Technická univerzita v Liberci

Fakulta mechatroniky a mezioborových inženýrských studií

Studijní program: B 2612 – Elektrotechnika a informatika

Studijní obor: 2612R011 – Elektrotechnické informační a řídící systémy

Vizualizace znakové řeči

Visialization of sign language

Bakalářská práce

Autor: **Jan Obrátil**

Vedoucí bakalářské práce: Ing. Josef Chaloupka, Ph.D.

Konzultant: Ing. Zbyněk Koldovský, Ph.D.

V Liberci dne 8. 5. 2006

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé bakalářské práce a prohlašuji, že **souhlasím** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mně požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do její skutečné výše).

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základe konzultací s vedoucím bakalářské práce a konzultantem.

Datum

Podpis

zde patří zadání bakalářské práce

Abstrakt

Tato bakalářská práce popisuje implementaci 3D grafického enginu, kterou lze následně použít pro vizualizaci znakové řeči. Program byl vytvořen za použití univerzálních knihoven OpenGL, standardních knihoven jazyka C a C++, grafických a vývojových nástrojů volně dostupných na internetu. Je zde rozvedena problematika 3D transformací v prostoru a metodika tvorby modelů a jejich pohybů včetně datové reprezentace v programu.

Klíčová slova: znaková řeč, znakový jazak, 3D grafika, Blender, Cal3D, 3D, C++, 3D transformace, quaternion, rotace ve 3D prostoru, tvorba pohybů ve 3D, OpenGL, tvorba 3D modelu lidského těla

Abstract

This work describes the implementation of 3D graphic engine that can be used for visualization of sign language. This implementation has been done by universal OpenGL graphic library, standard libraries of C and C++ language, graphics and developer tools free available from Internet. There are explained mathematical problems of 3D transformation in space and methodology of model creation and its data representation in program.

Keywords: sign language, 3D graphics, Blender, Cal3D, 3D, C++, 3D transformation, quaternion, rotation in 3D space, creation of move in 3D space, OpenGL, creation of 3D model of human body

OBSAH

Úvod

V dnešní moderní době je využití techniky, počítačů, každodenní praxí snad většiny z nás. Technika umožňuje realizovat nejrůznější projekty a analýzy, které byly ještě před nedávnem nerealizovanou teorií vědců. Za relativně krátkou dobu se podařilo mnohem více využít zvyšující se výkon výpočetní techniky, rychleji se rozvinuly stávající vědní obory a vznikly nové. Dynamicky se rozvíjejícím oborem je počítačová grafika v trojrozměrném prostoru.

Tato bakalářská práce prezentuje projekt využití trojrozměrné grafiky k zobrazení pohybujícího se modelu postavy k vizualizaci znakového jazyka.

Cílem tohoto projektu je nalézt řešení při vytváření prostředí vhodného pro znázornění znakové řeči pomocí 3D modelu člověka, který by měl být schopen animovat pohyb horních končetin podle zadaných slov, otáčet libovolně postavou a snadno modifikovat jeho vlastnosti, pohyby a funkčnost pomocí vstupních souborů.

Tento projekt obsahuje metodiku při vytváření programu, který má uvedené vlastnosti splňovat. Jsou zde řešeny všechny problémy, které bylo zapotřebí v projektu překonat. Výsledek bakalářské práce může být podkladem pro navazující práce realizující překlad českého jazyka do znakové řeči.

Znaková řeč byla vybrána pro její důležitost pro sluchově postižené lidi, aby nahradila pro ně nepopoužitelnou verbální komunikaci daleko srozumitelnější znakovou.

1 Jiné podobné systémy

V následujících odstavcích jsou představeny tři podobné systémy.

1.1 Sign Smith Studio

Sign Smith Studio[1] je produkt firmy Vcom3D, Inc. sídlící v USA. Jedná se o ucelené řešení tvorby znakového jazyka pomocí animované postavy a následné distribuce těchto dat.

Systém pracuje jako překladač psané angličtiny do amerického znakového jazyka[2]. Téměř jakýkoli text lze zadat do tohoto systému a výstupem je animace znakové řeči, kterou lze uložit do různých formátů:

- Videosekvence spustitelná v multimediálním přehrávači za použití běžně dostupných videokodeků.
- Animovaný GIF použitelný pro obohacení webových prezentací.
- Pomocí programu Sign Smith Showtime! na straně klienta se přímo zobrazí nativní datový soubor. Tato varianta lze použít pro zobrazení dat jak na vyměnitelných médiích tak pomocí streamování dat po Internetu.
- Pomocí programu Sign Smith Showtime! Server na straně serveru, který předzpracuje data a pošle je klientovi do webového prohlížeče. Tuto variantu je bohužel nepřenositelná mezi platformami, protože vyžaduje na straně klienta webový prohlížeč Microsoft Internet Explorer, je totiž použita ActiveX technologie.

Sign Smith Studio dále obsahuje kompletní knihovnu gest, výrazů obličeje, editovací nástroj pro tvorbu, úpravu, mazání a skriptování gest. Pro větší věrohodnost jsou synchronizovány pohyby gest s pohyby rtů a syntézy řeči s výrazem v obličeje pro dodání důrazu v určitých pasážích sdělení[1].

1.2 DePaul University American Sign Language Project

Tento projekt vznikl na univerzitní půdě v Chicagu v roce 1999 a dosud se na vývoji podílelo cca 46 vývojářů. Jedná se, stejně jako předchozí zmiňovaný systém, o překladač

[1] V materiálech dostupných na webu je tento produkt také nazýván Vcommunicator Studio. Pravděpodobně produkt Vcommunicator Studio byl přejmenován na Sign Smith Studio a část prezentací zůstala na webu se starým názvem.

[2] American Sign Language nebo ve zkratce ASL.

ze psané angličtiny do znakového jazyka pro anglo-americké jazykové prostředí.

V současné době se pracuje na vývoji nástroje, který by překládal mluvenou angličtinu do příslušných animací ve znakovém jazyce.

Na stránkách projektu už nebylo zveřejněno více informací, které by byly pro předmět této bakalářské práce zajímavé. Dostupné animace demonstrují pěkné grafické provedení modelu, které je velice realistické a čitelné[2].

1.3 Auslan Tuition System

Systém je vyvíjen v The University of Western Australia jako zástupce systému pro znakovou řeč v Australském znakovém jazyce[3].

Systém se skládá ze dvou programů

- Auslan Tuition System
- Auslan Sign Editor

První slouží pro zobrazování znakového jazyka a druhý pro editaci a přidávání nových gest.

Z prezentace na webu projektu vyplývá, že se jedná hlavně o program pro výuku znakového jazyka a je k tomu i patřičně uzpůsoben. Program nabízí i možnost dialogu vyobrazením dvou postav s animací jejich rozhovoru.

Animace v Auslan Tuition System fungují na principu keyframes, tedy klíčových pózách, které jsou nastaveny pro určitou pózu a dobu. V praxi tedy vytvoření příslušného posunku vypadá tak, že se nastaví čas a póza, kterou má postava v daný čas zaujmout a o nějaký čas později se nastaví další póza atd. Tím se vytvoří sada póz s časem, keyframů, které slouží jako kostra pohybu. Při přehrání posunku se polohy končetin mezi těmito keyframy interpolují a tím se dosáhne plynulého pohybu[3].

[3] Australian Sign Language nebo zkráceně Auslan.

2 Znaková řeč

Znakový jazyk je doplňkem verbální komunikace a běžně podvědomě používán, aniž by si to lidé uvědomovali. Každý podvědomě používá kývání při souhlasu či vrcení hlavou při nesouhlasu apod. Tyto posunky či mimika obličeje tvoří nezanedbatelnou část lidské komunikace a dokáže vyjádřit daleko výstižněji pocity, důraz či naléhavost sdělení. Pokud jsme odkázání pouze na znakovou řeč, pak je mimická složka ještě důležitější.

V 2. polovině 18. století se začaly objevovat církevní školy pro neslyšící, kde se vyučovala a rozvíjela znaková řeč a kde se dětem dostalo i vzdělání, které do té doby bylo výsadou církve a urozených lidí. Děti se učily znakový jazyk používaný mnichy zasvěcenými věčnému mlčení. Děti byly soustředěny do větších komunit, ve kterých mohly mezi sebou komunikovat a dále obohacovat znakový jazyk.

Koncem 19. století byl význam znakového jazyka zpochybněn a převládl názor, že i neslyšící se dokáže naučit mluvit a porozumět odezíráním ze rtů rodičů, nebude-li mít jinou možnost komunikace. Stačil k tomu jen zákaz znakového jazyka i přirozených posunků. Od padesátých let 19. století tento názor převládl a učitelé ve školách tento jazyk nesměli používat a to jak u nás, tak v jiných evropských zemích. Výsledkem byl izolovaný vývoj znakových jazyků na jednotlivých školách a tak vznikly nejen národní jazyky, ale i nářečí jednotlivých škol, které se od sebe podstatně liší.

Až v současné době je u nás snaha znakový jazyk sjednotit, přesto podoba českého znakového jazyka čerpající z pražského nářečí není jednotná a v praxi je možné narazit na odlišné znaky.

V posledních letech se změnil názor na výuku neslyšících včetně postoje veřejnosti. Zájem o znakový jazyk roste nejen u neslyšících. Znakový jazyk má velmi široké uplatnění, např. ve speciální pedagogice byla vypracována metoda, díky níž lze lépe komunikovat s autistickými dětmi, dokonce lze znakový jazyk použít při léčbě koktání[4].

2.1 Český znakový jazyk

Obecně bychom mohli gramatiku vyjadřování ve znakové řeči shrnout do tří typů:

1. Gramatika, která přesně kopíruje gramatiku češtiny včetně skloňování a koncovek (známá také jako *znakovaná čeština*). Jedná se o uměle vykonstruovaný jazyk slyšícími lidmi, aby co nejpřesněji kopíroval gramatiku českého mluveného jazyka. Při znakové řeči ústy se vyslovuje česká věta a k jednotlivým slovům se přiřazují

příslušné znaky. Tento typ gramatiky se prakticky nepoužívá, snad jen při výuce češtiny.

2. Volnější forma předchozí varianty, která vynechává koncovky, je využívaná při znakování odborných textů, na úřadech a bývá také vidět při tlumočení televizních programů. Neslyšící jí většinou špatně rozumí, neboť kopíruje gramatiku mluvené češtiny, která nemá se znakovým jazykem moc společného.

