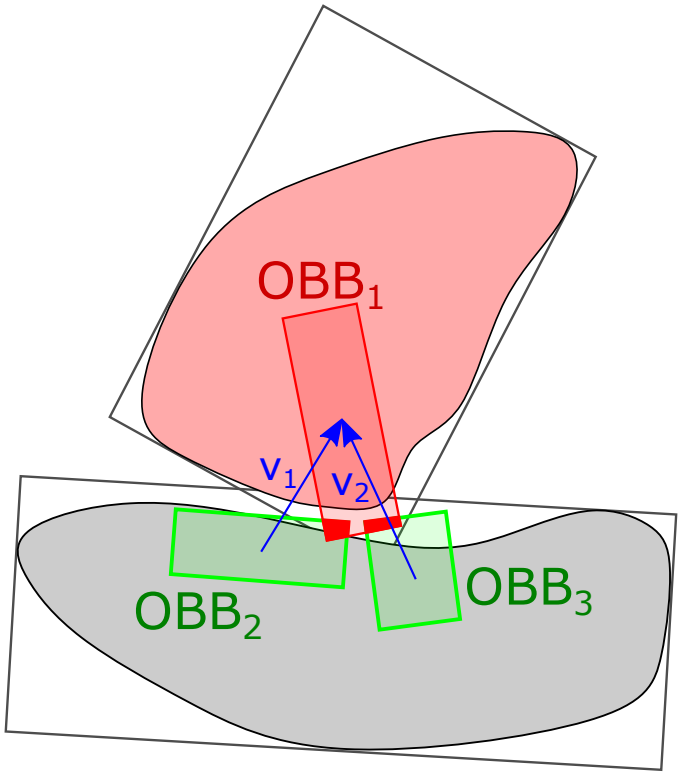
Detekci a řešení kolizí je pak nutné řešit skrze zmíněné managery. Je možné polozit dotaz na kolizi dvou různých manažerů (kosti vůči svalům) a na kolize v rámci jednoho manažera (svaly vůči svalům). Před každým dotazem je navíc třeba provést aktualizaci všech kolizních objektů. Pro kosti postačí aktualizovat transformaci dané kosti, pro svaly je pak třeba aktualizovat všechny vrcholy svalů a nad nimi konstruovat nové BVH stromy.

Dotaz na detekci kolizí dvou či jednoho manažera může vracet všechny páry trojúhelníků v kontaktu, kdy každý z páru trojúhelníků náleží příšlušným kolizním objektům. Pro každý pár trojúhelníků je k dispozici hloubka průniku kolidujících listů BVH stromů, které obsahují dané trojúhelníky. Dále kontaktní pár obsahuje normalizovaný vektor směřující od středu jednoho kontaktního BVH listu ke druhému. Nabízí se tedy možnost posunout všechny vrcholy kolidujícího trojúhelníku svalu po zmíněném vektoru o hloubku průniku a tím kolizi vyřešit. V případě kolize dvou svalů pak např. každý z těchto bodů posunout o polovinu hloubky průniku směrem od sebe.

Jako primitivum BVH stromů pro deformovatelná tělesa je možné použít AABB kvádry či sloučeninu OBB orientovaných ohraničujících kvádrů a tzv. RSS (rectangular swept sphere), kdy jsou hrany OBB kvádru jaksi zaobleny tvarem koule. Tato sloučenina primitiv (dále jen OBBRSS) umožňuje efektivní detekci kolizí (složka OBB) a efektivní výpočet vzdálenosti primitiv (složka RSS). Vzhledem k tomu, že je knihovna napsána za použití tzv. šablon (templates), neměl by být problém implementovat oba druhy primitiv.

Kolize např. dvou objektů reprezentovaných BVH stromem s OBB pri-

mitivy ovšem často nemusí poskytovat jednoznačný výsledek. Naopak se může stát, že je kolizí na jedno OBB primitivum vzhledem k jinému objektu mnoho. Problematika je znázorněna na obrázku 9.3.



Obrázek 9.3: Uvažujme objekt svalu (růžová) s daným podprostorem ohraničujícího OBB kvádru **OBB$_1$** a objekt kosti (šedá) s danými podprostory ohraničujících OBB kvádrů **OBB$_2$** a **OBB$_3$**. Červeně vyznačené plochy průniků kvádrů značí detekované kolize. Pro trojúhelník nacházející se v podprostoru **OBB$_1$** jsou tedy k dispozici dva ve skutečnosti jednotkové vektory k posunu **v$_1$** a **v$_2$**. Takto lze sesbírat řešení kolizí přes všechny žádané objekty a tím např. vyřešit problém kolize s více kostmi znázorněný na obrázku 9.1.

V případě vyobrazeném na obrázku 9.3 je jedna z možností řešení takové kolize např. sečíst výsledné kolizní vektory a posunout kolidující trojúhelník o maximální hloubku průniku.

Neboť jsou návrhy obou knihoven v tuto chvíli dostatečně popsány, budou v následující knihovně popsány některé implementační detaily integrací obou knihoven.

10 Implementace integrace knihoven

Obě knihovny byly implementovány do jediného projektu stávajícího řešení zdokumentovaného v práci M. Červenky z roku 2019 [4]. Při integraci knihoven byla snaha poskytnout uživateli možnost výběru z množiny detekcí kolizí, tedy vzájemnou nezávislost knihoven a stávající voxelizace při sestavování projektu. Neboť samotný projekt je sestavován pomocí nástroje CMake, bude v první sekci této kapitoly popsán nástroj i jeho využití pro zajištění společného překladu s oběma knihovnami.

10.1 Integrace pomocí CMake

CMake je open-source multiplatformní nástroj pro automatizovanou konfiguraci a sestavení projektů programovacích jazyků C++ a C. S tímto nástrojem je např. možné sestavit několik konfigurací projektů najednou, vytvářet preprocesorová direktiva či sestavit projekt pro různé verze požadovaného vývojového prostředí. Všechny možnosti a nastavení konfigurace se typicky nachází v souborech **CMakeLists.txt** či v souborech s příponou **.cmake**. Před samotnou konfigurací je potřeba, aby uživatel vybral požadované konfigurace a poskytnul příslušné systémové cesty. Pro každou knihovnu byla přidána do seznamu možností projektu možnost sestavení **BUILD_DISCREGRID** a **BUILD_FCL** pro příslušné knihovny Discregrid a FCL.

Možnosti byly přidány úpravou souboru **CMakeLists.txt** ve zdrojovém adresáři projektu. V tomto souboru je např. pro Discregrid přidána řádka `include{Discregrid}`, která zajistí nalezení a provedení CMake skriptu nacházejícího se v souboru **Discregrid.cmake** v podadresáři **CMake**. Právě zde je definovaná možnost **BUILD_DISCREGRID**, jež pokud je uživatelem zvolena, odstartuje hledání potřebných závislostí a adresářů této knihovny. V případě že jsou adresáře a knihovny nalezeny či poskytnuty explicitně uživatelem, je k cíli sestavení projektu přidán jak zdrojový, tak hlavičkový soubor třídy `SDF`, reprezentující integraci knihovny Discregrid, stejně tak jako jsou s cílem spojeny Discregrid knihovny s příponou **.lib** a přidána cesta k `include` adresáři knihovny, obsahující její rozhraní. Stejným způsobem je integrace zajištěna pro knihovnu FCL. V obou případech je navíc nutné spojit s cílem i závislosti knihoven.

10.2 Podmíněný překlad

Aby nedošlo ke snaze překladu např. třídy `SDF` v případě, že není zvolena možnost **BUILD_DISCREGRID**, je v souboru **Discregrid.cmake** přidána definice preprocesorové direktivy **BUILD_DISCREGRID**. Preprocesorovou podmínkou existence této direktivy je ohraničen jak zdrojový, tak hlavičkový soubor třídy `SDF` na první řádce kódem definicí podmínky `#ifdef BUILD_DISCREGRID` a na poslední řádce ukončením podmínky `#endif`. Stejně tak jsou ohraničeny soubory implementace detekce kolizí knihovnou FCL podmínkou existence direktivy kódem `#ifdef BUILD_FCL` na prvních a kódem `#endif` na posledních řádkách souborů. Tato direktiva je taktéž definována, pakliže uživatel v CMake konfiguraci o přeložení knihovny FCL žádá.

V případě, že jsou vybrány obě možnosti **BUILD_DISCREGRID** i **BUILD_FCL**, je potřeba rozhodnout, která metoda detekce a řešení kolizí je při spuštění programu použita. Proto je pro knihovnu Discregrid definována další direktiva `#define USE_DISCREGRID` a pro knihovnu FCL direktiva `#define USE_FCL` pouze v případě, že není definována direktiva `BUILD_DISCREGRID`. Tím je zaručeno, že se beze změny definic spustí program s detekcí a řešení kolizí knihovnou Discregrid. Definice těchto direktiv lze samozřejmě patřičně v kódu zakomentovat a mezi metodami na daný běh programu libovolně alternovat. Pokud nejsou definována direktiva pro použití Discregrid ani FCL, je použit algoritmus voxelizace.

