Systém pro výuku hry na bicí s využitím inverzní kinematiky.

Diplomová práce

Martin Linda

Fakulta elektrotechnická
České vysoké učení technické v Praze
Katedra počítačů

Praha 2005

Anotace

Tématem mé diplomové práce bylo vytvoření výukového a vizualizačního nástroje pro hru na bicí. Při implementaci animační části byly využity principy inverzní kinematiky.

V tomto dokumentu je rozebírán účel takové vizualizace. Výraznou měrou se zde budeme věnovat principům a problémům animace virtuálního humanoida, zvláště pak použitému modulu inverzní kinematiky. Velká pozornost bude věnována také návrhu uživatelského rozhraní pro sestavení libovolné sady bicích a adaptaci programu na ni.

Abstract

The goal of the diploma work was to create The drumset tutorial system. There were used principles of inverse kinematics in the implementation part of this project.

In This document we bring the purpose of such vizualization. We will discuss the principles and problems of animation of virtual humanoid, especially the inverse kinematics module. We will specialize also in user interface for creatig arbitrary drumset and in the adaptation of the aplication to the drumset.

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady uvedené v kapitole reference.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu § 60 Zákona č.121/2000 Sb. , o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 27. ledna 2005

Obsah

1. Úvod

V této kapitole bude čtenář seznámen s motivací k vytvoření diplomové práce na dané téma a s obsahem tohoto dokumentu

1.1 Motivace

Základní myšlenka tohoto projektu vznikla v roce 2003, kdy jsem hledal téma semestrální práce z předmětu vizualizace. Ve svých mimostudijních aktivitách se ve velké míře věnuji hudbě a obzvláště hře na bicí. Mnozí bubeníci často zjistí, že když chtějí jinému bubeníkovi popsat rytmus, který má zahrát, je to slovně problematické, často až nemožné a ve většině případů by bylo potřeba předvést, jak tento rytmus zabubnovat. Podobný problém ovšem nastává v situaci, kdy bubeník nějaký rytmus slyší a chce se jej naučit, případně, pokud chce nějaký nový rytmus vymyslet. Proto se zde naskytla myšlenka programu, ve kterém by bubeník mohl zadávat noty a grafický 3D model bubeníka by rytmus následně vizualizoval. Tato úloha byla nakonec zrealizována jako výše zmíněná semestrální práce.

Rozšíření úlohy do podoby diplomové práce bylo následně koncipováno jako výukový nástroj pro hru na bicí s maximální možností přizpůsobení se konkrétnímu bubeníkovi. To se týká zejména možnosti kompletního sestavení vlastní sady bicích, na kterou má být rytmus přehrán a to včetně přiřazení vlastních zvuků jednotlivým bubnům a možnosti ovlivnění techniky hry. Značný důraz měl být kladen i na kvalitu vizualizace. Rozhodl jsem se implementovat model a animaci celé postavy a použít komplexní algoritmy inverzní kinematiky pro výpočet pohybu v reálném čase.

Tento projekt jsem se snažil pojmout jako samostatně fungující a v praxi použitelný nástroj a soustředil jsem se hlavně na praktické provedení úlohy. K tomu jsem byl nucen ostatně i tím, že v dané oblasti v podstatě neexistuje teoretický základ, s výjimkou algoritmů inverzní kinematiky použitých pro animaci, o který bych se mohl opírat. Většina problémů a jejich řešení je rozebírána až v kapitole 4 zabývající se konkrétním řešením úlohy. V některých oblastech vývoje jsem proto narazil na slepé uličky a v jiných zase na velké možnosti dalšího výzkumu.

1.2 Organizace dokumentu

Dokument je rozdělen do sedmi kapitol a tří dodatků a jsou v něm dvě hlavní části. První se zabývá teoretickým pozadím bubnování, jeho výukou a stávajícími technikami animace v této oblasti využitelnými. Druhá část pak pojednává o našem řešení úlohy a jeho implementaci.

Obsah této kapitoly byl již popsán výše. Druhá kapitola přibližuje čtenáři problematiku hry na bicí a ve třetí kapitole jsou popsány použitelné způsoby animace postavy a požadavky na pohybový model. Tím je uzavřena teoretická část.

Čtvrtá a v celém dokumentu zřejmě stěžejní kapitola rozebírá náš přístup k problému a popisuje zvolené postupy pro řešení dílčích částí úlohy. Pátá kapitola popisuje samotnou implementaci a kapitola šestá popisuje a hodnotí testování aplikace.

Poslední kapitola obsahuje závěr a shrnutí možné práce do budoucna. Následují tři dodatky, z nichž první popisuje instalaci programu a jeho ovládání, druhý popisuje organizaci zdrojového kódu a třetí popisuje obsah CD-ROM přiloženého k této práci.

2. Úvod do problematiky

Tato kapitola obsahuje stručný popis hry na bicí, řeší variabilitu složení sady bicích a smysl vizualizace hry na bicí pro její výuku.

2.1 Hra na bicí

Hra na bicí nástroje je velmi osobitá hudební disciplína a rozhodně patří k těm složitějším. laik by mohl namítnout, že při bubnování musí člověk ovládat pouze čtyři kanály řízení – dvě ruce a dvě nohy, zatímco například při hře na kytaru deset – deset prstů na rukou. Nicméně hlavní obtížnost hry na bicí spočívá v nutnosti používat všechny čtyři končetiny naprosto nezávisle a ve správnou chvíli provést správný pohyb s každou z nich. Avšak člověk má akce rukou a nohou výrazně propojeny, zatímco prsty na rukou dokážeme vcelku bez problémů ovládat nezávisle. A právě pro toto uvolnění končetin je zapotřebí značná dávka cviku i talentu.

Tato aplikace řeší dva základní aspekty hry na bicí. V praxi musí bubeník jednotlivé noty či sluchové vjemy úderů převést do pohybu danou končetinou. Při tom je nutné provést úder ve správnou chvíli. U rukou je navíc potřeba vyřešit, kterou rukou, do kterého bubnu udeříme.

2.2 Různé styly a technika hry

Zásadním aspektem bubnování je také technika úderu do bubnu. Je rozdíl, pokud do bubnu udeříme koncem paličky či její hranou. Záleží na síle úderu a na tom, zda úder je veden spíše zápěstím, či spíše loktem. Rozdíl je nejen zvukový, ale má vliv i na další pohyb ruky. Na bicí se tak dá hrát různými styly. Liší se technika hry v rockové hudbě nebo v jazzové.

Tento aspekt techniky hry však není v této práci řešen. Pohybový model by si v takovém případě žádal značné rozšíření a poměrně rozsáhlý výzkum způsobu pohybu reálného bubeníka a jeho simulace, s důrazem na rozvinutí metod inverzní kinematiky a inverzní dynamiky.

2.3 Složení sady

Pro hru na bicí je samozřejmě důležité, z jakých bubnů se sada skládá. Jednotlivé sady se mohou lišit v druhu, počtu a umístění jednotlivých bubnů. Na složení sady pak závisí i způsob hry. Druh a umístění bubnů se projeví například na přiřazování úderů rukám. Na druhu a umístění bubnu totiž záleží, zda má bubeník tendenci do něj udeřit spíše levou nebo spíše pravou rukou. Špatné střídání rukou vzhledem k těmto prioritám by pak některé úseky činilo hůře zahratelné, či dokonce nemožné.

Každý bubeník má svou vlastní oblíbenou sestavu bicích. Z výše zmíněných důvodů je tedy důležité, aby to bylo v aplikaci zohledněno. V ideálním případě, by měl mít bubeník možnost nechat si rytmus předvést na modelu ekvivalentním své sadě. To je v této aplikaci plně umožněno tím, že je implementován editor pro sestavení a editaci vlastní sady bicích. Zbytek programu potom funguje adekvátně vzhledem ke složení sady.

2.4 Odlišnosti individuálních sad bicích

Klasická základní sada se obvykle skládá z následujících bubnů:

Velký buben, malý buben, dva kotle, velký kotel a činely *hi-hat*, *ride* a *crash*. Běžněji se však o nich hovoří jako o *kopáku* (velký buben), *virblu* (malý buben), *přechodech* (kotle), *kotli* (velký kotel) a názvy činelů se nepřekládají

Ilustrace 1: **Ilustrace vzhledu částí sady bicích.** Ilustrace jsou převzaty z [Orc04].

Tato sestava se může výrazně měnit v počtu i druhu bubnů. Počet přechodů se pohybuje od jednoho až do pěti v různých velikostech. Počet činelů může být prakticky neomezený, přidávají se k nim činely typu *splash*, *china* a dalších typů, případně zvonce a další perkusní nástroje. Někdy jsou v sadě dva virbly. Místo pedálu u kopáku pro jednu nohu, lze použít dvojpedál, tedy dva propojené pedály pro obě nohy, poháněj ící dvě paličky na jednom

kopáku. Jindy se použijí dva kopáky. A v neposlední řadě umístění bubnů pro leváka se zásadně liší od umístění pro praváka.

Ilustrace 2: **rozdílná složení sady bicích.** Vlevo klasická základní sestava, vpravo rozsáhlá sestava velmi pokročilého bubeníka. Ilustrace jsou převzaty z [Orc04].

2.5 Výuka hry na bicí a přínos vizualizace

Jak již bylo zmíněno v první kapitole v odstavci o motivaci práce, v praxi jsem došel k závěru, že bubeník se rytmus snáze naučí, nebo jej lépe analyzuje a pochopí, pokud vidí přímo pohyby, které je nutné vykonat pro zahrání daného rytmu. V opačném případě musí analyzovat jednotlivé údery, ze kterých je rytmus složen a vyřešit, kdy je má provést a jakou rukou. To může být v některých případech poměrně složité a zejména pro začátečníky to může znamenat problém. Naše úloha tento převod zajišťuje.

Pro splnění tohoto účelu je potřeba, aby animace bubeníka co nejvíce odpovídala reálnému pohybu a aby algoritmus přiřazení úderů rukám rovněž odpovídal realitě. Navíc je nutné, aby obě tyto části aplikace byly adaptabilní na složení sady.

3. Vizualizace pohybu - techniky animace

V této kapitole se seznámíme se stávajícími způsoby animace, s jejich výhodami a nevýhodami a s důvody pro výběr konkrétního řešení a jeho principy.

3.1 Druhy animace, jejich výhody a nevýhody

3.1.1 Keyframe animation

Keyframe animation je technika, kdy jsou zaznamenána takzvaná klíčová okna animace a stav animovaného objektu mezi těmito okny se vypočítá interpolací, případně pomocí nějaké obecnější funkce. Pro postavu bubeníka tak stačí zaznamenat natočení kloubů pro pozici ruky v okamžiku úderu do bubnu a v pozici klidové. Pomocí interpolace mezi dvěma natočeními dostaneme požadovaný pohyb ruky. Přechod mezi dvěma klíčovými okny lze potom modulovat tak, aby výsledný pohyb ruky nebyl lineární, ale aby reálněji simuloval pohyby skutečného bubeníka.

Výhodou této techniky je její poměrně nízká implementační náročnost. K jejímu použití není potřeba znát žádné pokročilé technologie pro simulaci kinematiky. Tato technika je i výpočetně nenáročná.

Nevýhodou tohoto přístupu je, že klíčová okna je potřeba zadat ručně a to pro každý buben a každou jeho pozici. Pokud chce uživatel sadu upravovat, musí pokaždé zadat znovu i klíčová okna, pro každý možný úder, tedy pro každou ruku na každém bubnu. Navíc, nalezení takové přechodové funkce, aby výsledný pohyb vypadal realisticky, nemusí být vždy jednoduché.

3.1.2 Motion capture

Tato technika umožňuje vytvořit velmi věrnou rekonstrukci pohybu. Je založena na záznamu reálného pohybu člověka a jeho následné aplikaci na modelu. Při záznamu jsou s určitou frekvencí ukládány hodnoty polohy z pohybových čidel umístěných na člověku, který předvádí požadovanou akci, například úder paličkou do bubnu. Při rekonstrukci pohybu jsou pak tyto hodnoty převedeny do rotací kloubů modelu. Je tak tedy realisticky rekonstruována každá fáze pohybu.

Výhodou této metody je vysoká realističnost rekonstruovaného pohybu. Výpočetní nároky jsou rovněž malé.

Podstatnou nevýhodou je však to, že zaznamenané sekvence nelze jednoduše měnit a tak je velmi obtížné přizpůsobit pohyb různým složením sady jinak, než opětovným záznamem každého pohybu. Navíc zařízení pro záznam pohybu je velmi drahé a pro amatérské použití v podstatě nedostupné.

3.1.3 Inverzní kinematika

Kinematika je proces, kdy počítáme pozici konce struktury skládající se z kloubů a segmentů tyto klouby propojujících, přičemž známe všechny úhly natočení v kloubech. Pro takové zadání existuje pouze jedno řešení. Inverzní kinematika řeší opačný problém. při znalosti cílového bodu hledáme úhly natočení kloubů takové, aby konec struktury tohoto bodu dosáhl. To může být obtížné a obvykle existuje mnoho nebo nekonečně mnoho řešení.

Tato technika však umožňuje výpočet realistické animace pohyb kostry v reálném čase. Při zadaném koncovém efektoru – kloubu kostry, který se má přiblížit k cílové pozici a zadané základně pohybu – kloubu kostry, kde začíná řetězec kloubů podílejících se na přiblížení, vypočítá natočení kloubů v řetězci tak, aby se pozice a případně i orientace koncového efektoru co nejvíce blížila k zadanému cílovému bodu. Případně můžeme požadovat i určité natočení koncového efektoru. Metoda umožňuje zahrnout do výpočtu například i omezení pohybu v kloubech a jejich tuhosti nebo gravitaci. Výsledný pohyb pak může být při správném nastavení těchto parametrů velmi realistický.

Výhodou této metody je fakt, že je nezávislá na složení sady bicích. Pohybovému modelu stačí, když zná místo, kam má palička udeřit a přiblížení se pak vypočítá v reálném čase s ohledem na konkrétní rozestavení bubnů i polohu postavy.

Nevýhodou metody jsou jednoznačně velké výpočetní nároky. Pro každý zobrazovaný snímek animace, se musí provádět velký počet složitých geometrických výpočtů, aby se klouby dostaly do požadovaných poloh. Jako velmi podstatná a v počátku této práce neočekávaná nevýhoda, se ukázala skutečnost, že inverzní kinematika sice zohledňuje anatomická omezení pohybu postavy pomocí omezení a tuhostí v kloubech, avšak pro realistickou simulaci specifického pohybu, jakým bubnování je, bylo potřeba metodu poněkud upravit, jak bude rozebráno dále.

3.2 Adaptabilita pohybového modelu

Jak již bylo zmíněno, program je koncipován tak, že aby se přizpůsoboval uživatelským požadavkům na složení sady. To znamená nejen umožnit skládání a editaci sady bicích, ale i následné přizpůsobení pohybu modelu bubeníka její konfiguraci. Například, ruce se budou hýbat do jiných míst po jiných drahách a také se budou střídat jiným způsobem na sadě s jedním činelem umístěným vlevo od bubeníka a jinak na sadě se dvěma činely umístěnými vpravo.

Tomuto požadavku z výše zmíněných animačních technik nejlépe vyhovuje inverzní kinematika a tuto techniku jsem také ve své práci použil.

4. Návrh řešení

V této kapitole se hlouběji podíváme na požadavky a záměry dané tématem úlohy a rozebereme jejich konkrétní řešení použitá v implementaci úlohy. Hlavní oblasti, kterými se budeme zabývat jsou: editor sady bicích a editor skladeb, grafický model, pohybový model, plánovač pohybu a zvukový subsystém.

4.1 Záměr aplikace a uživatelské rozhraní

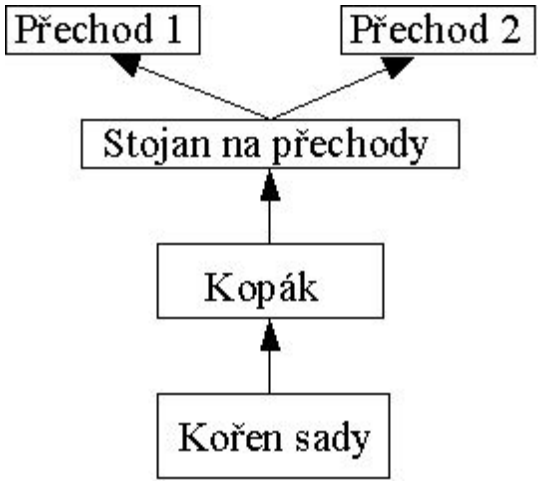
Aplikace je koncipována jako nástroj, který má uživateli pomoci naučit se, jak zabubnovat určitý rytmus. V úvodu byl vznesen požadavek, aby rytmus bylo možné vizualizovat na libovolné sadě bicích. To zajišťuje editor pro sestavení sady bicích. Rovněž je žádoucí, aby uživatel mohl rytmus snadno zadávat. K tomu je určen editor skladby. Sada bicích a následně i postava bubeníka, která požadovaný rytmus zabubnuje se zobrazuje ve vizualizačním okně, které je pro oba editory společné.

4.1.1 Editor sady bicích

Jak bylo řečeno v předchozích kapitolách, pro co nejlepší výukovou funkci programu je potřeba, aby model sady bicích v programu odpovídal sadě bicích, kterou uživatel ve skutečnosti používá. Z tohoto důvodu byl vytvořen editor pro sestavení a editaci sady bicích (dále editor sady). Při jeho návrhu byly implementovány různé funkce a omezení, které uživateli usnadní práci s tímto editorem tak, aby uživatel byl programem veden a aby mu v každém okamžiku byly umožňovány pouze smysluplné akce.

4.1.1.1 Hierarchie sady

Editor sady je komplexní nástroj pro vytváření a editaci sady bicích. Sada se skládá z jednotlivých částí - z bubnů (tímto pojmem zahrnuji i činely) a z částí stojanů. Sada je hierarchicky stromově uspořádána. Každá část může mít potomky, jejichž geometrické transformace jsou od ní odvozeny a jsou počítány relativně k jejím transformacím. Pokud tedy pohneme s částí sady, pohyb se projeví i na jejích potomcích.



Ilustrace 3: **Příklad hierarchie v sadě bicích.**

Velmi důležitým atributem u každého bubnu je preference úhozu, tj. zda úder do něj bude veden pravou či levou rukou. Z jeho nastavení pak plánovač pohybu určuje, která ruka

konkrétní úder v konkrétní chvíli provede. Přesný význam atributu a funkce plánovače pohybu budou diskutovány níže.

