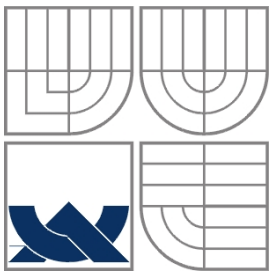
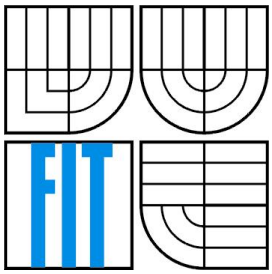


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

3D ANIMACE POSTAVY V POČÍTAČOVÉ GRAFICE

ANIMATION OF 3D CHARACTER IN COMPUTER GRAPHICS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MICHAL PEČENKA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAROSLAV PŘIBYL

BRNO 2008

Abstrakt

Cílem projektu je seznámit čtenáře s technikami používanými při real-time animaci 3D postav. Práce se soustředí zejména na dva typy animace: snímkovou animaci a kloubovou animaci. Popsány jsou algoritmy pro softwarovou i hardwarovou deformaci modelu, interpolaci klíčových snímků, kombinaci a prolínání animací, inverzní kinematiku a ragdoll. Výsledkem projektu je framework pro animaci 3D postav, který se skládá z implementované animační knihovny, příkladů demonstrujících činnost knihovny a nástrojů pro export animace z 3D Studia Max a MilkShape 3D.

Klíčová slova

snímková animace, kloubová soustava, kost, kloub, kůže, interpolace klíčových snímků, deformace modelu, kvaternion, matice, slerp, OpenGL, DirectX, shader, ragdoll, inverzní kinematika

Abstract

The main goal of this project was to familiarize readers with the techniques used in real-time animation of 3D characters. This work is focused on two types of animation: keyframe animation and skeletal animation. There are described algorithms for software and hardware accelerated model deformations, keyframe interpolations, animation blending, inverse kinematics and ragdoll. The result of this project is a framework, which consists of an animation library, examples demonstrating library functions and tools for export animations from 3D Studio Max and MilkShape 3D.

Keywords

keyframe animation, articulated system, bone, joint, skin, keyframe interpolation, skin deformation, quaternion, matrix, slerp, OpenGL, DirectX, shader, ragdoll, inverse kinematics

Citace

Pečenka Michal: 3D animace postavy v počítačové grafice. Brno, 2008, diplomová práce, FIT VUT v Brně.

3D animace postavy v počítačové grafice

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Jaroslava Přibyla
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Michal Pečenka
16. května 2008

Obsah

1 Úvod

Počítačová animace je v dnešní době velmi rychle se rozvíjejícím odvětvím počítačové grafiky. Hlavním hnacím motorem tohoto oboru je herní a filmový průmysl. Ale i firmy s jiným zaměřením dbají na co nejvyšší vizuální kvalitu svých produktů. Důležitou součástí vizuální stránky je právě počítačová animace, která přináší pohyb do jinak statických scén.

Nedílnou součástí počítačové animace je animace postav. Hlavním cílem je vytvořit animaci, kterou nelze rozeznat od reality. Proto se vymýšlejí stále nové a dokonalejší algoritmy, nové způsoby deformace kůže, věrohodnější modelování povrchu těla a vlasů nebo pokročilé metody animace látek. Kromě výsledného vzhledu jsou, zejména v počítačových hrách, kladeny požadavky na rychlost výpočtu a současně co nejmenší zatížení procesoru. Hlavně pro herní průmysl platí, že výpočetní kapacita je a vždy bude omezená, nelze tedy donekonečna zvyšovat počet vykreslovaných bodů a tím zajistit odpovídající kvalitu výstupu. Z tohoto důvodu je v dnešní době kladen velký důraz na vývoj nových technik pro snadnější, rychlejší a kvalitnější zobrazení animace.

První používanou technikou animace postav byla metoda per-vertex animace. V této metodě se trajektorie každého animovaného vrcholu musela nastavovat ručně. I když je tato technika docela snadno implementovatelná, v současných aplikacích se používá minimálně. Důvodem je obrovská paměťová náročnost a pro animátora složitá tvorba animací. Dnešní animace postav se provádí většinou pomocí tzv. animace kloubové soustavy. Kloubová soustava neboli kostra je velmi užitečná věc. Díky kostře, lze vytvořit animaci i velmi složitých modelů, docela jednoduchým způsobem. Kloubová soustava poskytuje jakousi abstrakci animovaného modelu. Místo náročné animace složitých objektů s velkým počtem vrcholů, stačí animovat relativně jednoduchou strukturu kloubů. Použitím kostry je manipulace s modelem postavy velmi jednoduchá, přičemž je tento systém vhodný pro počítačové zpracování. Kloubová soustava má však i další východy, např. vytváření reálných animací pomocí zařízení motion capture. Navíc není nutné vytvářet pro každou postavu novou sadu animací, ale lze již vytvořené animace sdílet mezi kloubovými soustavami.

Tato práce popisuje techniky, jakými lze zobrazit animaci 3D postavy, respektive libovolných živých či neživých objektů, na obrazovce počítače. První část práce je věnována teorii a vysvětlení základních pojmů patřících do oblasti počítačové grafiky a animace. Nechybí zde popis kloubové a per-vertex animace, grafických knihoven, shaderů, dopředné či inverzní kinematiky. V závěru je zmíněna i procedurální animace.

V další kapitole jsou detailně rozebrány algoritmy pro zobrazení animací. Mimo jiné metody interpolace klíčových snímků ve skeletální i snímkové animaci, algoritmy deformace modelů podle kloubových soustav, nejpoužívanější techniky inverzní kinematiky a ragdoll. Zmíněny jsou i možnosti implementace vybraných algoritmů na moderních grafických kartách.

Předposlední kapitola je věnována vlastnímu návrhu animační knihovny. Uvedena je struktura knihovny a hlavní cíle implementace. Hlavní část této kapitoly zabírá popis použitých datových struktur pro uložení snímkové a kloubové animace včetně objektově orientovaného návrhu rozhraní knihovny.

Výsledkem této práce je framework pro animaci 3D postav, který se skládá z implementované animační knihovny, příkladů demonstrujících činnost knihovny a skriptů pro export animace z 3D Studia Max a MilkShape 3D.

2 Teoretický úvod

2.1 Grafické knihovny

V současnosti existuje mnoho grafických knihoven a stále nové vznikají. Každá z těchto knihoven má svůj specifický obor použití, pro který byla navržena. Lze najít knihovny pro webové aplikace, mobilní či embedded zařízení [22], pro výkonové aplikace nebo třeba pro velmi realistický rendering. Přitom není vůbec neobvyklé, že některé knihovny plynule přecházejí napříč různými platformami. Příkladem je knihovna OpenGL, která je primárně určena pro osobní počítače, ale ve své odlehčené verzi je používána i v embedded systémech.

Jelikož je tento projekt napsán v jazyku C++ a určen pro osobní počítače, lze použít v podstatě dvě knihovny: OpenGL nebo DirectX.

2.1.1 DirectX

Jedná se o grafické API pevně svázané se systémem Windows (případně s platformou Xbox), které obsahuje nástroje pro tvorbu her a multimediálních aplikací. DirectX [23] se skládá z několika částí, z nichž se každá specializuje na určitou oblast. DirectX Graphics (DirectDraw, Direct3D) se orientuje na práci s dvojrozměrnou a trojrozměrnou grafikou, DirectInput poskytuje funkce pro obsluhu periferních zařízení (klávesnice, myši, joysticky, gamepady), DirectSound je rozhraní pro práci se zvukem, Direct Show podporuje tvorbu multimediálních aplikací (přehrávání a zpracování videa a zvuku) atd.

Knihovna DirectX vznikla původně jako nástroj pro vývoj počítačových her. Kupodivu se docela rychle rozšířila i do jiných oborů (např. 3D Studio Max podporuje renderování s použitím Direct3D). Klíčem úspěchu knihovny byla podpora nejnovějších funkcí grafických adaptérů a masová propagace ze strany Microsoftu.

Počátky vývoje spadají do roku 1994, kdy vedoucí společnost počítačového trhu – Microsoft, přišla s novým operačním systémem Windows 95. Do té doby používaný operační systém MS-DOS poskytoval vývojářům přímý přístup ke grafickým adaptérům, periferiím a ostatním připojeným zařízením. Tomu se Windows 95 se svým chráněným režimem nemohl výkonově rovnat. Microsoft se tedy rozhodl vytvořit nové grafické API, které by donutilo vývojáře opustit starý styl programování. V roce 1995, po jednom roce vývoje, je vypuštěna první verze DirectX.