10.3 Knihovna Discregrid

Knihovna Discregrid byla integrována podle návrhu popsaného v sekci 9.2 bez větších komplikací díky tomu, že je možné položit na vygenerované SDF dotaz pro určitý bod v prostoru. Implementací knihovnou Discregrid bylo tedy možné jaksi zastoupit předešlou voxelizaci, neboť v původním řešení třída `voxel` reprezentující voxelizaci implementuje rozhraní `collision`. K implementaci tohoto rozhraní bylo nutné především doplnit metodu pro generaci SDF pro jakoukoliv kost, tedy získat všechny vrcholy, trojúhelníky a meze dané kosti a podsunout je metodě Discregridu pro generaci SDF. Pro generaci těchto polí bude zvoleno rozlišení $64 \times 64 \times 64$ jako vhodná kompenzace paměťové náročnosti a kvality detekce kolizí.

10.3.1 Detekce a řešení kolizí

Dotaz na kolizi je pak proveden pouze jedním dotazem na SDF a jestliže je výsledná hloubka průniku záporná i s daným přičteným okrajem kosti,

nastává kolize. Kolize je pak detekována, pakliže platí vztah 10.1:

$$hloubka\_průniku < 0 + okraj\_kosti \tag{10.1}$$

Přidání okraje by mělo v mnoha testovacích případech předcházet kolizi úplně.

K řešení kolize byla použita jak hloubka průniku, tak poskytnutý gradient průniku. Gradient totiž stačí skalárně vynásobit absolutní hodnotou hloubky průniku a tento výsledný vektor přičíst ke kolidujícímu bodu. Na dotazovaný bod je nutné aplikovat před dotazem na kolizi transformaci dané kosti a následně provést opačnou transformaci bodu výsledného. Takto byla vyřešena kolize jednoho bodu vzhledem k jedné kosti.

10.3.2 Kolize se všemi kostmi

Rozšíření algoritmu na detekci přes všechny kosti bylo provedeno jednoduchou úpravou ve dvou částech dle návrhu z předešlé kapitoly. Místo navrácení opravené pozice dotazovaného bodu nyní metoda v případě kolize navrátí pouze gradient vynásobený absolutní hodnotou hlouby penetrace a na tento vektor již není aplikována inverzní transformace kosti. Místo toho jsou všechny takto nalezené vektory přičteny k původní kolidující pozici $\mathbf{p}$, a tak je docíleno detekce kolizí přes všechny kosti ve scéně. Tato modifikace je popsána následujícím algoritmem 2:

Algoritmus 2 Řešení kolizí jednoho svalu přes všechny kosti

```
1: for all muscle_vertices i do
2:    initialize resolved_i = p_i
3:    for all bone_models j do
4:       initialize detection = get_detection(j)
5:       if detection->collision(p_i, &gradient) is true then
6:          resolved_i += gradient
7:       end if
8:    end for
9: end for
```

10.4 Knihovna FCL

K implementaci detekce a řešení kolizí pomocí této knihovny bylo nutné v inicializaci první iterace simulace vygenerovat kolizní objekty nejen pro

kosti, ale i pro svaly. Tyto kolizní objekty reprezentuje v projektu vlastní třída, jež může daný objekt interně reprezentovat v kontextu FCL BVH stromem buď AABB kvádrů či OBBRSS ohraničujících orientovaných kvádrů se zakulacenými hranami. Kolizní objekty jsou taktéž registrovány u příšlušných manažerů.

10.4.1 Detekce a řešení kolizí

Vhledem k podstatě knihovny (rychle procházet a testovat množiny BVH stromů na kolize) není vhodné integrovat knihovnu stejným způsobem, jako knihovnu Discregrid. Místo dotazu na kolizi jednoho bodu vzhledem ke koliznímu objektu kosti je zde nutné položit dotaz na všechny kolize dvou množin kolizních objektů, který navrací všechny dvojice trojúhelníků v kolizi. Tyto dvojice bohužel nejsou knihovnou nijak řazeny, a tak nezbylo nic jiného, než prozkoumat, jakému objektu (kosti či svalu) s jakým identifikátorem každý kolizní trojúhelník náleží a řešit kolize pouze pro trojúhelníky náležející modelům svalů.

Řešení jedné kolize bylo oproti návrhu integrace provedno ne za pomoci kolizního vektoru vynásobeného příšlušnou hloubkou průniků primitiv, ale místo toho byla využita normála daného kolizního trojúhelníku náležejícího modelu kosti. Normály byly pro všechny body kolidujícího trojúhelníku přes všechny kolize primitiv sečteny, znormalizovány a následně vynásobeny maximální hodnotou nad příšlušnými hloubkami průniků. Po tomto výsledném vektoru byl posunut bod **p** na nekolidující pozici vzhledem ke všem kostem ve scéně.

V následující kapitole budou uvedeny výsledky experimentů provedených s uvedenými implementacemi detekce a řešení kolizí včetně porovnání se stávající metodou voxelizace.

11 Dosažené výsledky

Tato kapitola popisuje testování výsledků dosažených oběma integrovanými knihovnami pro detekci a řešení kolizí. Budou představena data, na kterých budou metody testovány, i prostředí a nástroje, které k experimentům budou použity. Aktuálně existují dvě relativně přímé možnosti jak testování provést. Jednou je využití Bash skriptů připravených M. Červenkou v rámci stávajícího řešení [4] a spouštět pouze projekt PBD, ve kterém jsou připraveny dva scénáře pro pohyb kosti se svaly. Tento projekt je součástí většího projektu Muscle Wrapping 2.0 (`https://gitlab.com/besoft/muscle-wrapping-2.0/`).

Druhou cestou je přímo spouštět tuto větší aplikaci s použitím metody PBD. Projekt Muscle Wrapping 2.0 je soustředěn na vývoj pluginu pro simulaci chování a určování tzv. mechanických akčních vláken svalových objektů okolo objektů kostí, sloužící k biomechanické analýze. Plugin je vytvářený pod licencí Apache License ver. 2.0 (`https://www.apache.org/licenses/LICENSE-2.0`) pro projekt OpenSim 4.0 poskytovaný platformou SimTK, která je určená pro biomedicínské výpočetní projekty (`https://simtk.org`).

Součástí Muscle Wrapping 2.0 je pět připravených scénářů pro simulaci pohybu kostí dolní končetiny usazené v kosti pánve. Scénáře jsou připraveny pro pozorování chování čtyř různých svalů: krátký přitahovač (*adductor brevis*), velký sval hýžďový (*gluteus maximus*), střední sval hýžďový (*gluteus medius*) a sval kyčelní (*iliacus*). Poslední scénář obsahuje oba hýžďové svaly najednou. Pro všechny scénáře je definovaný pro kosti končetiny stejný pohyb, ke kterému dochází při dynamické kontrakci kyčelních flexorů tj. všechny vyjmenované svaly jistou mírou přispívají k tomu, aby se končetina přibližovala směrem k hrudi, tedy dochází k rotaci kosti *femur* v kyčelním kloubu. V tomto případě interaktivní simulace je ovšem zdrojem pohybu kost, což vysvětluje např. již zmíněný problém natlačení kosti do svalu, který je třeba řešit a ve výsledcích zohlednit.

Pro testování je zvolen přístup Muscle Wrapping 2.0 z důvodu širší škály připravených testovacích scénářů. Proto budou v nadcházející sekci popsány soubory definující svaly a kosti používané pro OpenSim 4.0 scénáře, stejně jako budou popsány soubory definující samotné scénáře.

11.1 Testovací data

Neboť je projekt vyvýjený v rámci sady nástrojů VTK, jsou ve scénářích 3D objekty kostí i svalů popsány VTK formátem 3D objektů s příponou **.vtk**, které jsou dá se říci interní reprezentací VTK původních OBJ objektů s příponou **.obj**. Každý soubor definuje vrcholy trojúhelníkové sítě objektu, kdy počet vrcholů dosahuje u modelů svalů hodnoty až 80000 (*gluteus maximus*) a u modelů kostí až 255000 vrcholů (*femur*). Všechny modely kostí i svalů jsou realnými daty co nejlépe aproximujícími části lidského těla.