3. Znakový jazyk, kterým se mezi sebou neslyšící dorozumívají. Tento typ gramatiky většinou ovládají pouze ti, kteří s neslyšícím dlouhodobě žijí nebo spolu často komunikují.

Slyšící obvykle pro komunikaci s neslyšícím používají mluvenou češtinu doplněnou o posunky.

Toto rozdělení je pravděpodobně na všech místech stejné a tudíž aplikovatelné i na ostatní znakovací jazyky (německý, americký, britský, ...).

2.2 Prstová abeceda

Prstová abeceda se znakovým jazykem příliš nesouvisí, ale je hojně využívána v případech, kdy není pro dané slovo příslušný znak, např. jména. Někdy se názvy skládají ze znaku a přidaných dalších písmen, třeba *Jihlava* se dá složit z písmen *J* a *I* prstové abecedy a znaku *hlava*.

Prstová abeceda existuje ve dvou variantách

dvouruční – písmenka jsou přehledně znázorněna oběma rukama, ale je to neefektivní a pomalé;

jednoruční – písmenka jsou znakována jednou rukou a je do jisté míry „standardizována“ doporučením Světové federace neslyšících. Dá se tedy tvrdit, že je mezinárodní, i když se v několika písmenech vyskytují odlišnosti.

2.3 Znakování programem

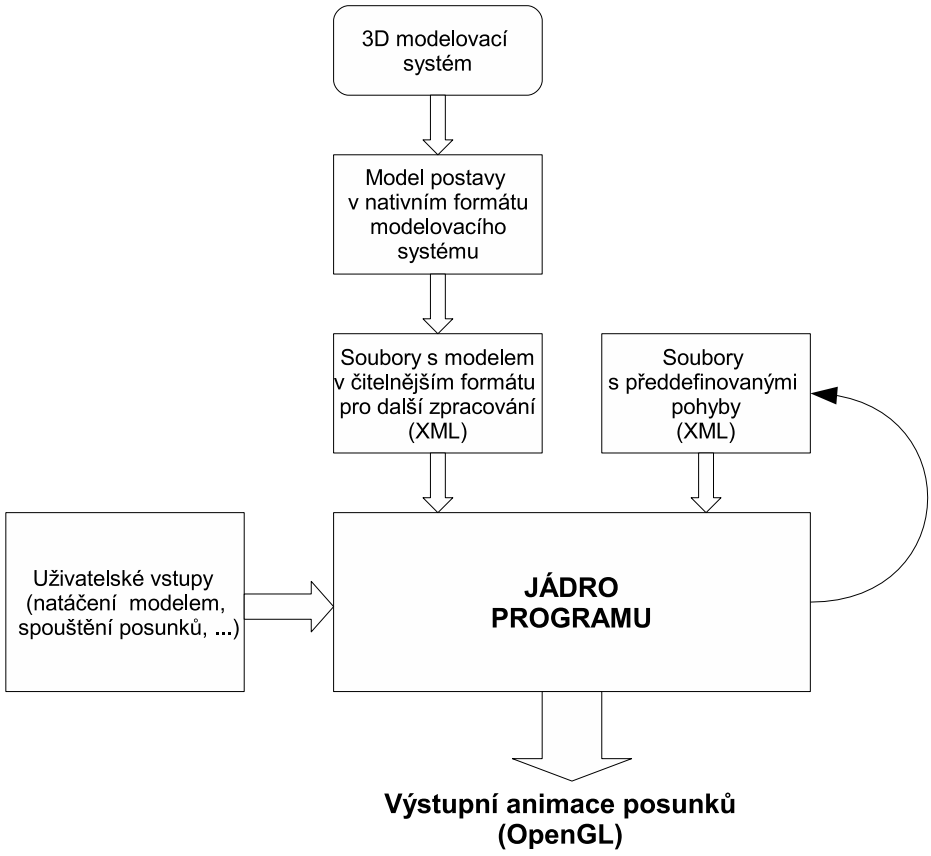
Z předchozího je zřejmé, že vytvoření programu univerzálně překládající český text do znakového jazyka tak, aby tomu neslyšící rozuměli, lze přirovnat k perpetum mobile. Tomuto cíli může jen přiblížit a vždy to bude vyžadovat obsluhu programu obeznámenou s principy znakové řeči. Bylo by možné by vytvořit překladač do znakované češtiny (viz kap. 2.1 na str. 10), což ovšem řeší tuto úlohu jen z části.

Základem úspěchu však vždy bude dobrá slovní zásoba, tedy databáze znaků, snadno kombinovatelná s možností tvorby synonym a vnořování znaků. Vnořováním znaků, je myšleno například výše popsaný znak *Jihlava*, kde by definice takového slova mohla být složena z již existujících znaků v databázi, tedy z písmen prstové abecedy *J* a *I* a znaku *hlava*, s tím, že pozdější úprava znaku *hlava*, ovlivní i znak *Jihlava*. Dále pak musí být implementován nějaký systém schopný poradit si se skloňovanými slovy na vstupu programu, protože ve znakovém jazyce se nečasuje, nepáduje apod.

3 3D model lidského těla a jeho animace

Úkolem této práce bylo vytvořit program zobrazující postavu člověka s pohyblivými horními končetinami tak, aby ze zadaného vstupu byl model schopen vyznakovat požadované posunky. Postavou by mělo být možné libovolně otáčet, aby se tím dal dobře nastavit úhel pohledu. Program by měl být univerzální se schopností jednoduše měnit vzhled postavy a editovat databázi posunků, aniž by se dělaly zásahy do vlastního kódu programu. Bylo by vhodné tento program postavit na široce podporovaných technologiích nevázané na konkrétní platformu ať softwarovou či hardwarovou tak, aby se dal program provozovat kdekoli. Tím by se zajistilo maximální možné nasazení softwaru, aniž by se zúžily možnosti nasazení tohoto produktu. Je zajímavé si představit možné budoucí použití na zařízeních typu herní konzole, které nemají s platformou x86 a systému Microsoft Windows nic společného.

Z nástinu cíle bakalářské práce je jasné, že se nejedná o triviální úkol a je nutné je rozdělit do více podúkolů, které jsou nezávislé na sobě. Ty pak lze v budoucnu propracovat více do detailu.



Obr. 1. Vstupy a výstupy programu.

Na obrázku 1 je principiální schéma vstupů a výstupů programu, jež vyhovuje základním požadavkům výše stanovených – snadno modifikovatelná databáze posunků (zde naznačena jako XML soubory) a snadno vyměnitelný model člověka.

3.1 Model člověka

Model člověka je hlavní součástí tohoto projektu. Prakticky se tvorba modelu setká se dvěma hledisky, které se navzájem vylučují: realističnost a hardwarová náročnost versus rychlost a jednoduchý model. Čím více bude model propracovaný do detailů, tím více bude náročný na hardware uživatele a naopak, pokud vytvoříme příliš strohý model, bude spatně čitelný. Musíme tedy volit vhodný kompromis někde uprostřed mezi těmito extrémy. Samozřejmě bychom si měli dát záležet na těch částech modelu, které jsou nejdůležitější pro pozorovatele, jako jsou prsty na rukou a obličejová část, kde je důležitá mimika popřípadě odezírání ze rtů modelu.

Dost významně může k čitelnosti modelu přispět umístění osvětlení a vrhající stíny rukou, které dodají modelu hloubku a pozorovatel tak zřetelněji rozezná polohu rukou. Toto je dost náročné laborování se zdroji světla na scéně a jejich parametry. Zdroj světla by neměl vytvářet příliš ostré stíny, ale zase musí být rozeznatelné. Na scéně musí být také světlo typu *ambient*, tedy všudypřítomné světlo, které zajistí, aby i části modelu ve stínu byly viditelné. Jinak by byly na modelu rušivá černá místa. Problematika světla je záležitostí až vlastního programu, ne modelovacího software. Použitý modelovací software a jeho exportní formát nemá podporu pro uchování dat o světelných podmínkách na scéně.

Pro tvorbu modelu se nejvíce bude hodit software, který sám o sobě umí vytvořit nějakou animaci, protože v takovém programu bude možné vytvořit klouby[4] a ty spolu s modelem importovat do našeho programu. Je velice důležité, aby i export z programu tyto vlastnosti dokázal načíst a uložit je do snadno parsovatelných datových struktur.

Před dalším pokračováním je zde uvedeno několik termínů dále se v textu vyskytujících týkající se modelování ve 3D. Budou použity anglické termíny v případech, kdy české výrazy nejsou dosud standardizovány.

vertex – jedná se o vrchol (bod) v prostoru definovaný třemi souřadnicemi kartézské soustavy[5] x, y, z;

[4] Nebo *kosti* v angličtině *bones*.

[5] Téměř bez výjimky se bude jednat o kartézskou soustavu.

face, polygon – ploška modelu definovaná zpravidla třemi vertexy, u polygonu možné i více vertexy, ale dost systémů tyto polygony stejně rozkládá na trojúhelníky;

scéna – virtuální prostor ve 3D, ve kterém se odehrává zobrazení modelů a animací;

bone, kost – prvek modelu využívající se při pohybech jednotlivých částí modelu; kost bývá definována jako umístění kloubu (středu otáčení) v kartézských souřadnicích a jako natočení vzhledem ke globálnímu souřadnému systému[6] v quaternionu[7] w, x, y, z;

globální souřadný systém (GSS) – souřadný systém scény, ve kterém probíhá zobrazení grafiky na výstupní zařízení;

lokální souřadný systém (LSS) – souřadný systém posunutý a natočený vzhledem ke GSS nebo jiného LSS.