4.1.1.2 Hitpointy

U každého bubnu můžeme definovat takzvané *hitpointy*, tedy body, do kterých je možné vést úder. Každý hitpoint má polohu nastavitelnou relativně ke středu bubnu k němuž náleží. Dále je možné nastavit typ hitpointu - zda bude úder veden koncem paličky či její hranou, případně zda se jedná o pedál pro levou či pravou nohu a procento, kterým síla úderu ovlivňuje hlasitost přehrávaného zvuku. Na jednom bubnu lze definovat i více hitpointů, což může odrážet různé způsoby úderu do bubnu. Například údery do hrany činelu, nebo do jeho středu se výrazně liší. Každý hitpoint také může mít definovánu jednu nebo více úrovní dynamiky úderu. Ke každé z těchto úrovní lze nahrát zvuk a nastavit rozsah síly úderu. Při různých silách úderu do bubnu se totiž charakter zvuku mění a toto řešení zajišťuje větší realističnost než pouhé nastavení hlasitosti zvuku.

Ilustrace 4: **Umístění hitpointu na bubnu.** Hitpoint určuje místo, kam má udeřit palička.

4.1.2 Editor skladeb

Jak již název napovídá v této části aplikace lze zadávat skladbu (rytmus), kterou chceme přehrát. Rovněž umožňuje nahrání sady bicích, na kterou se má rytmus přehrát. A v neposlední řadě zajišťuje přehrávání skladby a s ním spojenou generaci animace bubeníka ve vizualizačním okně.

4.1.2.1 Hudební teorie – oblast metrorytmických vztahů

Na tomto miste považuji za vhodné uvést některé termíny z hudební teorie, které jsou relevantní pro mojí práci. Čerpal jsem z [Gra99].

Relativní délka tónu se udává různými druhy not. Základní poměr dvou následujících druhů not je 1:2. Je ale možné i jiné dělení, třeba na tři (trioly), na pět (kvintoly) i na skupiny jiné velikosti. Taková skupina také může zabírat různě velký prostor – je možné mít kvintolu namísto půlové nebo čtvrťové noty, nebo například místo tří not osminových. V lichých metrech se mohou naopak vyskytovat sudé skupiny, kdy například místo tří not jsou dvě (duola) nebo čtyři (kvartola). Kvartola může také nahradit tři nebo pět not v sudých metrech.

Absolutní délka intervalu je dána teprve tempem skladby. To se udává v počtu úderů za minutu – BPM (z angl. *beats per minute*), přičemž jeden úder představuje základní počítací

jednotku skladby udanou metrickým označením (například 120 čtvrťových intervalů za minutu).

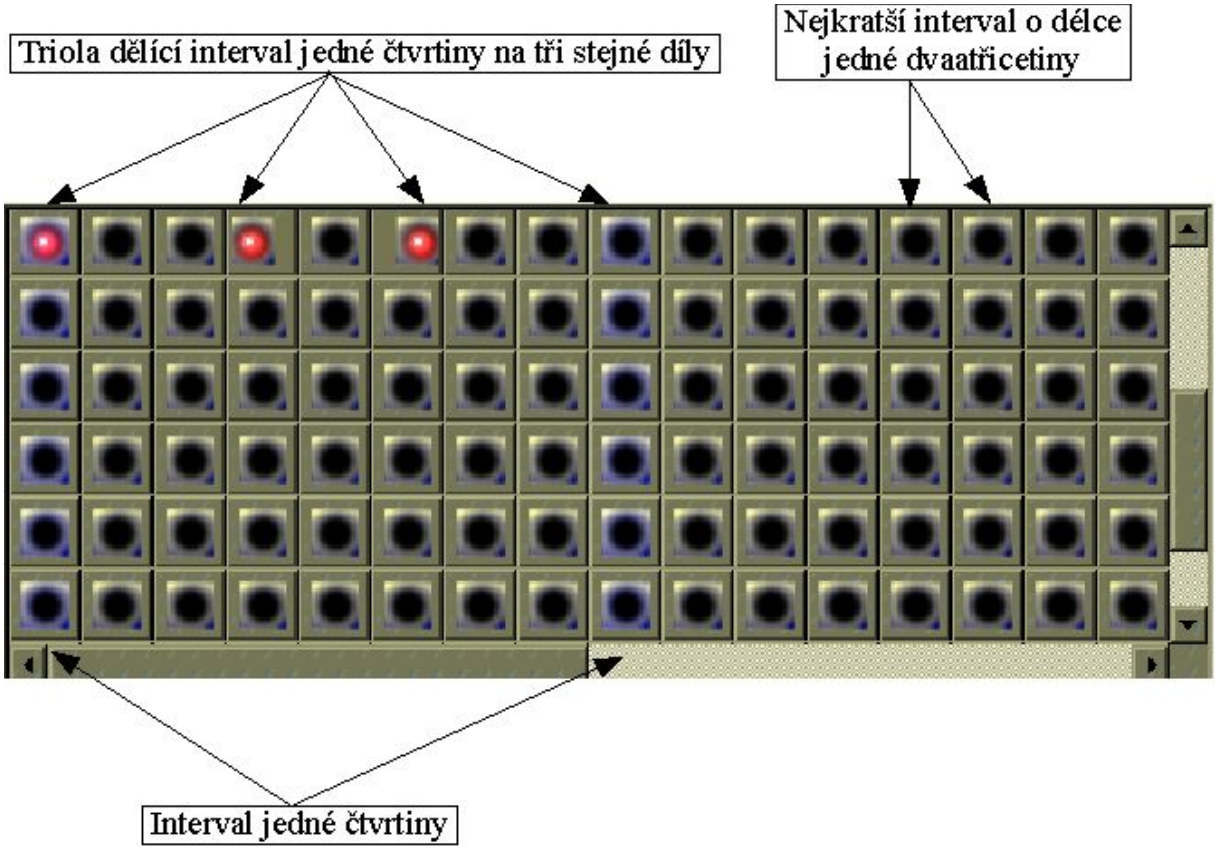
Metrické označení je základní měřítko, metrum, které udává velikost taktu. Uvádí se zlomkem, kde ve jmenovateli je základní počítací jednotka, v čitateli jejich počet v taktu. Metrum 4/4 tedy udává, že takt obsahuje 4 čtvrťové intervaly. Skladba může být celá v jednom metru, nebo se metrum může měnit, v extrémním případě v každém taktu.

Takt je organizační struktura skladby, grafické zobrazení určitého krátkého úseku hudby. Jeho délka je udána metrickým označením. Měřítkem taktu je metrum, jeho obsahem je rytmus.

4.1.2.2 Koncept skladby a notový záznam

V konceptu skladby v této aplikaci je zavedena kvantizace úderů (omezení intervalů mezi údery na celočíselný násobek základního nejkratšího intervalu) Jako nejkratší interval byla zvolena dvaatřicetinová nota, jelikož toto rozlišení se z hlediska rychlosti pohybů běžného bubeníka jeví jako dostatečné. Navíc je ještě možné úderu přiřadit posunutí o jednu třetinu základního intervalu vpřed nebo vzad. Toto posunutí nám pak umožní zadávat trioly, přičemž nejkratší interval nahrazený triolou je zde jedna osminová nota. Skladba je buď v metru 4/4 nebo 3/4 a její tempo se může pohybovat v rozmezí 1 až 200 BPM.

Skladba je rozdělena do homogenních bloků, jejichž délka může být maximálně jeden takt, minimálně jedna dvaatřicetina. Bloky jsou zobrazen jako mřížka, jejíž jeden sloupec reprezentuje jeden dvaatřicetinový interval. Mřížky všech bloků mají stejný počet řádků, kde každý řádek obsahuje údery pro jeden určitý hitpoint.



Ilustrace 5: **Mřížka skladby a její intervaly.** Ukazuje, jak triola dělí sudý interval na tři díly.

Každá skladba je vždy vztažena ke konkrétní sadě. Je tomu tak proto, že noty, které se zadávají, reprezentují údery do jednotlivých hitpointů. Skladba má tedy smysl vždy pro jednu konkrétní sadu.

V editoru skladby lze také spouštět a zastavovat přehrávání skladby. Skladbu lze přehrávat buď celou, nebo pouze jeden takt opakovaně. Při přehrávání skladby je ve vizualizačním okně generována animace bubeníka přehrávajícího zadaný rytmus.

4.1.2.3 Automatická omezení

Při vkládání not je třeba ošetřit, aby uživatel mohl zadat pouze smysluplný a ve skutečnosti zahratelný rytmus. Je zřejmé, že nelze zahrát více jak dva údery paličkami v jednom okamžiku. Dále se v praxi nevyskytuje situace, kdy by bubeník bouchal do dvou hitpointů na jednom bubnu najednou. Poslední omezení je pro činel hi-hat. Nemá totiž smysl, aby byl prováděn "úder" sešlápnutím pedálu činelu hi-hat a zároveň paličkou. Všechna tato omezení editor kontroluje a na jejich porušení je uživatel upozorněn výstražným signálem a není mu v takovém případě povoleno notu vložit.

4.1.3 Vizualizační okno

Toto je centrální okno aplikace a je společné pro editor sady a pro editor skladby. Je v něm zobrazován model sady, části sady jsou zobrazeny včetně materiálů a textur, ve scéně je nastaveno osvětlení. vetší realističnost obrazu a lepší prostorovou orientaci je sada umístěna do jednoduché místnosti.

V editoru sady se navíc zobrazují na bubnech hitpointy v editoru skladby je naopak zobrazena postava bubeníka posazená k bicím na stoličku. Při přehrávání skladby je počítána animace této postavy, která předvádí, jak zabubnovat daný rytmus.

V tomto okně je rovněž snímán pohyb myši, kterou lze ovládat kameru určující pohled na scénu a v editoru sady i části sady bicích.

4.1.4 Vzájemná komunikace částí systému

4.1.4.1 Přepínání režimů aplikace

S aplikací se dá pracovat ve dvou režimech. V režimu editoru sady bicích a v režimu editoru skladby. Mezi těmito režimy lze jednoduše přepínat a funkce a rozhraní aplikace se okamžitě změní. Sada bicích z jednoho režimu se při tom přenáší i do druhého. Respektive zůstává stále přítomna ve vizualizačním okně.

4.1.4.2 Automatická aktualizace

Tímto termínem rozumíme přizpůsobení editoru skladby při změně sady bicích. Pokud totiž změníme v editoru sady bicích její složení – přidáme či odebereme buben nebo nahrajeme novou sadu, tak se tomu automaticky přizpůsobí i editor skladby. Tedy počet řádků mřížky a jejich přiřazení jednotlivým hitpointům. Stejně tak, pokud v editoru skladby nahrajeme sadu, lze jednoduše přepnout do módu editoru sady bicích a tuto sadu editovat.

Ilustrace 6: **Automatické přizpůsobení aplikace.** Podle složení sady se nastaví počet řádků mřížky skladby a přiřadí se jednotlivým hitpointům v sadě.

4.2 Grafický model

Ve vizualizačním okně jsou zobrazovány jednak 3D modely bicích a stojanů a jednak 3D model postavy bubeníka. Toto vše je zasazeno do 3D scény reprezentující jednoduchou místnost, která uživateli pomáhá lépe se orientovat v prostoru.

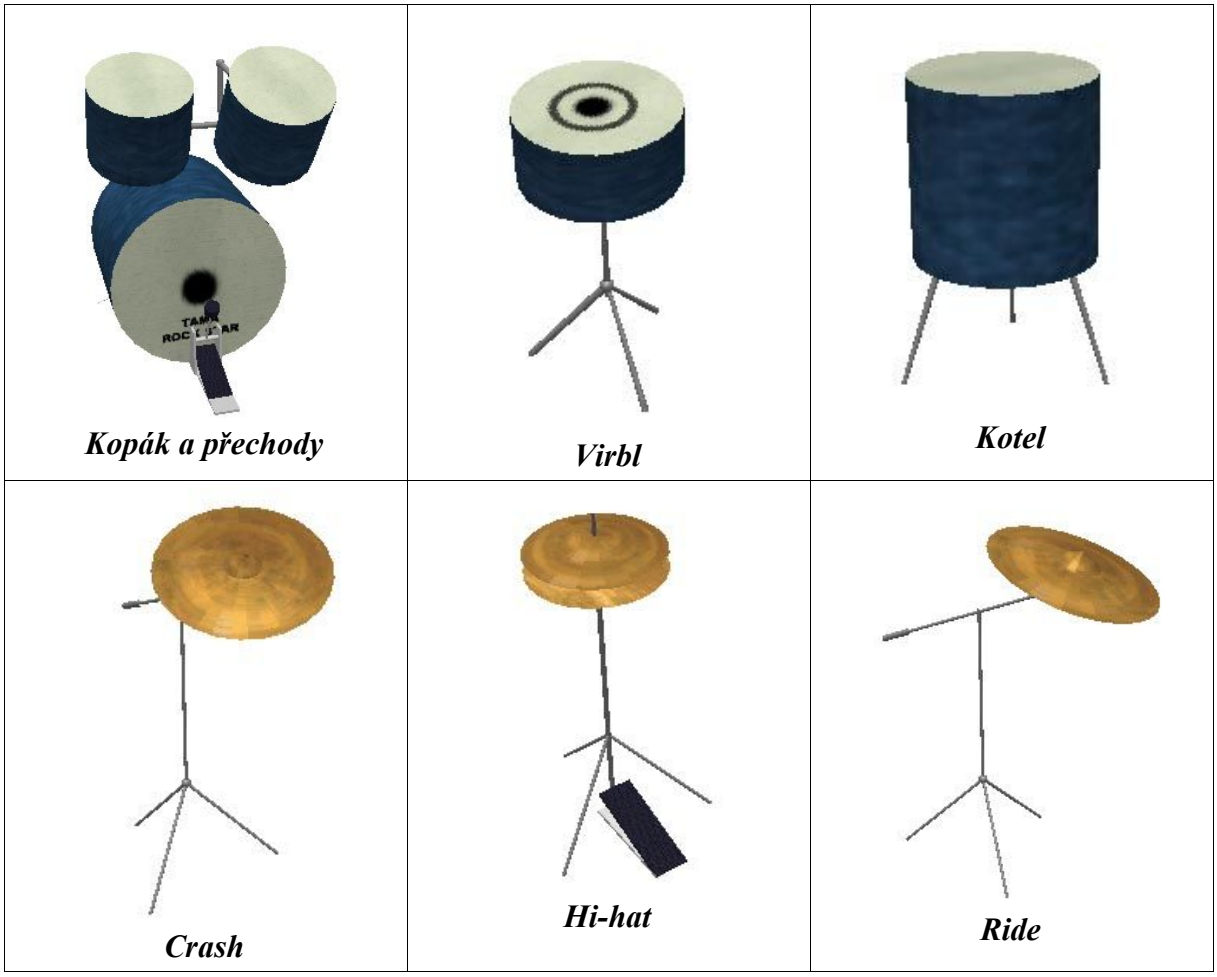
4.2.1 Modely bicích

4.2.1.1 3D model

Bubny mají většinou válcový, kuželovitý nebo diskovitý tvar. Proto i v této aplikaci jsou jejich modely vytvořeny z válců, disků a kuželů a jejich sjednocení. Pouze některé části, jako například pedál u kopáku, jsou vytvořeny z jednotlivých polygonů.

Rozměry bubnů, konkrétně průměry a hloubky odpovídají skutečným velikostem bubnů daných typů, pouze u vyklenutí činelů není simulace přesná. Detaily jako obroučky či přichytávací šrouby na obvodech bubnů jsou vynechány, jelikož nejsou pro simulaci podstatné.

Na povrch bubnů jsou aplikovány textury a jsou nastaveny materiály tak, aby výsledný dojem byl pokud možno realistický. Povrch činelů tak je výrazně lesklý s texturou bronzu, povrch blan umístěných na bubnech je matný s texturou plastu.



Ilustrace 7: Ukázky modelů částí sady použitých v aplikaci.

4.2.1.2 Pohybový model bicích

Pro realistický pocit z vizualizace sady je samozřejmě důležité, aby bubny reagovaly na energii úderu, tedy aby se po úderu rozhýbaly a to navíc adekvátně k jeho síle.

Některé z typů bubnů tedy mají přiřazený pohyb, který vykonají po úderu paličkou. Každý typ má tento pohyb jiný a samostatně definován. Virbl a kotel se po úderu zatřesou. Je to simulováno malou náhodnou rotací kolem os X a Z. Přechody se při úderu lehce sklopí a vzápětí se vrátí do původní polohy. Kopák se při sešlápnutí pedálu lehce zatřese. Zde je dobře vidět efekt hierarchie sady, jelikož přechody umístěné na kopáku se tím rozpohybují také.

4.2.2 Model postavy

Postava bubeníka je vystavěna podle standardu H-Anim [Han03], což je standard pro popis humanoida v jazyce VRML [Vrl97]. Odděluje kostru humanoida a jeho tělo a zavádí v kostře hierarchii kloubů a segmentů mezi nimi. Pro animaci postavy pak přistupujeme pouze k její kostře. Celý humanoid je složen z uzlů dvou typů z uzlů typu Joint a z uzlů typu Segment. Uzly typu Joint reprezentují klouby kostry a uzly typu Segment reprezentují části těla humanoida. Navíc je možné definovat uzel Site, který reprezentuje místo na těle humanoida, například špičku prstu nebo oko. Model humanoida je uložen ve VRML souborech a je uložena odděleně jeho kostra podle standardu H-Anim a 3D modely částí jeho těla. Tento model je v aplikaci načítán pomocí parseru standardu H-Anim a parseru 3D modelu. Více

informací o standardu H-Anim a VRML a načítání modelu humanoida se nachází v kapitole o implementaci.

4.3 Pohybový model

Pohybový model použitý pro animaci postavy je stěžejní částí této práce a zřejmě nejkomplikovanější oblastí implementace.