Scénáře jsou ve formátu XML, kde jsou na úvod definovány příslušné soubory pro simulaci scény a pohybu v ní v rámci nástroje OpenSim 4.0, které jsou taktéž ve formátu XML. Následně jsou definovány geometrie objektů svalů s příslušnými body úponů na kosti. Scénář dále určuje strategii pro dekompozici svalů, jež se používá pro vyplnění prostorů svalů zmíněnými akčními vlákny. Nakonec je možné určit, kterou z implementovaných metod scénář použije, tedy v tomto případě bude použita metoda PBD implementována v rámci projektu M. Červenky 2019 [4].

11.2 Prostředí testování

Všechny testy proběhnou na verzi CPU Core i5-7200U 2.5 GHz pro osobní notebook s operačním systémem Microsoft Windows 10 Home. Ke spuštění aplikace bude použito Microsoft Visual Studio Community 2019 16.9.1 s překladačem Microsoft Visual C++ 2019. Aplikace bude přeložena pomocí CMake verze 3.17.2 v optimalizované `release` verzi s využitím OpenMP paralelizace.

11.3 Porovnání Discregrid, FCL a voxelizace

Softwarové zaručení kvality deformace komplexních objektů v objemné scéně bohužel často není zcela možné, neboť např. zde neexistuje objektivně správná posloupnost pozic vrcholů, se kterou by se daly výsledky scénáře srovnávat. Testování kvality deformace tedy proběhne pouze na úrovni vizuálního porovnání metod implementovaných integrovanými knihovnami s původní metodou voxelizace.

Poté je možné s využitím měřících nástrojů např. vývojového prostředí porovnat paměťovou náročnost jednotlivých metod. Nakonec je vhodné ověřit, zdali i za použití nové metody modely svalů zachovávají objem, neboť je to jediné v rámci možností měřitelné omezení PBD, které by metoda neměla do

velké míry ovlivňovat.

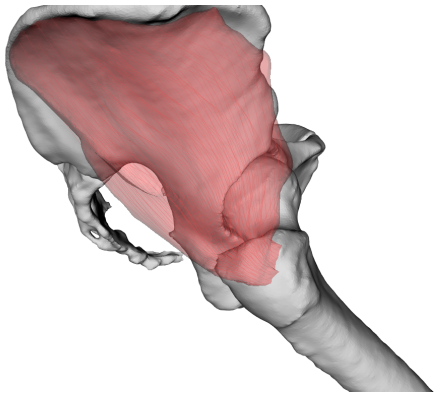
V závěru této kapitoly budou shrnuty překážky a nedostatky jednotlivých metod i testovacího prostředí, stejně jako budou diskutovány možné alternativy a modifikace těchto metod.

11.3.1 Vizuální porovnání deformace

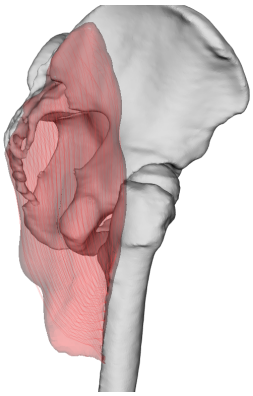
Vizuální porovnání proběhne celkem nad třemi scénami dostatečně vypovýdajícími o funčknosti, stabilitě a kvalitě každé z metod. Těmito scénami budou vždy popsaný pohyb dolní končetiny (v podstatě kosti *femur*) se svaly *gluteus medius*, *gluteus maximus* a *iliacus*.

Každá scéna obsahuje celkem 47 snímků, přičemž každý snímek proběhnou 3 iterace metody PBD. Pro každý vybraný snímek bude ukázáno chování všech tří dostupných metod (sloupce Voxelizace, Discregrid, FCL), zatímco pořadí snímku bude vypsáno vlevo od sekvence obrázků. Výběr vždy několika snímků bude záviset na tom, zdali v okolí daného snímku dochází k zásadním změnám chování některé z metod, přičemž bude hodnoceno, jak se metody celkově v daný scénář chovají a jak se v chování vzájemně liší či v čem se shodují.

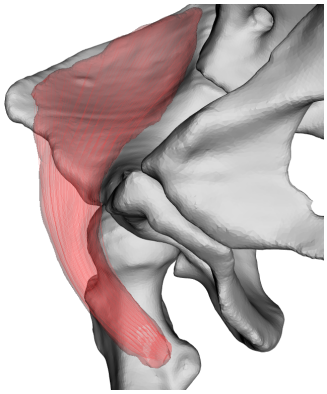
Počáteční stavy modelů testovaných svalů v simulaci jsou znázorněny na obrázcích 11.1 - 11.3.



Obrázek 11.1: Počáteční stav simulace svalu *gluteus medius*.



Obrázek 11.2: Počáteční stav simulace svalu *gluteus maximus*.

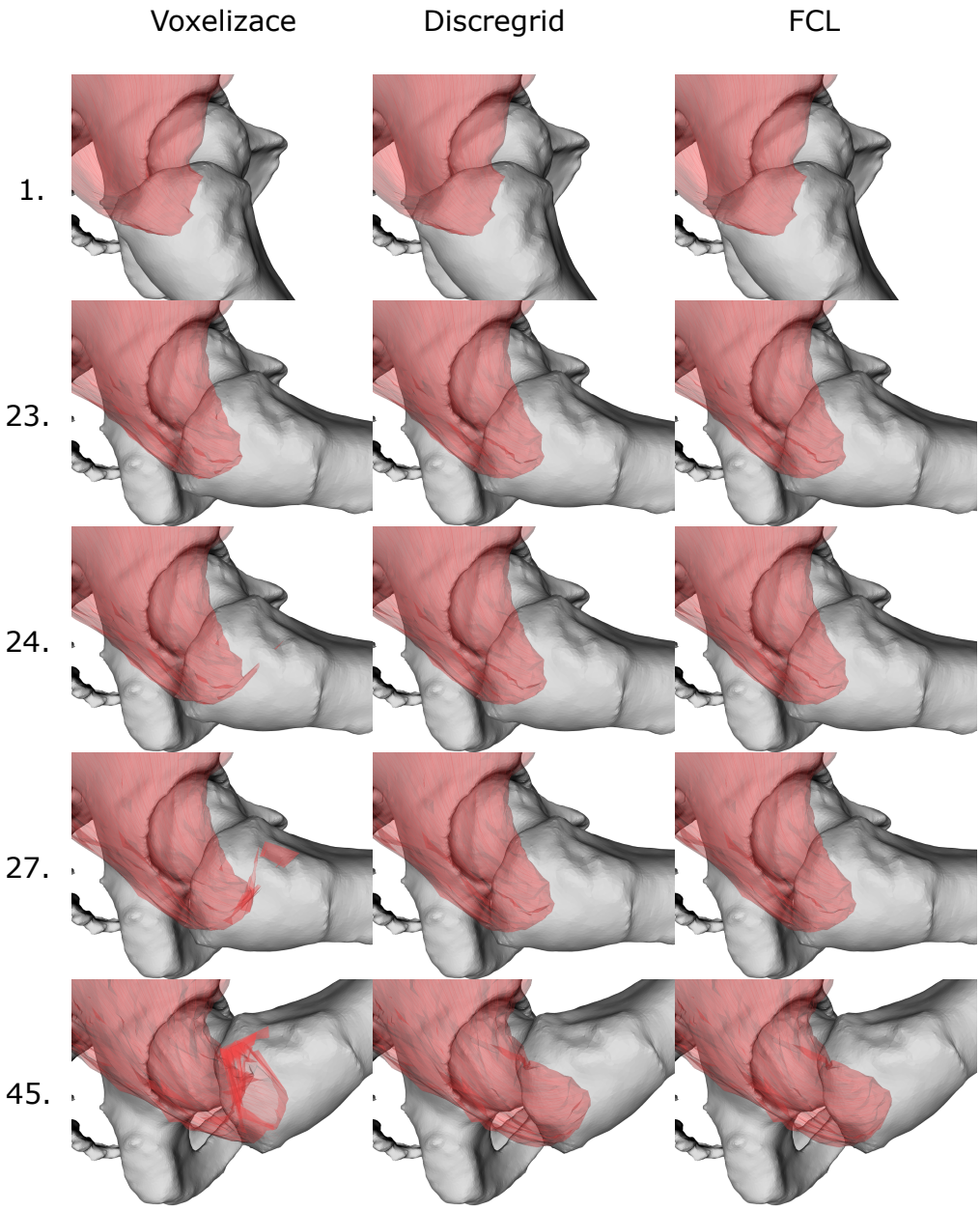


Obrázek 11.3: Počáteční stav simulace svalu *iliacus*.