keyframe – póza modelu definovaná v čase

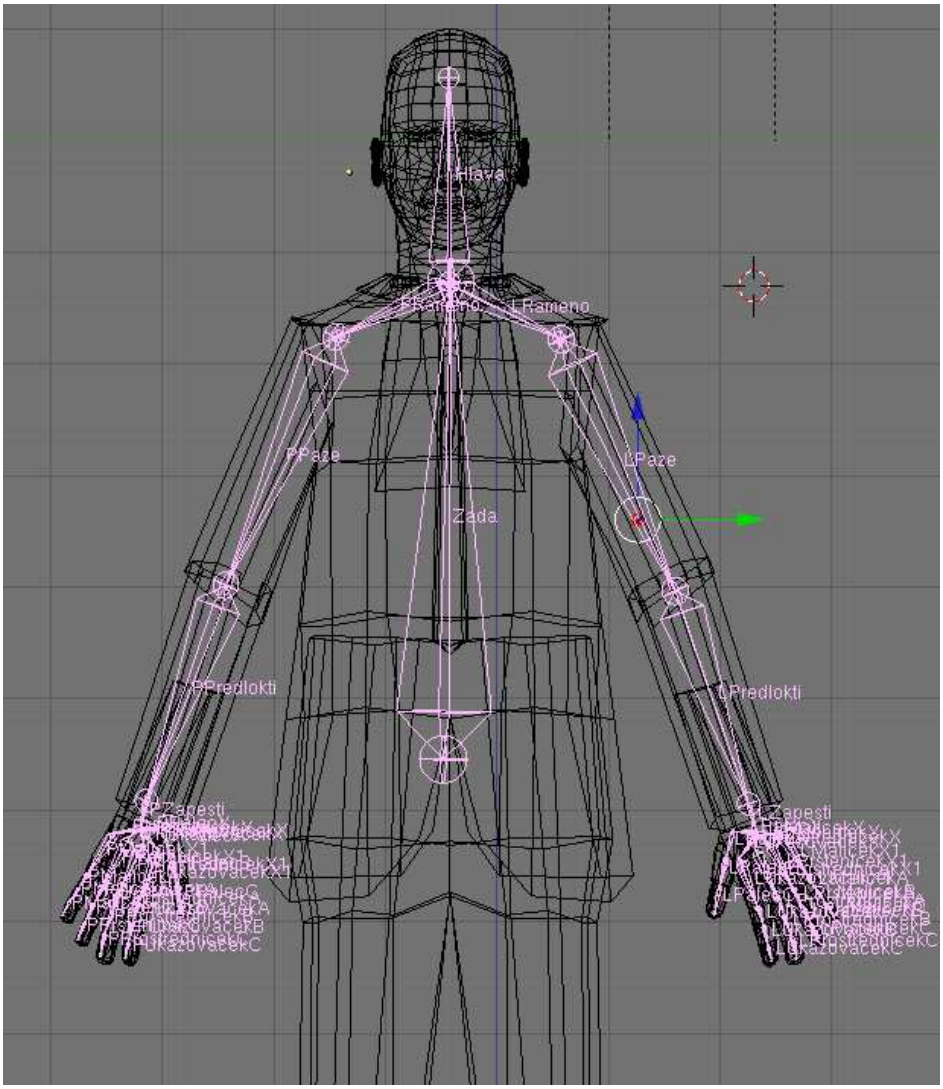
3.1.1 Systém kostí

Nejefektivnějším systémem pro animaci postav nebo obecně pro rozpohybování nějakého objektu je systém kostí. Jedná se o analogii převzatou z živočišné říše. Hotový model, který je sestaven pouze z vertexů s polygony, je doplněn kostmi s asociovanými vertexy. Tím při pohybu jedné kosti pohneme i všemi jejími vertexy a pokud na ní závisí další kosti[8], pohnou se i ony (včetně jejich vertexů a dalších případných kostí).

Na základě tohoto principu se dá dobře navrhnout systém pohybů, který rekurzivně hýbe kostrou modelu.

Kvůli přehlednosti návrhu by kosti měly být konzistentní, tzn. celá kostra by měla mít pouze jednu (referenční) kost, která nemá žádného předka (parent bone). To by mělo usnadnit případný pohyb a umístění modelu v prostoru, které by se pak dalo elegantně omezit na umístění a natočení této referenční kosti v prostoru. Měly by být také asociovány všechny vertexy ke kostem, aby při pohybu referenčních kostí se pohnul celý model a nekazily to natažené polygony přes celou scénu.

[6] Nebo také v lokálním souřadném systému parent kosti.
[7] Více o quaternionech na str. 23
[8] Běžný termín pro kost závislou na jiné je *child bone*.



Obr. 2. Kostra modelu člověka

3.2 Pohyby modelem

V předchozích statích byl popsán způsob aplikace systému kostí na výsledný model. Nyní je zapotřebí rozebrat způsob, kterým budou tyto kosti animovány v čase.

Tento projekt vychází z principu systému pohybů založeného na *keyframes* (více na str. 9 nebo v [3]). Vstupními daty může být jednorozměrné pole keyframů, kdy každý keyframe v sobě ponese informace o natočení všech kostí v lokálním souřadnicovém systému parent kosti. Při implementaci je třeba si uvědomit, že natočení každé kosti je vztaženo k parent kosti, čímž zásadně roste algoritmická náročnost tohoto problému. Kdyby byl použit GSS a porovnány keyframy ruky s nataženým loktem a pokrčeným loktem, změny by byly v natočení kloubů veškerých kostí závisejících na předloktí. Tato

vlastnost je však nepřijatelná. U keyframů s LSS by byla změna pouze v natočení kosti v lokti a zbytek zůstane podle očekávání.

Tak, aby keyframe vyhovoval daným požadavkům, musí mít v sobě informaci o čase, za který má model zaujmout danou pózu a seznam kostí a jejich natočení vzhledem k LSS parent kosti.

4 Matematický popis transformace bodu v prostoru

Tak, aby bylo možné vytvářet pohybující se 3D počítačový model, je zapotřebí použít algebraické mechanismy, které budou pro výpočty vhodné. Teoreticky lze vybrat z více matematických nástrojů, ale ne všechny postupy jsou správně aplikovatelné v počítačovém programu, kdy je nutné uvažovat i numerické nepřesnosti ve výpočtech, které mohou ve svém důsledku dost podstatně ovlivnit výsledek transformace modelu.

V následujícím textu budou použity následující konvence:

- matice rotace bude značena jako $\mathbb{R}$;
- matice translace (posunutí) $\mathbb{T}$;
- vektor obecně jako $\vec{v}$.

4.1 Transformace bodu v prostoru

Transformace bodu v prostoru může být zobecněna zápisem

$$\vec{r} = \mathbb{A} \cdot \vec{r_0} \tag{1}$$

kde transformovaný vektor o souřadnicích $\vec{r}$ vznikne součinem matice transformace $\mathbb{A}$ a původních souřadnic bodu $\vec{r_0}$.

Matice transformace $\mathbb{A}$ se skládá z matice rotace $\mathbb{R}$ a z matice translace $\mathbb{T}$, tedy

$$\mathbb{A} = \mathbb{R} \cdot \mathbb{T} \tag{2}$$

Matice rotace $\mathbb{R}$ má tvar

$$\mathbb{R} = \begin{pmatrix} r_{11} & r_{21} & r_{31} & 0 \\ r_{12} & r_{22} & r_{32} & 0 \\ r_{13} & r_{23} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3}$$

Matice translace $\mathbb{T}$ má tvar

$$\mathbb{R} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4}$$

kde t_x, t_y a t_z jsou příslušná posunutí na osách x, y a z.

Výsledná obecná matice transformace $\mathbb{A}$ má tedy tvar

$$\mathbb{A} = \begin{pmatrix} r_{11} & r_{21} & r_{31} & t_x \\ r_{12} & r_{22} & r_{32} & t_y \\ r_{13} & r_{23} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{5}$$

Transformovaný vektor $\vec{v}$ má tuto podobu

$$\vec{v} = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{6}$$

Matice rotace má tyto vlastnosti:

- regulární

$$\det \mathbb{R} \neq 0 \tag{7}$$

- speciální ortogonální

$$\mathbb{R}^{-1} = \mathbb{R}^{\mathrm{T}} \tag{8}$$

  platí i

$$\det \mathbb{R} = 1 \tag{9}$$

- součet kvadrátů členů v řádku nebo sloupci je 1

$$\sum_{i=1}^{4} r_{ni} = \sum_{i=1}^{4} r_{in} = 1 \tag{10}$$

  kde $n \in \{1, 2, 3, 4\}$

4.2 Skládání transformací

Další důležitou vlastností je skládání transformací, kdy je zapotřebí provést více transformací najednou, např. máme transformace $\mathbb{A}_{12}$ a $\mathbb{A}_{23}$ a ty jsou pak aplikovány na souřadnicový vektor nějakého bodu $\vec{b_0}$, pak lze provést

$$\vec{b_1} = \mathbb{A}_{12} \cdot \vec{b_0} \tag{11}$$

$$\vec{b_2} = \mathbb{A}_{23} \cdot \vec{b_1} \tag{12}$$

nebo složit transformace $\mathbb{A}_{12}$ a $\mathbb{A}_{23}$ do jedné $\mathbb{A}_{13}$

$$\mathbb{A}_{13} = \mathbb{A}_{12} \cdot \mathbb{A}_{23} \tag{13}$$

a transformovat

$$\vec{b_2} = \mathbb{A}_{13} \cdot \vec{b_0} \tag{14}$$

4.3 Reprezentace rotací podle Eulera

Jednou ze základních transformací bodu v prostoru je *rotace*. Bez rotace by bylo obtížné modelem pohybovat, takže správná implementace rotace je základem funkčnosti celého programu.

Matice rotace je sestavena podle následujících vzorců:

- rotace podle osy x o úhel α

$$\mathbb{R}_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{15}$$

- rotace podle osy y o úhel β

$$\mathbb{R}_y = \begin{pmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{16}$$

- rotace podle osy z o úhel γ

$$\mathbb{R}_z = \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{17}$$

4.3.1 Rotace nejsou komutativní

Problém si lze ukázat na jednoduché úloze, kdy je zapotřebí otáčet bodem kolem počátku. Otáčení kolem jednotlivých os bude zadáno ze vstupní periferie (myš, joystick, klávesnice, ...) a podle toho se bude patřičně otáčet kamerou. Z této myšlenky vznikne toto schéma programu:

1. načíst hodnoty změn úhlů ze vstupního zařízení
2. tyto hodnoty přičíst k aktuálním hodnotám úhlů v proměnných
3. převést hodnoty úhlů na matici transformace podle vzorců 15, 16, 17 a 13.
4. vložit matici do `GL_MODELVIEW`
5. překreslit scénu
6. skočit na bod 1 (aby vše bylo v „nekonečné“ smyčce)

Je to jednoduché schéma, ale není příliš funkční. Zdůvodněním může být následující příklad:

Na stole leží kniha. Hřbet knihy je na levé straně kolmo na hranu stolu a titulní strana desek směřuje nahoru. Osa x je na hraně stolu, osa y prochází hřbetem knížky. Osy xy definují rovinu stolu. Osa z prochází kolmo stolem.