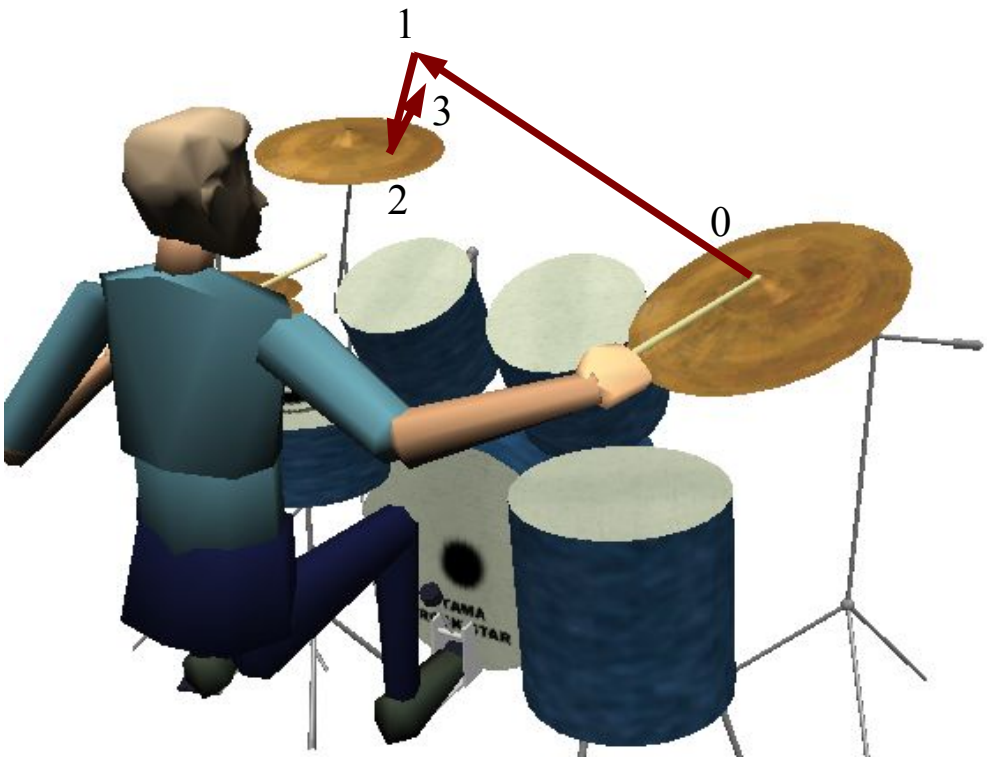
4.3.1 Charakteristické vlastnosti pohybu při bubnování

Jako v každém pohybu, lze i v bubnování vysledovat určité zákonitosti dané účelem tohoto pohybu. Například při chůzi jsou ruce svěšeny podél těla a jsou v rameni lehce rotovány v opačném smyslu než protilehlá noha. Při běhu se do pohybu již daleko výrazněji zapojují lokty a zápěstí. Při bubnování je situace poněkud složitější, jelikož hlavním účelem pohybu paže je zasáhnout koncem (případně jinou částí) paličky požadovaný bod na požadovaném bubnu. Správnost pohybu z hlediska techniky hry a vizuální přirozenost pohybu ovšem závisí významně na požadované síle a typu úderu. Navíc trajektorie jednotlivých kloubů ruky se liší v závislosti na začátku a cíli trajektorie celého úderu.

4.3.1.1 Trajektorie pohybu

Trajektorií pohybu rozumíme trajektorii pohybu koncového efektoru, tedy konce, případně středu paličky. Její začátek je vždy tam, kde se koncový efektor nachází v okamžiku zahájení úderu a její konec je v cílovém bodě určitého bubnu. To nám sice zajišťuje znázornění doteku paličky a bubnu, ale ještě to nevypadá příliš jako úder. Pokud by se totiž palička pouze dotkla bubnu a zůstala tak, vypadalo by to samozřejmě velmi nepřirozeně. Je proto potřeba simulovat ještě „odskočení“ paličky od bubnu. Tento odraz paličky lze simulovat například tak, že po dosáhnutí cílového místa na bubnu, toto cílové místo přesuneme zhruba 20 cm nad buben a asi 5 cm zpět k postavě bubeníka.

I tak je ještě trajektorie příliš plochá a nevystihuje správně reálný pohyb. Jako vylepšení se nabízí umístit nejprve cíl kousek (např. 15 cm) nad buben a v momentě úderu povolit pohyb pouze v zápěstí, což spolu s posunutím cíle do požadovaného bodu na bubnu a zpět znázorní švih zápěstím v této fázi úderu při pohybu ruky reálného bubeníka.



Ilustrace 8: **Vedení trajektorie konce paličky.** Z bodu 0 (aktuální pozice paličky) vedeme trajektorii asi 20cm nad buben do bodu 1. Potom je úder veden shora dolů do místa úderu do bodu 2. Nakonec se trajektorie vrací nad buben a lehce zpět k bubeníkovi do bodu 3.

Poslední aspekt trajektorie pohybu, který jsem zkoumal byl fakt, že dráha reálného pohybu není plochá, tedy poloha koncového bodu není lineární interpolace mezi začátkem a cílem pohybu, ale probíhá v určitém oblouku. Velikost, výška, sklon a umístění extrému tohoto oblouku jsou však závislé na vzájemné poloze ruky, startu a cíle pohybu a jeho věrná simulace je tak velmi obtížná.

Dále je nutné vyřešit jev, kdy právě nepoužívaná ruka po úderu nebude neustále napřažená nad místem posledního úderu, ale bubeník má tendenci ji po určité době stáhnout do klidové polohy. Tuto dobu je třeba stanovit tak, aby se ruka nestahovala do klidové polohy zbytečně často, ani aby zbytečně dlouho nečekala nad bubnem.

Ilustrace 9: **Klidová pozice rukou.** Pokud je ruka delší dobu v klidu, vrací se do této pozice.

4.3.1.2 Dynamika pohybu

Dynamika úderu by se rovněž měla projevit na výsledné podobě pohybu. Pohyb bude jistě vypadat jinak pokud do činelu udeříme jen lehce, pak se hýbe hlavně zápěstím, nebo pokud do něj udeříme opravdu silně a úder vedeme celou rukou. Výpočet dynamiky a jejího vlivu na pohyb ruky je ovšem značně složitý a je to samostatná oblast problematiky inverzní kinematiky. V této verzi aplikace tedy implementována není a dynamika úderu je znázorněna pouze amplitudou pohybu po úderu do některých bubnů a samozřejmě intenzitou a charakterem zvuku.

4.3.2 Animace

V této části textu se budeme zabývat zejména použitou technikou animace humanoida.

4.3.2.1 Modul inverzní kinematiky

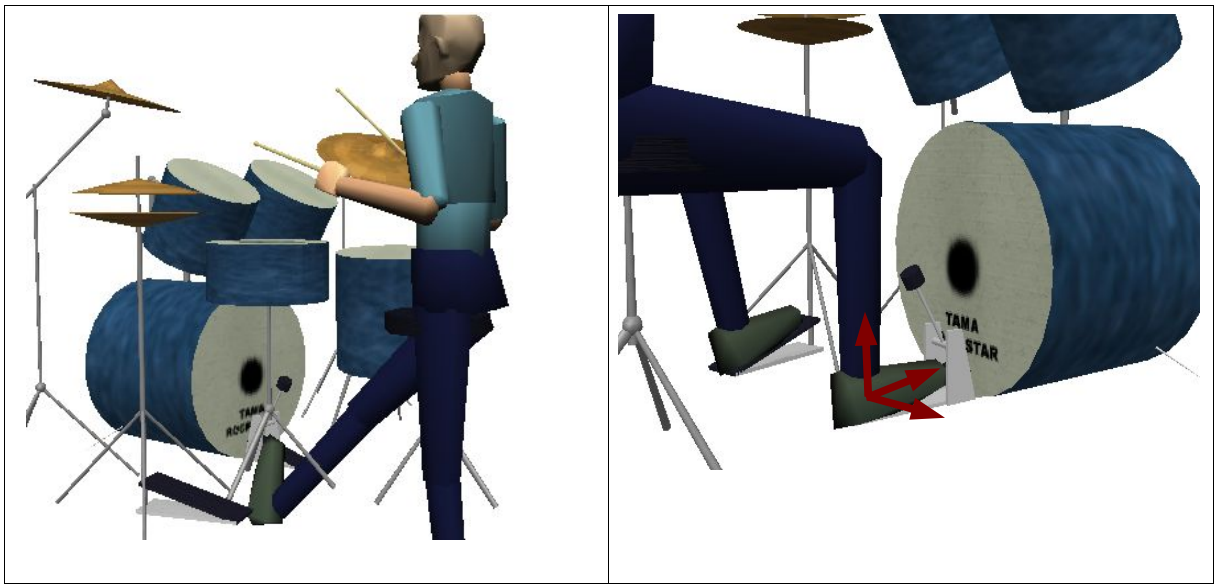
Pro výpočet animace rukou je použit Modul inverzní kinematiky. Tento modul umožňuje výpočet na kostře humanoida, který pro zadaný řetěz kloubů nalezne jejich natočení tak, že konec řetězu dosáhne nebo se v maximální možné míře přiblíží zadané cílové pozici. Tento nástroj nám umožňuje zadat libovolný cíl úderu, následně aplikovat výpočet na ruku humanoida držící paličku a tím dostat konec paličky do cílové pozice na bubnu. Nemusíme tedy předem znát umístění jednotlivých bubnů v sadě a hitpointů na těchto bubnech, ale pomocí výpočtu inverzní kinematiky se jejich rozestavení automaticky přizpůsobíme. Tato skutečnost je pro naší aplikaci zcela zásadní, jelikož je tím umožněno přehrání rytmu na libovolné sadě bicích. Podrobný popis principů a funkcí modulu IK je v kapitole o implementaci.

Jak bude rozebráno dále, při tvorbě aplikace jsem došel k závěru, že technika animace pomocí inverzní kinematiky není sama o sobě dostatečná pro realistickou simulaci pohybu rukou bubeníka, ale je zapotřebí tento výpočet podrobně řídit tak, aby respektoval nejen anatomická omezení, ale i záměr a filozofii simulovaného pohybu. Možná řešení tohoto problému jsou rovněž popsána v kapitole o implementaci.

4.3.2.2 Pohyb nohou

Pohyb nohou je řešen poněkud jinak, než pohyb rukou. Je to dáno především tím, že nohy zůstávají stále na jedné pozici a pohyb pří sešlápnutí pedálu kopáku nebo činelu hi-hat je vykonáván převážně v kotníku.

Nejprve je potřeba vyřešit umístění nohou na pedálech. Noha musí být umístěna nejen na správném místě, ale i ve správné orientaci ke pedálu. Pokud by orientace řešena nebyla, tak by se nejspíše noha v kotníku ohnula nahoru a pata by byla pod pedálem. To z toho důvodu, že základní poloha panáčka je ve stoje, pouze výška jeho těžiště je posunuta dolu do polohy vhodné pro posazení na stoličku. Tudíž na začátku jsou nohy pod úrovní bubnů a jejich přiblížení nahoru na pedály by probíhalo výše zmíněným způsobem. Je tedy potřeba pomocí orientace špičky chodidla hlídat, aby byla stále ve správné poloze a noha se ohýbala pouze v kyčli a v koleni.



Ilustrace 10: **Nastavení nohy do správné pozice na pedál.** Na levém obrázku vidíme pokus o přiblížení do cílové pozice bez výpočtu orientace chodidla. Noha se pak k pedálu přibíží zdola. Napravo je znázorněn postup s výpočtem orientace chodidla, který zajistí, že noha zaujme správnou pozici.

Když jsou nohy správně umístěny, tak jsou dvě možnosti. Buď jejich pohyb bude řešen pomocí inverzní kinematiky tak, že koncový efektor bude špička nohy a jeho poloha se bude měnit podle polohy hitpointu na pedálu. To je sice obrácený postup, než ve skutečnosti, tedy místo aby noha tlačila na pedál, tak pedál tahá za nohu, avšak vizuálně zde není rozdíl. Pouze je třeba noze vytvořit určitý předstih, jelikož přiblížení algoritmu inverzní kinematiky k cílovému bodu v reálném čase není úplné a to že noha následuje pohyb pedálu by mohlo být poznat. Malý předstih pohybu nohy před pedálem tento nedostatek řeší. Zůstává ale otázkou, zda poměrně vysoké výpočetní nároky (je třeba počítat přiblížení minimálně pro čtyři klouby na každé noze a to i s výpočtem orientace koncového efektoru, kdy výpočetní náročnost stoupne takřka dvojnásobně), stojí za poměrně malý vizuální efekt, jelikož amplituda pohybu nohy na pedálu je malá.

Druhou možností je použít inverzní kinematiku pouze pro umístění nohou na pedály a potom již jen provádět rotaci nohy v kotníku tak, aby sledovala pohyb pedálu. To je samozřejmě

celkem triviální postup s minimální výpočetní náročností. Pohyb sice nepůsobí tak reálně, jako při použití inverzní kinematiky, ale jeho názornost je dostatečná.

4.3.2.3 Animace hlavy

Pro větší živost pohybu a výsledný dojem je animována rovněž hlava humanoida. Je natáčena tak, aby se dívala stále do středu spojnice konců paliček. Je také implementováno jisté zpoždění tohoto natáčení pohledu, aby se zamezilo rychlému cukání, při rychlých změnách polohy paliček. Na tento výpočet není používán celý aparát inverzní kinematiky, ale jsou použity pouze některé jeho pomocné funkce, pro zjištění potřebného úhlu pohledu, na jehož hodnotu je pak nastaveno natočení v kloubu krčního obratle.

Ilustrace 11: **Animace hlavy.** Rotace v krčním obratli je nastavována tak, že vektor z něj vycházející a kolmý na osu hlavy směřuje do středu spojnice konců paliček.

4.4 Plánovač pohybů

Tato část programu řeší přidělování cílů rukám, spouštění jejich pohybu a spouštění pohybu bicích. Pohyb rukou a pedálů bubnů je samozřejmě nutné spustit s určitým předstihem, před okamžikem úderu. Navíc při řešení problému přidělení cílů rukám je potřeba znát jistou historii jejich pohybů. Je zde proto fronta událostí. Každé pole této fronty obsahuje události úderů rukou a nohou v jednom okamžiku. Následující pole obsahují události po sobě jdoucích nejkratších intervalů skladby.

4.4.1 Přidělování úderů rukám

Při bubnování je nutné se vždy rozhodnout, kterou rukou daný úder provedeme, případně, při úderu do dvou bubnů současně, jak údery rukám přiřadíme. To je důležité jednak pro efektivitu pohybu, kdy je rychlejší do bubnu udeřit rukou, která je mu blíže a jednak pro ošetření kolizí rukou.

Pro výběr ruky pro úder je samozřejmě důležitá poloha bubnu, ale bubeník má také tendenci pro úder do některých bubnů preferovat určitou ruku podle jejich typu. Například hi-hat je umístěna vlevo od bubeníka, ale bubnuje se na ni většinou pravou rukou. Tato preference bubnu je v této aplikaci nastavována pomocí systému priorit, kdy každý buben má nastaven

parametr priorita udávající, kterou ruku budeme při úderu do něj preferovat. A podle této priority se také plánovač pohybu rozhoduje jak údery rukám přiřadí.

4.4.2 Priorita

Plánovač v jednom okamžiku obdrží maximálně čtveřici událostí (jednu pro každou nohu a jednu pro každou ruku), které obsahují údaje o úderu, který má být proveden. Údaj priority se může pohybovat v celočíselném rozsahu –3 až 3. Význam hodnot je takový, že –3 znamená extrémní prioritu pro přiřazení úderu levé ruce 0 znamená zcela neutrální prioritu a 3 znamená extrémní prioritu pro přiřazení úderu pravé ruce. Priorita je nastavována pro celý buben a to v editoru sady bicích. Při přidání bubnu se priorita nastaví automaticky podle typu bubnu. Je ale možné ji změnit, tak aby odpovídala změněné pozici bubnu, nebo rozdílným preferencím uživatelů. Podle této priority, ale i podle typu a umístění bubnu se pak rozhoduje o přidělení událostí rukám.

Algoritmus, který bude podrobně popsán v kapitole o implementaci, by měl zajistit, že ruce se budou střídat tak, aby bubnování bylo vzhledem k jejich pohybu efektivní a aby se při tom zbytečně nekřížily. Tím se navíc ve většině případů vyhneme kolizím rukou. Pro úplné odstranění kolizí rukou při jakémkoliv rozestavení bubnů by však bylo potřeba implementovat i detekci kolizí a následně upravovat dráhy pohybů rukou.

4.5 Zvukový subsystém

Zvukový subsystém zajišťuje nahrávání a přehrávání zvuků bubnů a umožňuje přehrát zvuky ve formátech WAV a MP3 v libovolné kvalitě. Lze přehrávat až 32 zvuků najednou, přičemž každému můžeme nastavit hlasitost. V extrémních případech, kdy v rychlém sledu spustíme přehrávání velkého množství dlouhých zvuků sice může být tento počet překročen, pak dochází k vypnutí zvuků, které byly spuštěny nejdříve, ale většinou se jeví jako dostatečný.

4.6 Časování programu a cyklus přehrávání skladby

Aplikace používá dva časovače. Jeden velmi přesný s vysokou prioritou, který zajišťuje přehrávání skladby a stará se o spouštění úderů a průběh animace rukou a nohou. Jeho perioda je závislá na tempu skladby a její délka se rovná jedné třetině délky nejkratšího intervalu notového záznamu, což je jedna třetina dvaatřicetinové noty. Výpočet animace rukou a nohou je na něj napojen proto, aby bylo možné při snížení tempa skladby a tím i zpomalení pohybu končetin, lépe analyzovat přehrávaný rytmus. Druhý časovač má nižší prioritu a jeho perioda se s tempem skladby nemění. Na něj je napojen výpočet pohybu bubnů, animace hlavy a překreslování scény.

Při přehrávání skladby se cyklicky s periodou přesného časovače opakuje následující algoritmus. Časovač má periodu rovnou jedné třetině délky nejkratšího intervalu skladby proto akce spojené s přehráváním skladby se provádějí jednou za tři cykly.

Algoritmus **přehrávání skladby**
Vstup: aktuální stav přehrávání skladby a animace
Výstup: následující stav přehrávání skladby a animace
Metoda:
pro každý cyklus časovače
    proveď výpočet dalšího okna animace rukou
    proveď výpočet dalšího okna animace nohou
pro každý třetí cyklus časovač
    v editoru skladby načti další sloupec mřížky taktu
    odešli všechny údery v tomto sloupci do plánovače pohybů
    v plánovači posuň frontu událostí a vyšli signály pro animaci rukou a bubnů a přehrávání zvuku

Tím je zaručena synchronizace animace a přehrávání skladby.

5. Implementace

V této kapitole se budeme věnovat konkrétnímu způsobu programové implementace úlohy. Specifikujeme platformu, na které byla úloha vyvíjena, zmíníme použité externí knihovny, nastíníme vnitřní architekturu programu a způsob implementace jednotlivých částí úlohy.

5.1 Vývojová platforma a cílový OS

Aplikace byla vyvíjena v prostředí *Microsoft Visual Studio 6* v programovacím jazyce *C++* [Mic98] na platformě *Microsoft Windows XP*. Nicméně, při programování v tomto jazyce byl dodržen *ANSI* standard, takže vzhledem k tomu, že i použité knihovny QT [QT00], FMOD [FM00], OpenGL [OGL15] a GLUT jsou platformově nezávislé, je celá aplikace přenositelná na úrovni zdrojového kódu.