Gluteus medius

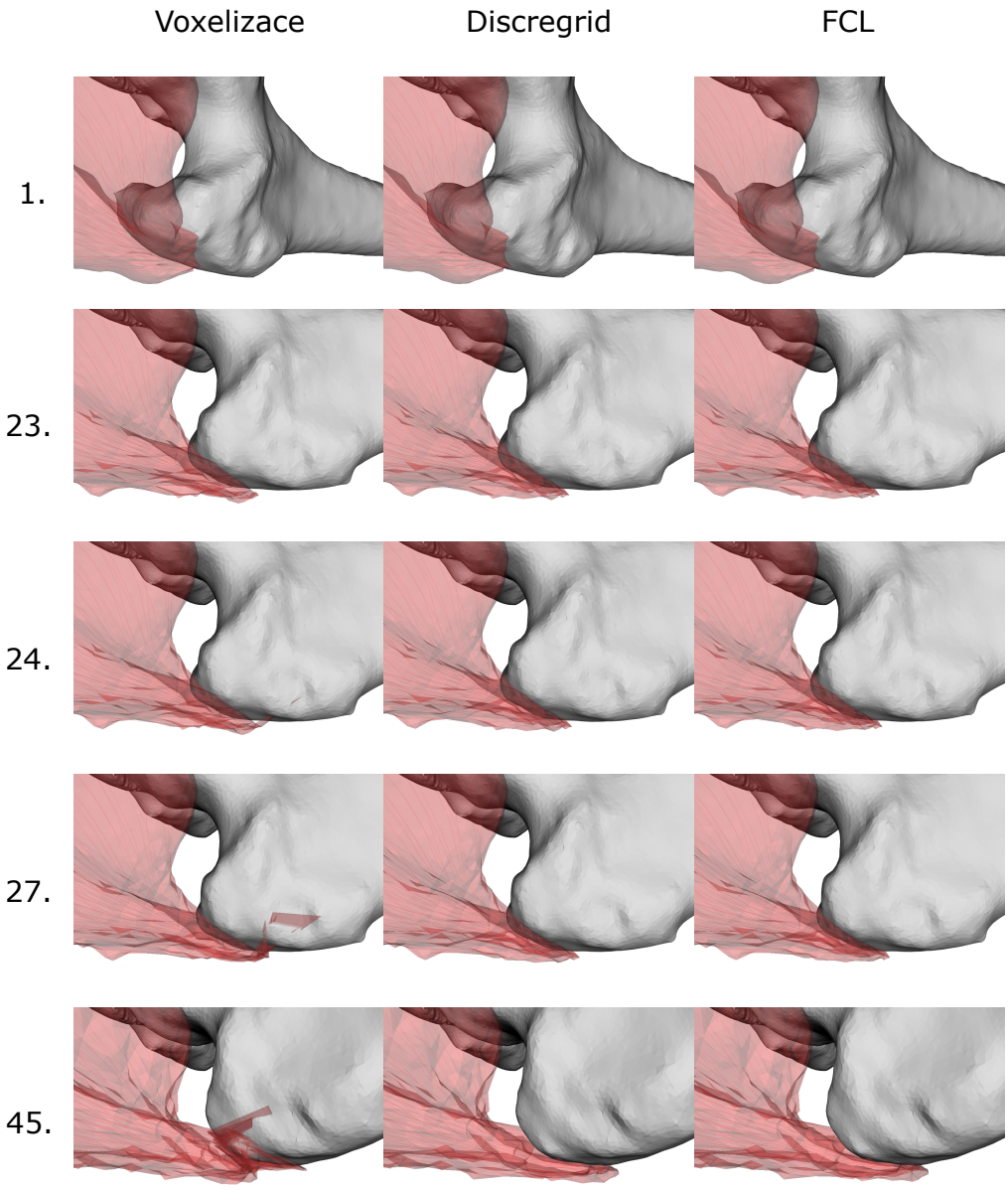
Střední sval hýžďový neboli *gluteus medius* podléhá při rotaci kosti *femur* v kyčelním kloubu téměř nepatrné deformaci v oblasti úponu na tuto kost, neboť část kosti *femur* tzv. velký chocholík při rotaci do tohoto svalu, jenž je na něm upnutý, naráží.

Řešení kolizí pro tento sval voxelizací, knihovnou Discregrid a knihovnou FCL je znázorněno na obrázku 11.4 ana obrázku 11.5.



Obrázek 11.4: Detail deformace svalu *gluteus medius* zboku.

Jak je vidět na obrázku 11.4, do 23. snímku simulace se metody chovají vizuálně téměř totožně. To platí pro metody Discregrid i FCL až do 45. snímku, zatímco začátkem 24. snímku metoda voxelizace začíná způsobovat nerealistické protínání svalu s kostí v oblasti velkého chocholníku. Metody Discregrid i FCL zde oproti voxelizaci poskytují srovnatelnou a vyhovující deformaci svalu.



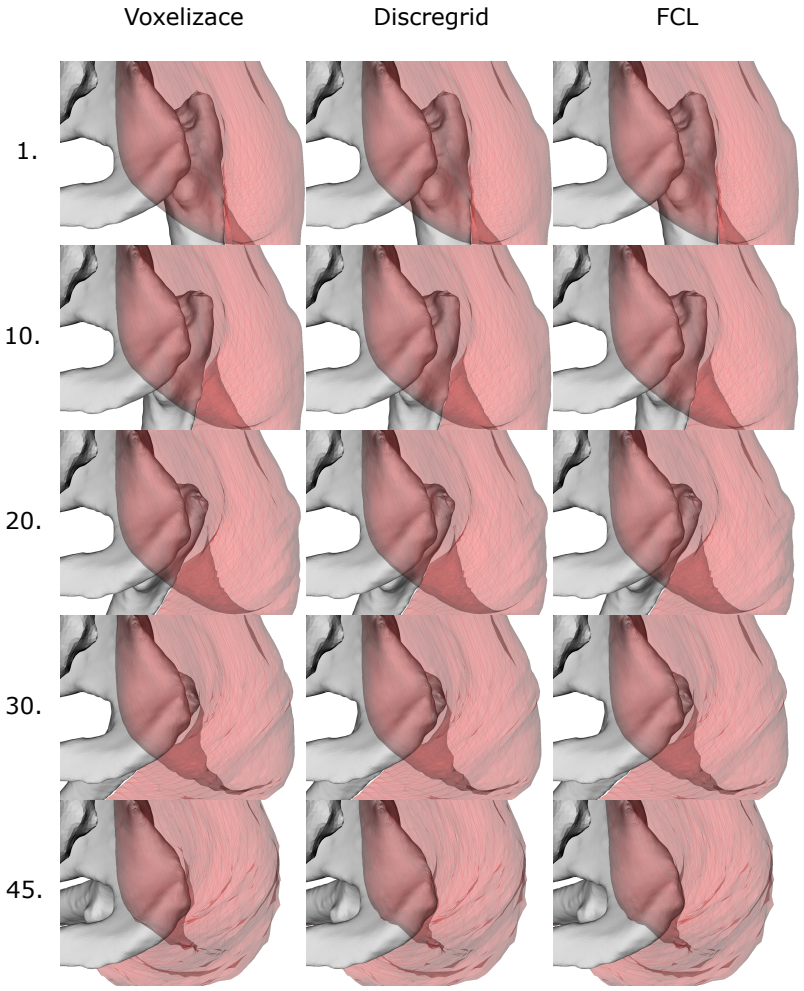
Obrázek 11.5: Detail deformace svalu *gluteus medius* shora.

Ve snímcích na obrázku 11.5 se začíná proti předchozímu úhlu zboku nerealistické protínání svalu a kosti objevovat zřetelně až ve snímku číslo 27, kdy jsou vrcholy modelu svalu vytlačeny na nekolidující pozici, ovšem na druhé straně kosti. Chování metod Discregrid a FCL je i shora velmi podobné a vyhovující.