Poté budou provedeny následující transformace:

1. Otočení knihy kolem osy z o 90°[9]. Kniha bude hřbetem na hraně stolu.
2. Otočení knihy kolem osy x o 180°. Kniha bude hřbetem směřovat od hrany stolu zadní stranou desek nahoru.

[9]Pro kladné otáčení funguje pravidlo pravé ruky (palec ve směru osy a prsty ve směru otáčení), kniha se tedy bude otáčet proti směru hodinových ručiček.

Po zadání těchto rotací do výše uvedeného schématu programu dojde k úplně jinému výsledku. Problém je v tvorbě matice rotace, která je většinou tvořena součinem $\mathbb{R}_x \cdot \mathbb{R}_y \cdot \mathbb{R}_z$ a protože součin matic není komutativní, tedy obecně $\mathbb{A}_1 \cdot \mathbb{A}_2 \neq \mathbb{A}_2 \cdot \mathbb{A}_1$, výsledkem je prohození rotací 1 a 2, takže konečná poloha knihy bude hřbetem na hraně stolu zadní stranou desek nahoru.

Aby program fungoval jak má, je třeba změnit způsob jeho práce s rotacemi. V programu bude uložená matice rotace, která bude při startu programu jednotková, a ta se bude násobit přírůstkem natočení podle jednotlivých os. Výsledná matice se použije pro transformaci objektu a uloží se do dalšího cyklu.

Opravené schéma programu může být následující:

1. načíst hodnoty změn úhlů ze vstupního zařízení
2. z těchto hodnot vytvořit matice rotace podle jednotlivých os
3. vynásobit zapamatovanou matici transformace maticemi z bodu 2
4. vložit matici do `GL_MODELVIEW`
5. překreslit scénu
6. skočit na bod 1

Toto programové schéma by mělo fungovat, tedy matematicky je korektní. Nutno poznamenat, že přírůstky k úhlům by měly být malé. (Neměly by přesahovat cca 10°, jinak se začnou projevovat výše popsané problémy.)

Programové schéma má však úplně jiný nedostatek a tím jsou zaokrouhlovací chyby a numerická nestabilita matice rotace. Když se bod otočí $3600\times$ o $0,1°$, nevrátí se přesně do výchozí pozice. To není až takovým problémem, prakticky vždy se s podobnými zaokrouhlovacími chybami v programu musí počítat, ale v tomto případě, to vadí daleko více, protože matice rotace ztrácí své vlastnosti (viz str. 19). Pokud matice rotace začne ztrácet své vlastnosti, začne provádět i jiné nežádoucí transformace[10]. Řešením může být občasné opravení (znormování) matice, ale to není systémové řešení.

Z výše uvedeného je zřejmé, že je nutné použít jiný matematický prostředek pro ukládání rotací. Je zapotřebí, aby tento prostředek uchovával pouze rotace a tím eliminoval zaokrouhlovací chyby pouze na nepřesnosti v rotacích[5].

[10]Např. zoomování scény ve směru jedné osy není očekávaný výsledek rotace.

4.4 Reprezentace rotací pomocí quaternionu

Quaternion je definován jako číslo s jednou reálnou a třemi imaginárními složkami. Lze tedy říci, že jde o rozšířené komplexní číslo. Zatímco klasická komplexní čísla mají tvar $a + ib$, kde $i^2 = -1$, quaternion je zapsán jako

$$w + ix + jy + kz \tag{18}$$

kde

$$\begin{aligned} ij &= k && (19)\\ jk &= i && (20)\\ ki &= j && (21)\\ ji &= -k && (22)\\ kj &= -i && (23)\\ ik &= -j && (24)\\ i^2 = j^2 = k^2 = ijk &= -1 && (25) \end{aligned}$$

Z výše uvedeného je zřejmé, že quaterniony nejsou komutativní, což je dobře využitelné (rotace také nejsou komutativní)[6].

Pro naše účely budeme používat jednotkový quaternion, který je definován jako

$$w^2 + x^2 + y^2 + z^2 = 1 \tag{26}$$

Quaternion můžeme převést na matici transformace

$$\mathbb{R} = \begin{pmatrix} w^2 + x^2 - y^2 - z^2 & 2xy - 2wz & 2wy + 2xz & 0 \\ 2wz + 2xy & w^2 - x^2 + y^2 - z^2 & 2yz - 2wx & 0 \\ 2xz - 2wy & 2wx + 2yz & w^2 - x^2 - y^2 + z^2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{27}$$

a použít ji běžným způsobem[7].

Další operace nutná pro skládání rotací je násobení quaternionů. Princip je stejný jako u maticového zápisu s tím rozdílem, že pořadí operací má opačný smysl

$$q_{13} = q_{23}q_{12} \tag{28}$$

Vlastní operace násobení quaternionů $q_1 q_2$, kde $q_1 = a + \vec{u}$ a $q_2 = t + \vec{v}$ [11], je definovaná jako

$$q_1 q_2 = at - \vec{u} \cdot \vec{v} + a\vec{v} + t\vec{u} + \vec{u} \times \vec{v} \tag{29}$$

kde operace $\vec{u} \cdot \vec{v}$ je skalárním součinem dvou vektorů a $\vec{u} \times \vec{v}$ je vektorovým součinem dvou vektorů. Po úpravě dostaneme

$$\begin{aligned} q_1 q_2 &= w_1 w_2 - x_1 x_2 - y_1 y_2 - z_1 z_2 + \\ &+ i(y_1 z_2 - z_1 y_2 + w_1 x_2 + x_1 w_2) + \\ &+ j(z_1 x_2 - x_1 z_2 + w_1 y_2 + y_1 w_2) + \\ &+ k(x_1 y_2 - y_1 x_2 + w_1 z_2 + z_1 w_2) \end{aligned}$$

Inverzní rotaci, podobně jako u maticových operací, získáme inverzí quaternionu q^{-1}, který je definován jako

$$q^{-1} = (w + ix + jy + kz)^{-1} = w - ix - jy - kz \tag{30}$$

Operací s quaterniony je samozřejmě daleko více, ale pro účely implementace rotací v programu zcela postačují tyto základní.

Schéma (z kap. 4.3 na str. 20) bude upraveno tak, aby operace s rotacemi byly prováděny na úrovni quaternionů, a výsledná transformace přeložena do maticového zápisu.

1. načíst hodnoty změn úhlů ze vstupního zařízení
2. z těchto hodnot vytvořit quaternion
3. vynásobit quaternion z bodu 2 zapamatovaným quaternionem a výsledek si zapamatovat
4. zapamatovaný quaternion převést na matici
5. vložit matici do `GL_MODELVIEW`
6. překreslit scénu
7. skočit na bod 1

[11] Někdy je quaternion zapisován jako reálná složka a vektor imaginárních složek.

5 Tvorba modelu

Model je tvořen podle zásad uvedených v kap. 3.1 na str. 14.

Pro tvorbu modelu je použit modelovací systém Blender. Je to vhodný program používaný filmovými studii pro tvorbu animací a má vše, co je potřeba pro tvorbu modelu s kostmi a jeho exportu. Program vydán pod GNU GPL licencí[12].

5.1 Modelování v Blenderu

Blender má svojí vlastní filozofii ovládání programu, která se výrazně liší od zavedených zvyklostí. Detaily použití zde nebudou uvedeny a zájemce odkáži na [8] nebo na oficiální dokumentaci [9].

Při modelování postavy je postupováno podle tutoriálů dostupných na Internetu. Postup vytváření postavy zde proto nebude dopodrobna uveden, protože je popsán v tutoriálech pro modelování hlavy [10], modelování ruky [11] a další [9].

Jakmile bude vlastní model hotov, následuje dosazení kostí do modelu, aby bylo možné v *pose mode* režimu pohybovat modelem. V tomto módu lze jednoduše ověřit správnost asociovaných vertexů k jednotlivým kostem pootočením kostí.

Pro pojmenování kostí byla zavedena následující konvence:

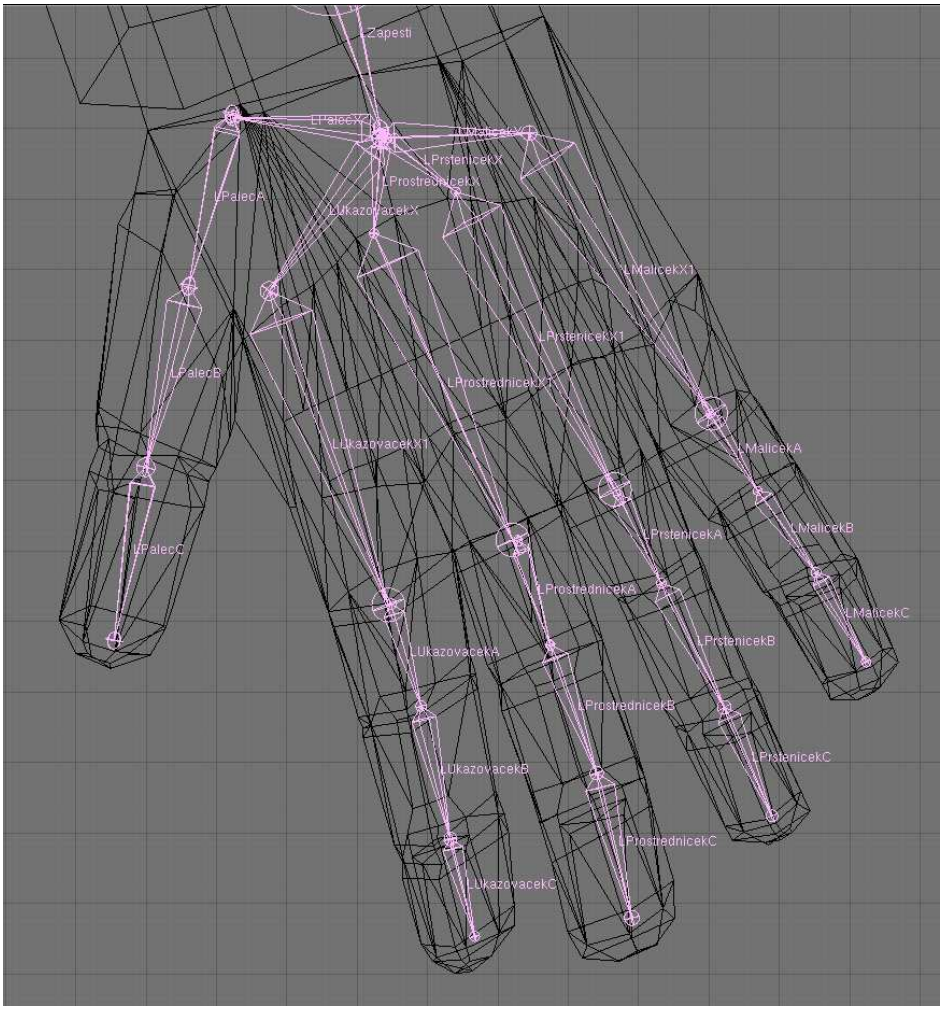
- Název kosti je odvozen od části těla, kde se nachází.
- Pokud existuje ta samá kost v páru, např. levá a pravá paže, je před jméno vloženo „L“ resp. „P“.
- Pokud má část těla další dělení, např. prsty na ruce, je na konci připojeno pořadové písmeno, např. „A“, „B“, ...
- Z důvodu omezení exportního modulu (více viz str. 27) nesmí název kosti obsahovat tečky (pozor na implicitní pojmenování ve formátu `bone.XXX`, kde XXX je pořadové číslo vytvořené kosti), mezery a další potenciálně nebezpečné znaky. Aby se předešlo problémům jsou v této bakalářské práci použity ve jménech kostí pouze velká a malá písmena anglické abecedy a čísla.
- V modelu lze použít pomocných kostí na místech, kde je to zapotřebí. Exportní formát má omezení, které znemožňuje umístění počátku (kloubu) potomka kosti na jiné místo, než je konec parent kosti. Je to dost nepříjemné omezení, protože