5.2 Použité knihovny

V jednotlivých oblastech aplikace bylo využito několik knihoven poskytujících nástroje pro snadnější implementaci těchto oblastí. Nyní uvedeme jejich stručný popis.

5.2.1 Knihovna QT

QT je multiplatformní *C++* knihovna vyvíjená firmou *Trolltech* a poskytuje nástroje pro plně objektově orientovanou implementaci grafického uživatelského rozhraní. Je dostupná pro platformy *Microsoft Windows – 95, 98, NT, 2000 a XP* a pro platformy *Unix / X11 - Linux, Sun Solaris, HP-UX, Digital Unix, IBM AIX, SGI IRIX* a další - [QT00].

QT zajišťuje:

- *velmi účinný mechanismus pro komunikaci objektů pomocí struktur signálů a slotů*
- *lehce nastavitelné vlastnosti objektů*
- *systém událostí a filtrů událostí*
- *sofistikovaný systém časovačů umožňující jednoduchou výstavbu grafického uživatelského rozhraní řízeného událostmi*
- *hierarchický systém objektových stromů a vlastnictví objektů*
- *tzv. guarded pointers, ukazatele, které jsou automaticky nastaveny na null, pokud odkazovaný objekt je zrušen.*

Některé tyto rysy jsou implementovány pomocí standardních *C++* technik, jiné, jako mechanismus slotů a signálů jsou zajišťovány pomocí *QT Meta Object Systému*.

Signály a sloty jsou používány pro komunikaci mezi objekty. Jejich mechanismus je centrální rys QT a pravděpodobně hlavní část knihovny, která ji nejvíce odlišuje od ostatních knihoven tohoto typu.

Všechny třídy, které dědí od tříd knihovny QT, mohou obsahovat signály a sloty. Signály jsou vysílány objektem, pokud se mění jeho stav způsobem zajímavým pro okolí, případně, pokud je k tomu dán explicitní příkaz. Tlačítko uživatelského rozhraní tedy vyšle signál o svém stlačení, pokud je na něm stisknut kursor myši, můžeme však i explicitně vyslat signál o významné události v běhu nějakého algoritmu. Signál neví nic o tom, kdo ho přijímá, čímž je zajištěna transparence komunikace jednotlivých komponent systému. Sloty jsou používány pro přijímání signálů, ale mohou být volány i jako běžné metody objektů. Slot neví, zda je k němu nějaký signál připojen či nikoliv, což opět zajišťuje komunikační transparenci. Můžeme

připojit libovolný počet signálů k jednomu slotu a signál může být připojen k libovolnému počtu slotů. Je také možné signál propojit přímo s dalším signálem (pak po vyslání prvního signálu bude vyslán i druhý signál).

To, co je hlavním smyslem vysílání signálů a jejich přijímání sloty, je přenos informace. Signály a sloty mohou být volány s libovolným počtem argumentů libovolného typu, přičemž tyto parametry musí být v propojených signálech a slotech kompatibilní. Jinak k požadovanému propojení nedojde, čímž je zajištěna striktní typová kontrola. Důležitým aspektem celého mechanismu je to, že sloty a signály mohou být propojeny z vnějšku jednotlivých objektů. Tím je dosaženo vysoké nezávislosti jednotlivých komponent systému.

Podrobnější informace o knihovně QT, zejména o mechanismu slotů a signálů, lze nalézt v manuálu knihovny QT.

5.2.2 Knihovna FMOD

FMOD je multiplatformní knihovna poskytující služby komplexního zvukového systému. Je dostupná pro platformy *Windows, Linux*, *Macintosh, GameCube, PS2 a Xbox.* Podporuje 3D zvuk, přehrávání *midi*, různých formátů modulů, formáty *wav, mp3, ogg vorbis, wma, aiff*, nahrávání zvuku, CD playback, CD extrakci, instrukce *mmx,* DSP efekty, použití spektrální analýzy, zajišťuje podporu synchronizace, standardy *ASIO, EAX 2 & 3*, použití v prostředích *C/C++/VB/Delphi* a další. [FM00]

FMOD poskytuje propracované aplikační rozhraní pro využití těchto funkcí. Tato knihovna navíc poskytuje velmi kvalitní zvukový mixér s velmi malými nároky na strojový čas. V této aplikaci byl sice použit pouze zlomek funkcí, které FMOD nabízí, nicméně jednoduchost jejich použití v implementaci plně ospravedlňuje použití takto komplexní knihovny. Jeden z velkých přínosů tohoto nástroje byl také ve využití jeho synchronizačních funkcí pro řízení animace a přehrávání skladby.

5.2.3 OpenGL

OpenGL – *"Open Graphics Library"* je softwarové rozhraní pro přístup ke grafickému hardwaru. Rozhraní sestává z několika set procedur a funkcí, které umožňují programátorovi specifikovat objekty a operace pro vytvoření grafického výstupu, specificky barevných obrazů 3D objektů. - [OGL15]

Pro programátora je OpenGL sada příkazů, které umožňují specifikaci geometrických objektů ve 2D a 3D, spolu s nástroji, které ovlivňují, jak budou tyto objekty vykresleny.

5.2.4 Knihovna Glut

GLUT je jednoduché na platformě nezávislé programové rozhraní (knihovna) určené primárně pro vytváření jednoduchých uživatelských rozhraní pro aplikace používající OpenGL (tuto funkci zde zajišťuje knihovna QT). GLUT má ovšem i některé další vlastnosti a schopnosti, které rozšiřují knihovnu *OpenGL* a které byly v této aplikaci využity. Jsou to zejména:

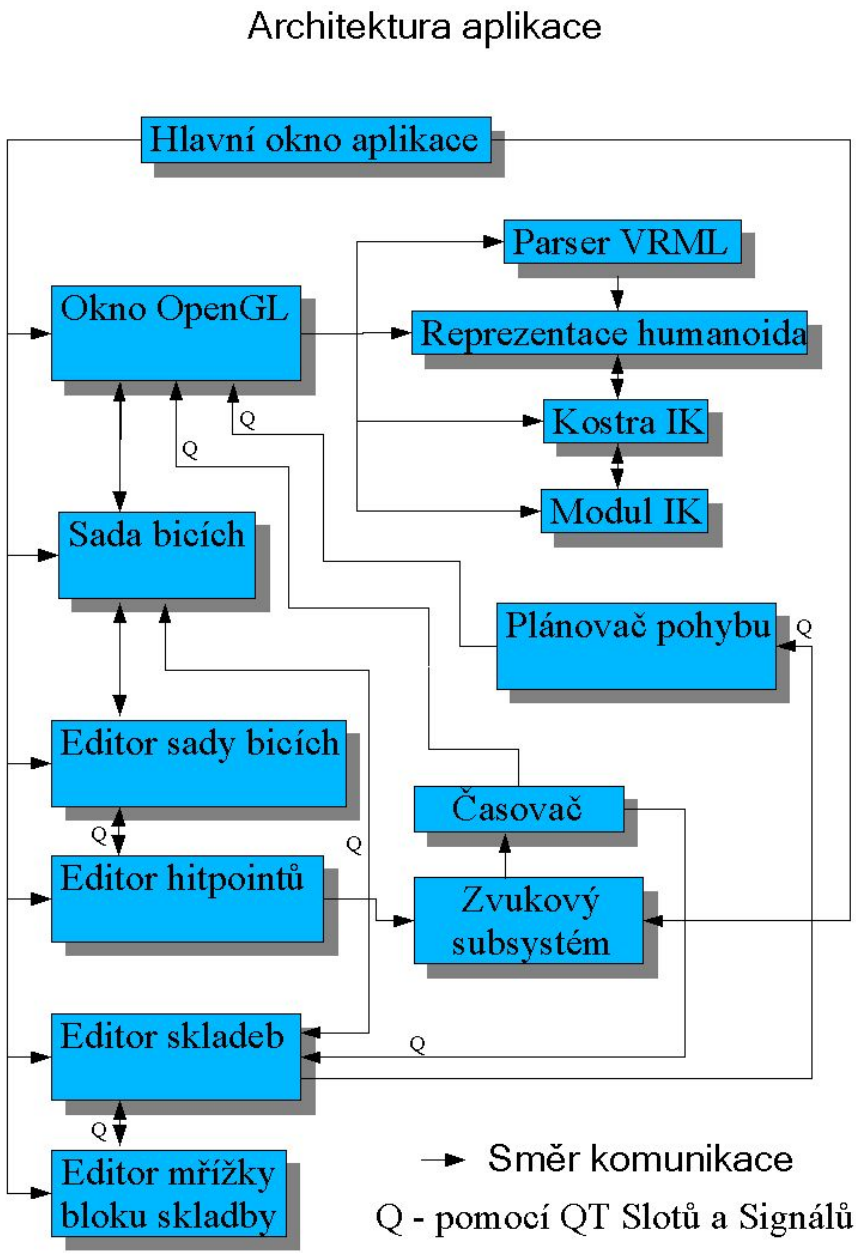
- *Možnost renderovaní základních geometrických objektů pomocí kvadrik – to jsou povrchy definované pomocí obecné kvadratické rovnice.* GLUT *má předdefinované funkce pro vykreslení některých kvadrik - válce, kužele, koule a disku*
- *Používání bitmapových a vektorových fontů*
- *Práce s barevnou paletou*

5.3 Architektura a hierarchie systému

Aplikace byla vytvořena co možná nejvíce modulárně, to znamená, že její jednotlivé části jsou do značné míry nezávislé a mohou být lehce změněny, nebo být použity v jiné aplikaci. Na této modularitě se značnou měrou podílí i použití knihovny QT a to zejména jejího mechanismu slotů a signálů.

5.3.1 Moduly

Základní rámec programu tvoří hlavní okno aplikace. V něm se vytvářejí instance všech ostatních částí programu a jsou zde navzájem propojovány. Toto okno rovněž obsahuje lištu s několika menu, kterými jsou přepínány módy aplikace, nastavovány parametry animace, případně další volby. Zbytek aplikace se skládá z následujících modulů: parser VRML a model postavy, okno OpenGL, editor sady bicích, editor skladby, plánovač pohybů rukou a zvukový subsystém. Tyto moduly jsou na sobě nezávislé a jsou z vnějšku z úrovně hlavního okna propojeny pomocí slotů a signálů knihovny QT. Kromě toho jediná závislost mezi moduly s výjimkou zvukového systému je ta, že všechny obsahují ukazatel na objekt reprezentující aktuální sadu bicích.



Ilustrace 12: **Architektura aplikace a komunikace mezi jednotlivými moduly.**

Nyní popíšeme implementaci jednotlivých modulů.

5.3.2 Parser VRML a model postavy

5.3.2.1 Standard VRML 2.0

Model postavy bubeníka je vytvořen pomocí jazyka VRML, který implementuje standard H-Anim a tento standard popisuje virtuálního humanoida ve VRML. VRML (Virtual Reality Modeling Language) je formát souboru pro popsání interaktivních 3D objektů a světů. VRML je navrženo pro použití na internetu, intranetu i lokálních klientech. VRML má také sloužit jako univerzální výměnný formát pro integraci 3D grafiky a multimedií.

VRML je v této práci používáno pouze pasivně pro uchování popisu modelu postavy a je načítáno pomocí jednoduchého VRML parseru. Zmíníme zde tedy pouze základní principy VRML a oblasti, jež tento parser řeší a pro další informace o tomto jazyce čtenáře odkážeme na literaturu [Vrl97].

5.3.2.2 Standard H-Anim

H-Anim specifikuje standardní způsob reprezentace humanoidů ve VRML 2.0. Hlavní cíle tohoto standardu jsou [Han03]:

- *Kompatibilita:* Humanoid by měl fungovat v jakémkoli VRML 2.0 kompatibilním prohlížeči.
- *Flexibilita:* Nejsou stanoveny žádné předpoklady o typu aplikace, která bude humanoida používat.
- *Jednoduchost.*

5.3.2.2.1 Popis

Lidské tělo se skládá z řady segmentů (například předloktí nebo lýtko) které jsou navzájem propojeny klouby (například loket nebo kotník). Pro to, aby aplikace mohla humanoida animovat, potřebuje získat přístup ke kloubům humanoida a měnit jejich úhly natočení. Aplikace může využívat i další informace, jako například limity natočení v kloubech nebo hmotnosti segmentů.

Soubor VRML humanoida obsahuje skupinu uzlů typu Joint (angl. kloub), které jsou hierarchicky uspořádány. Uzel typu Joint může obsahovat další uzly typu Joint a může obsahovat také uzly typu Segment, které popisují část těla asociovanou s tímto uzlem. Soubor také obsahuje jeden uzel typu Humanoid, který uchovává popisná data jako jméno autora, copyright a dodatečné informace. Tento uzel také uchovává reference na všechny uzly typů Joint a Segment. Přes tento uzel tedy aplikace získá přístup k jednotlivým částem kostry.

5.3.2.2.2 Uzly

Pro zjednodušení vytváření humanoida byly do VRML specifikace přidány tři uzly. Každý z nich je definován pomocí PROTO.

- Uzel typu Joint

Každý kloub v těle je reprezentován uzlem typu Joint. Základní implementace Joint uzlu je transformační uzel, který je používán k definování vztahu každé části těla k jejímu přímému otci. To je ovšem pouze jedna možnost implementace. Některé systémy mohou například používat jednu polygonální síť pro reprezentaci humanoida, místo oddělených uzlů typu IndexedFaceset pro každou část těla. V takovém případě uzel typu Joint bude zodpovědný za pohyb vertexů korespondujících s konkrétní částí těla a všech jejích potomků. V uzlu typu Joint jsou uchovávány ještě další specifické informace. Konkrétně, jméno slouží k identifikaci uzlu v hierarchii, limity a stiffness (tuhost) slouží pro podporu inverzní kinematiky. A samozřejmě uzel typu Joint obsahuje pole odkazů na své potomky.

PROTO Joint uzlu vypadá následovně

```
PROTO Joint [
    exposedField   SFString     name                ""
    exposedField   SFVec3f      translation         0 0 0
    exposedField   SFRotation   rotation            0 0 1 0
    exposedField   SFVec3f      scale               1 1 1
    exposedField   SFRotation   scaleOrientation    0 0 1 0
    exposedField   SFVec3f      center              0 0 0
    exposedField   MFNode       children            []
    exposedField   MFFloat      ulimit              [ 0 0 0 ]
    exposedField   MFFloat      llimit              [ 0 0 0 ]
    exposedField   SFRotation   limitOrientation    0 0 1 0
    exposedField   MFFloat      stiffness           [1 1 1]
]
```

Ilustrace 13: **Prototyp uzlu Joint.** Převzato z [Han15]

Většina polí odpovídá polím transformačního uzlu i když jejich interpretace může být rozdílná. Další pole mají následující význam. Pole ulimit udává maximální povolené natočení kloubu v úhlech okolo os X, Y a Z. Obdobně llimit udává minimální povolené natočení. Stiffness udává tuhost kloubu, tedy odpor k natočení kolem jednotlivých souřadných os v mezích 0 – maximální odpor, nulová možnost natočení, až 1 – nulový odpor. Pole center udává absolutní pozici středu kloubu v souřadném systému postavy.

- Uzel typu Segment

Každá část těla je uložena v uzlu typu Segment. Tento uzel je typicky implementován jako uzel, obsahující reprezentaci dané části těla. Ale může to být i uzel popisující skupinu takových tvarů, nebo transformační uzel, který určuje pozici části těla v rámci jejího souřadného systému. Uzel typu Segment bude také obsahovat své jméno pro identifikaci v hierarchii a odkazy na své potomky.

```
PROTO Segment [
        exposedField   SFString     name                ""
        exposedField   SFFloat      mass                0
        exposedField   SFVec3f      centerOfMass        0 0 0
        field          SFVec3f      bboxCenter          0 0 0
        field          SFVec3f      bboxSize            -1 -1 -1
        exposedField   MFNode       children            [ ]
    ]
```

Ilustrace 14: **Prototyp uzlu Segment.** Převzato z [Han15]

Pole mass udává celkovou hmotnost části těla, kterou uzel popisuje. Informace o bounding boxu mohou být využity například při detekci kolizí.

- Uzel typu Site

Uzel typu Site reprezentuje místo na těle humanoida, které chceme nějak identifikovat. Například špička ukazováčku, nebo oči. I tento uzel může mít potomky, můžeme tak třeba do dlaně přidat nějaký předmět.

```
PROTO Site [
    eventIn         MFNode      addChildren
    eventIn         MFNode      removeChildren
    exposedField    MFNode      children        []
    exposedField    SFString    name            ""
    exposedField    SFRotation  rotation        0 0 1 0
    exposedField    SFVec3f     translation     0 0 0
]
```

Ilustrace 15: **Prototyp uzlu Site.** Převzato z [Han15]

- Uzel typu Humanoid

Uzel typu Humanoid slouží k uložení textových dat popisujících humanoida a také obsahuje reference na uzly typu Joint a Segment.

```
PROTO Humanoid [
    exposedField    SFVec3f      size              0.5 1.75 0.3
    exposedField    SFString   name                ""
    exposedField    MFString   info                []
    exposedField    SFString   version             "1.1"
    exposedField    MFNode     joints              []
    exposedField    MFNode     segments            []
    exposedField    MFNode     sites                 [ ]
    exposedField    MFNode     viewpoints            [ ]
    exposedField    MFNode     humanoidBody          [ ]
    exposedField    SFVec3f    center              0 0 0
    exposedField    SFRotation rotation            0 0 1 0
    exposedField    SFVec3f    scale               1 1 1
    exposedField    SFRotation scaleOrientation    0 0 1 0
    exposedField    SFVec3f    translation         0 0 0
    field           SFVec3f    bboxCenter          0 0 0
    field           SFVec3f    bboxSize            -1 -1 -1
    ]
```

Ilustrace 16: **Prototyp uzlu Humanoid.** Převzato z [Han15]

Pole tohoto uzlu kombinují pole uzlů typu Segment i Joint. Pouze pole children je zde nahrazeno poli joints, segments, sites, viewpoints, které obsahují odkazy na uzly daných typů. Navíc je zde ještě pole humanoidBody, které v podstatě obsahuje pouze odkaz na kořen hierarchie uzlů popisující humanoida.

5.3.2.2.3 Model humanoida

Humanoid by měl být vymodelován tak, že stojí, dívá se ve směru osy Z s osou Y ukazující nahoru a osou X ukazující doleva. Počátek (0, 0, 0) by měl být umístěn v úrovni země mezi nohama humanoida. Nohy by měly být postaveny naplocho na zemi v šíři boků od sebe. Spodek chodidla by měl být na úrovni Y=0. Ruce mají být rovné a souběžně se stranami těla a dlaněmi otočenými dovnitř. Toto je základní pozice humanoida.