Gluteus Maximus

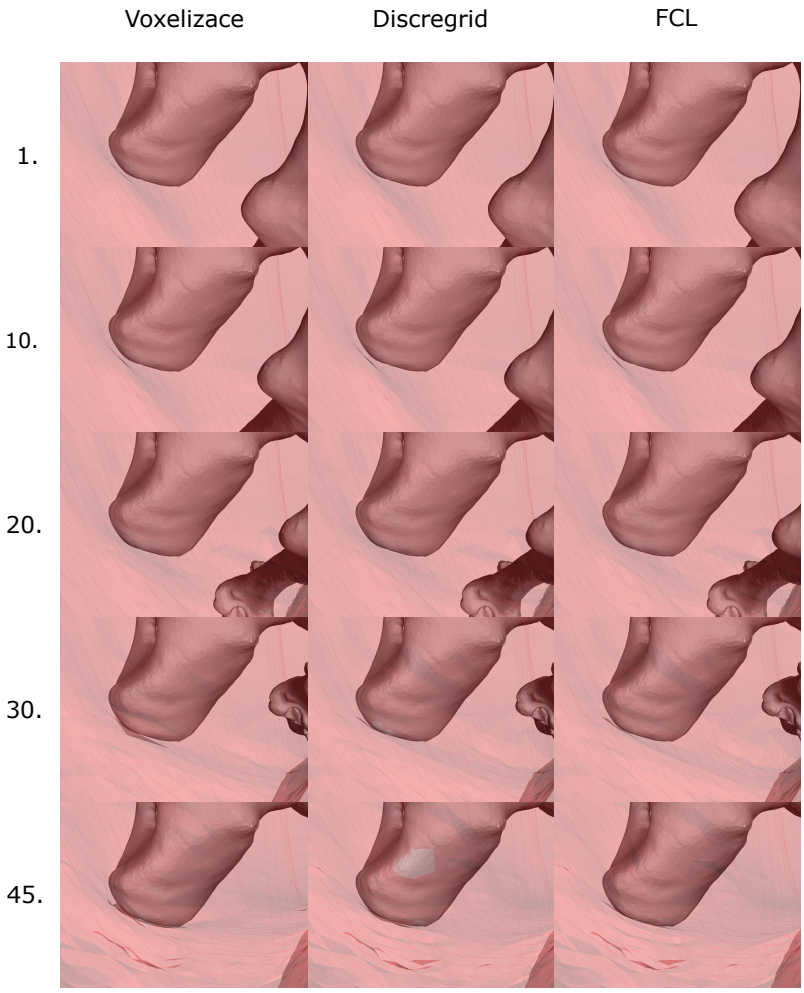
Druhý zkoumaný sval, velký sval hýžďový neboli *gluteus maximus* je pohybem kosti *femur* natahován a tlačen do oblasti tzv. *os ischii* neboli sedací kosti. Oproti předchozímu svalu zde dochází ke kolizi v jiném smyslu, tedy že sval vstupuje do stacionární kosti, nikoliv případ, že do svalu vstupuje dynamicky se pohybující kost.

Řešení kolizí pro tento sval voxelizací, knihovnou Discregrid a knihovnou FCL je znázorněno na obrázku 11.6 a na obrázku 11.7.



Obrázek 11.6: Detail deformace svalu *gluteus maximus* zpoza.

Na snímcích obrázku 11.6 se chování metod zásadním způsobem neliší. Na posledním snímku 45 je pouze např. zřetelné, že metody Discregrid i FCL dovolují detailnější řešení kolizí a tudíž jimi simulovaný sval jaksi obtéká kost sedací těsněji než v případě metody voxelizace.



Obrázek 11.7: Detail deformace svalu *gluteus medius* zpoza, detail v okolí kosti sedací.

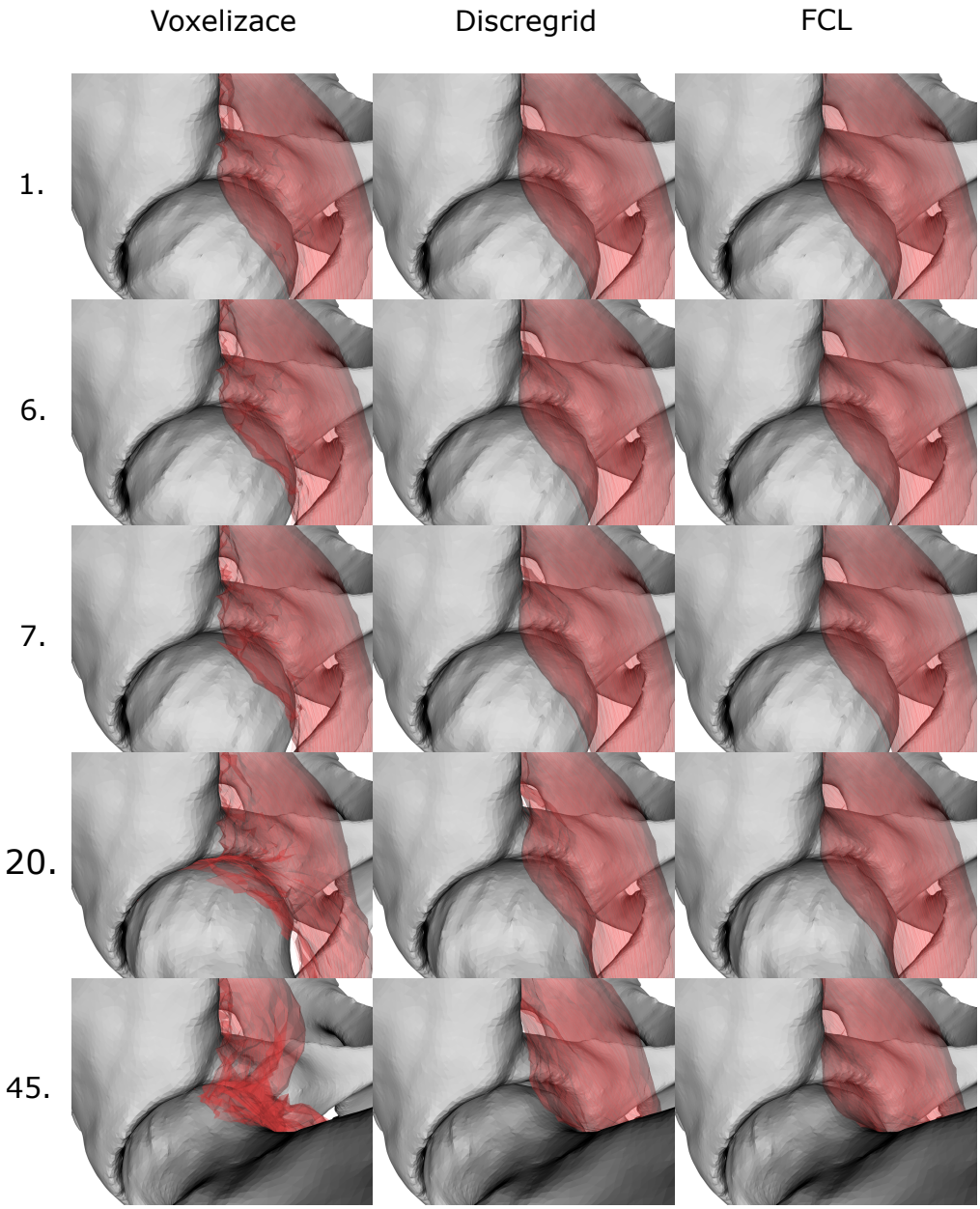
Pozorováním snímku 30 na obrázku 11.7 se dá pro metodu Discregrid vypozorovat, že dochází k průniku některých trojúhelníků s kostí (světlejší část). Na snímku 45 je potom tento průnik zřetelnější. Tento jev je nejspíše způsoben příliš nízkým rozlišením SDF této kosti. Model svalu ovšem proniká do modelu kosti jen z velmi malé části, a tak by se dal tento výsledek považovat za vyhovující. Metody voxelizace i FCL zde fungují bez problémů.

Iliacus

Pro sval kyčelní neboli *iliacus* se vyskytují oba případy kolizí, neboť se kost *femur* pohybuje přímo do svalu a v závislosti na řešení této kolize pak může docházet i k tomu, že je sval kvůli jiným PBD omezením (zachování tvaru, objemu, vzdáleností,...) v jiné části např. v blízkosti stacionární kosti tzv. *os*

pubis neboli stydké kosti do této kosti vtlačován. Navíc je tento sval v těsné blízkosti tzv. *articulatio coxae* neboli kyčelního kloubu, kde existuje mezi kostmi relativně úzký prostor, do kterého může být sval taktéž vtlačován.