Možná díky takto rychlému vývoji, se knihovna DX 1.0 nevyhnula určitým chybám, přesto poskytovala požadovaný výkon a tím splnila svůj účel. Během následujícího roku vznikly další dvě verze (DX 2.0 a DX 3.0). Od té doby vycházejí nové verze přibližně jednou za rok (podrobnější popis historie viz. [23]).

Co se tohoto projektu týká, dalo by se z celé knihovny DirectX využít jen Direct3D, což je část určená k vykreslování 3D grafiky. Direct3D se původně skládalo ze dvou částí: retained mód a immediate mód. Retained mód byl graf scény postavený na technologii COM, ve skutečnosti se ale nikdy příliš neujal. Herní vývojáři požadovali větší kontrolu nad činností hardwaru, než mohl tento mód nabídnout. Z tohoto důvodu přestal Microsoft od verze DirectX 3.0 retained mód dále vylepšovat.

S využitím immediate módu lze přistupovat k funkcím grafického hardware na té nejnižší možné úrovni. Výhodou je možnost fyzického usměrnění instrukcí a jejich spouštění, z čehož vyplývá maximální rychlost prováděných operací. Lze získávat různé informace přímo z grafické karty a následně je využívat v programu. V immediate módu lze pracovat jen se základními grafickými entitami (body, úsečky, trojúhelníky, polygony). Důležité je, že v tomto módu nelze využít zabudovaný geometrický engine (graf scény), navíc musí vývojář vlastnoručně implementovat různé funkce vykreslovacího řetězce, což je výhoda, protože použití vlastních algoritmů pro vykreslení a správu scény, je obvykle mnohem efektivnější a rychlejší, než obecný geometrický engine poskytovaný retained módem.

Knihovna DirectX je určena výhradně pro operační systémy z rodiny Windows, mezi platformami je tedy prakticky nepřenositelná. Existují však různé wrappery, které umožňují provozovat některé verze DX i na jiných OS. Zatím jsou však jejich možnosti značně omezené a ve většině případů znatelně zaostávají ve výkonu.

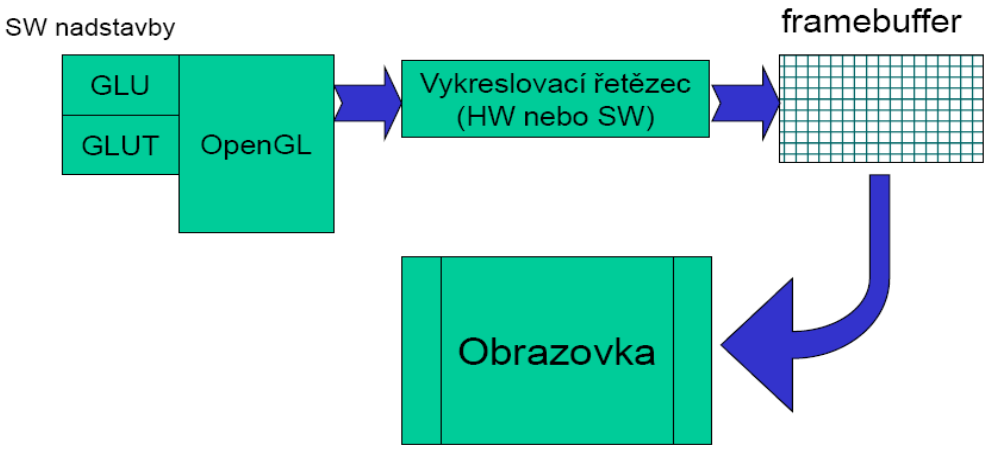
2.1.2 OpenGL

Knihovna OpenGL [14,16] (Open Graphics Library) byla navržena firmou SGI (Silicon Graphics Inc.) jako aplikační programové rozhraní (Application Programming Interface - API) k akcelerovaným grafickým kartám resp. celým grafickým subsystémům. Předchůdcem tohoto API byla programová knihovna IRIS GL (Silicon Graphics IRIS Graphics Library). Knihovna OpenGL byla navržena s důrazem na to, aby byla použitelná na různých typech grafických akcelerátorů. Dokonce i když není grafický akcelerátor nebo nějaká jeho funkce k dispozici, použije se implementovaná alternativa tzv. softwarová emulace. V současné době lze knihovnu OpenGL použít na všech dostupných operačních systémech, počínaje různými verzemi unixových systémů (včetně Linuxu a samozřejmě IRIXu), OS/2, Sun, až po platformy Microsoft Windows.

Knihovna OpenGL (na rozdíl od IRIS GL nebo Direct3D) byla vytvořena tak, aby byla nezávislá na použitém operačním systému, grafických ovladačích a správcích oken. Proto také neobsahuje žádné funkce pro práci s okny, pro vytváření grafického uživatelského rozhraní (GUI), ani pro zpracování událostí. Tyto funkce je možné volat buď pomocí konkrétního správce oken nebo lze použít některou z nadstaveb OpenGL, třeba knihovnu GLUT [15] (OpenGL Utility Toolkit). Pro dosažení co největší nezávislosti na použité platformě zavádí knihovna OpenGL vlastní primitivní datové typy, například GLvoid, GLbyte, GLboolean, GLint, nebo GLdouble.

Programátorské rozhraní knihovny OpenGL je vytvořeno tak, aby byla knihovna použitelná v téměř libovolném programovacím jazyce. Primárně je k dispozici hlavičkový soubor pro jazyky C a C++. V tomto souboru jsou deklarovány nové datové typy, symbolické konstanty a sada cca. 120 funkcí tvořících vlastní rozhraní knihovny. Podobné soubory s deklaracemi však existují i pro další programovací jazyky jako například Fortran, Object Pascal, Ada, C# či Java.

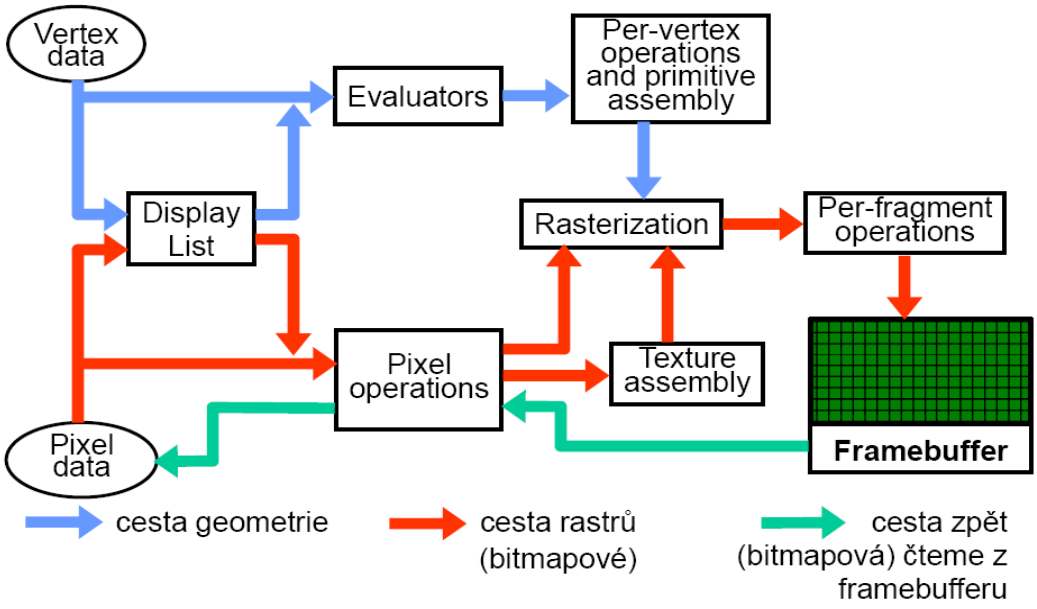
Z programátorského hlediska se OpenGL chová jako stavový automat. To znamená, že během zadávání příkazů pro vykreslování, lze průběžně měnit vlastnosti vykreslovaných primitiv (barva, průhlednost) nebo celé scény (volba způsobu vykreslování, transformace) a toto nastavení zůstane zachováno do té doby, než ho explicitně změníme. Výhoda tohoto přístupu spočívá především v tom, že funkce pro vykreslování mají menší počet parametrů a že jedním příkazem lze globálně změnit způsob vykreslení celé scény. Vykreslování scény se provádí procedurálně - voláním funkcí OpenGL se vykreslí výsledný rastrový obrázek. Výsledkem volání těchto funkcí je rastrový obrázek uložený v tzv. framebufferu (paměť snímku), kde je každému pixelu přiřazena barva, hloubka, alfa složka popř. i další atributy. Z framebufferu lze získat pouze barevnou informaci a tu je možné následně zobrazit na obrazovce – viz. obrázek 2.1 .