V této pozici by měly všechny úhly být nulové. Navíc i pole translation by mělo být nulové a pole scale nastavené na hodnoty (1, 1, 1). Jediné pole, které nemá základní hodnotu nulovou je pole center. To je používáno k určení bodu, okolo kterého se kloub (a všichni jeho potomci) bude otáčet.

Ilustrace 17: **Postava humanoida v základní poloze pro modelování.** Převzato z [Han15]

5.3.2.3 Parser standardu H-Anim

V této aplikaci je používána postava bubeníka založená na standardu H-Anim, bylo proto potřeba implementovat parser, který tuto postavu načte a načtená data převede do vnitřní reprezentace programu.

Parser načte hierarchickou strukturu reprezentující kostru jakéhokoli humanoida splňujícího zásady specifikace H-Anim 1.1 a převede ji do hierarchie tříd, které odpovídají jednotlivým VRML uzlům. Jsou načítány uzly typu Joint, Segment, Site a Humanoid.

Při načítání souboru parser nejprve přeskočí specifikaci PROTO, jelikož počítá se splněním standardu a posune se v souboru až na místo, kde narazí na uzel Humanoid. Načte pole version, name a info, pokud jsou přítomny a pokračuje až k poli humanoidBody. Zde pak začne načítat hierarchii uzlů typu Joint a Segment. Pro každý uzel je vytvořen jeho ekvivalent ve vnitřní reprezentaci programu a je vytvořena hierarchická struktura plně odpovídající kostře popsané v načítaném souboru.

Parser respektuje možnost nepřítomnosti jednotlivých polí uzlů a v takovém případě jsou u jejich ekvivalentu ve vnitřní reprezentaci nastaveny defaultní hodnoty. Respektuje i možnost libovolného pořadí jednotlivých polí a další zákonitosti syntaxe souboru VRML.

Jediná omezení kladená na obsah souboru humanoida jsou, že nesmí obsahovat uzly jiných typů než Humanoid, Joint, Segment a Site. Jsou pak v lepším případě ignorovány, v horším je nahlášena chyba nekompatibility souboru. A požadavek, aby těla jednotlivých segmentů byla uložena v separátních souborech, na které je v uzlu typu Segment odkazováno pomocí uzlu typu Inline. Při nedodržení této podmínky nebudou těla načtena, případně bude nahlášena chyba nekompatibility souboru.

5.3.2.4 Parser 3D modelu

Popis 3D modelu jednotlivých segmentů těla humanoida je uložen v separátních souborech a je na ně odkazováno pomocí VRML uzlu typu Inline vloženého v uzlech typu Segment. Obsah tohoto souboru je pak načítán pomocí zvláštní sekce parseru VRML.

Hlavním účelem této práce nebylo implementovat celý parser VRML, což by byl úkol poměrně náročný a zdlouhavý. Z toho důvodu se parser omezuje pouze na jeden určitý způsob reprezentace těla segmentu.

Parser předpokládá, že soubory s reprezentací části těla budou obsahovat jeden nebo více uzlů typu Shape, ve kterých bude specifikován materiál a uzel typu IndexedFaceSet, který popisuje síť polygonů. Případně může být uzel typu Shape obalen do uzlu typu transform určujícího jeho geometrické transformace. V uzlu typu IndexedFaceSet je načteno pole coordinate, ve kterém jsou určeny jednotlivé vrcholy mřížky. Dále je načteno pole coordIndex, které určuje z jakých vrcholů se skládají polygony mřížky. Texturovací souřadnice v této implementaci načítány nejsou.

Jelikož aplikace používá grafickou knihovnu OpenGL, bylo potřeba pole coordIndex upravit tak, aby zadávalo pouze polygony se stejným počtem vrcholů, jelikož OpenGL neumožňuje zobrazovat najednou polygony s různým počtem vrcholů. Toho bylo dosaženo triangulací každého polygonu, který obsahoval vyšší počet vrcholů než tři.

Ilustrace 18: Model načítaný pomocí implementovaného parseru jazyka VRML.

V této části parseru jsou také automaticky vypočítány normálové vektory pro každý vrchol. Pro tento úkon je třeba znát orientaci polygonu. Ta by měla být pro všechny polygonu v jednom uzlu typu IndexedFaceSet stejná a mělo by ji udávat pole ccw. Bohužel v některých souborech popisujících použitého humanoida orientace polygonů neodpovídala nastavení

tohoto pole a bylo třeba tyto případy řešit zvlášť. U souborů, které mají pole ccw nastaveno na FALSE a u souborů u kterých jsem zjistil, že je to také potřeba, je obrácena orientace polygonů. Vektorovým součinem je vypočtena normála pro každý trojúhelník a poté je přičtena ke každému z jeho vrcholů. Nakonec jsou normály renormalizovány, tedy jsou převedeny na vektory o jednotkové velikosti.

5.3.3 Editor sady bicích

Implementace tohoto editoru je téměř celá postavena na knihovně QT. Skládá se ze dvou panelů, z nichž první je určen pro manipulaci s částmi hierarchie sady bicích a druhý pro tvorbu hitpointů a manipulaci s nimi. Oba tyto panely jsou realizovány jako nezávislé komponenty, které spolu komunikují pouze pomocí signálů QT a to tak, že panelu pro úpravu hitpointů je zaslán ukazatel na aktuálně zvolený buben sady, na kterém mají být hitpointy upravovány, případně, je mu sděleno, že k aktuální části sady hitpointy přidat nelze a panel se pak přepne do pasivního módu.

Ukazatel na hierarchii sady je i v okně OpenGL, kde se sada zobrazuje. Takže jakákoliv změna provedená v editoru se v grafickém okně ihned projeví. Popis ovládání a funkcí tohoto editoru čtenář nalezne v dodatku A.

Ilustrace 19: **Editor sady bicích**

5.3.4 Editor skladeb

Tento editor je stejně jako editor sady bicích postaven na knihovně QT. Editor se skládá rovněž ze dvou částí (panelů). První panel slouží pro nastavení parametrů editace skladby a manipulaci se skladbou. Druhý zobrazuje mřížku pro zadávání úderů do skladby.

Ilustrace 20: **Editor skladeb**

5.3.4.1 Přehrávání skladby

V editoru skladeb je rovněž řízeno přehrávání skladby. Při spuštění přehrávání se vyšle signál, který ve zvukovém subsystému spustí přehrávání pomocné skladby o délce jedné dvaatřicetinové noty a stejném tempu, jako skládaný rytmus. V každé cyklu je pak volána callback funkce knihovny FMOD, a v ní je vyslán signál, na který jsou napojeny ostatní moduly. Je na něj napojen rovněž slot v tomto editoru, který zajišťuje přehrávání. V tomto slotu je vždy načten aktuální sloupec aktuálního bloku skladby a noty v něm zapsané jsou odeslány signálem do modulu plánovače pohybů rukou. Tam jsou zanalyzovány a přiřazeny rukám a nohám a jsou zde podle nich vyslány signály pro spuštění pohybu rukou a pohybu bubnů a pro přehrávání zvuků.

5.3.5 Zvukový subsystém

Zvukovou stránku aplikace zajišťuje knihovna FMOD. Při nahrání zvuku v editoru sady bicích, případně při nahrání celé hotové sady se zvuk z disku načte pomocí funkcí knihovny

FMOD do paměti a je vrácen jeho index v poli zvuků, pomocí kterého k tomuto zvuku později přistupujeme. Tento index je uchováván v hitpointu, ke kterému daný zvuk náleží. Počet zvuků není nijak omezen, pouze si musíme uvědomit, že zvuky v komprimovaném formátu MP3 jsou po nahrání do paměti dekomprimovány a při příliš velkém využití operační paměti, může systém provádět swapování na disk, což se většinou projeví zpomalením aplikace.

5.3.5.1 Využití pro implementaci přesného časovače

V předchozí kapitole jsem zmínili, že zvukový subsystém je využit pro časování aplikace. Je to umožněno následovně. Při spuštění přehrávání skladby v aplikaci je v knihovně FMOD spuštěno přehrávání prázdné cyklické skladby o stejném tempu a délce jedné třetiny dvatřicetinové noty (to je perioda ve které jsou počítány jednotlivé snímky animace). V každém cyklu této skladby FMOD automaticky volá callback funkci, jejíž tělo můžeme libovolně upravit. V našem případě je zde umístěn příkaz k vyslání signálu třídy TimerEvent, který okamžitě spouští rutiny ostatních modulů, které je třeba řídit.

Popis ovládání a funkcí tohoto editoru čtenář nalezne v dodatku A.

5.3.6 Modul inverzní kinematiky

Modul inverzní kinematiky slouží pro výpočet animace humanoida v reálném čase.

Algoritmy tohoto modulu vznikly jako diplomová práce mého kolegy Bohuslava France [Fra05], rovněž v tomto semestru na katedře výpočetní techniky fakulty elektrotechnické na ČVUT. Práce Bohuslava France byla vytvořena pro platformu jazyka Java s přímou vazbou na jazyk VRML. Rozhodl jsem se tyto algoritmy použít a upravit modul tak, aby splňoval požadavky naší aplikace.

Zatímco původní algoritmy jsou psané v jazyce Java, pro použití modulu v této aplikaci bylo zapotřebí jej přepsat do jazyka C++ a vazbu na VRML přizpůsobit mé vnitřní reprezentaci VRML uzlů humanoida, které jsou zde načítány pomocí výše zmiňovaného parseru standardu H-Anim. K tomuto kroku mě vedla skutečnost, že C++ pracuje lépe v režimu reálného času než Java a bylo nutné vzít v úvahu otázky výkonu, na který aplikace klade také poměrně vysoké nároky.

5.3.6.1 Přepis modulu IK

Přepis IK modulu bylo nutno vzít v úvahu že VRML a OpenGL mají opačnou orientaci osy Z.

Další problém byl fakt, že C++ na rozdíl od Javy neobsahuje funkci takzvaného garbage collectoru a bylo tak nutné implementovat správné uvolňování objektů z paměti.

Kromě těchto dvou bodů proběhl přepis modulu IK bez problémů. Pro každou třídu byl vytvořen hlavičkový soubor s deklarační částí a cpp soubor s tělem třídy. IK modul není přepsán celý, ale jsou vynechány části kódu pro non-realtime zpracování výpočtu a přídavné třídy pro zobrazování animace, které pro naší aplikaci nejsou potřebné. V celém přepisu je pokud možno dodržena struktura a metody tříd, funkční stránka přepsaného modulu je v podstatě identická s původní verzí.

5.3.6.2 Funkce modulu IK

Modul IK je používán tak, že mu předáme ukazatel na kořen humanoida a ukazatele na základní uzel a na uzel koncového efektoru. Potom ještě specifikujeme cílovou destinaci a tolerance pro její dosažení.

Modul IK nalezne řetěz kloubů, které v kostře humanoida nejkratší cestou spojují základní uzel s efektorem. Na tomto řetězu potom postupně mění rotace v jednotlivých kloubech tak, aby efektor se dostal co nejblíže k cílové pozici. Pokud je v cílové destinaci zadána i orientace, snaží se navíc efektor do této orientace natočit. Pokud je zadána pouze orientace, provádí se pouze natočení koncového efektoru. Při výpočtu se provede maximálně 10 iterací tohoto přiblížení. Pokud se efektor dostane dříve do cílové pozice a orientace s požadovanými tolerancemi, výpočet je ukončen dříve.

5.3.6.3 Algoritmus výpočtu

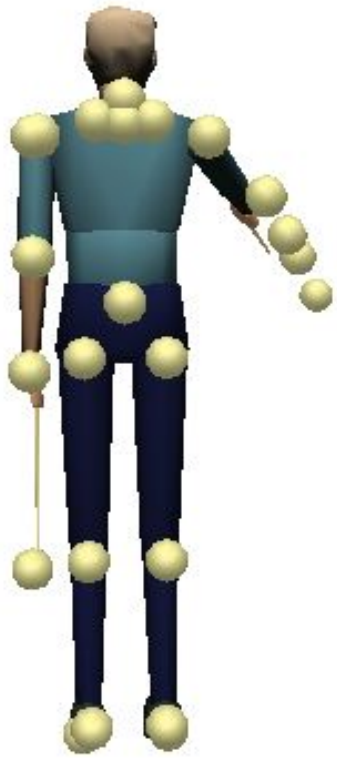
Výpočet nejlepšího přiblížení k cílové pozici popisuje následující algoritmus:

Algoritmus **výpočet IK**
Vstup: cílová pozice a orientace, základní uzel, efektor, tolerance pozice, tolerance orientace
Výstup: natočení kloubů mezi základním uzlem a efektorem takové, že efektor je co nejblíže k cílové pozici
Metoda:
vytvoř nejkratší řetězec kloubů mezi základním uzlem a efektorem
opakuj
    pro každý kloub v řetězci počínaje efektorem
        spočítej úhel natočení k cílové pozici
        pro každý kloub humanoida počínaje kořenem
            přepočítej pozici a orientaci
dokud
    pozice a orientace efektoru není v zadané toleranci od cílové pozice a orientace nebo neprojde maximální počet iterací

Tento algoritmus se vykoná pro každý snímek zobrazovaný na trajektorii úderu.

5.3.6.4 Paralelní kostra IK

Výpočet inverzní kinematiky neprobíhá přímo na kloubech humanoida, nýbrž na IK ekvivalentech těchto kloubů. Je to proto, že kloub humanoida je vytvořen jako ekvivalent uzlu VRML splňujícího standard H-Anim, zatímco uzel IK je vytvořen trochu odlišně tak, aby měl vlastnosti a parametry vhodné pro výpočet inverzní kinematiky.



Ilustrace 21: **Znázornění paralelní kostry inverzní kinematiky.**

Po nahrání souboru humanoida a vytvoření hierarchické struktury jeho uzlů typu Point, Segment a Site, je podle ní vytvořena paralelní struktura uzlů IK (dále kostra IK). V každém uzlu typu Joint je vytvořen odkaz na uzel typu IK Joint, který je podle něj vytvořen a jehož parametry jsou podle něj nastaveny, obdobně jsou přidány uzly typu IK Segment a IK Site. Všechny uzly IK jsou uspořádány do stejné hierarchie, jakou tvoří původní uzly VRML. Navíc je v každém uzlu IK ještě odkaz na jeho přímého předchůdce (rodiče) v hierarchii. Ten je potom využíván při výpočtu IK a umožňuje průchod hierarchií od potomků směrem ke kořeni.

Výpočet IK tedy neprobíhá na uzlech VRML humanoida, ale na paralelní kostře IK. Po dokončení výpočtu jsou aktualizovány rotace v uzlech typu Joint podle ekvivalentních uzlů typu IK Joint.

5.3.6.5 Možnosti modulu IK

Modul IK dovoluje poměrně komplexní kontrolu a nastavení parametrů výpočtu inverzní kinematiky. Základní funkce je taková, že modulu IK předáme kořenový uzel kostry IK, základní uzel řetězu kloubů, na kterém se bude výpočet provádět, koncový uzel - efektor tohoto řetězu a cílovou destinaci, ke které se má efektor přiblížit. Cílová destinace může udávat pouze polohu, nebo pouze orientaci, případně obojí. Výpočet je tomu přizpůsoben. Efektor je uzel, který se má dostat do místa určeného cílovou destinací, případně má zaujmout orientaci, která je v cílové destinaci určena. Může to být uzel typu IK Joint, nebo IK Site. Základní uzel řetězu kloubů je uzel ke kterému je výpočet v kostře IK prováděn a může to být pouze uzel typu IK Joint. Pokud tedy chceme hýbat rukou, jako koncový efektor zvolíme například konec ukazováčku a potom si musíme rozmyslet, jak zvolit základní uzel. Pokud se má humanoid například někam natáhnout, nebo něco sebrat, zvolíme jako základní uzel až

kořen humanoida. Pokud chceme pouze pohnout rukou a tělo humanoida nechat v klidu, zvolíme jako základní uzel například rameno.

V našem případě jako koncový efektor volíme konec paličky a jako základní klouby volíme ramena humanoida. Při pohybu nohou řízeném výpočtem IK volíme jako koncový efektor špičku chodidla a jako základní uzel kyčel.

Nastavení orientace koncového efektoru může být důležité. Například ve výše zmíněném případě nastavení nohou do pozic na pedály bychom se bez výpočtu koncové orientace téměř neobešli, případně by bylo potřeba nohy donastavit do správných pozic ručně bez použití inverzní kinematiky. Na druhou stranu, pokud chceme pouze dosáhnout paličkou do určitého bodu, tak nastavení koncové orientace ve většině případů nebudeme potřebovat.

Chování výpočtu modulu IK a způsob dosažení cílové pozice lze velkou měrou ovlivnit pomocí nastavení limitů a tuhostí v kloubech. V každém kloubu lze nastavit minimální a maximální povolené natočení okolo jednotlivých os kloubu. Navíc lze nastavit tuhost kloubu, která udává, jak těžko se kloub otáčí okolo jednotlivých os. Při nastavení tuhosti na nulu nelze s kloubem hýbat vůbec, naopak při hodnotě jedna neklade kloub žádný odpor. Pomocí tohoto parametru lze například ovlivnit, v kterých kloubech a v jakém směru se končetina ohne nejdříve. Je přirozené, že ruka se ohne nejdříve v lokti a teprve potom v zápěstí, než naopak. Pomocí limitů lze nastavit anatomická omezení pohybu v kloubech. Například v lokti bude povolena rotace pouze kolem osy X a například v rameni bude povolena kolem všech os s největším rozsahem kolem osy X a nejmenším rozsahem kolem osy Y. Tato omezení navíc můžeme zúžit tak, aby lépe odpovídala požadovanému pohybu. Takže třeba v rameni povolíme menší rotaci kolem osy Z tak, aby se loket nemohl dostat příliš vysoko a podobně. Nastavení těchto limitů a tuhostí si ovšem žádá velkou dávku trpělivosti a experimentování, jelikož sice udržují pohyb v určitých mezích, ale právě tato omezení mohou v některých situacích ztěžovat dosažení určité pozice. Nastavení tuhostí v kloubech zase ztěžuje dosažení cílové destinace, takže je pak potřeba provést více iterací přiblížení, což samozřejmě obnáší vyšší výpočetní nároky.