[12] Více na `http://gnu.cz/article/32/`

Obr. 3. Detail na dlaň ruky

Blender toto umožňuje, ale kostra vyexportovaného modelu je pak nepoužitelná. Z toho důvodu je možné vytvářet si vlastní kosti, které nebudou použity jako parametry při nastavování póz.

- Kosti, které budou použity jako parametry póz, jsou: `Zada`, `Hlava`, `LRameno`, `PRameno`, `LPaze`, `PPaze`, `LPredlokti`, `PPredlokti`, `LZapesti`, `PZapesti`, `LPalecA`, `LPalecB`, `LPalecC`, `PPalecA`, `PPalecB`, `PPalecC`, `LMalicekA`, `LMalicekB`, `LMalicekC`, `PMalicekA`, `PMalicekB`, `PMalicekC`, `LPrstenicekA`, `LPrstenicekB`, `LPrstenicekC`, `PPrstenicekA`, `PPrstenicekB`, `PPrstenicekC`, `LProstrednicekA`, `LProstrednicekB`, `LProstrednicekC`, `PProstrednicekA`, `PProstrednicekB`, `PProstrednicekC`, `LUkazovacekA`, `LUkazovacekB`, `LUkazovacekC`, `PUkazovacekA`, `PUkazovacekB`, `PUkazovacekC`.
- Další typy kostí, které byly zavedeny, protože exportní formát nepodporuje uklá-



Obr. 4. Detail na kostru jednotlivých prstů s pomocnými kostmi

dání barev, jsou kosti s prefixem „dummy“. Tyto kosti pojmenovávají jednotlivé části modelu a nijak neslouží pro pohyb s modelem. Je to dost nečisté řešení, nicméně nebyl čas na jiné, protože by zabralo řádově měsíce práce. Všechny kosti začínající na „dummy“, budou v modelu ignorovány a použity na těch místech v programu, kde jsou tyto dummy kosti využívány. Tyto kosti nemají v modelu žádného předka (parent bone).

5.2 Export z Blenderu do Cal3D

Vybraným formátem pro export modelu do programu se stal formát knihovny Cal3D[13], který byl nejlepším řešením ze všech exportních formátů podporovaných Blenderem.

[13]Bližší informace o Cal3D na `http://cal3d.sourceforge.net/`

Bylo zvažováno použití celé knihovny Cal3D jako základu pro manipulaci s modelem, ale tato myšlenka byla zavrhnuta, protože knihovna umožňuje pouze animovat již vytvořené animace. Knihovna Cal3D byla vytvořena primárně pro použití v různých hrách, v kterých je většina animací předem daných. To ovšem není tento případ, kdy bude potřeba korigovat pohyb rukou podle aktuální pozice a požadavku uživatele, proto tato knihovna nebyla použita. Možná se jedná o chybu, protože práce na implementaci importu dat a jejich další zpracování, byla velice problematická.

Knihovna Cal3D se stále vyvíjí a co se týče API i formátu modelu se může dost měnit. Exportní modul v Blenderu byl použit ve verzi 0.9, což je verze, která je běžně dostupná v instalaci Blenderu. V archívu knihovny Cal3D je možné najít soubor se specifikací tohoto formátu ve verzi 0.10, ale části, které jsou použity programem, se neliší. Knihovna je celkem dobře zdokumentována, i když by se hodil v některých specifických oblastech podrobnější popis. Specifikaci tohoto formátu je možné najít v příloze na str. 40.

V následujících statích jsou uvedeny některé rady, které mohou ostatním pomoci při exportu z Blenderu do Cal3D formátu:

- Před exportem do Cal3D formátu uložte model, ukončete Blender, spusťte Blender, otevřete model a až pak jej exportujte. Pokud budete exportovat průběžně, bude exportní skript hlásit varování a výsledný export bude jiný než předpokládaný!

- Pokud při exportu skript havaruje a bude hlásit chybu něco ve smyslu dělení nulou, bude se pravděpodobně jednat o chybu ve funkci `vector_normalize()` kolem řádku 315 v souboru `blender2cal3d.py` v adresáři `/usr/lib/blender/scripts`. Chyba se přestane objevovat po vložení řádku `l = l + 0.00001` před řádek s `return`. Tato chyba se projevovala ve verzi exportního skriptu 0.9 i nejnovější 0.91.

Otázkou stále zůstává vyměnitelnost modelu. Vyměnitelností modelu je myšlena kompletní výměna modelu člověka třeba za postavičku Barta Simpsona, aniž by to narušilo parametry pohybů. Přenositelnost parametrů natočení kloubů bude zachována pouze za předpokladu zachování, kromě názvů kostí, pozice (vzájemné natočení) kostí v modelu. To je dost nepraktické a tak je zapotřebí vymyslet jakousi konfigurační mezivrstvu mezi natočením jednotlivých os kloubů a natočením, které skutečně žádáme. V podstatě se jedná o jeden konfigurační soubor přiložený k modelu nesoucí kalibraci natočení každého kloubu. Implementace takového zpracování korekčních dat ze souboru

není až takovým problémem, problémem však je získání těchto korekčních dat, které je bez podpory GUI nerealizovatelné. To je velice pracné a proto tuto část v programu neimplementuji, nicméně je to jeden z bodů budoucího vývoje tohoto programu.

5.3 Struktura programu

Program byl koncipován co nejjednodušeji a co nejvíce modulárně. Parametry programu včetně toho, jaký model se má načíst a jaké pohyby vykonat, jsou vkládány na příkazovou řádku programu. To umožňuje, alespoň v těchto ranných stádiích vývoje programu, snadno měnit nastavení, aby s každou změnou nastavení nebylo zapotřebí složitě přepisovat GUI. Prozatím bude program fungovat na tomto principu a do budoucna je plánováno GUI, které by bylo oddělené od vlastního zobrazujícího programu. Zajímavý je návrh komunikace programu a GUI na síťové vrstvě. To by umožnilo běh programu s modelem odděleně na jiném stroji než je GUI, nehledě na to, že by s programem nemuselo komunikovat GUI, ale třeba nějaký server apod.

V současné chvíli program funguje spuštěním z příkazové řádky nebo z dávkového souboru definující příslušné parametry programu. Provede se kontrola parametrů, z parametrů se vyberou konfigurační údaje (soubory), načtou se soubory s modelem. Tyto modelové soubory se rozparsují a uloží do datových struktur programu. Proběhne otevření souborů s pohyby a jejich načtení do paměti. Program spustí grafický výstup přes OpenGL knihovny vstoupením do „nekonečné“ smyčky, ve které se renderuje model a provádí pohyby. Po provedení všech posunků je možné využít jednoduchý editační nástroj pro tvorbu pohybů.

5.4 Metodika práce s daty

5.4.1 Datová reprezentace modelu v programu

Jako první jsou do paměti načteny kosti do datové pole typu `Armature`. Z těchto kostí jsou separovány kosti pojmenované s prefixem `dummy`. Tyto kosti jsou ukládány do jiného pole, aby nepřekážely plnohodnotným kostem. (O dummy kostech je zmínka v kap. 5.1 na str. 25.) Kosti se pak uspořádají do stromové struktury tak, aby se s nimi dalo dobře manipulovat.

Dále se načtou data vertexů. Opět podobně, jako výše u kostí, se vytvoří pole vertexů datového typu `Point`. Poté se načtou plošky (faces) do pole datového typu `Face` a vytvoří se propojení mezi konkrétní ploškou a příslušnými třemi vertexy. Dále

se přiřadí vertexy ke kostem, které je ovlivňují, a přiřadí se k nim barva (pokud je to možné, přiřadí se podle jména *dummy* kosti).

Po načtení těchto dat se provedou některé další inicializační procedury, mezi které patří vypočítání středů a natočení jednotlivých kostí.

Výpočet středů natočení kostí byl jednou z nejvíce problematických částí programu a to hned z několika důvodů: vyžadoval naprostou bezchybovost velkého množství částí programu, které ještě nebyly pořádně otestovány (nebylo ještě na čem). Problém tedy byl jak algoritmicky problematický výpočet, tak drobné chyby navzájem se ovlivňující na různých místech programu a bylo tedy velice obtížné určit, jestli se chyba týká vlastního algoritmu nebo něčeho jiného. Problém byl i v nepochopení některých dat v modelovém souboru, protože v dokumentaci je vysvětlení některých položek omezeno na jednoduchý popis.