Obrázek 2.1: Vykreslování pomocí OpenGL [14]

Pomocí funkcí poskytovaných knihovnou OpenGL lze vykreslovat obrazce a tělesa složená pouze ze základních geometrických prvků, kterým se říká *grafická primitiva* (např. bod, úsečka, trojúhelník, polygon, bitmapa a pixmapa). Existují však rozšiřující knihovny, které funkcionalitu dále zvyšují. Jednou ze základních knihoven používaných společně s OpenGL je knihovna GLU (OpenGL Utilities), která umožňuje využívat *tesselátory* (rozložení nekonvexních polygonů na trojúhelníky), *evaluátory* (výpočet souřadnic bodů ležících na parametrických plochách) a vykreslovat *kvadriky* (koule, válce, kužely a disky). Mezi další hojně používané nadstavbové knihovny patří Open Inventor a OpenSceneGraph, pomocí kterých lze konstruovat celé scény složené z hierarchicky navázaných objektů. V porovnání s Direct 3D retained módem, kde se také pracuje s hierarchií scény, jsou však tyto knihovny mnohem mocnější a přitom disponují poměrně jednoduchým rozhraním.

Správu a rozšiřování standardu OpenGL má od roku 1992 nastarosti nezávislé konsorcium ARB (Architecture Review Board), do kterého patří firmy jako Hewlett-Packard, Digital Equipment Corporation, IBM, Intel, nVidia, ATI, Silicon Graphics a další (u založení byl i Microsoft). ARB je skupina společností, které schvalují rozšíření a vydávání dalších verzí OpenGL. Kromě toho také vydávají různé testy a jinak podporují rozvoj standardu OpenGL.



Obrázek 2.2: Vykreslovací řetězec podrobněji

Již od počátku je do OpenGL zahrnut i pružný systém rozšíření tzv. extensions. Každý výrobce grafického hardware tak může specifikovat svá rozšíření oproti standardu, např speciální funkce u nových grafických karet atd. Každé takové rozšíření má svůj specifický unikátní název, který začíná identifikací původce daného rozšíření (např. nVidia má značku NV). Označení EXT patří společným rozšířením a značka ARB reprezentuje extenze schválené konsorciem.

2.2 Shadery

S příchodem prvních grafických akcelerátorů (3dfx Voodoo v roce 1995) došlo k výraznému urychlení vykreslované 3D grafiky. Tento výkonový skok byl dosažen implementací časově náročných algoritmů přímo do čipu grafické karty, až do teď prováděl tyto výpočty procesor. Na jednu stranu přinesl tento přístup velký výkonový posun, ale na stranu druhou kvalita obrazu zůstala v podstatě stejná. Problém byl v tom, že grafické akcelerátory té doby neumožňovaly žádnou flexibilitu, byly takzvaně fixed-function. Fixed-function znamená, že algoritmy v grafických čipech byly pevně specifikované a tím limitovaly vývojáře, kteří je chtěli využívat ve svých aplikacích.

Modelem pro získání požadované flexibility se stal jazyk známého programu Photorealistic RenderMan od úspěšné firmy Pixar. Pomocí tohoto jazyka bylo možné popsat a následně vykreslovat fotorealistické scény (RenderMan stojí za prvním počítačově renderovaným celovečerním filmem – Toy Story). Výpočet scény byl však prováděn softwarově a tedy neumožňoval použití v interaktivní počítačové grafice.

Asi 6 let po příchodu grafických akcelerátorů, byla uvedena první cenově dostupná grafická karta, která implementovala systém podobný jazyku RenderMan, ale přímo procesorem této karty (GeForce3 v roce 2001). Díky této technologii bylo možné vykreslovat realistickou grafiku i v interaktivních aplikacích.

Principem programovatelných GPU (Graphics Processing Unit) je možnost specifikovat chování transformačních a rasterizačních jednotek na grafickém čipu. To znamená, že na rozdíl od pevně dané funkcionality dřívějších karet, je nyní plně v rukou programátora, jak budou 3D modely implementovány grafickým akcelerátorem.

Shader [18] je počítačový program určený pro zpracování přímo na grafické kartě. Tento program je napsán nejčastěji v jazycích Cg (univerzální jazyk od firmy nVidia), HLSL (jazyk pro platformu DirectX od firmy Microsoft) nebo GLSL (OpenGL) a později překladačem přeložen do assembleru přímo pro danou grafickou kartu. Tyto jazyky jsou si navzájem dosti podobné a syntaxí se blíží jazyku C. V současné době se shadery dělí na tři typy: vertex, pixel a geometry.

2.2.1 Vertex shader

Vertex shader [18, 19] je program, který nahrazuje modul zpracování vrcholů, provede se tedy s každým vrcholem vstupní geometrie.

Každý vrchol je specifikován svými parametry (pozice, barva, normála, texturovací koordináty atd.). Podstatou vertex shaderu je na základě těchto vstupních parametrů, které mohou být dále rozšířeny o uživatelsky definované globální proměnné, provést specifikovanou transformaci. Nejčastěji prováděné transformace jsou: výpočet osvětlení, pohyb s daným vrcholem, úprava normály, transformace texturovacích souřadnic, násobení vrcholu globální, modelovou a projekční maticí aj. Po aplikaci shader funkce na vstupní vrchol je výsledkem poloha daného vrcholu na obrazovce. Vertex shader nemůže měnit počet vrcholů (nemůže vrcholy přidávat či ubírat), ani typ respektive topologii právě zpracovávané primitivy.

Možnost programové změny transformačního řetězce představuje pro zkušeného programátora velmi silný nástroj pro implementaci nejrůznějších grafických efektů, například mapování obrázků okolního prostředí na zobrazované těleso (environment mapping), vykreslování skutečně hrbolatých povrchů (displacement mapping nahrazení „trikového" bump mappingu) či dokonce výpočet a vykreslování parametrických křivek a ploch.

Celý tento systém je realizován paralelní architekturou na grafickém čipu (dnešní grafické karty mají zpravidla desítky shader jednotek, případně několik paralelních jader). Výpočet probíhá

v reálném čase, což je velký pokrok od dob, kdy se podobné programy řešily softwarově na CPU za pomoci několika desítek počítačů.

2.2.2 Geometry shader

Jedná se o nový typ shaderu, který byl poprvé zaveden v DirectX 10 (Windows Vista), jehož úkolem je zpracovávat celé geometrické útvary (trojúhelníky, úsečky, body) složené z vertexů, které jsou již transformovány předchozím vertex shader programem. Geometry shader [18, 20] obdrží geometrický útvar jako pole vrcholů a může provádět operace typu: výpočet normály trojúhelníku, výpočet délky úsečky atd.

Novou možností je přidávat respektive rušit příchozí vrcholy a tím upravovat geometrii objektů. Z čehož plyne i to, že padá limitace jednoho jediného vertexu na vstupu a stejného jediného (byť nějak zpracovaného) vertexu na výstupu klasických vertex shaderů. Takovými technikami lze např. generovat sprity z jednoho vrcholu, řídit tesselaci modelů přímo na GPU apod. Geometry shader jednotku lze v pipeline úplně vypnout, v tomto případě jsou vrcholy posílány skrz, aniž by došlo k nějaké změně.

V současnosti podporují geometry shadery pouze grafické karty nVidia GeForce řady 8 nebo vyšší a některé karty z řady ATi Radeon 2x00 nebo lepší.

2.2.3 Pixel shader

Vrcholy, které prošly vertex respektive geometry shaderem, přichází na vstup pixel shaderu. Pixel shader [18, 19] neboli rasterizační program je implementován v samostatné programovatelné jednotce, která je, podobně jako výše popsaná jednotka pro vertex shader, taktéž umístěna na grafickém procesoru (korektnější název pro tento shader by byl fragment shader, neboť kromě samotných barev pixelů je možné programově měnit i další vlastnosti fragmentů například jejich hloubku).