Při výpočtu přiblížení musíme ještě brát v úvahu, že výpočet probíhá od základního uzlu směrem ke koncovému efektoru, takže uzly blíže k základnímu uzlu budou natáčeny spíše, než uzly vzdálenější. Ruka se tak více ohne v ramenní, než v zápěstí. A také, výpočet natočení probíhá podle jednotlivých os odděleně, přičemž nejdříve se spočítá natočení podle osy Z, potom podle osy X a nakonec podle osy Y. Díky tomu mají klouby tendenci se natáčet podle os právě v tomto pořadí priorit. To ovšem může být někdy problém, jelikož například v kyčli bychom preferovali natočení podle osy Y a teprve potom podle osy Z, zatímco výpočetní modul má tendenci opačnou. Tento rozpor se dá do jisté míry kompenzovat nastavením tuhostí rotace v příslušném kloubu.

5.3.6.6 Výpočetní nároky a optimalizace pro výpočet v reálném čase

Výpočetní nároky IK modulu mohou být tedy asymptoticky vyjádřeny takto:

$O(m \cdot i \cdot n^2)$

m = počet snímků pro jeden úder

i = počet iterací v jednom snímku

n = počet kloubů celého humanoida

Výpočetní nároky algoritmu tedy závisí nejvíce na počtu kloubů humanoida. Pokud byl algoritmus aplikován na humanoida, který měl definovány všechny klouby lidské kostry, pak výpočet jednoho trval několik vteřin. To je však pro tuto aplikaci neúnosné, jelikož zde je potřeba jeden snímek spočítat v čase několika milisekund.

Vzhledem k této skutečnosti tedy bylo potřeba snížit počet kloubů humanoida na nezbytné minimum. V kostře byly ponechány pouze klouby zápěstí, lokte a ramene, kotníku, kolena a kyčle a potom střed humanoida, nejvyšší zádový obratel a první krční obratel. Po tomto omezení se již podařilo čas výpočtu snížit na přijatelnou mez.

Na počtu iterací závisí kvalita přiblížení do cílové polohy. Jako naprosto dostačující se jeví provést deset iterací, aby se koncový efektor dostal do pozice s tolerancí pět centimetrů. Ve většině případů se výpočet zastaví ještě dříve. Jiná situace nastává, pokud povolíme i výpočet orientace koncového efektoru. Potom výpočet skoro vždy využije všech deset iterací a ani potom ještě nemusí být cíl s požadovanou tolerancí dosažen. To se samozřejmě projeví na zvýšení času výpočtu. Sice jako multiplikativní konstanta, ale při nízkém počtu kloubů bohužel dosti patrná (v našem případě vzrostl čas výpočtu někdy až na dvojnásobek).

Jelikož rychlost pohybu je závislá na tempu skladby (aby bylo možné pohyb zpomalit, pro lepší pochopení rytmu), snímky animace se v této aplikaci také počítají s frekvencí závislou na tempu skladby. Jsou spočítány tři snímky pro jednu dvaatřicetinovou notu, tedy konkrétně pro tempo 120 BPM je to jeden snímek za 20.83 ms.

Při testování aplikace na stroji s procesorem AMD Athlon 1.4 Ghz, na platformě MS Windows XP a grafickou kartou Nvidia Geforce 2 MX byl výpočet pohybu všech čtyř končetin bez zapojení orientace koncového efektoru, zvládnut v reálném čase i pro nejvyšší povolené tempo skladby 200 BPM. Což znamená 12,5 ms na jeden snímek. Při zapojení výpočtu orientace koncového efektoru již ale docházelo k výpadkům a čas výpočtu v některých případech značně přesahoval požadovanou mez na jeden snímek.

5.3.6.7 Problémy IK modulu a jejich řešení

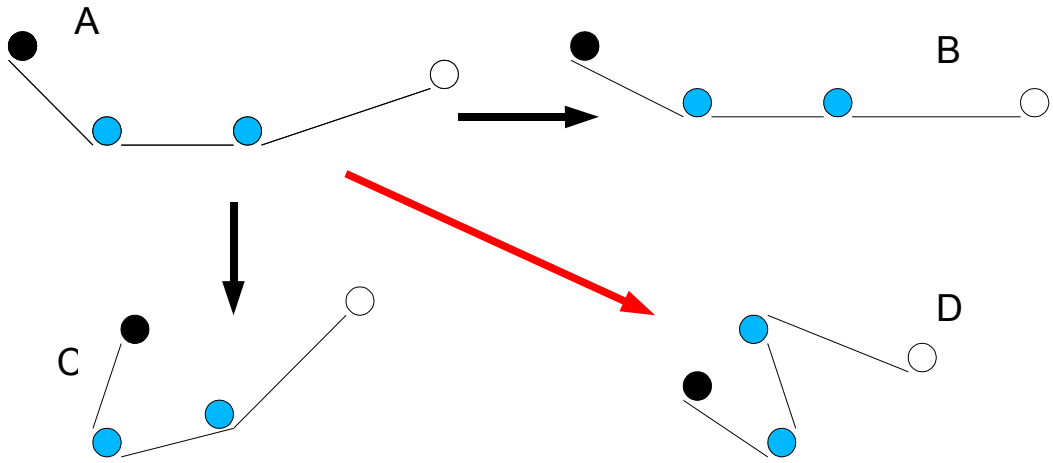
5.3.6.7.1 Nedostatečná omezení polohy rukou

Při použití IK modulu pro výpočet animace bubeníka se vyskytlo několik problémů zásadního charakteru. Zejména je to skutečnost, že IK modul sice respektuje nastavená anatomická omezení, avšak neřeší již účel a styl pohybu. To znamená, že IK modul se snaží přiblížit koncový efektor do cílové destinace nejjednodušším možným způsobem v rámci daných omezení. To může v některých situacích působit velmi nepřirozeně. Při úderu, který začíná vysoko, třeba nad činelem a končí nízko, například na virblu, reálný bubeník stáhne ruku k sobě a do virblu udeří s loktem poměrně nízko a u těla. Oproti tomu, IK modulu se jeví jako nejjednodušší řešení nechat loket vysoko a ruku ohnout dolu, takže palička se sice svým koncem dotýká virblu, ale míří na něj shora, místo aby udeřila spíše naplocho.

Ilustrace 22: **Nedostatečná omezení polohy rukou.** Pozice ruky sice ještě je anatomicky reálná, avšak nikoliv přirozená pro hru na bicí.

Tento problém je nejvýraznější u pohybů, které jsou vedeny způsobem, že koncový efektor se pohybuje proti zbytku končetiny směrem k tělu. Je to tím, že pokud se efektor vzdaluje, končetina se narovnává a prostor možných řešení k dosažení cílové pozice se zmenšuje. Pokud se však efektor pohybuje k tělu, počet možných natočení končetiny roste a roste i pravděpodobnost, že algoritmus vybere řešení, které nebude přirozené pro prováděný pohyb.

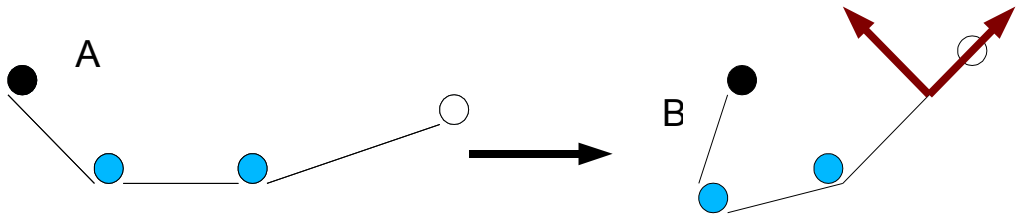
Tento fenomén se bohužel nedá uspokojivě vyřešit pomocí nastavení limitů a tuhostí v kloubech i když by teoreticky právě k řešení takových situací měly být určeny. Lze sice nastavit omezení tak, aby loket nešel přes určitou mez, ale pak zase ruka na některé bubny dosáhne pouze za cenu nepřirozeného natočení, kterého se IK modulu ani nemusí podařit dosáhnout. A po rozsáhlém zkoušení nastavit tuhosti tak, aby ovlivňovaly správně pořadí natočení kloubů, musím konstatovat, že při bubnování není možné najít vždy vyhovující nastavení, jelikož priorita pořadí natočení kloubů se u různých úderů mění. A nastavení tuhostí se projeví také v pomalejším dosažení cílové pozice.



Ilustrace 23: **Nedostatečná omezení polohy rukou.** Černé kolečko – základní uzel, bílé – efektor, modré – mezilehlé klouby. Při pohybu efektoru od základního uzlu se paže natahuje a prostor možných natočení se zmenšuje. Nastavení do cílové polohy B proběhne v pořádku. Při pohybu efektoru směrem proti základnímu uzlu se prostor řešení zvětšuje a hrozí, že místo správného řešení C algoritmus zvolí D.

5.3.6.7.2 Omezení polohy rukou pomocí orientace koncového efektoru

Pro řešení tohoto jevu se nabízí několik možností. Jako první bylo vyzkoušeno zapojení orientace koncového efektoru do výpočtu. Orientace paličky a tím i zápěstí by tak byla dána a zbytek ruky by již neměl příliš velkou volnost, aby se natočil do nepřirozené polohy. Objevilo se zde ale několik úskalí, která znemožnila použití této metody



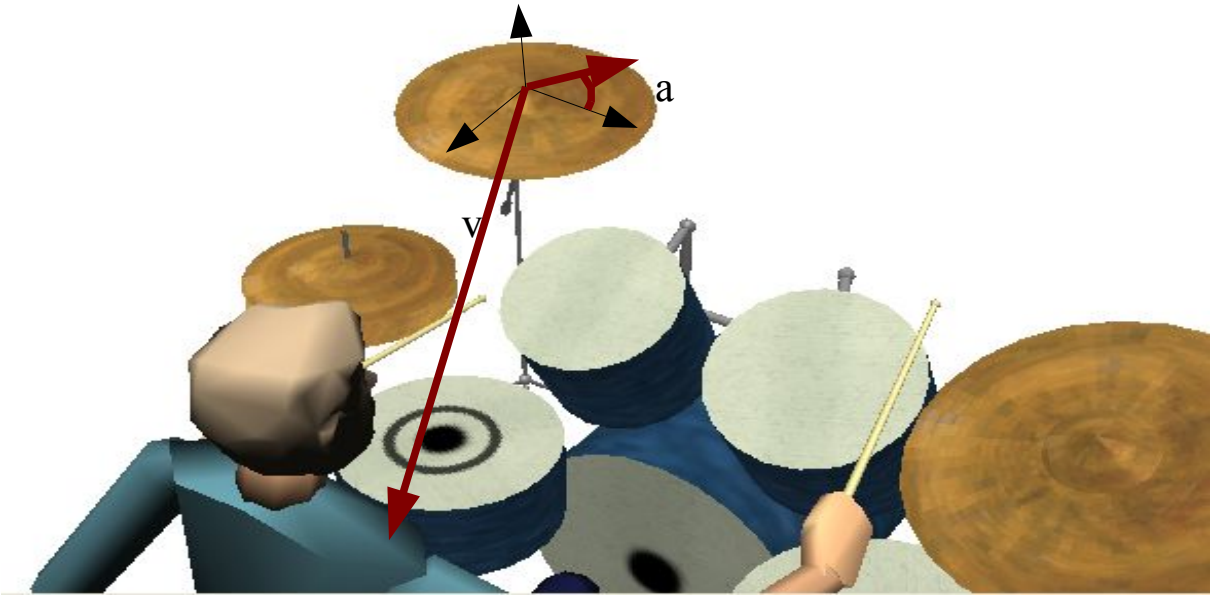
Ilustrace 24: O**mezení polohy rukou pomocí orientace efektoru.** Pokud je dána orientace efektoru, je dána i orientace segmentu mezi ním a dalším kloubem v paži. Množina řešení a tím i pravděpodobnost nepřirozeného nastavení kloubů pro dosažení cílové pozice se výrazně zmenšuje.

Klouby ruky se sice většinu času podaří udržet v přirozených mezích, jelikož orientace paličky jim nedovolí se z nich příliš vzdálit, ale jistá volnost pohybu v lokti a v rameni stále zůstává a při pohybu, kdy efektor tlačí na zbytek ruky stále může dojít k jejich nepřirozenému natočení.

Dále se bohužel nepodařilo uspokojivě vyřešit způsob určení orientace konce paličky. Ta se mění podle umístění cílového bubnu vzhledem k umístění humanoida a je pro každou ruku jiná. Určení úhlu, který má svírat palička s plochou bubnu problém není. Známe pozici i natočení bubnu a následný analytický výpočet není příliš složitý. Nepříjemnou skutečností ale je, že nelze jednoduše určit natočení paličky kolem osy bubnu, dokud není ruka v koncové pozici úderu. Toto natočení se totiž mění s polohou bubnu a ruky a jeho výpočet lze provést

až po přiblížení ruky do cíle. Tuto komplikaci se mi podařilo obejít pouze za cenu jistých zobecnění a nepřesností.

A to buď tak, že určím úhel natočení kolem osy X, který má palička s bubnem svírat a složím ho s vektorem, který určuje pozice cílového hitpointu a pozice ramene humanoida. To zajistí, že palička udeří do bubnu pod požadovaným úhlem a její orientace směrem k rameni bude celkem vyhovující. Poměrně často však nastane situace, kdy to, že palička míří směrem k rameni, nastaví ruku do polohy, která není zcela přirozená. To nastává v podstatě vždy, když se ruka dostane blíže k tělu a palička by tak měla být natočena více od těla.



Ilustrace 25: **Určení potřebné orientace paličky v momentu úderu.** Vektor orientace paličky při úderu do bubnu určíme složením vektoru z místa úderu k základnímu uzlu (k rameni) a úhlu, který má palička svírat s rovinou bubnu.

Druhou možností je provést výpočet bez orientace koncového efektoru a cíl pohybu při tom umístit lehce nad cílový hitpoint. Po dosažení tohoto bodu přesuneme cíl přímo do cílového hitpointu a provedeme výpočet i se zahrnutím orientace. Tu určíme podobně jako v předchozím případě s tím rozdílem, že neuvažujeme vektor směrem k rameni, ale pouze k zápěstí, což zaručí, že klouby ruky zůstanou navzájem přibližně v pozicích, kterých již dosáhly. Zde tedy nenastává problém s nevyhovujícím určením orientace jako v předchozím případě, zase však hrozí větší nebezpečí, že ruka zaujme špatnou pozici již při přesunu nad hitpoint, jelikož v té době není zapojená orientace.

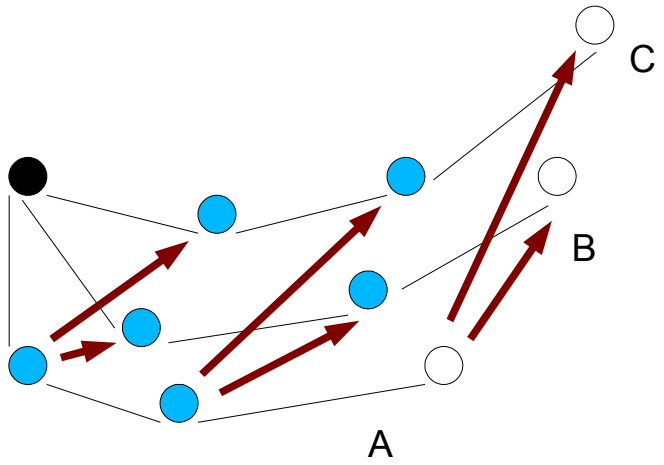
Ani jedna z těchto možností výpočtu orientace cílového efektoru tedy nepřináší zcela uspokojivé výsledky. Navíc jako velmi podstatná nevýhoda tohoto řešení se jeví fakt, že zapojení orientace do výpočtu s sebou nese zvýšené nároky na čas výpočtu přiblížení ruky do cíle, zejména kvůli potřebě vyššího počtu iterací k jeho dosažení. Toto zvýšení sice není znát asymptoticky, takže nad vyšší počet cca nad 10 kloubů není příliš znatelné. Ale působí jako malá multiplikativní konstanta, která se na průměrné složitosti výpočtu pro mnou použité 4 klouby na jedné ruce bohužel projeví téměř dvojnásobným zvýšením výpočetních nároků.

5.3.6.7.3 Dynamické nastavování limitů a tuhostí

Jako druhé řešení se jsem viděl možnost dynamického nastavování limitů a tuhostí v kloubech. Idea je taková, že pohyby by byly podle určitých kritérií (vzdálenost startu a cíle, vzdálenost cíle od ramene, výška bubnu, typ bubnu) rozděleny do kategorií a tyto kategorie by měly přednastavený script, který by v jednotlivých fázích pohybu měnil limity a tuhosti v kloubech tak, aby pohyb byl přirozený a bylo zaručeno, že ruka se dostane do požadované pozice. Nalezení těchto kategorií a jejich skriptů si však žádá rozsáhlé testování a experimentování a z časových důvodů proto bylo tato varianta opuštěna.

5.3.6.7.4 Použité řešení – výpočet přiblížení z klidové pozice

Řešení, které nakonec bylo v této aplikaci použito, je velmi jednoduché, ale spolu s drobnými úpravami řízení trajektorie přináší poměrně dobré výsledky. Jeho princip spočívá v tom, že výpočet přiblížení neprobíhá spojitě tak, že se koncový efektor přibližuje vždy z místa, kterého právě dosáhl do dalšího místa na trajektorii. Místo toho se výpočet provádí vždy z klidové pozice ruky do požadovaného místa. Zobrazovány jsou samozřejmě pouze polohy ruky po dosažení pozice na trajektorii pohybu, takže návraty do klidové polohy zůstávají pozorovateli skryty.



Ilustrace 26: **Řešení problému omezení rukou.** Paralelní kostra IK je po dosažení cílového bodu trajektorie vrácena do základní pozice a přiblížení k dalšímu bodu trajektorie probíhá opět odtud. Jsou však zobrazovány pouze koncové pozice.

Jelikož klidová pozice ruky je u těla a ruka v podstatě nikdy nebude více skrčena, tak při výpočtu ani nikdy nenastane situace, kdy koncový efektor tlačí na zbytek ruky a nenastanou nechtěná vychýlení s tím spojená. A protože výpočet je prováděn vždy z této klidové pozice, tak nehrozí ani při delších periodických pohybech vychýlení kloubů do poloh pro pohyb bubeníka nepřirozených.

Nevýhodou této metody je skutečnost, že pohyb ruky je v průběhu úderu poněkud plošší a méně přirozený, než při použití výpočtu z aktuální polohy ruky. Navíc, jelikož není zapojen výpočet orientace konce paličky, musí být pomocí správného průběhu trajektorie ošetřeno, že palička udeří do bubnu ve správném úhlu. V této aplikaci je použit postup, kdy trajektorie je vedena z poslední dosažené pozice do pozice 20 cm nad cílovým hitpointem. Potom je cílová pozice posouvána dolů až se dotkne hitpointu a následně vedena zpět 20 cm nad hitpoint a 10

cm zpět směrem k rameni. To zaručuje, že palička udeří do bubnu vždy shora pod určitým malým úhlem a je tím i simulován odraz paličky od bubnu.