Středy otáčení lze (nebo alespoň „mělo by“) získat dvěma způsoby:

- Z globálních parametrů (viz specifikace v příloze položky local translation a local rotation) v XML souborech v `<LOCALTRANSLATION>` a `<LOCALROTATION>`, kde takový střed otáčení by měl být nalezen po rotaci bodu daného pomocí `LOCALTRANSLATION` o `LOCALROTATION`.
- Z lokálních parametrů, kdy každý kloub kosti byl vyjádřen posunutím `TRANSLATION` v souřadném systému parent kosti a natočením kosti `ROTATION` také vzhledem k LSS parent kosti.

V programu není ani jedna z těchto uvedených implementací, protože vždy po naprogramování jedné nebo druhé docházelo k chybným výsledkům rotací. Nakonec bylo implementováno od každé varianty něco jiného, kdy natočení jednotlivých kloubů bylo získáno z parametru `LOCALROTATION` a středy otáčení rekurzivně z `TRANSLATION`.

Neúspěch v implementacích byl s největší pravděpodobností způsoben v té době ještě neodladěnými metodami pro práci s quaterniony. Později byla třeba odhalena chyba v násobení quaternionů. Chyba byla do programu zanesena slepým opsáním vzorce ze stránek na Internetu. Po zkontrolování vzorce i z jiných zdrojů na internetu vyplynulo, že chybných webů je více. Nezbývalo, než spornou část odvodit ručně z definice (viz vzorec 29 na straně 24).

5.4.2 Datová reprezentace pózy a pohybu modelu v programu

Jak již bylo uvedeno výše (v kap. 3.2 na str. 16), jednotlivé pózy modelu jsou definovány jako relativní natočení kostí vzhledem k parent kosti. Pokud k těmto parametrům bude přidán i čas, za který má model stanovenou pózu zaujmout, bude definován plnohodnotný keyframe. Pohyby, tedy jednotlivé znaky, jsou reprezentovány několika keyframy, které dohromady dávají pohyb.

Při spuštění se načtou ze souborů (databáze) jednotlivé pohyby a uloží se do paměti. V momentě spuštění animace se zjistí relativní natočení kloubů (vzhledem k parent kosti) a cílové relativní natočení kloubů (cílové natočení je uloženo v keyframu). Mezi těmito natočeními se provede slerp funkce pro časy t_1, t_2, ..., t_n, kde n = ${t_{kf}}$[14] a t_{kf} je požadovaný čas pro dosažení pózy. Čas t_0 by měl být 0 a čas $t_n = t_{kf}$ by měl být celkový čas potřebný pro dosažení pozice definované keyframem. Jednotlivé polohy kloubů spočítané slerp funkcí jsou uloženy do paměti a po té animovány. Po dosažení keyframu se opět zjistí aktuální pozice a vypočítá se cesta do dalšího keyframu atd.

Funkce slerp je definována takto

$$\mathrm{slerp}(q_1, q_2, t) = \frac{\sin(1-t)\varphi}{\sin\varphi} q_1 + \frac{\sin t\varphi}{\sin\varphi} q_2 \tag{31}$$

kde $t \in (0, 1)$ a φ je úhel mezi dvěma quaterniony.

5.4.3 Formát souboru s keyframy

Keyframy se ukládají do XML formátu s následující strukturou:

```
<keyframes>
  <keyframe time="_cas_">
    <quat boneName="_jmenoKosti_"
  w="_w_" x="_x_" y="_y_" z="_z_" />
  </keyframe>
</keyframes>
```

Veškerá data jsou zabalena do tagu `<keyframes>` obsahující všechny další keyframy jejichž vlastnosti jsou v tagu `<keyframe>`. Tento tag má atribut „time“ vyjadřující čas v milisekundách, za který model tuto pózu zaujme.

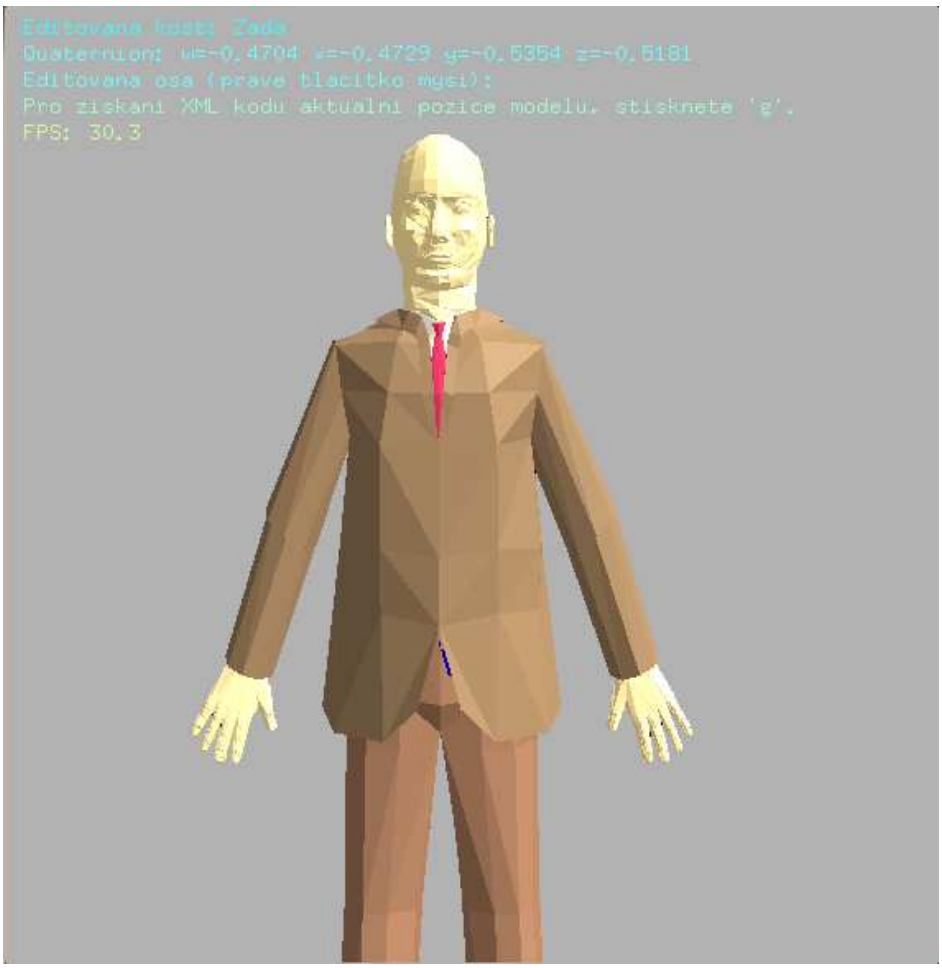
Parametry pózy jsou v tagu `<quat>` příslušného keyframu. Tento tag má atributy

[14] Frames Per Second – počet snímků za sekundu.

- „boneName“ udává jméno kosti, ke které se příslušný parametr vztahuje
- „w“, „x“, „y“, „z“ udávají quaternion relativního natočení kloubu vzhledem k parent kosti

Tagů `<quat>` bude pravděpodobně v keyframu více – jeden tag na jeden kloub modelu, ale je možné některé vynechat. Program se s tím vypořádá a vynechaný kloub ponechá v jeho předchozí poloze.

5.5 Stav dokončení programu



Obr. 5. Pohybující se model zobrazený programem

5.5.1 Interface programu

Vzhledem k tomu, že bylo rozhodnuto nedělat žádné GUI, které by se muselo při každé změně programu předělávat, ovládá se program přes klávesové zkratky. Může se to zdát pro dost krkolomné, nicméně tím byl ušetřen čas, který mohl být věnován na řešení skutečných problémů.

Seznam zkratek a jejich akcí:

\+ – přiblížení modelu (zoom in)

− – oddálení modelu (zoom out)

o – posune model dále

p – posune model blíže

k – posune model doleva

l – posune model doprava

n – posune model dolu

m – posune model nahoru

levé tlačítko myši – model rotuje podle vodorovné osy při vertikálním tahu myši nebo rotuje podle svislé osy při horizontálním tahu myši

v – rotuje výběr jednotlivých kloubů

b – rotuje výběr jednotlivých kloubů na druhou stranu

x – vybere osu otáčení x

y – vybere osu otáčení y

z – vybere osu otáčení z

pravé tlačítko myši – pokud je vybrán kloub a je vybrána osa otáčení, pohybuje označeným kloubem podle vybrané osy otáčení při horizontálním tahu myši

g – vypíše do konzole aplikace XML kód keyframu s aktuálním natočením kloubů

f – přepnutí do fullscreenu

w – přepnutí do okna

Esc – ukončení programu

Zajímavá funkce je pod klávesou „g“, ta vypíše XML kód keyframu do konzole. Takto lze velmi jednoduchým způsobem editovat XML soubor s keyframy nebo vytvářet další posunky. Více o tvorbě keyframů se dočtete v kap. 5.5.3.

5.5.2 Parametry programu

Důležité parametry jsou programu předávány z příkazové řádky. Prvním parametrem musí být jméno konfiguračního souboru modelu. Je to jeden ze souborů, který vznikne při exportu z nativního formátu Blenderu do formátu Cal3D, a má příponu `cfg`. Modelem v programu může být kterýkoli model obsahující názvy kostí popsané v kap. 5.1.

Dalšími parametry jsou pak jména souborů jednotlivých keyframů, které se mají vyznakovat. Tyto parametry jsou nepovinné. Když nebudou uvedeny, model bude nehybný a lze vytvářet keyframy pro nové posunky.

5.5.3 Tvorba souborů s keyframy

Jak již bylo uvedeno výše, lze model nastavit do určité pózy, zmáčknout klávesu „g“ a z konzole programu metodou copy&paste zkopírovat příslušný blok XML dat do textového editoru. V textovém editoru je zapotřebí jen opravit časový údaj z implicitních 1000 ms na požadovanou hodnotu a zkontroluje se, jestli jsou všechny `<keyframe>` v jednom velikém `<keyframes>`. Po zapsání změn na disk je možné soubor s keyframy vyzkoušet vložením ho jako parametr na příkazové řádce. Model by měl vykonat zadané pohyby podle keyframů v souboru a zastavit se na posledním z nich.

6 Implementace programu

Pro program byl zvolen programovací jazyk, který bude co nejvíce multiplatformní a přitom výkonný. Velmi zajímavou myšlenkou bylo vytvoření programu v Javě, ale byly zde pochybnosti o rychlosti takového programu.

Jako programovací jazyk byl tedy zvolen ANSI standard jazyka C++. Byla snaha vše psát s využitím ANSI C a ANSI C++ nebo knihoven, které tyto standardy používají. Jádro programu je postaveno na platformě nezávislém řešení.