Pomocí pixel shaderu je možné programově změnit původní operace prováděné nad vykreslovanými fragmenty. Mezi tyto operace patří zejména multitexturing, modulace textur, bumpmapping apod. Kromě modifikace těchto základních operací je možné programově aplikovat různé konvoluční filtry nebo další efekty na pixelové úrovni nezahrnuté v původním vybavení grafického akcelerátoru. Může se jednat o různé hranové filtry, filtry pro rozmazání vykreslovaných fragmentů apod. Pixel shadery lze také použít při vytváření složitějších efektů, například záblesků, rozšířeného bumpmappingu, jitteringu, tělesově orientovaného antialiasingu apod.

Programovatelný rasterizační řetězec opět nakládá se vstupními hodnotami (atributy) zcela libovolně, podobně jako vertex shader. K dispozici jsou veškeré informace, které obsahuje pevná (fixed-function) rasterizační jednotka. Vždy však musí být zaručeno, že výstupem z pixel shaderu budou již zmíněné informace o hodnotě fragmentu, tj. zejména jeho výsledná barva.

2.3 Kinematika

Kinematika je část mechaniky, která se zabývá klasifikací a popisem různých druhů pohybu, ale nezabývá se jeho příčinami. Naproti tomu dynamika zkoumá pohyb z hlediska působení vnějších sil. Kinematika k vyhodnocení pohybu používá veličiny jako pozice, rychlost, zrychlení atd. nesleduje však dynamické veličiny jako např. hybnost a energii, kterými se zabývá dynamika. Kinematika se dělí na dvě základní části, přímá neboli dopředná kinematika a inverzní kinematika.

2.3.1 Dopředná kinematika

Přímá neboli dopředná kinematika (forward kinematics) počítá, v případě kloubové soustavy, polohu koncového členu kinematického řetězce (tzv. end effectoru) ze známých poloh jednotlivých kloubů (viz. rovnice 2.1 kde X reprezentuje polohu koncového efektoru a $\vec{\Theta}$ konfiguraci kloubové soustavy). U kloubových soustav je typicky k dispozici transformační matice pro každý kloub (rotace + pozice). Postup, při kterém jsou nastavovány transformace jednotlivých členů soustavy, za účelem změny polohy této soustavy, se nazývá přímá kinematika. Výsledná poloha modelu (případně koncového efektoru) je dána spojenou transformací – postupné vynásobení transformačních matic jednotlivých členů soustavy.

$$\vec{\Theta} \rightarrow X \tag{2.1}$$

$$(\Theta_1, \Theta_2, ..., \Theta_n) \rightarrow X \tag{2.2}$$

$$X = f(\vec{\Theta}) \tag{2.3}$$

Snímková animace neobsahuje žádný kinematický řetězec nebo něco podobného, nelze tedy dopočítat polohu koncového členu respektive koncového efektoru. U snímkové animace je poloha celého modelu (všech vrcholů) dána buď pro každý snímek animace, nebo jen v některých snímcích, přičemž se zbylé snímky dopočítávají. Pokud je nutné mít k dispozici něco jako koncový efektor, např. když je potřeba měnit předměty v rukou postav, lze toho docílit třeba tím, že za koncový člen budou považovány některé z vrcholů modelu a jejich poloha bude tedy poloha koncového efektoru. Další možnost je umístit do modelu a jeho animace neviditelné objekty, jejichž pozice případně rotace budou reprezentovat polohy koncových členů.

2.3.2 Inverzní kinematika

Přímá kinematika hledá polohu koncového členu kinematického řetězce ze známých poloh jednotlivých kloubů. Inverzní kinematika [8] má oproti přímé zadánu polohu koncového členu (tzv. end effectoru) a hledá tomu odpovídající transformace jednotlivých kloubů. Tato úloha je mnohem složitější než úloha přímé kinematiky. Může mít dokonce nekonečně

mnoho řešení. Pro správnou funkci inverzní kinematiky je nutné nastavit pohybová omezení kloubů v kinematickém řetězci, tím dojde k vyloučení nereálných poloh jednotlivých částí. Inverzní kinematiku lze použít jen pro kloubovou soustavu, těžko by se daly dopočítat vrcholy modelu, pokud by tento neobsahoval kosti.

$$X \rightarrow \vec{\Theta} \tag{2.4}$$

$$X \rightarrow (\Theta_1, \Theta_2, \ldots, \Theta_n) \tag{2.5}$$

$$\vec{\Theta} = f^{-1}(X) \tag{2.6}$$

Algoritmy inverzní kinematiky nejsou triviální. Pro velmi jednoduché kinematické řetězce s nízkým počtem kloubů lze aplikovat algebraické řešení (soustava algebraických rovnic), což je nejrychlejší známý způsob. Bohužel pro rozsáhlejší kloubové soustavy dochází k explozi počtu řešení a už dále není možné takové soustavy tímto způsobem řešit. Další zajímavé algoritmy pro řešení úlohy inverzní kinematiky nabízí skupina analytických metod. Zde se nachází metody založené na pseudoinverzi jakobiánu, lineárním či kvadratickém programování.

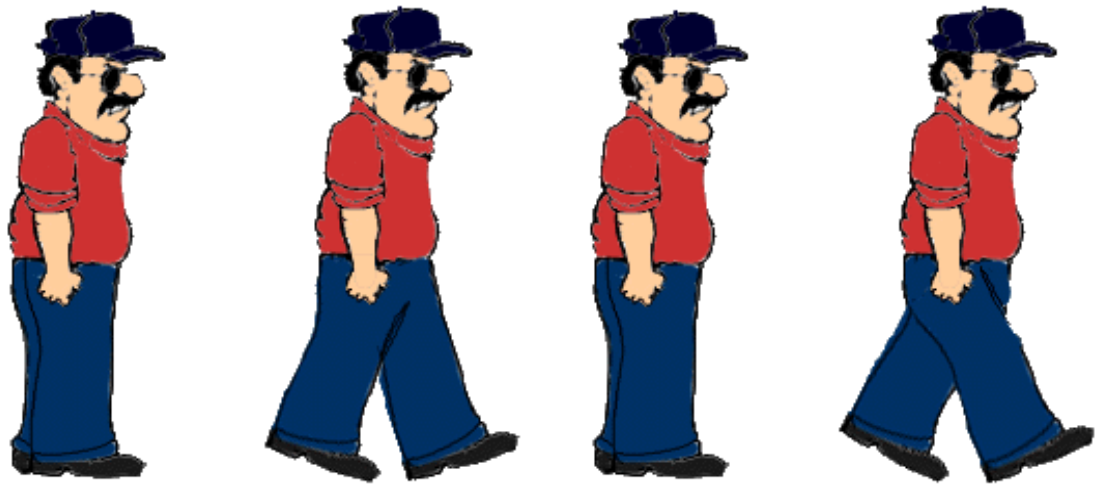
Inverzní kinematiku lze řešit také pomocí heuristických algoritmů. Nejznámějším zástupcem této třídy je bezesporu metoda CCD (cyclic coordinate descent). Principem CCD [25] je geometrické iterativní řešení. Algoritmus postupně minimalizuje chybu každé kosti, dokud není dosaženo požadované polohy s dostatečnou přesností. Dosažení požadované polohy však není garantováno. Se zavedením DOF (degree of freedom – omezení pohybu) dochází ke vzniku lokálních minim, které tento algoritmus nedokáže v základní podobě odhalit. Výhodou CCD algoritmu je: rychlost, relativně malá výpočetní náročnost a schopnost akceptovat více podmínek (lze zadat pozice několika koncových efektorů).

Inverze jakobiánu [27, 28] využívá následující skutečnosti: pokud existuje závislost koncového efektoru na stavovém vektoru kloubové soustavy, musí existovat i závislost inverzní. K výpočtu je využíváno diferenciálních rovnic. Klíčovou součástí těchto rovnic je invertovaná matice jakobiánu, která však nemusí vždy existovat kvůli redundancím ve stavovém prostoru. Proto vznikla metoda pseudoinverze jakobiánu, která inverzi matice nahrazuje transpozicí. Jedná se o iterativní algoritmus, který oproti dříve uvedeným metodám poskytuje nejlepší výsledky, zato však patří mezi nejpomalejší. Navíc je nutné řešit nestability v mezních polohách.

I když existují i další metody výpočtu inverzní kinematiky, velmi často se využívá kombinace několika algoritmů tak, aby byl výsledek nejlepším řešením daného problému.

2.4 Snímková animace

Snímková animace (morph target animation, per-vertex animation) je jednou z metod 3D počítačové animace. Počátky snímkové animace však sahají mnohem dále. Snímková animace se používala už v prvních kreslených filmech, kdy animátor vytvořil sadu statických obrázků, které se následně měnily před kamerou s cílem vytvoření plynulého pohybu.