5.3.6.7.5 Problém orientace paličky

V předchozích odstavcích byl zmíněn problém orientace paličky při úderu. Pokud je úder veden koncem paličky, tak by palička měla do bubnu udeřit tak, aby byla nad ním a aby s jeho plochou svírala poměrně malý úhel, podle pozorování skutečného pohybu se jedná o cca 5 až 30 stupňů. Pokud však je úder veden hranou paličky, tak by měla být skloněná opačně a do hrany bubnu udeřit pod úhlem cca -5 až -30 stupňů.

V odstavci pojednávajícím o možnosti omezení polohy rukou pomocí orientace paličky byla diskutována obtížnost zjištění správné orientace paličky a i výpočetní náročnost tohoto řešení a z výše uvedených důvodů není výpočet orientace paličky pomocí inverzní kinematiky implementován. V odstavci o použitém řešení je potom popsán způsob, jakým zajišťujeme, že úder koncem paličky je veden pod správným úhlem. Problém ale nastává v druhém případě, při vedení úderu hranou paličky do hrany bubnu.

Po experimentech s různým vedením trajektorie byl nalezen způsob, který po úderu hranou paličky zajišťuje, že úhel, který svírá s plochou bubnu, bude v požadovaných mezích. Je nutné vést trajektorii nejdříve do bodu pod úrovní bubnu přibližně 20 cm do strany, potom cíl posuneme na úroveň bubnu a tím zajistíme, že palička bude ve správném úhlu k jeho ploše. Následně již můžeme cíl přesunout do bodu úderu na hraně bubnu. Je to ovšem metoda velmi nedokonalá, jelikož výsledný pohyb působí na rozdíl od úderu koncem paličky, který je veden shora, značně nepřirozeně. Pro elegantnější řešení by zřejmě bylo potřeba použít výpočet orientace konce paličky.

5.3.7 Okno OpenGL

Toto je centrální okno aplikace. Je zde realizována vizualizace sady bicích a postavy bubeníka. Je zde primárně vytvářen objekt reprezentující sadu bicích, jsou zde vytvářeny instance těl jednotlivých typů bubnů, je zde načítána postava bubeníka a vytvořena její paralelní kostra IK. Je zde inicializováno rozhraní OpenGL, nastaveny parametry zobrazení 3D scény a scéna je zde vykreslována. Rovněž je v tomto okně snímán pohyb myši a interpretován pro ovládání kamery, případně v módu editoru pro sestavení sady pro pohyb s částmi sady. A konečně je zde i prováděn výpočet animace bubeníka.

5.3.7.1 Inicializace

Při inicializaci okna se provede několik úkonů. Je vytvořen objekt reprezentující prázdnou sadu a po skončení inicializace se aplikace pokusí nahrát sadu pojmenovanou dafault uloženou v adresáři Drumsets. Pokud sada není na této cestě nalezena, aplikace automaticky nastartuje v módu editoru sady bicích. Pokud je sada úspěšně nahrána, jsou přidány panely editoru skladby a aplikace je spuštěna v tomto módu.

Následně je nahrán model postavy. Děje se tak pomocí parseru VRML, který načte kostru humanoida podle standardu H-Anim a vytvoří její vnitřní reprezentaci. Také načte těla jejích jednotlivých částí. Podle této reprezentace je potom postava vykreslována. Po načtení postavy je ještě vytvořena paralelní kostra IK, na které probíhá výpočet animace. Podle nastavení této IK kostry je vždy po výpočtu aktualizována reprezentace VRML humanoida, která slouží k jeho zobrazení.

Při inicializaci OpenGL je vytvořena instance třídy, která popisuje těla jednotlivých typů bubnů možnostmi knihovny GLUT. Tyto popisné sekvence příkazů jsou zde zkompilovány do takzvaných display listů, které urychlují jejich pozdější provádění, tedy vykreslování bubnů.

Po vytvoření display listů, načtení postavy bubeníka a úspěšném načtení sady, je bubeník usazen na stoličku a jeho ruce a nohy jsou nastaveny do základní polohy. K tomu je použit aparát inverzní kinematiky. Pravá noha je nastavena na pedál kopáku a levá noha na pedál hi-hat (samozřejmě pokud je sada obsahuje). Ruce jsou pak nastaveny do klidové polohy, kdy jsou lokty u těla, předloktí vodorovně a paličky lehce nahoru. Když je bubeník nastaven v základní poloze, tak jsou uložena natočení ve všech kloubech kostry IK. Od této základní polohy se potom při animaci počítají všechna přiblížení k jednotlivým bodům trajektorie.

5.3.7.2 Tělo a pohyb bubnů

Každý buben pro své vykreslení volá podle svého typu potřebný display list nebo display listy, přičemž před jejich zobrazením se ještě provedou určité transformace a každý buben má také rutinu, která mění parametry těchto transformací a je periodicky volána časovačem s konstantní periodou nezávislou na tempu skladby, čímž je simulován pohyb bubnu. .Funkce, které mění polohu bubnu byly stanoveny experimentálně tak, aby co nejlépe vyjadřovaly pohyb skutečných bubnů a jsou následující:

Pohyb činelu je simulován rotací kolem osy X. Tento úhel natočení α se mění podle následující funkce:

$$\alpha = \frac{\frac{t}{5}\cdot\cos((A-t)\cdot 18\ )\cdot A}{80},\ \text{na počátku}\ \ A=\frac{140\cdot v}{128}\ ,\ t=80$$

kde *t* je počet cyklů časovače s periodou 50 ms od okamžiku úderu a *v* je síla úderu v rozpětí od 1 do 128.

Je tak simulován periodický pohyb kolem osy X a postupný útlum amplitudy, jejíž počáteční hodnota je závislá na síle úderu. Podobně činel ride je rozpohybován podle funkce:

$$\alpha = \frac{\sqrt{\frac{t}{5}}\cdot\cos((A-t)\cdot 18\ )\cdot A}{100},\ \text{na počátku}\ \ A=\frac{300\cdot v}{128}\ ,\ t=80$$

A konečně hi-hat provádí rotaci kolem osy zpodle funkce

$$\alpha = \frac{\sqrt{\frac{t}{3}}\cdot\cos((A-t)\cdot 4.5)\cdot A}{20},\ \text{na počátku}\ \ A=\frac{60\cdot v}{128}\ ,\ t=40$$

5.3.7.3 Zobrazení humanoida

Tělo humanoida není na rozdíl od těl bubnů vykreslováno pomocí display listů, ale pomocí takzvaných *vertex arrays*. To je technika, kdy v paměti jsou v polích uloženy souřadnice vrcholů a souřadnice normálových vektorů, případně i texturovací souřadnice a funkci OpenGL se předají ukazatele na tuto paměť a specifikuje se typ grafických elementů, které jsou těmito vrcholy zadány (trojúhelníky, čtyřúhelníky, apod.). OpenGL pak podle zadaných

parametrů vykreslí grafické elementy specifikované v těchto polích. Jelikož data jsou v paměti uložena hned za sebou, je tato technika velmi rychlá.

5.3.7.4 Řízení animace

V tomto modulu je spuštěn časovač, který s konstantní frekvencí překresluje scénu a druhý časovač s konstantní frekvencí, kterým jsou řízeny pohyby bubnů. Tento časovač navíc řídí i pohyb hlavy.

Okno OpenGL je pomocí slotů a signálů QT propojeno s dalšími moduly aplikace. Pokud je aplikace v módu editoru sady bicích, tak při pohybu myši jsou vysílány signály sloužící pro manipulaci s částmi sady. Modul má sloty pro příjem událostí spouštějících pohyby rukou, tedy pro každou ruku je vytvořen slot, který přijímá od plánovače pohybu signál určující, kam má ruka udeřit. Dále je zde slot, který je napojen na signál vysílaný z callback funkce knihovny FMOD, volané časovačem této knihovny při přehrávání skladby. V tomto slotu je prováděn výpočet trajektorie úderu a následný výpočet polohy ruky pomocí inverzní kinematiky.

Trajektorie úderu je vedena tak, že pozice konce paličky je nejprve po jednu šestnáctinovou notu přesouváno ze současné pozice k cílovému hitpointu. Z této dráhy je zobrazeno šest okének. Potom je pro simulaci odskočení paličky cíl přesunut nad buben a tato dráha je zobrazena ve třech okénkách, tedy během jedné dvaatřicetinové noty. Pokud je ruka více jak jeden takt v klidu, je pomalu (během půl taktu) přesunuta do klidové pozice.

5.3.8 Plánovač pohybu

5.3.8.1 Spouštění pohybů

Pro spouštění pohybů s požadovaným předstihem plánovač obsahuje potřebnou frontu událostí. Ta má délku tří intervalů skladby a buňky jednotlivých cyklů obsahují údery pro ruce a nohy v daném intervalu a mají následující významy: první cyklus jsou události, které byly právě načteny a rozhoduje se, jak budou přiděleny. Po jejich přidělení se ještě před dalším cyklem spustí pohyb rukou. V druhém se u událostí náležících nohám, vyšle signál pro spuštění pohybu pedálu. V třetím cyklu dochází ke kontaktu paličky a bubnu a jsou vyslány signály pro přehrání zvuku a pro spuštění pohybu bubnu.

5.3.8.2 Přidělování cílů rukám

Údery je nutné rukám přidělovat s ohledem na techniku hry a na možnost vzájemné kolize rukou. Za tímto účelem je zkoumána priorita bubnů, do kterých má být veden úder, poloha těchto bubnů a historie úderů u každé ruky. Rozhodovací algoritmus je následující:

Algoritmus **plánovač_pohybu**
Vstup:
počet cílů
pri1 – priorita prvního bubnu
pri2 – priorita druhého bubnu
hit1 – cílový hitpoint na prvním bubnu
hit2 – cílový hitpoint na druhém bubnu
hi-hat – je TRUE pokud aspoň jeden hitpoint je umístěn na činelu hi-hat, jinak je FALSE
last – ruka, která naposled udeřila do činelu hi-hat nebo 0 pokud poslední úder nebyl do činelu hi-hat
stilR – počet nejkratších intervalů skladby, po které byla pravá ruka v klidu
stilL – počet nejkratších intervalů skladby, po které byla levá ruka v klidu
Výstup:
hitR – cílový hitpoint levé ruky nebo 0
hitL – cílový hitpoint pravé ruky nebo 0
Metoda:

```
hitR = 0
hitL = 0
pokud (počet cílů = 0) konec //žádná ruka
pokud (počet cílů = 1)          //pouze jedna ruka
        hitR = hit1
        pokud (hi-hat = TRUE)       //na činelu hi-hat se ruce střídají
                pokud (last = pravá)
                        hitR=0
                        hitL=hit1
        jinak pokud (-2 < pri1 < 2)    //jinak se rozhodne podle priority
                pokud (stilR < 2)       //a podle toho, jak byla která ruka dlouho v klidu
                        hitR=0
                        hitL=hit1
        jinak pokud (pri1 < -1)
                pokud (stilL > 1)
                        hitR=0
                        hitL=hit1
        jinak pokud (pri1 > 1)
                pokud (stilR < 2)
                        hitR=0
                        hitL=hit1
jinak   //obě ruce
        hitR=hit1
        hitL=hit2
        pokud (pri1 < pri2)    //zohledníme prioritu
                hitL=hit1
                hitR=hit2
        jinak pokud (pri1 = pri2)       //zohledníme i pozici
                pokud (pozice hit1 je vlevo od pozice hit2)
                        hitL=hit1
                        hitR=hit2
        pokud (pozice hitL je vpravo a nad hitR)
                prohoď hitL, hitR
konec
```

6. Testování

6.1 Návrh experimentu

Výsledky naší úlohy není možné vzhledem k jejímu praktickému charakteru objektivně posoudit. Pro ověření funkčnosti aplikace a zejména jejího výukového záměru byl vytvořen uživatelský test. Jeho úkoly byly následující:

1. Posuďte na defaultní sadě bicích a připravených skladbách kvalitu a věrohodnost animace bubeníka z hlediska estetického a z hlediska hry na bicí.
2. Vyzkoušejte s defaultní sadou bicích složit vlastní rytmy a zjistěte, zda předvedení rytmu programem splňuje vaše očekávání či nikoliv.
3. Vyzkoušejte sestavit vlastní sadu bicích a posuďte práci v editoru k tomu určeném. Potom zopakujte druhý bod.
4. Pokuste se posoudit, zda vizuální předvedení rytmu vám pomohlo rytmus pochopit, případně zahrát.

Účastník testu byl pouze seznámen se základním ovládáním aplikace, zbytek testu již procházel bez cizí pomoci. Po dokončení testu se účastník ke každému bodu volně písemně vyjádřil.

Aplikaci testovalo pět subjektů jejichž demografické rozložení shrnuje následující tabulka, přičemž zkušenosti jsou hodnoceny známkami 1 až 5, kde 1 je nejlepší.

Věk	***Zkušenosti s hrou na bicí***	***Zkušenosti s výpočetní technikou***	***Hra na jiný hudební nástroj***
25	*4*	*1*	*ne*
27	*3*	*2*	*ne*
24	*2*	*4*	*ne*
26	*3*	*2*	*ano*
26	*3*	*3*	*ano*

Tabulka účastníků testu

6.2 Výsledky testu

Testovaných subjektů bylo poměrně málo, avšak u jednotlivých bodů testu se v odpovědích vyskytovaly často podobné dojmy. Jejich shrnutí je následující:

1. Uživatelé považují animaci většinou za věrohodnou, pouze v některých případech úderů a nastavení bubnů nebyl dojem dostatečně přirozený. Vizuální dojem velmi pozitivně podporuje animace bubnů.
2. Přidělování úderů rukám vypadá ve většině případů smysluplně, avšak některé případy přidělení nepůsobí jako nejefektivnější řešení.
3. Editor pro sestavení sady bicích je hodnocen veskrze kladně a to svým záměrem umožnit sestavení vlastní sady i svým provedením. V druhé části úkolu bylo kladně hodnoceno automatické přizpůsobení programu uživatelské sadě.
4. Uživatelé se shodli, že při zpomalení předváděného rytmu aplikace skutečně napomáhá k jeho analýze a jeho pochopení.

6.3 Závěry z testování

Experimenty prokázaly, že aplikace splňuje původní záměr. Aplikace uživateli umožňuje sestavit vlastní sadu bicích a úspěšně se jí přizpůsobí. Její výuková funkce byla hodnocena v naprosté většině případů kladně a i estetický dojem z animace byl pozitivní.

7. Závěr a práce do budoucna

7.1 Zhodnocení

V tomto dokumentu jsme se zaměřili na specifikaci aplikace, která slouží pro výuku hry na bicí. Nastínili jsme smysl takové aplikace a uvedli jsme principy a techniky v oblasti hry na bicí a v oblasti animace. Rozebrali jsme problémy použitého animačního modelu využívajícího inverzní kinematiku a nabídli jsme jejich řešení. Nakonec jsme popsali architekturu a konkrétní implementaci cílů této aplikace.

Finální aplikace umožňuje:
- sestavení vlastní sady bicích s možností použití vlastních zvuků
- editaci rytmů pro libovolnou sadu
- vizualizaci rytmů v reálném čase
- zajišťuje, že technika hry se vždy přizpůsobí konkrétní sadě.
- Zpomalení skladby a manipulaci s kamerou pro snadnou analýzu zadaného rytmu

V aplikaci byly použity algoritmy inverzní kinematiky a byl vytvořen jejich modul v jazyce C++, jež může být jednoduše využit jinými projekty. V rámci této práce také vznikl parser načítající ze souborů VRML postavu humanoida splňující standard H-Anim 1.1. I ten může být snadno použit v jiné aplikaci.

Pro vyhodnocení funkčnosti aplikace byl vytvořen uživatelský test. Jeho výsledky potvrzují, že aplikace smysluplně vizualizuje zadané rytmy pro uživatelem vytvořenou sadu a lze ji použít pro lepší pochopení a naučení určitého rytmu.

V zadání naší práce byly dva požadavky:
- využití algoritmů inverzní kinematiky
- vytvoření výukového nástroje pro hru na bicí

První požadavek byl splněn implementací modulu pro výpočet inverzní kinematiky, který je používán pro výpočet animace rukou.

Splnění druhého požadavku nelze zatím objektivně zhodnotit. Nicméně subjektivní uživatelský test ukazuje, že koncepce a implementace aplikace je správná.

Tyto body byly hlavními požadavky specifikované pro naši práci. Můžeme tedy konstatovat, že zadání naší práce bylo úspěšně vyřešeno.

7.2 Práce do budoucna

Možnosti rozšíření této aplikace jsou poměrně široké. Na tomto místě zmíníme tři body, které by mohly být v budoucnu implementovány.

- **Využití key-frame animation**

Toto je poměrně zajímavá varianta řešení problému nedostatečného omezení polohy rukou, který byl popisován v kapitole o implementaci animace rukou.

Použití inverzní kinematiky při výpočtu animace, bylo potřebné díky požadavku adaptability na jakékoliv rozestavení bicích. To znamená, že program má být schopen zjistit koncové

pozice úderů pro libovolnou sadu a dráhy pohybů humanoida jim přizpůsobit. Jakým mechanismem má být pohyb veden mezi těmito pozicemi, již ale určeno není. Stačí tedy pomocí aparátu inverzní kinematiky najít koncové pozice úderů (čímž máme zajištěnu adaptabilitu na libovolnou sadu) a pohyb mezi těmito pozicemi lze řešit již jiným výpočetně méně náročným způsobem; nabízí se využití techniky key-frame animation.

- **Problém orientace paličky**

V současné verzi aplikace není tento aspekt hry na bicí implementován. Zde by bylo potřeba výrazně rozšířit možnosti modulu inverzní kinematiky a řízení trajektorie ruky.

- **Vylepšení 3D modelu bicích a humanoida**

Vizuálně přitažlivější a detailnější modely bicích a humanoida by jistě zvýšily estetický dojem z celé animace. Bylo by však potřeba implementovat import některého externího formátu (3D Studio, Maya), nebo rozšířit možnosti stávajícího parseru jazyka VRML.

- **Dynamika pohybu**

Pro výrazně reálnější pocit z pohybu bubeníka by bylo zajímavé použití technologie inverzní dynamiky. Inverzní kinematika samotná totiž respektuje anatomická omezení nastavená v kostře, avšak nikoliv záměr a smysl simulovaného pohybu. Pokud by se do výpočtu zapojil i dynamický element, výsledná animace by měla více odpovídat reálnému pohybu. Navíc by bylo možné jemněji simulovat techniku hry, například odrazy paličky po úderu nebo různý charakter pohybu při různé síle úderu.