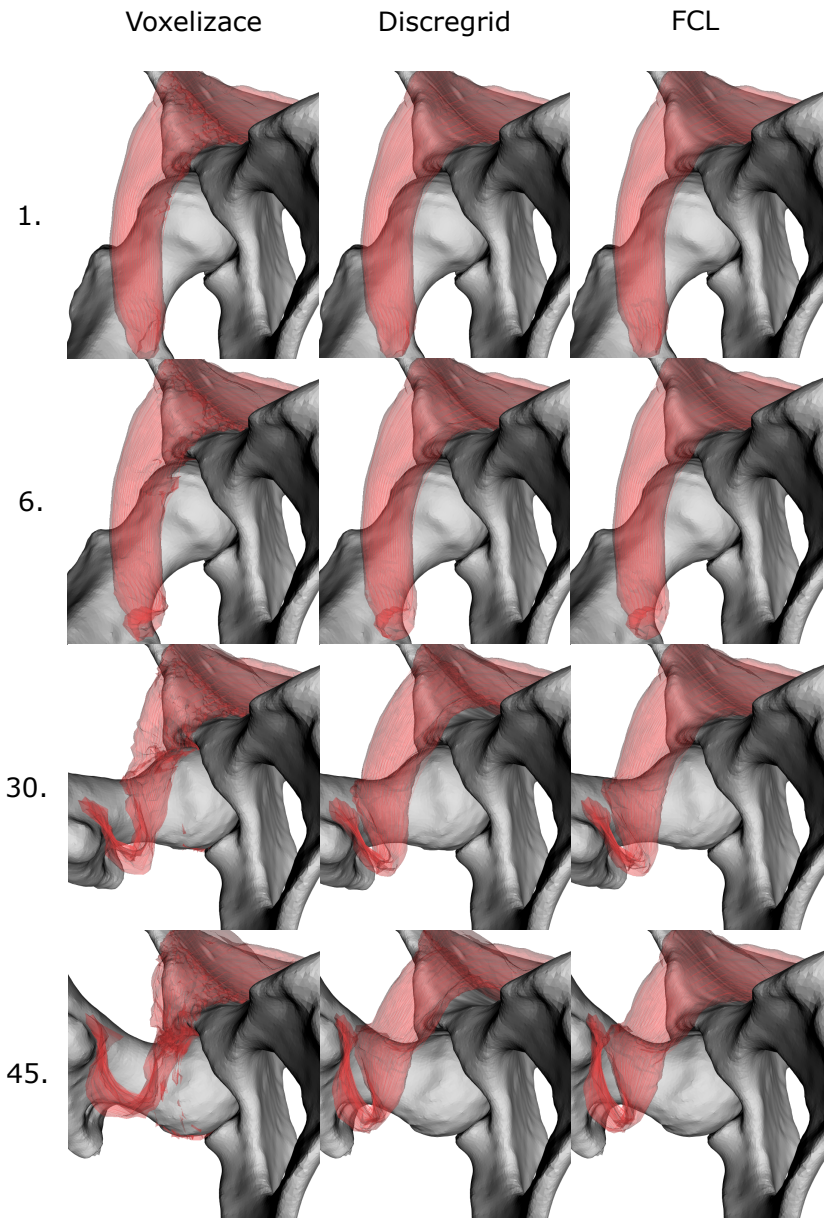
Řešení kolizí pro tento sval voxelizací, knihovnou Discregrid a knihovnou FCL je znázorněno na obrázku 11.8 a na obrázku 11.9.



Obrázek 11.8: Detail deformace svalu *iliacus* zboku.

Ze snímků na obrázku 11.8 je zřejmé již od 7. snímku, že se výsledky simulace všech metod liší. V případě metody voxelizace oproti dvěma zbylým

metodám začíná model svalu vstupovat do prostoru kyčelního kloubu. Sval je až do konce simulace nějakým způsobem navíjen. Metody Discregrid a FCL zde toto navíjení nezpůsobují. Dále metoda Discregrid dohromady s jinými omezeními PBD způsobuje, že je sval na okraji kosti pánve vytlačován směrem od pánve. To je dáno tím, že jsou metodou Discregrid kolize tohoto svalu s kostí *femur* řešeny pouze s malými posuny bodů a sval se při pohybu kosti snaží zachovat tvar a objem. Stává se v jistém smyslu více vzpřímeným.



Obrázek 11.9: Deformace celého svalu *iliacus* zepředu.

Na 7. snímku obrázku 11.9 lze vidět v případě metody voxelizace začátek

pronikání svalu do oblasti kyčelního kloubu, stejně jako na úhlu zboku. Dále je na snímku 30 pro metodu Discregrid zřetelná snaha o zachování tvaru svalu v jeho horní části, kde je sval jaksi vypružován a užší. Na posledním snímku 45 je vidět, že deformace s metodami Discregrid a FCL probíhá až na malé detaily velmi podobně a vhodně. Naopak sval deformovaný spolu s řešením kolizí metodou voxelizace již úplně prochází hlavou kosti *femur*.

11.3.2 Časová náročnost

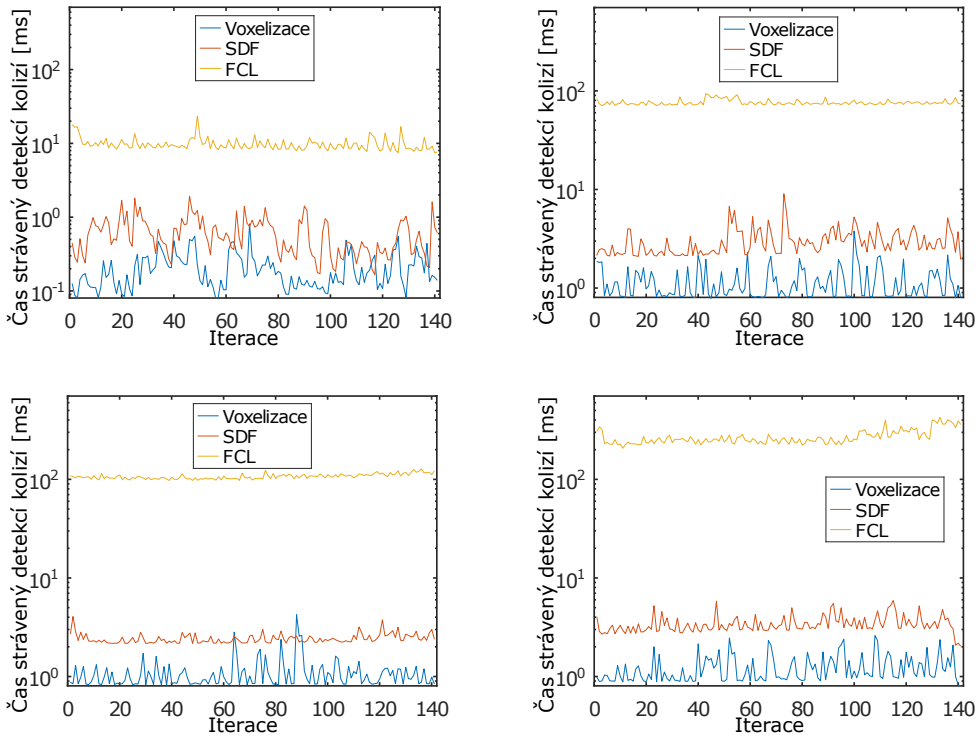
Test časové náročnosti jednotlivých metod proběhne experimentálním způsobem. Budou testovány všechny dostupné modely svalů *adductor brevis*, *gluteus medius*, *gluteus maximus* a *iliacus*, každý vzhledem ke všem metodám pro detekci a řešení kolizí tedy voxelizaci, Discregrid a FCL. Časy řešení kolizí nebudou promítnuty, neboť se napříč metodami zásadním způsobem neliší.

Simulace bude spuštěna s měřením času detekce kolizí danou metodou v každé iteraci simulace. Testování proběhne dvěma způsoby, a to nejprve se **třemi** iteracemi za snímek, následně bude proveden test s **tisícem** iterací za snímek. Snímků bude opět vždy 47. Výstupem všech testování budou grafy porovnání výsledných časů pro dané metody. Výsledky proběhlých testů se třemi iteracemi za snímek jsou vizualizovány v grafech na obrázku 11.10.