Obrázek 2.3: Snímková animace ve 2D

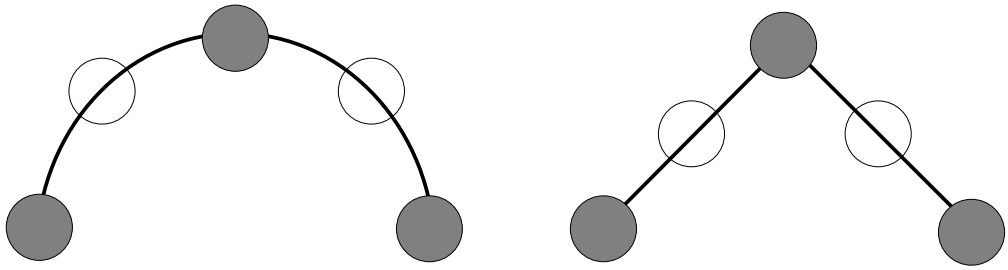
Počátkem 90. let se do popředí dostala počítačová animační technika zvaná morphing. A to díky popové hvězdě – Michaelu Jacksonovi. Ne, opravdu se nejedná o vtip. Král popu použil morphing v klipu k písni Black and White a tím stvořil fenomén, který přetrval až do současnosti. V daném klipu jsou momenty, kdy kamera zabírá horní část těla právě zpívající osoby a vždy po několika sekundách tento člověk plynule změní vzhled (morfuje) a stane se z něj někdo úplně jiný. Morphing v daném klipu vypadá opravdu úžasně a určitě stojí za shlédnutí i v dnešní době.

V tomto textu se pod pojmem snímková animace respektive morphing označuje plynulá změna tvaru 3D objektů v čase. Morphing (zkratka pro morph target animation) se často používá jako alternativa ke skeletální animaci. Snímková animace může být uložena v podstatě dvěma způsoby: buď se ukládají polohy všech vrcholů modelu pro každý snímek animace a renderer pouze zobrazuje předpočítané vrcholy pro daný snímek, nebo se ukládají jen klíčové snímky a pozice vrcholů mezi těmito snímky se musí dopočítávat. První zmíněný způsob uložení animace je ovšem extrémně náročný na spotřebu paměti. Pro plynulou animaci je nutné zobrazit alespoň 24 snímků za vteřinu, průměrný model má okolo 3000 vrcholů, kde každý vrchol obsahuje polohu v prostoru a normálu. Pokud by byly souřadnice uloženy jako 16 bitová reálná čísla, tak by pouhá jedna sekunda animace zabírala 864 kB paměti (24*3000*6*2B) a to nejsou započítány koordináty textury a další atributy, které není nutné ukládat pro každý snímek animace.

Obrázek 2.4: Morph target animation [17]

Používání snímkové animace má oproti skeletální animaci několik výhod. Výtvarníci mají mnohem větší kontrolu nad animovaným objektem, protože mohou upravovat pozice jednotlivých vrcholů v každém snímku animace a nejsou omezeni žádnou kloubovou soustavou nebo něčím podobným. Toho se hojně využívá například při animaci látek, kůže, mimiky obličeje či jiných objektů, které radikálně mění svou strukturu a tvar během animace, protože je obtížné (někdy dokonce nemožné) navázat tyto objekty na kostru respektive kloubovou soustavu.

Bohužel, jsou tu i nějaké nevýhody použití snímkové animace. Kromě již zmíněné paměťové náročnosti, je per-vertex animace obvykle i časově náročnější než skeletální animace, protože se při interpolaci musí počítat každý vertex zvlášť. Navíc pokud nejsou klíčové snímky vhodně vybrány nebo není použita kvalitní metoda interpolace, nevypadá výsledek příliš dobře. Například při základní metodě lineární interpolace, kdy body v mezi-snímcích leží na přímce, která spojuje odpovídající vrcholy v klíčových snímcích.



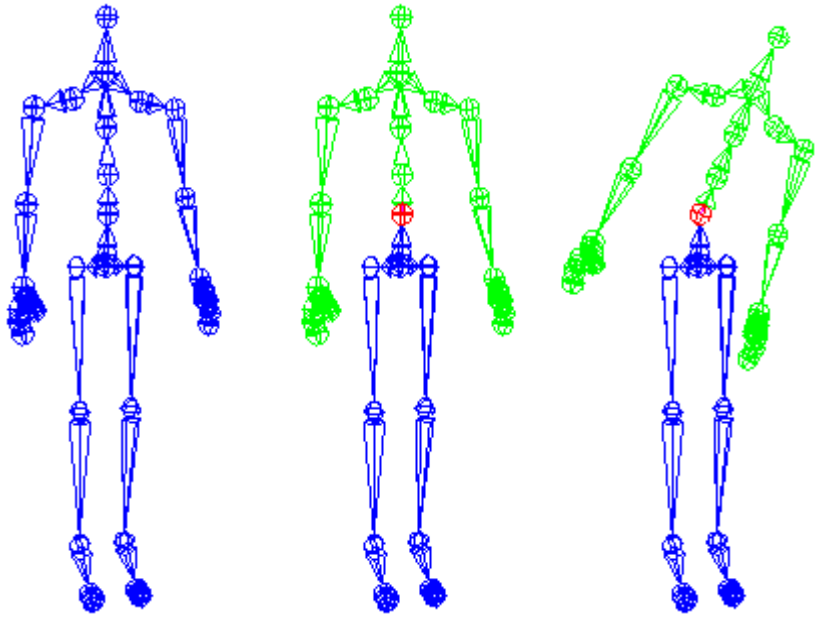
Obrázek 2.5: Spline a lineární interpolace klíčových snímků

Občas je nutné animaci vytvořenou v jednom programu přenést do jiné aplikace, třeba kvůli renderingu. Aby se předešlo problémům s exportem respektive importem animace v nativním formátu určité aplikace, je často tento problém řešen konverzí nativního formátu do snímkové animace a následným exportem. Tato operace je nezbytná, třeba když každý z použitých programů implementuje jinak skeletální animaci či jiné speciální efekty.

2.5 Animace kloubové soustavy

2.5.1 Kloub

Kloub, někdy označován jako kost (bone), je nejmenší část kloubové soustavy. Pomocí kloubů je definována transformace kloubové soustavy. Pozice kloubu i jeho rotace je relativní vzhledem k nadřazenému kloubu. Proto změna transformace, byť jen jednoho kloubu v hierarchii, ovlivní všechny podřazené klouby. Pokud jsou relativní vzdálenosti kloubů stálé a mění se pouze jejich rotace (tzn. délky kostí jsou konstantní), jedná se o rigidní kloubovou soustavu (rigid – tuhý, nepružný). Kloubová soustava člověka (kostra) a všech ostatních živočichů je rigidní.



Obrázek 2.6: Změna transformace kloubu ovlivní jeho potomky

Každý kloub je tedy definován svou transformací, která je relativní vzhledem k nadřazenému kloubu. Jedná-li se o kloubovou soustavu v trojrozměrném prostoru, používá se pro uložení této transformace 16ti prvková homogenní matice (4x4). Touto maticí lze popsat různé druhy transformací jako například: posun, rotace, změna měřítka, zkosení, atd. Prakticky se však v kloubové soustavě a její animaci využívá jen posun a rotace, v ojedinělých případech změna měřítka (scale).

Často je nutné vyjádřit transformaci kloubu v globální souřadné soustavě. Globální transformace určitého kloubu je dána součinem relativní transformace tohoto kloubu s globální transformací kloubu nadřazeného (rovnice 2.7), potřebné matematické základy jsou uvedeny zde [12].

$$M_i = R_i \cdot M_{i-1} \tag{2.7}$$

$$M_i = R_i \cdot R_{i-1} \cdot R_{i-2} \cdot \ldots \cdot R_0 \tag{2.8}$$

$$M_0 = R_0 \tag{2.9}$$

Kde M_i je neznámá globální transformační matice určitého kloubu. R_i je relativní transformační matice počítaného kloubu a M_{i-1} je globální transformační matice nadřazeného kloubu. Po jednoduché matematické úpravě vznikne rovnice (2.8), které se říká postupná transformace. Na konci této rovnice se vyskytuje matice R_0, což je relativní transformace kořenového kloubu. Protože ke kořenovému kloubu neexistuje kloub nadřazený, je jeho relativní transformace vztažena ke středu globálního souřadného systému. Proto je relativní transformace kořenového kloubu zároveň transformací globální (2.9).