8. Poděkování

Autor tohoto textu děkuje všem, kteří se jakoukoli měrou podíleli na vzniku tohoto textu a implementaci rozebírané aplikace, či autora v této činnosti nějak podporovali.

Obzvláště ing. Adamu J. Sporkovi a ing. Vladimíru Štěpánovi za cenné rady a vedení při vzniku práce, Bohuslavu Francovi za poskytnutí algoritmů inverzní kinematiky a podpory při, jejich využívání a úpravě, ing. Petru Kadlecovi za cenné rady při implementaci úlohy a všem zúčastněným za ochotu a trpělivost při testování aplikace.

9. reference

[Gra99] Grace R.: *Hudba a zvuk na počítači,* Praha: Grada, ISBN 80-7169-519-X, 1999

[Kur02] Kurfürst P.: *Hudební nástroje,* Praha: Toga, ISBN 80-902912-1-X, 2002

[Hol64] Holzknecht V., Poš Vl.: *Kniha o hudbě,* Praha: Orbis, 1964

[Orc04] Orcígrová A.: *Bicí souprava,* Praha, prezentace, 2004

[Mic98] The Microsoft Corporation: *MSDN Library Visual Studio 6.0 release,* Software development enviroment documentation, 1998

[OGL15] Segal M., Akely K.: *The OpenGL Graphics System: A Specification (Version 1.5)* [online], URL: http://www.opengl.org/documentation/specs/version1.5/glspec15.pdf,

[NeV91] Nenadál K., Václavíková D.: *Turbo C - popis jazyka,* Praha: Grada, ISBN 80-85424-08-8, 1991

[Her00] Herout P.: *Učebnice jazyka Java,* České Budějovice: Kopp, ISBN 80-7232-115-3, 2000

[Fra05] Franc B.: *Iverzní kinematika*, Praha 2004, Diplomová práce na FEL ČVUT, vedoucí práce Vladimír Štěpán

[IK04] *Inverse Kinematics, URL:* http://freespace.virgin.net/hugo.elias/models/m_ik.htm, 2004

[Han03] *Specification for a Standard VRML Humanoid,* URL: http://h-anim.org/Specifications/H-Anim1.0, 2003

[Vrl97] Mezinárodní standard ISO/IEC 14772-1:1997 – VRML 97

[QT00] Trolltech: *QT Reference Documentation,* Software development documentation, 2000

[FM00] Firelight Technologies: *FMOD Reference Documentation,* 2000

10. Dodatek A – Instalace programu a ovládání

10.1 Instalace programu

Program je možné spustit přímo z CD, případně jej lze nakopírovat na disk a spouštět odtud. Důležité je, aby byl nakopírován přesně celý obsah CD i s podadresáři. Aplikaci spouští soubor vdr.exe.

10.2 Hardwarové a softwarové požadavky

Aplikace je zkompilována pro platformu Microsoft Windows. Její běh je vyzkoušen na verzích XP, 2000 a NT. Na verzi Windows 98 byly zaznamenány problémy s časovačem knihovny QT.

Aplikace ke svému provozu potřebuje grafickou kartu podporující OpenGL a zvukovou kartu. Pro hladký běh programu byl jako plně dostačující konfigurace otestován stroj s procesorem AMD Athlon 1.4 Ghz, paměťové požadavky jsou minimální pouze je nutné vzít ohled na celkovou velikost zvuků (po dekomprimaci), které nahrajeme k sadě bicích. Minimální podporované rozlišení obrazovky je 1024x768 bodů.

10.3 Ovládání aplikace

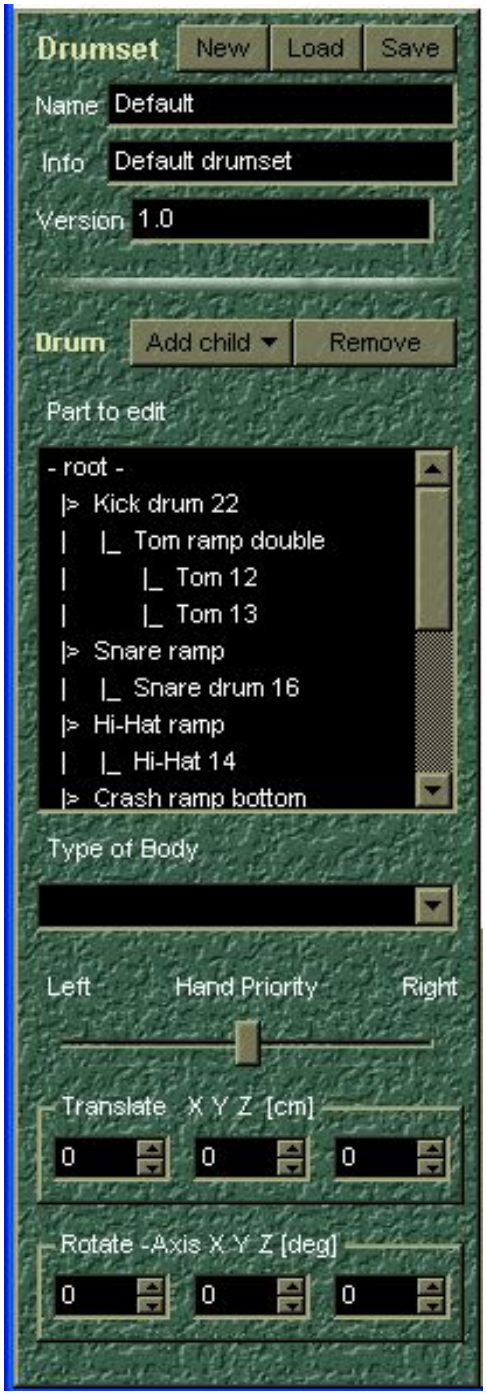
Aplikace funguje ve dvou režimech – v režimu editoru skladby a v režimu editoru sady bicích.

10.3.1 Editor sady bicích

10.3.1.1 Hierarchie sady

V editoru je možné vytvořit novou (prázdnou) sadu, uložit ji, nebo uloženou sadu opět nahrát. Sadu je možné popsat názvem, její verzí a dodatečnými informacemi. V každém okamžiku nám editor ukazuje seznam vytvořené hierarchie součástí sady, ve kterém můžeme zvolit konkrétní část, se kterou chceme manipulovat, případně, ke které chceme přidávat potomky. Prázdná sada obsahuje pouze kořenový uzel bez zobrazitelného těla. K některým částem sady je možné přidávat potomky, případně lze část ze sady odstranit. Při odstranění části sady se odstraní i celý strom jejích potomků. Každému bubnu můžeme zadat jméno a přiřadit zobrazitelné tělo. Výběr těla součásti sady probíhá pomocí menu, jehož nabídka je nastavena podle daného kontextu – typu bubnu. Pokud jsme do sady přidali buben typu činel, pak jeho tělo se dá změnit pouze na jiný typ činelu. Navíc, potomky můžeme přidávat také pouze pomocí kontextového menu a to pouze v maximálním počtu určeném pro danou součást. Takže k rampě pro dva přechody lze přidat pouze různé typy přechodů a to maximálně dva, k rampě na činel pouze různé typy činelů a to maximálně jeden a například k činelu nelze přidat nic. Na začátku lze přidat pouze kopák, stojan na činel, stojan na hi-hat, stojan na virbl nebo stojan na kotel.

U každé části sady můžeme nastavovat geometrické transformace – polohu a natočení. Děje se tak buď pomocí pohybu myší v hlavním grafickém okně, nebo jemněji pomocí tlačítek a číselných polí na panelu editoru. I zde jsou nastavena jistá omezení. Každým typem bubnu lze hýbat pouze v určitých směrech a natáčet ho podle určitých os. Navíc tento pohyb je ještě omezen jistým rozsahem. Je tak zajištěno, že model sady zůstane celistvý.



Ilustrace 27: **Editor sady bicích – panel hierarchie sady**

Jako další usnadnění práce s programem funguje automatické přidání potomka do správného místa na rodiči. Tedy střed činelu se umístí na konec rampy, nebo hi-hat na vršek stojanu. Uživateli pak stačí pouze lehce doladit natočení bubnů a umístit rampy na správná místa ve scéně.

U každého bubnu také můžeme nastavit preferenci, kterou rukou se má provést úder do tohoto bubnu. Tento atribut je nastaven automaticky podle typu bubnu při jeho přidání do hierarchie a může pak být podle potřeby změněn.

10.3.1.2 Hitpointy

Ke každému bubnu je automaticky přidán defaultní hitpoint, či hitpointy. Hitpointy nelze definovat na všech částech sady, ale pouze na částech k tomu logicky určených. Tedy ne na stojanech a rampách, kromě stojanu na hi-hat, kde je automaticky přidán hitpoint pro levou nohu.

Každý hitpoint může mít dále definovánu jednu nebo více úrovní dynamiky úderu. Ke každé z těchto úrovní lze nahrát zvuk a nastavit rozsah síly úderu. Při různých silách úderu do bubnu se totiž charakter zvuku mění a toto řešení zajišťuje větší realističnost než pouhé nastavení hlasitosti zvuku. Rozsahy těchto úrovní lze nastavit. Nicméně uživatel se může spokojit pouze s jednou úrovní dynamiky.

Ilustrace 28: **Editor sady bicích – panel hitpointů**

10.3.1.3 Ukládání sady

Ukládání sady je řešeno následovně. Popis složení částí sady, hitpointů a úrovní dynamiky je uloženo do jednoho textového souboru. Potom je vytvořen adresář jehož jméno je složeno ze jména souboru sady a k němu je připojeno slovo sounds. Do tohoto adresáře jsou nakopírovány všechny zvuky v dané sadě použité.

10.3.2 Editor skladeb

Editor se skládá ze dvou částí (panelů). Ovládání prvního panelu umožňuje nahrát, uložit nebo vytvořit novou skladbu a nastavit její parametry. Dále jsou zde prvky pro manipulaci s takty a nastavení parametrů vkládaných not, offsetu a síly úderu. Na panelu je také umístěno ovládání přehrávání skladby. V editoru lze samozřejmě nahrát libovolnou sadu bicích, potom se vytvoří i nová skladba. Lze vytvořit novou skladbu, uložit ji, nebo uloženou skladbu znovu nahrát. Při ukládání skladby lze zvolit, zda se do adresáře, kam se skladba ukládá, zkopíruje i celá sada s ní spojená, nebo zda se pouze uloží cesta k této sadě. Obě volby mají své opodstatnění. První zajišťuje přenositelnost skladeb na jiné počítače, druhá šetří diskovým prostorem. Pokud se při nahrání skladby nenalezne sadu na odpovídající cestě, je vyvolán dialog, který se uživatele dotáže na její umístění.

10.3.2.1 Struktura skladby

Skladba se skládá z jednotlivých oddělených bloků. Editor v daném okamžiku zobrazuje a umožňuje editovat vždy pouze mřížku jednoho bloku. V editoru lze pro každou řádku mřížky nastavit hlasitost, nebo jí ztlumit, případně zvolit, že bude přehrávána pouze tato řádka. Bloky lze přidávat, mazat a kopírovat, nelze měnit jejich posloupnost. Skladba se může skládat z libovolného počtu bloků.

Je možné zvolit, zda skladba bude ve čtyřčtvrťovém nebo tříčtvrťovém rytmu. Lze rovněž nastavit její tempo. To je omezené rozsahem 1 až 200 BPM.

Pro zadávané noty lze nastavit sílu úderu v rozsahu 1 až 128 a u čtyřčtvrťového rytmu ještě offset, který může být implicitně nula, nebo plus nebo minus jedna třetina (zde jedna třetina z dvaatřicetinové noty). To je z toho důvodu, aby bylo možné zadávat trioly (rozdělí interval čtyř úderů rovnoměrně na tři údery), které jsou ve hře na bicí poměrně často používané. U tříčtvrťového rytmu offset umožněn není.

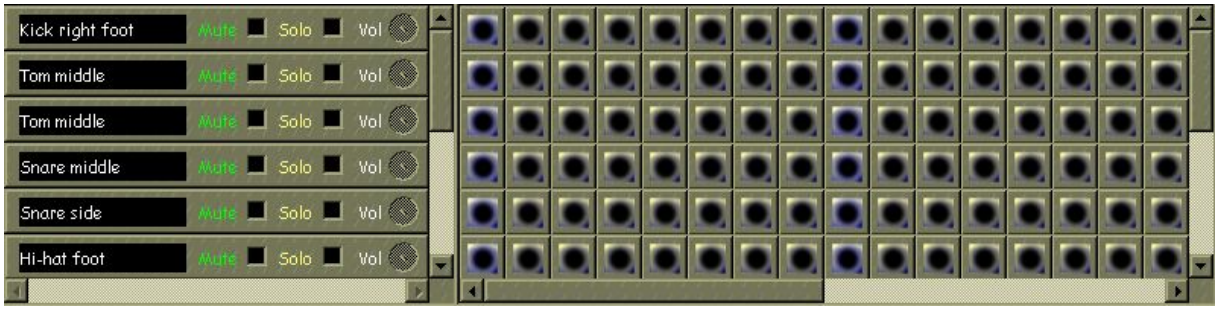
Ilustrace 29: **Editor skladby – ovládací panel**

10.3.2.2 Mřížka

Druhá část editoru je panel zobrazující mřížku aktuálního bloku skladby a ovládací prvky jejích jednotlivých řádků. Blok je zobrazen jako mřížka tlačítek, jejíž délka odpovídá délce aktuálně zvoleného bloku, a kde každý řádek odpovídá jednomu hitpointu sady. Při stisknutí tlačítka myší se na odpovídající pozici mřížky vloží nota s momentálně nastavenou sílou úderu a offsetem. Při opakovaném stisknutí tlačítka se snižuje síla úderu zadané noty, po dvaceti až k nule. Potom je nota vymazána. Při stisku pravého tlačítka myši se nota ihned

vymaže. Síla úderu je na tlačítkách znázorněna intenzitou světla umístěného na tlačítku. Offset je pak znázorněn posunutím tohoto světla.

Editor také automaticky kontroluje, zda nejsou porušena omezení na vkládání not (nelze zadat noty pro více jak dvě ruce najednou a podobně). Na jejich porušení je uživatel upozorněn výstražným signálem a není mu v takovém případě povoleno notu vložit.



Ilustrace 30: **Editor skladby – mřížka pro zadávání úderů**

11. Dodatek B – Přehled zdrojového kódu

11.1 Třídy uživatelského rozhraní

- vdrWindow – hlavní okno aplikace
- QGLWindow – okno OpenGL. Zobrazuje 3D scénu, uchovává objekt postavy bubeníka a objekt sady bicích, řídí animaci
- DreditWidget – Panel editoru skladby. Obsahuje popis struktury patternu a skladby
- Dreditor – Panel pro zobrazení a editaci patternu.
- BeatLine – Jedna řádka patternu
- ChanLine – Ovládání jedné řádky patternu
- DrsetWidget – Panel editoru sady bicích
- HitpointWidget – Panel pro editaci hitpointů

11.2 Třídy VRML parseru a reprezentace humanoida

- Parser – parser VRML, načte soubor VRML popisující humanoida a vytvoří jeho reprezentaci v programu
- Node – virtuální třída reprezentující uzel VRML, ale i obecně prvek scény. Od ní jsou děděny třídy Joint, Segment, Site a Drum
- Human – ekvivalent uzlu VRML typu Humanoid
- Joint – ekvivalent uzlu VRML typu Joint
- Segment – ekvivalent uzlu VRML typu Segment
- Site – ekvivalent uzlu VRML typu Site
- Body – popis těla segmentu, obsahuje strukturu Shape, která uchovává souřadnice vrcholů trojúhelníků reprezentujících tělo segmentu

11.3 Třídy reprezentace sady bicích

- DrumSet – reprezentuje hierarchii sady bicích
- Drum – reprezentuje buben, nebo jinou část sady. Popisuje pohyb bubnu a volá display listy pro jeho zobrazení
- DrumBody – obsahuje display listy popisující těla jednotlivých typů bubnů
- Timage – slouží k nahrání obrázku ve formátu tga a vytvoření textury v OpenGL

11.4 Třídy modulu inverzní kinematiky

- ikPoint – reprezentuje bod
- ikVector – reprezentuje vektor v 3D prostoru a obsahuje funkce pro operace s vektory
- ikMatrix – reprezentuje matici 4x4 a obsahuje funkce pro operace s maticemi
- ikDestination – reprezentuje destinaci určující pozici a orientaci v této pozici
- ikAxisRotation – rotace kolem obecné osy
- ikElement – ik ekvivalent uzlu VRML. Od něj jsou děděny třídy ikJoint a ikSite
- ikJoint – ik ekvivalent uzlu VRML typu Joint
- ikSite – ik ekvivalent uzlu VRML typu Site
- ikChain – vytváří řetěz uzlů od základního uzlu k efektoru nejkratší cestou
- ikIteration – provede jednu iteraci výpočtu přiblížení efektoru k cílové destinaci
- ikFrame – provede celý výpočet přiblížení tím, že volá v cyklu ikIteration

11.5 Ostatní třídy

- NoteParser – plánovač pohybů. Přiřazuje údery rukám
- Event – událost reprezentující jeden úder. Je zasílána NoteParseru.
- SoundEngine – zvukový subsystém nahrávající a přehrávající zvuky a vysílající signál časovače v callback funkci knihovny *FMOD*.
- TimerEvent – událost jejíž instance vysílá v callbacku knihovny FMOD signál pro řízení animace a přehrávání skladby

12. Dodatek C – Obsah CD

Adresář BParts	adresář obsahuje soubory *VRML* popisující těla jednotlivých částí humanoida.
Adresář Doc	obsahuje manuál a dokumenty pojednávající o vývoji programu
Adresář Drumsets	Obsahuje defaultní sadu bicích i se zvuky
Adresář Songs	Obsahuje ukázkové skladby
Adresář Sounds	Obsahuje zvuky uživatelského rozhraní
Adresář src	Obsahuje zdrojové soubory a projekt pro prostředí *Microsoft Visual Studio 6*
Adresář Textures	Obsahuje textury ve formátu bmp použité v uživatelském rozhraní a textury ve formátu tga použité na modelech bicích.
fmod.dll, glut32.dll, qt-mt230nc.dll	Knihovny potřebné pro běh aplikace
vdr.exe	Binární distribuce aplikace
Human.wrl	Soubor *VRML* popisující postavu bubeníka
Vdrtimer120.xm	Prázdná skladba sloužící k časování aplikace