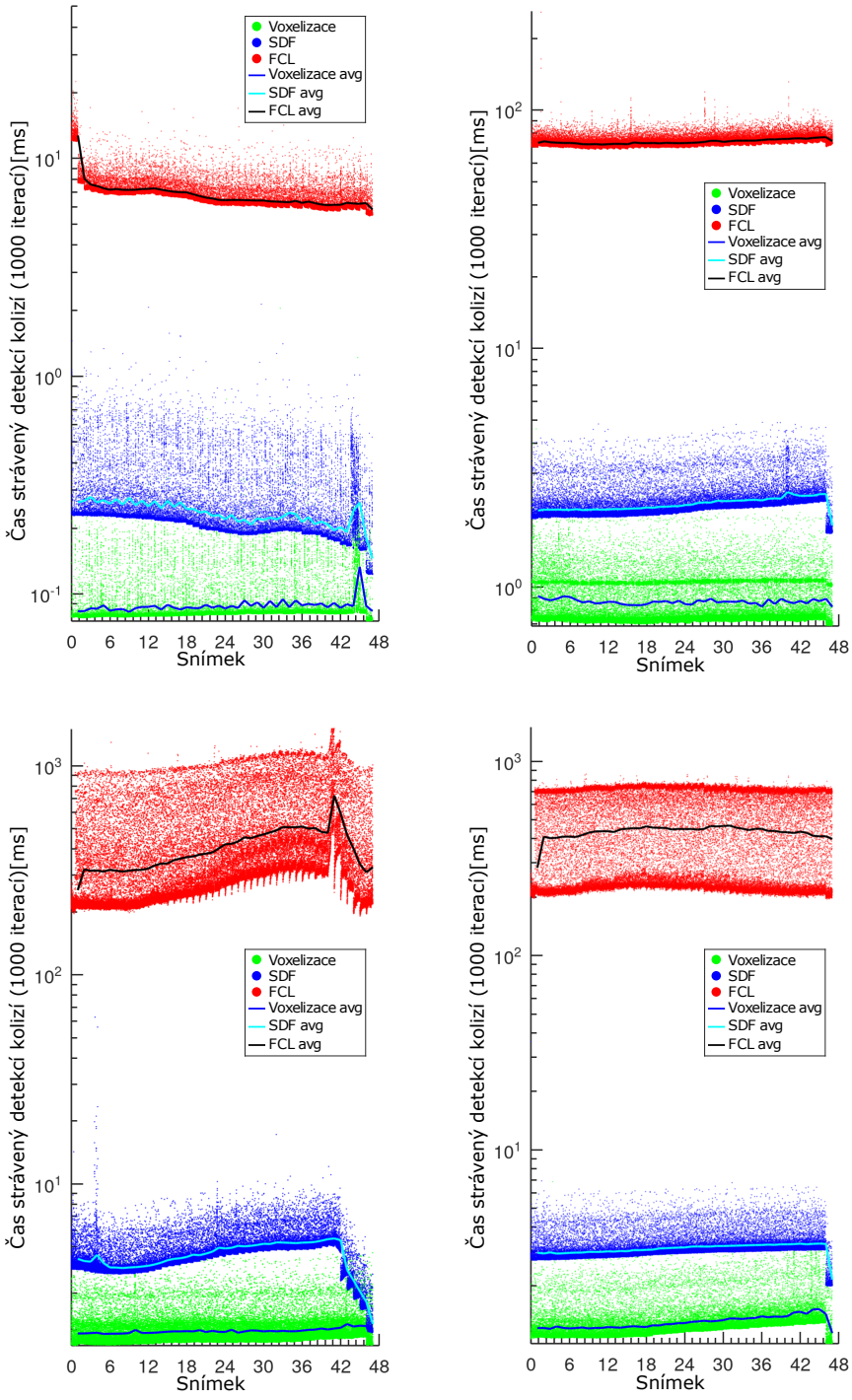
Z uvedených výsledných grafů lze vypozorovat následující seřazení metod podle rychlosti výpočtu (od nejrychlejší po nejpomalejší):

1. voxelizace
2. knihovna Discregrid (SDF)
3. knihovna FCL

Ze všech grafů vyčnívá v hodnotách právě první, pro sval *adductor brevis*, neboť se při simulaci deformace tohoto svalu velké množství kolizí neděje. Ve zbylých třech grafech, zatímco se metody voxelizace a Discregrid v časové efektivitě drží alespoň v řádové blízkosti (přibližně 10 ms na iteraci), metoda detekce kolizí knihovnou FCL je podstatně pomalejší (přibližně 200 ms na iteraci). Tento marginální rozdíl je potvrzen a zdůrazněn výsledky testů s tisícem iterací za snímek grafech na obrázku 11.11.



Obrázek 11.10: Grafy porovnání metod pro časy detekci kolizí napříč třemi metodami pro svaly *adductor brevis* (první řádka vlevo), *gluteus medius* (první řádka vpravo), *gluteus maximus* (druhá řádka vlevo) a *iliacus* (druhá řádka vpravo). Každý graf znázorňuje pro jednotlivé iterace časy strávené detekcí kolizí v milisekundách. Celkově v každém grafu vystupuje 141 iterací, tedy 47 snímků × 3 iterace.



Obrázek 11.11: Grafy porovnání metod pro časy detekci kolizí v milisekundách napříč třemi metodami pro svaly *adductor brevis* (první řádka vlevo), *gluteus medius* (první řádka vpravo), *gluteus maximus* (druhá řádka vlevo) a *iliacus* (druhá řádka vpravo). Celkově v každém grafu vystupuje 47000 iterací, tedy 47 snímků × 1000 iterací. Pro lepší porovnání hodnot jsou přidány křivky průměrů v jednotlivých snímcích.

11.3.3 Zachování objemu

Testování zachování objemu bylo provedeno pomocí existující metody ve stávajícím řešení M. Červenky [4] nad všemi dostupnými modely svalů. Použitá metoda funguje tak, že z každého trojúhelníků na povrchu tělesa sestrojí čtyřstěn s bodem v počátku souřadného systému a vypočítá jeho objem. Objemy se přes čtyřstěny sečtou a jelikož do součtu objemy nevstupují v absolutní hodnotě, objemy trojúhelníků, jejichž normála je vzhledem k rovině trojúhelníku na stejné straně jako počátek, mají zápornou hodnotu. Naopak pokud je počátek na druhé straně než normála, má objem kladnou hodnotu.

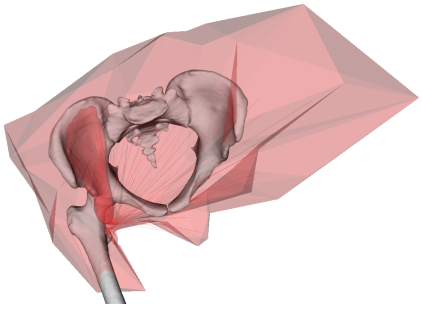
Objem byl pro všechny metody a všechny svaly napříč všemi snímky konzistentní. Výsledné a zároveň počáteční hodnoty uvedené pouze pro modely svalů jsou vypsané v tabulce 11.1. Omezení zachování objemu tedy přetrvává pro všechny metody.

Model svalu	Objem přes všechny metody pro všechny snímky $[cm^3]$
Gluteus medius	271.01600
Gluteus maximus	746.5630
Iliacus	102.3250
Adductor brevis	54.6708

Tabulka 11.1: Tabulka objemů naměřených pro dané modely svalů v kubických centimetrech.

11.4 Poznatky

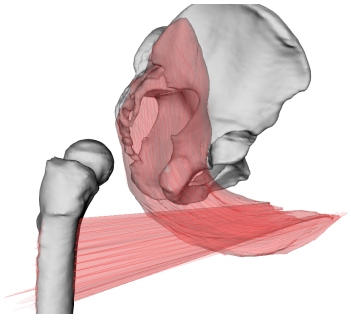
Pro integraci knihovny FCL nebyly použity navrhované kolizní vektory, neboť jejich testování produkovalo nerealistické řešení kolizí. Tato chyba je znázorněna na obrázku 11.12, kde sval *iliacus* obaluje celou kost pánve. Částečné řešení tohoto problému by mohlo být omezení vzdálenosti korekce pozice bodu na nějakou rozumnou hodnotu, avšak stále zůstává problém nesprávnosti směrů těchto vektorů.



Obrázek 11.12: Nerealistické roztáhnutí svalu *iliacus* za použití kolizních vektorů knihovny FCL.

V kontextu knihovny FCL se v průběhu implementace naskytl problém při snaze o využití BVH stromů s primitivy AABB kvádrů. Zatímco by tento přístup podle dokumentace knihovny měl být pro kolize elastických těles časově nejefektivnější, je toto zrychlení provedeno nejspíše velkou mírou paralelizace a optimalizace např. interní reprezentace těles v manažerovi. To nejenže znemožňuje konzistenci interních reprezentací těles v manažerovi a reprezentací těles v metodě PBD bez zavedení větší režie, ale i produkuje výjimky různých druhů, nejspíše kvůli neošetřeným kritickým sekcím knihovny. Knihovna FCL ovšem vykazuje dobré vizuální výsledky, a tak by mohlo být vhodné nekonzistenci dat a výjimky ošetřit a tím dosáhnout vyšší výpočetní rychlosti. K tomu v rámci této práce bohužel nezbyl čas.

Dalším problémem, který je významný zejména pro metodu testování je chyba testovacího prostředí, nejspíše některého z vizualizačních nástrojů, vyskytující se ve všech simulačních scénářích vyobrazena na obrázku 12.13, kde v posledním snímku 47 dochází k transformaci stehenní kosti mimo kyčelní kloub. Tento snímek proto zkresluje konzistentně všechny testovací scénáře (jak je zřetelně vidět např. na obrázku 11.11 v grafech svalů *gluteus maximus* a *iliacus*), naštěstí ovšem pouze v posledním kroku simulace.



Obrázek 11.13: Chyba posledního snímku simulačních scénářů, kde je kost stehenní transformována mimo kyčelní kloub.

Vhodným rozšířením by mohlo býti zavést detekci kolizí přes všechny kosti i pro metodu voxelizace. Bylo by možné místo postupné aplikace kolizních omezení pro každý bod v kolizi s více kostmi tato omezení sečíst (tj. přičíst k pozici bodu vektor, který danou kolizi řeší).