2.5.2 Kost

Kost (bone) spojuje dva klouby v kloubové soustavě. Jeden z propojených kloubů je rodič a druhý je potomek. Každá kost má definovanou délku – vzdálenost propojených kloubů. Délka kosti je určena transformací potomka, konkrétně jeho polohou (transformace potomka je relativní vzhledem k rodiči). Pokud jsou délky všech kostí v soustavě konstantní, říká se této struktuře rigidní kloubová soustava.

Obvykle mají kosti význam pouze pro vizuální reprezentaci kloubové soustavy. Většinou se pojmem kost označuje kloub. Pokud se to týká kloubové soustavy, je holenní kost kolenním kloubem.

2.5.3 Limity kloubů

Obecný kloub, který je charakterizován transformační maticí, se může neomezeně pohybovat. Může vykonávat všechny transformace, které lze popsat 16ti prvkovou maticí. V praxi však není žádoucí, aby se klouby mohly libovolně pohybovat.

Jedním ze způsobů, jak omezit pohyb kloubu, je snížení počtu stupňů volnosti. Jeden stupeň volnosti (DOF – Degree Of Freedom) udává jednu dimenzi pohybu. U kloubových soustav mají obvykle jednotlivé klouby až 6 DOF (pozice + rotace). Snížení počtu stupňů volnosti znemožní kloubu pohyb v určitých dimenzích. Například lidský kolenní kloub má 1 DOF (může se otáčet pouze kolem jedné osy). Redukcí DOF u všech kloubů lze z obecné kloubové soustavy vytvořit soustavu rigidní.

Někdy však snížení stupňů volnosti nestačí a je třeba omezit rozsah hodnot v jednotlivých dimenzích. Například dříve zmíněný kolenní kloub se nemůže otáčet v plném rozsahu 360ti stupňů. V takovém případě je nejjednodušší omezit rotaci kloubu jen na určitý interval hodnot.

V praxi se limity kloubů používají hlavně v inverzní kinematice, kde toto omezení zabrání tomu, aby se kloubová soustava dostala do nereálné polohy. Ze stejného důvodu se limity kloubů používají i ve fyzikální simulaci (např. ragdoll).

2.5.4 Kloubová soustava

Kloubová soustava je hierarchická struktura vzájemně propojených kloubů. Tato struktura má podobu obecného stromu. Každý kloub v kloubové soustavě má svého předchůdce – nadřazený (rodičovský) kloub, toto neplatí pro kořen stromu. Kořen stromu je nejvyšší kloub v kloubové soustavě, jako jediný z celé struktury nemá žádného rodiče. V každém stromu se nachází právě jeden kořen. Každý kloub může mít jen jednoho rodiče, ale více potomků – podřazených kloubů (tzv. následníků). Pokud kloub nemá žádné potomky, jde o tzv. koncový kloub, respektive koncový efektor (end effector). V terminologii datových struktur se koncovému kloubu říká list (leaf) a kloubu, který má potomky, se říká uzel (node).

2.5.5 Kůže

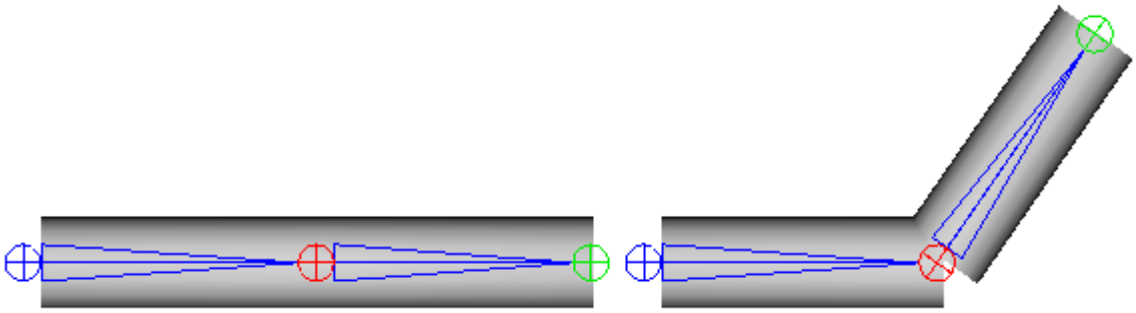
Kloubová soustava nemá sama o sobě žádnou fyzickou reprezentaci – není vidět. Při použití v robotice to příliš nevadí, protože relativní transformace kloubů přímo určují natočení jednotlivých robotických ramen nebo jejich částí.

V počítačové grafice bývá vetšinou ke kloubové soustavě přiřazena její grafická reprezentace. Jedná se nejčastěji o model složený z trojúhelníků. Tyto trojúhelníky představují povrch modelu. Takovému modelu se pak říká kůže (skin).

Obrázek 2.7: Model s přiřazenou kloubovou soustavou

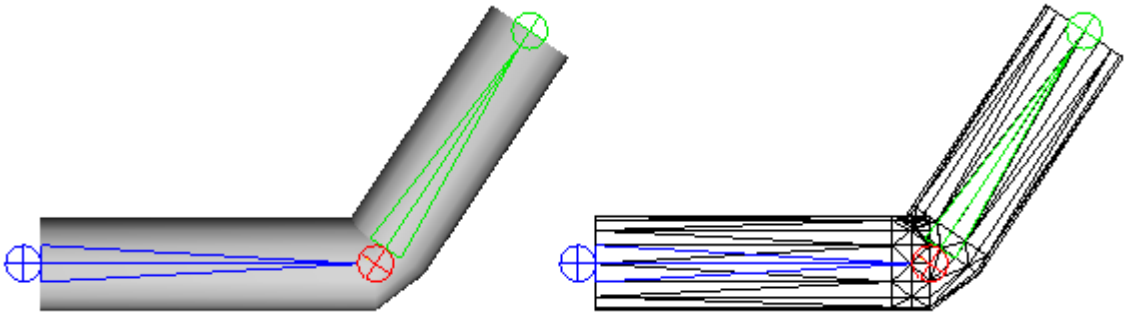
Samotný model pro reprezentaci kloubové soustavy nestačí, je nutné definovat, jak bude povrch modelu deformován danou kloubovou soustavou. Způsobů jak deformovat povrch modelu pomocí kloubové soustavy existuje mnoho. Jednotlivé přístupy se liší především výpočetní náročností a kvalitou výstupu.

Historicky nejstarší způsob deformace modelu pomocí kloubové soustavy předpokládal rozdělení modelu na objekty, které odpovídají jednotlivým kostem v kloubové soustavě. Následně stačilo spočítat změny transformace kostí (kloubů) vůči jejich výchozí pozici a tyto změny aplikovat na objekty přiřazené jednotlivým kostem. Tento způsob deformace je nejjednodušší s odpovídající kvalitou výstupu. Transformované části povrchu na sebe nemusí úplně doléhat, což má za následek vznik nerovností a děr v modelu.



Obrázek 2.8: Kosti deformují jednotlivé objekty

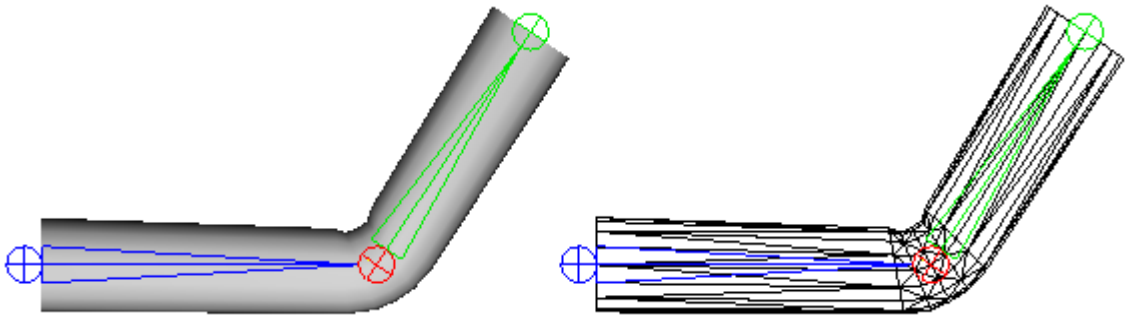
Jako vizuálně lepší řešení se ukázalo, nepřiřazovat jednotlivým kloubům objekty, ale jen určité vrcholy z modelu. V tomto případě není model rozdělen na objekty odpovídající jednotlivým kostem, ale je z jednoho kusu. Každý vrchol z modelu je přiřazen právě k jedné kosti (kloubu). Při transformaci kostí nevznikají v povrchu modelu díry, protože se nepohybují celé objekty, ale pouze vrcholy.