Program je primárně psán ve vývojovém prostředí KDevelop verze 3.3.2 s použitím překladače z rodiny GCC verze 4.0.2 v operačním systému GNU/Linux. Zdrojové kódy jsou bez problémů přeložitelné pod operačním systémem Windows překladačem Mingw32, což je windowsový port překladače GCC. Program by měl jít přeložit i v Microsoft Visual Studiu.

6.1 Volba knihoven

Kromě standardních knihoven C a C++ byla použita knihovna OpenGL pro zobrazování 3D grafiky a její rozšíření GLUT a pro parsování XML souborů knihovna *Small, simple, multi-platform XMLParser library* z `http://iridia.ulb.ac.be/~fvandenb/tools/xmlParser.html`. Tato knihovna je napsána v ANSI C++ a vydána pod LGPL licencí[15].

Jediné knihovny, na kterých je tento program závislý, je knihovna OpenGL s jejím rozšířením GLUT. Knihovny OpenGL bývají standardní součástí systému, ale GLUT obvykle nebývá, takže je knihovnu (soubor `glut32.dll`) zapotřebí nainstalovat do systému nebo nakopírovat do adresáře se spustitelným souborem programu. Výše zmíněná knihovna pro parsování XML souborů je již obsažena v programu.

[15] Bližší informace o LGPL licenci na adrese `http://gnu.cz/article/34/`

7 Závěr

Výsledkem této bakalářské práce je vytvoření systému pro zobrazení 3D modelu člověka s pohyblivými horními končetinami tak, aby bylo možné interpretovat znakovou řeč. Umožňuje pohybovat modelem podle instrukcí ze zadaných souborů a obsahuje i jednoduchý nástroj, kterým lze tyto pohybové soubory vytvářet. Program byl vytvořen s použitím základních standardizovaných knihoven nezávislých na platformě, čímž lze aplikaci provozovat na jakémkoli zařízení, pro které je možné přeložit kód napsaný v ANSI C/C++ a existuje podpora 3D akcelerace pomocí knihovny OpenGL.

V průběhu řešení se ukázalo, že výsledný produkt je dosti složitý a komplexní systém, který je ještě zapotřebí dále zdokonalovat a vyřešit řadu problémů, z nichž každý by svým rozsahem odpovídal samostatné odborné práci. Také bude zapotřebí dotvořit uživatelsky přátelské rozhraní umožňující program používat i široké laické veřejnosti a především sluchově postiženým lidem. Nelze tedy výsledek bakalářské práce chápat jako hotový produkt, ale jako první krok ve vývoji rozsáhlého a robustního systému pro vizualizaci znakové řeči.

Navrhované kroky dalšího postupu v projektu jsou diskutovány v následující stati.

7.1 Vize do budoucna

Tento projekt realizoval 3D model pohybující se podle předem zadaných povelů, ale pořád se nemůže rovnat konkurentům z jiných univerzit nebo komerčního prostředí. Je ještě zapotřebí udělat hodně práce spadajících do nejrůznějších oborů.

7.1.1 Model

Rozhodně je zapotřebí upravit model, který není zrovna nejpovedenější. Bylo by zapotřebí sehnat grafika, který má už nějaké zkušenosti s 3D modelováním postav. Tvorba modelu je velice zdlouhavá činnost, ve které jde o detaily, a je zapotřebí mít nějaké zkušenosti. Dost podstatně to ovlivňuje celkový dojem z této aplikace.

7.1.2 Pohyb částí modelu

Bylo by dobré doimplementovat natáčení jednotlivých částí modelu na bázi *influences*. Influence je parametr, který je také exportován s modelem. Vychází z faktu, že je možné, aby vertex patřil k několika kostem. Každá taková kost tento vertex nějak *ovlivňuje*. Parametr influence tedy udává míru takového ovlivnění. Např. vertex je ovlivňován

třemi kostmi rovnoměrně, tj. 33% od každé. Pokud jedna kost rotuje ve svém kloubu o 3°, vertex se otočí v jejím směru pouze o 1°.

Tato změna v implementaci by přinesla daleko reálnější deformace, horší je to ovšem s realizací. Byl zde pokus o implementaci, ale výsledkem bylo divné chování vertexů různých influences. Pravděpodobně se jednalo o nepřesnosti v implementaci slerp funkce v době implementace influencí, ale toto riziko zde stále je. Pokud nebude implementována přesná funkce, která nalezne třetinu nebo polovinu mezi dvěma quaternionama, budou se jednotlivé části modelu rozjíždět po dráze rotace a výsledkem bude hodně zdeformovaný model.

Další oblastí, kterou je zapotřebí promyslet je výpočet interpolace pohybů tak, aby byla přirozenější. V současném stavu není možné vytvořit pohyb po kružnici. Pravděpodobně by se celý výpočet budoucí dráhy pohybů odehrával před započetím celého pohybu.

7.1.3 Standardizace parametrů póz

Toto téma bylo již naznačeno v kap. 5.2 na str. 27. Je zapotřebí vytvořit kalibrační mezivrstvu, která nám zajistí přenositelnost pohybů mezi různými modely s jinými tvary kostry (ne však s odlišnými názvy kostí). Implementovat tuto mezivrstvu by nebyl takový problém, ale problémem je implementovat GUI, s jehož pomocí se bude vytvářet tato konfigurace. Zároveň by se touto kalibrací mohly ošetřit meze pohybů, které také nejsou řešené.

7.1.4 Interface programu – GUI

GUI a v podstatě jakýkoli jiný uživatelsky přátelský přístup k programu v současné době neexistuje. To by se mělo změnit.

Na str. 29 v kap. 5.3 bylo naznačeno, jak by takové GUI mohlo vypadat. Opravdu by mohlo být postaveno na komunikaci přes TCP/IP protokol na síťové vrstvě. Vnitřně by program s GUI komunikoval přes vlastní protokol postavený na XML. Tím by byl dán další rozměr možného nasazení programu.

Provedení a technologie, na které bude postaveno GUI, nic nemění na faktu, že musí existovat editor posunků na úrovni použitelné pro běžného uživatele. Úspěchu nemůže být dosaženo, dokud nebude v databázi dostatečné množství posunků.

7.1.5 Databáze znakového jazyka

Další záležitostí je převedení souborů s posunky do databáze kvůli lepší odezvě a vlastnostem, které se s použitím databázového systému získají. Jedním z možných kandidátů na takovou databázi by mohlo být SQLite[16].

Je zapotřebí předělat strukturu uložených keyframes, aby si uchovávaly informace o synonymech v českém jazyce a aby umožňovaly vnořené reference. Tím myslím skutečnost, že některá slova lze složit z již existujících posunků. Slovo by pak mohlo obsahovat odkaz na posunek a příslušné další pohyby tento posunek rozšiřující. Zjednodušila by se údržba, protože po upravení odkazovaného posunku by se změnil i posunek, který jej využívá v jiném slově.

7.1.6 Syntéza řeči a mluvící hlava

Dalším možným rozšířením je dosazení do programu syntézy řeči, která by syntetizovala slovo příslušné danému posunku, s doprovodem mimiky v obličeji modelu. Toto by mohlo být součástí budoucího interaktivního výukového programu znakového jazyka, které by usnadnilo výuku nejen slyšícím lidem řečovou syntézou, ale i neslyšícím pomocí mimiky obličeje a odezírání ze rtů modelu.

[16]Domovská stránka SQLite je `http://www.sqlite.org`

SEZNAM POUŽITÉ LITERATURY

[1] Inc. Vcom3D. Authoring with studio. [Online; navštíveno 26. 4. 2006]. Dostupné z: `http://www.vcom3d.com/Studio.htm`.

[2] DePaul University. Depaul university american sign language project – project overview. [Online; navštíveno 27. 4. 2006]. Dostupné z: `http://asl.cti.depaul.edu/project_info.html`.

[3] Jason C. Wong. The auslan system sign editor user manual. [Online; navštíveno 27. 4. 2006]. Dostupné z: `http://auslantuition.csse.uwa.edu.au/docs/SEManual.pdf`.

[4] Mgr. Marie Růžičková. *Učíme se českou znakovou řeč*. SEPTIMA, Praha, 1997.

[5] Steve Baker. Euler angles are evil. [Online; navštíveno 30. 4. 2006]. Dostupné z: `http://sjbaker.org/steve/omniv/eulers_are_evil.html`.

[6] Wikipedia. Quaternion — wikipedia, the free encyclopedia, 2006. [Online; navštíveno 30. 4. 2006]. Dostupné z: `http://en.wikipedia.org/w/index.php?title=Quaternion&oldid=49818266`.

[7] Wikipedia. Quaternions and spatial rotation — wikipedia, the free encyclopedia, 2006. [Online; navštíveno 1. 5. 2006]. Dostupné z: `http://en.wikipedia.org/w/index.php?title=Quaternions_and_spatial_rotat%ion&oldid=50096033`.

[8] Blender Foundation. Blender home page. [Online; navštíveno 1. 5. 2006]. Dostupné z: `http://www.blender.org`.

[9] kolektiv autorů. Blender documentation volume i – user guide. [Online; navštíveno 1. 5. 2006]. Dostupné z: `http://www.blender3d.org/documentation/htmlI/book1.html`.

[10] Olivier Saraja. Using subsurf for head modeling. [Online; navštíveno 1. 5. 2006]. Dostupné z: `http://www.linuxgraphic.org/section3d/blender/pages/didacticiels/head-s%ubsurf/index-ang.html`.

[11] Lyubomir Kovachev. Rigging a hand and a foot. [Online; navštíveno 1. 5. 2006]. Dostupné z: `http://www.blender3d.org/documentation/htmlI/x7613.html`.

[12] Bruno Heidelberger. Cal3d – character animation library. [Online; navštíveno 3. 5. 2006]. Dostupné z: `http://cal3d.sourceforge.net/`.