11.4.1 Shrnutí výsledků testování

Z vizuálního porovnání deformací vyplývá, že metody Discregrid i FCL problémy stávající metody voxelizace řeší v plné míře. Z pohledu časové efektivity ovšem knihovna FCL postrádá, a tak se jako nejvhodnější metodou zdá zatím býti detekce a řešení kolizí pomocí knihovny Discregrid. V oblasti zachování objemů těles dosahují všechny tři metody stejného, správného výsledku.

12 Závěr

V práci došlo k detailnímu popisu a analýze pěti dostupných metod, řešících problém rychlé detekce a řešení kolizí ve 3D prostoru. Z nich byly vybrány veřejné implementace v podobně knihoven Discregrid, která používá metodu Distance fields, a FCL, založené na metodě BHV stromů. Tyto implementace byly nezávisle na sobě integrovány do stávajícího řešení M. Červenky 2019 [4] a otestovány na simulacích deformace čtyř různých svalů kolidujících s kostmi při pohybu jedné z nich. Metoda detekce a řešení kolizí pomocí integrace knihovny Discregrid obstála ve všech testovaných ohledech tj. řešení kolizí je přesné, metoda je z pohledu časové náročnosti srovnatelná s původním řešením v práci M. Červenky 2019 [4] a zachovává objemy modelovaných svalů.

K výsledné metodě byla navíc přidána vlastnost detekce kolizí vzhledem ke všem kostem, která se dá aplikovat i na stávající řešení, popř. jiné metody detekce kolizí. Touto vlastností oplývá i druhá integrovaná metoda, která stejně jako první metoda vykázala během testování přesnou deformaci svalů, ovšem za cenu nízké časové efektivity. Nicméně i tato druhá metoda nadále skýtá potenciál uplatnění v kontextu deformace svalů metodou PBD za podmínky optimalizace výpočetního času.

Nakonec shledávám zadání práce za splněné, neboť byla nalezena požadovaná metoda poskytující přesnou a rychlou detekci kolizí v kontextu deformace svalů metodou PBD.

13 Přehled zkratek

- **PBD** – Position Based Dynamics
- **FEM** – Finite Element Method
- **FVM** – Finite Volume Method
- **MSS** – Mass Spring System
- **BSP** – Binary Space Partitioning
- **BVH** – Bounding Volume Hierarchy
- **AABB** – Axis-Aligned Bounding Box
- **SDF** – Signed Distance Field
- **LDI** – Layered Depth Images
- **VTK** – Visualization Toolkit
- **OBB** – Oriented Bounding Box
- **BSD** – Berkeley Software Distribution
- **BSH** – Bounding Sphere Hierarchy
- **FCL** – Flexible Collision Library
- **SoP** – Sweep and Prune
- **kDOP** – Discrete Oriented Polytope
- **GPU** – Graphical Processing Unit
- **OBBRSS** – Oriented Bounding Box Rectangle Swept Sphere
- **OBJ** – 3D Wavefront objekt File
- **XML** – Extensible Markup Language
- **CPU** – Central Processing Unit
- **GHz** – gigahertz
- **ms** – milisekunda
- **cm^3** – centimetr krychlový

Literatura

[1] Bender, J. – Müller, M. – Macklin, M. A Survey on Position Based Dynamics, 2017. In *Proceedings of the European Association for Computer Graphics: Tutorials*, EG '17, Goslar, DEU, 2017. Eurographics Association. doi: 10.2312/egt.20171034. Dostupné z: https://doi.org/10.2312/egt.20171034.

[2] Berndt, I. – Torchelsen, R. – Maciel, A. Efficient Surgical Cutting with Position-Based Dynamics. *IEEE Comput. Graph. Appl.* May 2017, 38, 3, s. 24–31. ISSN 0272-1716. doi: 10.1109/MCG.2017.45. Dostupné z: https://doi.org/10.1109/MCG.2017.45.

[3] Cutler, B. *Rigid Body Dynamics, Collision Response, & Deformation* [online]. Rensselaer Polytechnic Institute, 2017. [cit. 2020/16/10]. Dostupné z: https://www.cs.rpi.edu/~cutler/classes/advancedgraphics/S15/lectures/08_deformation.pdf.

[4] Červenka, M. Deformace svalových vláken systémem vázaných částic. Plzeň, 2019. 62s. Diplomová práce na Fakultě aplikovaných věd Západočeské univerzity v Plzni na katedře informatiky. Vedoucí diplomové práce Josef Kohout.

[5] Červenka, M. – Kohout, J. Fast and Realistic Approach to Virtual Muscle Deformation. In:BIOSTEC 2020. Sebútal: ScitePress, 2020. s. 217-227. ISBN 978-989-758-398-8, ISSN 2184-4305.

[6] Faure, F. et al. SOFA: A Multi-Model Framework for Interactive Physical Simulation. In Payan, Y. (Ed.) *Soft Tissue Biomechanical Modeling for Computer Assisted Surgery*, 11 / *Studies in Mechanobiology, Tissue Engineering and Biomaterials*. Springer, June 2012. s. 283–321. doi: 10.1007/8415_2012_125. Dostupné z: https://hal.inria.fr/hal-00681539.

[7] Fontes, E. *FEM vs. FVM* [online]. COMSOL INC., Nov 2018. [cit. 2021/05/05]. Dostupné z: https://www.comsol.com/blogs/fem-vs-fvm/.

[8] Klein, J. – Zachmann, G. Time-Critical Collision Detection Using an Average-Case Approach. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, VRST '03, s. 22–31, New York, NY, USA, 2003. Association for Computing Machinery. doi: 10.1145/1008653.1008660.

Dostupné z: https://doi.org/10.1145/1008653.1008660. ISBN 1581135696.

[9] Macklin, M. et al. Unified Particle Physics for Real-Time Applications. *ACM Trans. Graph.* July 2014, 33, 4. ISSN 0730-0301. doi: 10.1145/2601097.2601152. Dostupné z: https://doi.org/10.1145/2601097.2601152.

[10] Mesit, J. Modeling And Simulation Of Soft Bodies. Electronic Theses and Dissertations, 2004-2019. 1647. Orlando, 2010. 165s. Disertační práce na Fakultě inženýrství a počítačových věd Univerzity střední Florida na katedře elektrického inženýrství a počítačové vědy. Školitel disertační práce Ratan Kumar Guha.

[11] Pan, J. – Chitta, S. – Manocha, D. FCL: A general purpose library for collision and proximity queries. In *2012 IEEE International Conference on Robotics and Automation*, s. 3859–3866, 2012. doi: 10.1109/ICRA.2012.6225337.

[12] Romeo, M. – Monteagudo, C. – Sánchez-Quirós, D. Muscle Simulation with Extended Position Based Dynamics. In *Proceedings of the XXVIII Spanish Computer Graphics Conference*, CEIG '18, s. 1–10, Goslar, DEU, 2018. Eurographics Association. doi: 10.2312/ceig.20181146. Dostupné z: https://doi.org/10.2312/ceig.20181146.

[13] Shellshear, E. 1D sweep-and-prune self-collision detection for deforming cables. *The Visual Computer*. 05 2014, 30, s. 553–564. doi: 10.1007/s00371-013-0880-7.

[14] Suriyakumar, V. et al. Open-source software for collision detection in external beam radiation therapy. s. 101351G, 03 2017. doi: 10.1117/12.2255644.

[15] Teschner, M. et al. Collision Detection for Deformable Objects. *Computer Graphics Forum*. 2005. Dostupné z: https://hal.inria.fr/inria-00394479.

Přílohy

Uživatelská příručka

Součástí práce je jednoduchý Windows Command Script (skript příkazového řádku operačního systému Windows) s názvem **RunCollisionSimulation.cmd**. Tento skript očekává sestavené všechny potřebné knihovny a kompletní množinu testovacích souborů. Obě tyto nutnosti jsou dodány jako součást skriptu. Jak již z názvu vyplývá, jedná se o plugin, a to do projektu OpenSim 4.0.

Skript pluginu je spustitelný pouhým rozkliknutím a na úvod žádá uživatele o specifikaci metody detekce kolize, která bude pro simulaci spuštěna. Je na výběr ze tří implementovaných detekcí kolizí a to voxelizace, Discregridu a FCL. Uživatel je tázán na metodu do té doby, než zadá jednu z iniciál metod. Následně má uživatel na výběr z pěti scénářů simulace svalů *gluteus medius* společně se svalem *gluteus maximus*, svalů *gluteus medius* a *gluteus maximus* zvlášť, svalu *adductor brevis* a svalu *iliacus*. Opět je uživatel tázán na svaly k simulaci do doby, než zadá jeden z identifikátorů těchto svalů. Následně se spustí samotná simulace.

Plugin je možné spouštět dokola s různými svaly a metodami, zatímco není třeba projekt opětovně sestavovat. Slouží k ukázce chování všech metod.