Obrázek 2.9: Na každý vrchol působí jedna kost

Graficky nejvydařenější deformace kůže je odvozena z předcházející metody. Rozdíl je v tom, že každý vrchol z modelu, který je deformován danou kloubovou soustavou, není přiřazen pouze

k jedné kosti, ale může být transformován obecně několika kostmi současně. V praxi se volí maximálně 4 kosti na jeden bod. Výsledná transformace vrcholu je dána váženým průměrem transformací od všech přiřazených kostí.



Obrázek 2.10: Na každý vrchol působí více kostí

V dnešní době se k deformaci kůže používá výhradně tato metoda, i když je výpočetně nejnáročnější.

2.5.6 Animace kloubové soustavy

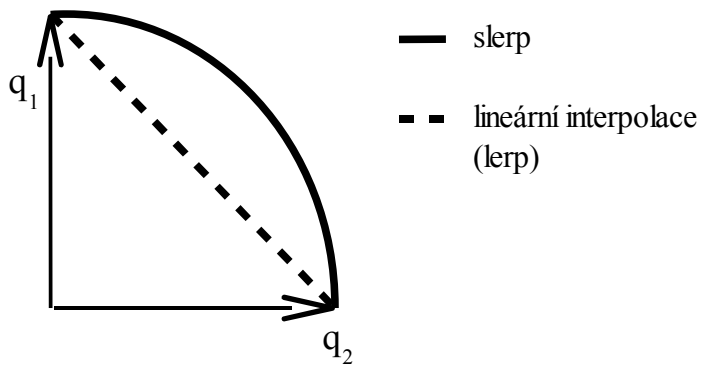
Základním principem animace je změna vybrané hodnoty v čase. V případě kloubových soustav se budou v čase měnit relativní transformace jednotlivých kloubů [13]. Ve většině případů se budou měnit pouze pozice a rotace kloubů.

Jednou z možností, jak uložit animaci kloubové soustavy, je zaznamenávat hodnoty transformačních matic jednotlivých kloubů v každém časovém okamžiku. Tento způsob je možné použít jen v případě, kdy jsou dopředu známy všechny časové okamžiky. Další nevýhodou je extrémní paměťová náročnost.

Podstatného zlepšení lze dosáhnout použitím tzv. animace klíčových snímků (keyframe animation). Klíčový snímek je nějaká významná poloha kloubové soustavy. Každý klíčový snímek je obvykle vztažen k určitému časovému okamžiku. Principem animace klíčových snímků je dopočítávání polohy kloubové soustavy mezi jednotlivými klíčovými snímky. Polohu kloubové soustavy v požadovaném čase lze zjistit interpolací dvou nejbližších klíčových snímků.

Další úspory paměti lze docílit jiným způsobem ukládání klíčových snímků. Změna oproti předcházející metodě je v tom, že klíčové snímky nezaznamenávají polohu celé kloubové soustavy, ale jen stavy určitých kloubů. Ukládají se jen hodnoty kloubů, které se vůči předcházejícímu klíčovému snímku změnily. Třeba v animaci, kde nějaká postava stojí a mává rukou, není nutné ukládat v každém klíčovém snímku polohy všech kloubů. Kompletní kloubovou soustavu stačí uložit jen v prvním a posledním snímku animace (kvůli interpolaci), v ostatních klíčových snímcích lze zaznamenat jen transformace kloubů přímo souvisejících s mávající rukou.

Komprimací ukládaných hodnot lze také ušetřit spoustu paměti a dokonce i zrychlit výpočet interpolace mezi klíčovými snímky. Ve skutečnosti se nejedná o komprimaci, ale o jiný způsob uložení dat. Trik spočívá v rozdělení transformační matice na dílčí transformace. Místo 16ti prvkové matice pro každý kloub se bude ukládat jen vektor polohy a vektor rotace daného kloubu. Stejně se nedají interpolovat přímo transformační matice, ale interpoluje se zvlášť poloha, respektive rotace kloubu. V trojrozměrné grafice bude mít vektor polohy 3 prvky [x, y, z]. Vektor rotace může mít 3 nebo 4 prvky podle toho, jestli bude rotace uložena jako eulerovy úhly [10] (roll, pitch, yaw) nebo jako kvaternion. Jako vhodnější se jeví uložení rotace ve formě kvaternionů [9], protože se tím ušetří výpočetní čas, který by jinak zabral převod eulerových úhlů na kvaterniony. Výhoda uložení rotace ve formě kvaternionů je při výpočtu interpolace mezi dvěma kvaterniony. Známý vzorec sférické lineární interpolace [9] (slerp) provede interpolaci rotace po nejkratším možném povrchu koule. Rozdělením transformační matice na rotaci a translaci zabere každý kloub v paměti zhruba polovinu původního místa (místo původních 16ti prvků se bude ukládat jen 3 + 4 = 7 prvků).



Obrázek 2.11: Lineární a sférická interpolace

2.6 Procedurální animace

Procedurální animace je technika, která se používá k automatickému generování animací, přičemž generování probíhá v reálném čase. To umožňuje tvorbu mnoha rozmanitých sekvencí reagujících například na aktuální chování uživatele, které by jinak musely být nahrazeny předdefinovanými animacemi.

Pomocí procedurální animace se obvykle simulují částicové systémy (kouř, oheň, voda), chování látek (oblečení), dynamika pevných těles, animace srsti či vlasů a animace postav. V počítačových a video hrách se procedurální animace používá pro jednodušší věci jako např. otáčení hlavy charakteru, když se hráč rozhlíží kolem sebe, ale i pro náročnější výpočty jakým je například ragdoll fyzika.

Ragdoll fyzika nachází uplatnění zejména při vytváření animace smrti postavy, kdy s použitím této techniky dopadne postava realisticky na zem, přičemž může dojít k interakci s okolními předměty. Ragdoll se obvykle skládá ze série rigidních těles [29], která jsou propojena v souladu s kloubovou soustavou modelu. Na tato tělesa působí fyzikální zákony, tím lze docílit velmi realistických efektů, které by se jen těžko daly vytvořit s použitím tradiční animace. Například postava může spadnout ze schodů nebo umřít na hraně propasti a následně se skoulet dolů, protože váha horní části těla s sebou stáhne zbytek.

3 Algoritmy

V následujících kapitolách budou podrobně rozebrány klíčové algoritmy pro snímkovou respektive kloubovou animaci postav v počítačové grafice.

Protože je tento projekt zaměřen primárně na interaktivní real-time grafiku, je nezbytné přijmout jistá omezení při výběru vhodných metod. Hlavním kritériem je rychlost, nelze však zanedbat ani kvalitu výstupů nabízených algoritmů. Vždy je proto nutné volit kompromis mezi rychlostí a kvalitou možných výstupů.

3.1 Snímková animace

Snímková (keyframe) animace je jednoduchá a efektivní cesta jak animovat 3D objekty. Jak už bylo popsáno dříve, snímková animace může být uložena v podstatě dvěma způsoby: buď se ukládají polohy všech vrcholů modelu pro každý snímek animace a renderer pouze postupně zobrazuje předpočítané vrcholy pro jednotlivé snímky, nebo se ukládají snímky jen v extrémních polohách tzv. klíčové snímky a pozice vrcholů mezi těmito snímky se musí dopočítávat. První zmíněný způsob uložení animace je ovšem extrémně náročný na spotřebu paměti a proto se téměř nepoužívá. Mnohem častěji je využívána interpolace klíčových snímků.