Příloha A

```
o-------------------------------------------------------------------------------o
|                                                                               |
|                       cal3d fileformat description                            |
|                                                                               |
|                                Version 0.10.0                                 |
|                              (12. january 2005)                               |
|                                                                               |
|        Copyright (C) 2001, 2002, 2003 Bruno 'Beosil' Heidelberger             |
|                                                                               |
o-------------------------------------------------------------------------------o

o-------------------------------------------------------------------------------o
| Table of Contents                                                             |
o-------------------------------------------------------------------------------o

  1 What is this document for?
  2 cal3d skeleton file (.csf)
  3 cal3d animation file (.caf)
  4 cal3d mesh file (.cmf)
  5 cal3d material file (.crf)
  6 cal3d skeleton xml file (.xsf)
  7 cal3d animation xml file (.xaf)
  8 cal3d mesh xml file (.xmf)
  9 cal3d material xml file (.xrf)
  10 Website
  11 Author


o-------------------------------------------------------------------------------o
| 1 What is this document for?                                                  |
o-------------------------------------------------------------------------------o

This document describes the format of the different files used in the cal3d
character animation library version 0.10.0.

*******************************************************************************
*******************************************************************************
*******************************************************************************

 IMPORTANT: As the cal3d library is still in heavy development, the fileformats
 described in this document will most likely become obsolete as soon as a new
```

```
 version of the library is released. Furthermore, the fileformat itself is not
 yet big-/little-endian independent and may contain fields that are either
 ignored or misinterpreted by the library.

*****************************************************************************
*****************************************************************************
*****************************************************************************


o---------------------------------------------------------------------------o
| 2 cal3d skeleton file (.csf)                                              |
o---------------------------------------------------------------------------o

Stored in this file is the hierarchy of bones that composes the skeleton.


description               length  type      comments
------------------------  ------  --------  ----------------------------------
[header]
  magic token             4       const     "CSF\0"
  file version            4       integer   700
  number of bones         4       integer

[first bone]
  length of bone name     4       integer
  bone name               var     string
  translation x           4       float     relative translation to parent bone
  translation y           4       float
  translation z           4       float
  rotation x              4       float     relative rotation to parent bone
  rotation y              4       float     stored as a quaternion
  rotation z              4       float
  rotation w              4       float
  local translation x     4       float     translation to bring a vertex from
  local translation y     4       float     model space into bone space
  local translation z     4       float
  local rotation x        4       float     rotation to bring a vertex from
  local rotation y        4       float     model space into bone space
  local rotation z        4       float
  local rotation w        4       float
  parent bone id          4       integer   index to parent bone
  number of children      4       integer
```

```
  [first child]
    child bone id          4       integer   index to child bone

  [all other children]
    ...

[all other bones]
  ...
```

```
o-------------------------------------------------------------------------------o
| 3 cal3d animation file (.caf)                                                 |
o-------------------------------------------------------------------------------o
```

All the keyframes of an animation are stored in this file. They are grouped by tracks (one track per animated bone) and contain the time, the relative position and the relative rotation to the parent bone.

description	length	type	comments
[header]			
magic token	4	const	"CAF\0"
file version	4	integer	700
duration	4	float	length of animation in seconds
number of tracks	4	integer	
[first track]			
bone id	4	integer	index to bone
number of keyframes	4	integer	
[first keyframe]			
time	4	float	time of keyframe in seconds
translation x	4	float	relative translation to parent bone
translation y	4	float	
translation z	4	float	
rotation x	4	float	relative rotation to parent bone
rotation y	4	float	stored as a quaternion
rotation z	4	float	
rotation w	4	float	
[all other keyframes]			
...			

```
[all other tracks]
  ...
```

```
o-------------------------------------------------------------------------------o
| 4 cal3d mesh file (.cmf)                                                      |
o-------------------------------------------------------------------------------o
```

This file contains all the mesh data, such as the weighted influences of the bones on each vertex. The mesh is splitted into submeshes to group faces with the same material thread.

description	length	type	comments
[header]			
magic token	4	const	"CMF\0"
file version	4	integer	700
number of submeshes	4	integer	
[first submesh]			
material thread id	4	integer	
number of vertices	4	integer	
number of faces	4	integer	
number of lod steps	4	integer	number of vertices to collapse
number of springs	4	integer	number of springs
number of maps	4	integer	
[first vertex]			
position x	4	float	position in model space
position y	4	float	
position z	4	float	
normal x	4	float	normal in model space
normal y	4	float	
normal z	4	float	
collapse id	4	integer	index to vertex to collapse to
face collapse count	4	integer	number of collapsing faces when this vertex gets collapsed
[first map]			
u coordinate	4	float	map coordinates
v coordinate	4	float	

```
    [all other maps]
      ...

    number of influences   4       integer

    [first influence]
      bone id              4       integer   index to bone
      weight               4       float     weight of influence (1.0 == 100%)

    [all other influences]
      ...

    {physical property (only stored when #springs > 0 !)]
      weight               4       float     weight of vertex for cloth-/hair-
                                             animation (0.0 == rigid)

  [all other vertices]
    ...

  [first spring]
    vertex id 1            4       integer   index to vertex
    vertex id 2            4       integer   "
    spring coefficient     4       float
    idle length            4       float     rest length of the spring

  [all other springs]
    ...

  [first face]
    vertex id 1            4       integer   index to vertex
    vertex id 2            4       integer   "
    vertex id 3            4       integer   "

  [all other faces]
    ...

[all other submeshes]
  ...
```

```
o-------------------------------------------------------------------------------o
| 5 cal3d material file (.crf)                                                  |
```

```
o-------------------------------------------------------------------------------o

Material properties like ambient, diffuse and specular color and the shininess
factor are stored in this file. If the material contains texture maps, the
(file-)names of the textures are stored here too.


description                length  type      comments
-------------------------  ------  --------  ---------------------------------
[header]
  magic token              4       const     "CRF\0"
  file version             4       integer   700
  ambient color red        1       byte
  ambient color green      1       byte
  ambient color blue       1       byte
  ambient color alpha      1       byte
  diffuse color red        1       byte
  diffuse color green      1       byte
  diffuse color blue       1       byte
  diffuse color alpha      1       byte
  specular color red       1       byte
  specular color green     1       byte
  specular color blue      1       byte
  specular color alpha     1       byte
  shininess                4       float
  number of maps           4       integer

  [first map]
    length of texture name 4       integer
    texture name           var     string    most likely a filename

  {all other maps]
    ...

o-------------------------------------------------------------------------------o
| 6 skeleton Xml file (.xsf)                                                    |
o-------------------------------------------------------------------------------o

this is the text/Xml file format of cal3d skeleton file,
see cal3d skeleton file (.csf) for more information

<HEADER MAGIC="XFS" VERSION="900"/>
<SKELETON NUMBONES="INT">
```

```
   <BONE ID="INT" NAME="STRING" NUMCHILD="INT">
      <TRANSLATION>FLOAT FLOAT FLOAT</TRANSLATION>
      <ROTATION>FLOAT FLOAT FLOAT FLOAT</ROTATION>
      <LOCALTRANSLATION>FLOAT FLOAT FLOAT</LOCALTRANSLATION>
      <LOCALROTATION>FLOAT FLOAT FLOAT FLOAT</LOCALROTATION>
      <PARENTID>INT</PARENTID>
      <CHILDID>INT</CHILDID>
      ...              {all other childs}
    </BONE>
    ... {all other bones}
</SKELETON>


o-------------------------------------------------------------------------------o
| 7 cal3d xml animation file (.xaf)                                             |
o-------------------------------------------------------------------------------o

this is the text/Xml file format of cal3d animation file,
see cal3d skeleton file (.csf) for more information

<ANIMATION VERSION="1000" DURATION="FLOAT" NUMTRACKS="INT">
   <TRACK BONEID="INT" NUMKEYFRAMES="INT">
      <KEYFRAME TIME="FLOAT">
         <TRANSLATION>FLOAT FLOAT FLOAT</TRANSLATION>
         <ROTATION>FLOAT FLOAT FLOAT FLOAT</ROTATION>
      </KEYFRAME>
      ....                      {all other keyframes}
   </TRACK>
   ....                         {all other animation}
</ANIMATION>

o-------------------------------------------------------------------------------o
| 8 cal3d xml mesh file (.xmf)                                                  |
o-------------------------------------------------------------------------------o

this is the text/Xml file format of cal3d mesh file,
see cal3d mesh file (.cmf) for more information

<MESH VERSION="1000" NUMSUBMESH="INT">
   <SUBMESH MATERIAL="INT" NUMVERTICES="INT" NUMFACES="INT" NUMLODSTEPS="INT"
NUMSPRINGS="INT" NUMTEXCOORDS="INT">
       <VERTEX ID="INT" NUMINFLUENCES="INT">
          <POS>FLOAT FLOAT FLOAT</POS>
```

```
          <NORMFLOAT FLOAT FLOAT</NORM>
          [<COLLAPSEID>INT</COLLAPSEID>]
          [<COLLAPSECOUNT>INT</COLLAPSECOUNT>]
          <TEXCOORD>FLOAT FLOAT</TEXCOORD>
          ...                              {all the other texture coordinates}
          <INFLUENCE ID="INT">FLOAT FLOAT FLOAT</INFLUENCE>
          ...                              {all the other bone influances}
          [<PHYSIQUE>FLOAT</PHYSIQUE>]
       </VERTEX>
       ....                                {all the other vertices}
       <SPRING VERTEXID="INT INT" COEF="FLOAT" LENGTH ="FLOAT"/>
       ...                                 {all the other springs}
       <FACE VERTEXID="INT INT INT"/>
       ...                                 {all the other faces}
    </SUBMESH>
    ...
</MESH>

o-------------------------------------------------------------------------------o
| 9 cal3d xml material file (.xrf)                                              |
o-------------------------------------------------------------------------------o

this is the text/Xml file format of cal3d material file,
see cal3d material file (.crf) for more information

<MATERIAL VERSION="1000" NUMMAPS="INT">
   <AMBIENT>FLOAT FLOAT FLOAT FLOAT</AMBIENT>
   <DIFFUSE>FLOAT FLOAT FLOAT FLOAT</DIFFUSE>
   <SPECULAR>FLOAT FLOAT FLOAT FLOAT</SPECULAR>
   <SHININESS>FLOAT</SHININESS>
   <MAP>STRING</MAP>
   ...                     {all other maps}
</MATERIAL>

o-------------------------------------------------------------------------------o
| 10 Website                                                                    |
o-------------------------------------------------------------------------------o

  The official website of cal3d can be found at: http://cal3d.sourceforge.net


o-------------------------------------------------------------------------------o
| 11 Author                                                                     |
```

```
o--------------------------------------------------------------------------------o

  This document was written by Bruno 'Beosil' Heidelberger.
  the Xml format has been added by Laurent Desmecht


o--------------------------------------------------------------------------------o
```