Interpolace (blending, morphing nebo tweening) je proces při kterém se vypočítává nová poloha mezi dvěma známými polohami. V případě snímkové animace jsou dopočítávány polohy všech bodů modelu mezi dvěma klíčovými snímky. Nejjednodušší metodou interpolace je tzv. lineární interpolace. Při této interpolaci se bod z jednoho klíčového snímku do následujícího klíčového snímku dostane po nejkratší možné dráze – po přímce. Pokud jsou známy časy jednotlivých klíčových snímků, neměl by být větší problém dopočítat hodnoty v mezi-snímcích. Základní vztah pro výpočet lineární interpolace ukazuje rovnice 3.1. Složitější metoda lin. interpolace, která zohledňuje i nestejné vzdálenosti mezi klíčovými snímky, je popsána v kapitole algoritmy kloubové soustavy.

$$p(t) = p_0 + t(p_1 - p_0) \quad t \in \langle 0,1 \rangle \tag{3.1}$$

Bohužel má lineární interpolace jednu podstatnou vadu a sice, že docela často dochází k deformaci modelu. Nejvíce je tento defekt patrný, když jsou jednotlivé klíčové snímky hodně daleko jeden od druhého. Například pokud je v jednom snímku zobrazena postava, která má ruce u těla a na dalším snímku má daná postava ruce vzpažené, dopadne lineární interpolace velmi špatně. V dopočítaných snímcích nebudou ruce postavy opisovat svým pohybem půlkruh, jak by asi každý čekal, ale budou se

pohybovat po nejkratší dráze z jednoho klíčového snímku do následujícího, což bude mít za následek deformaci modelu. Naštěstí existují postupy jak tento nechtěný jev eliminovat. První způsob spočívá v přidání dalších klíčových snímků do animace, tím se sice nevyřeší pohyb bodů po nejkratší dráze, ale bude-li klíčových snímků více respektive budou-li blíže u sebe, nebude tento defekt tolik patrný. Druhým řešením je použití lepší interpolační metody.

Interpolace pomocí Hermitovy kubiky na jednu stranu odstraňuje nepříjemné vlastnosti lineární interpolace, na druhou stranu si bere daň v podobě vyšší výpočetní náročnosti a nutnosti zadat o další dva krajní body více.

$$p(t) = (2t^3 - 3t^2 + 1)\, p_0 + (t^3 - 2t^2 + t)\, m_0 + (-2t^3 + 3t^2)\, p_1 + (t^3 - t^2)\, m_1 \qquad (3.2)$$

$$m_i = \left(\frac{1-\alpha}{2}\right)((p_i - p_{i-1}) + (p_{i+1} - p_i)) \qquad (3.3)$$

Kromě dalších dvou krajních bodů se ve výpočtu vyskytuje ještě hodnota α, která reprezentuje sílu jakou je křivka přitahována k jednotlivým tangentám m_0 a m_1. I když se tato hodnota může dle potřeby měnit, je ve většině případů konstanta $\alpha = 0$ naprosto vyhovující.

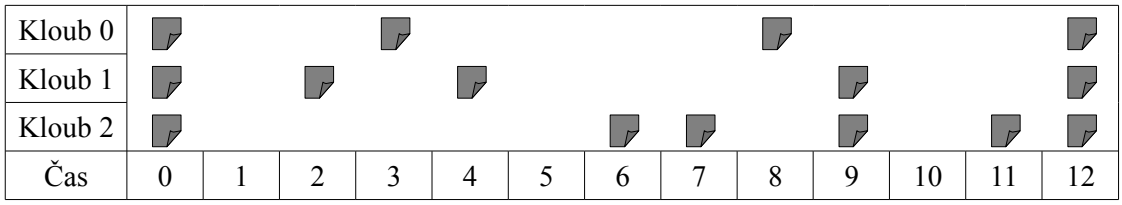
3.2 Animace kloubové soustavy

Algoritmy pro animaci kloubové soustavy by měli sloužit k reprodukci uložené animace. Ve většině programů je právě animace kloubové soustavy [13] stěžejní úlohou, která ovlivní všechny další algoritmy týkající se práce s kloubovou soustavou (např. skládání animací, prolínání animací atd.).

Animace kloubové soustavy patří do tzv. přímé kinematiky [11]. Jsou předdefinovány relativní transformace jednotlivých členů kloubové soustavy v určitých časech a je nutné dopočítat polohu kompletní kloubové soustavy, případně polohy koncových kloubů respektive koncových efektorů. Protože by nebylo příliš výhodné ani ekonomické ukládat transformace kloubů pro každý snímek animace, musel se vymyslet způsob redukce ukládaných dat. Obecně používaná redukce dat spočívá v uložení dílčích transformací jednotlivých kloubů místo kompletní 16ti prvkové transformační matice pro každý kloub. Pro kloubové soustavy to většinou znamená uložení informace o změně polohy respektive rotace kloubu. Ve výjimečných případech se ukládá i změna měřítka (scale). Tyto dílčí transformace lze snadno interpolovat, proto je animace kloubové soustavy uložena nejčastěji ve formě tzv. klíčové animace. Klíčová animace kloubové soustavy poskytuje spojitou animaci, adaptivní paměťovou náročnost a přijatelnou rychlost. Protože se v klíčové animace neukládá pozice kloubové soustavy v každém snímku, je nejdůležitějším algoritmem při reprodukci uložené kloubové animace interpolace klíčových snímků.

3.2.1 Interpolace klíčových snímků

Animace kloubové soustavy je v dnešní době nejčastěji zaznamenána jako posloupnost klíčových snímků. V každém klíčovém snímku je uložena transformace kloubové soustavy v určitém čase. Mezi jednotlivými klíčovými snímky (mezi jejich časy) nemusí být konstantní časové vzdálenosti. Příklad časového rozložení klíčových snímků je na obrázku 3.1.



Obrázek 3.1: Příklad časového rozložení klíčových snímků

Z obrázku je kromě časového rozložení klíčových snímků patrná ještě jedna podstatná věc: klíčové snímky jsou vázány k jednotlivým kloubům, tzn. není v nich uložena transformace celé kloubové soustavy, ale jen daného kloubu. Tím se šetří paměť, protože se ukládá poloha jen těch kloubů, které se v animaci mění.

Transformace kloubu není uložena jako matice 4x4, ale je rozdělena na dílčí transformace. Většinou se v klíčovém snímku ukládá vektor pozice a vektor rotace daného kloubu. Rozdělení transformační matice se dělá ze dvou hlavních důvodů: jednak se tím ušetří velké množství paměti (místo 16ti hodnot matice se ukládá jen 7 hodnot – pozice + rotace ve formě kvaternionu), ale hlavní důvod je ten, že nelze interpolovat přímo transformační matice.

Při dopočítávání klíčových snímků se interpoluje zvlášť pozice a zvlášť rotace (případně další transformace). Z výsledných interpolací jednotlivých dílčích transformací kloubu se nakonec sestaví kompletní transformační matice platná pro daný čas. Výsledkem interpolace kloubu musí být matice, protože se využívá při výpočtu deformace modelu.

Výpočet transformační matice kloubu pro čas *T*:

- Nalezení dvou nejbližších klíčových snímků pro vstupní čas *T*.
- Normalizace vstupního času *T*.
- Výpočet pozice kloubu v čase *T*.
- Výpočet rotace kloubu v čase *T*.
- Sestavení transformační matice pro čas *T*.

Při reprodukci animace kloubové soustavy se musí hodnota času *T* postupně inkrementovat od prvního do posledního snímku animace. Hodnota *T* je obvykle reálné číslo. Většinou je požadováno, aby se animace přehrávala na každém počítači stejnou rychlostí. Toho lze dosáhnout jednoduše tím,

že hodnota *T* bude růst v závislosti na rychlosti daného PC. Většinou se používá časovač, který měří dobu výpočtu a zobrazení jednoho snímku. Když bude hodnota *T* inkrementována tímto časem, dosáhne se nezávislosti na rychlosti počítače (např. na 2x výkonnějším počítači bude doba vykreslení snímku 2x menší, tím pádem bude hodnota *T* růst 2x pomaleji).

Když je nutné spočítat polohu celé kloubové soustavy v čase *T,* je nezbytné provést výše popsaný výpočet s každým kloubem této soustavy.

V prvním kroku algoritmu se hledají dva nejbližších klíčové snímky pro zadaný vstupní čas *T*. Jednotlivé klíčové snímky jsou pro každý kloub uloženy chronologicky podle hodnoty jejich časů. K jednotlivým snímkům se nejčastěji přistupuje pomocí indexu (jsou uloženy v poli, seznamu nebo vektoru). Hledá se takový nejmenší index, aby hodnota času klíčového snímku na daném indexu byla větší nebo stejná než zadaný vstupní čas *T*. Tím je nalezen nejbližší větší klíčový snímek (větší ve smyslu času). Nejbližší menší klíčový snímek je hned vedle (má o jedničku menší index).

Po nalezení dvou klíčových snímků, které jsou vhodné pro interpolaci, je nutné normalizovat hodnotu vstupního času *T*. Algoritmy pro interpolaci pozice, respektive rotace, předpokládají hodnotu *T* v rozsahu <0,1>.

$$Tn = \frac{T - T_{i-1}}{T_i - T_{i-1}} \qquad (3.4)$$

Tn je normalizovaná hodnota vstupního času *T*. T_i a T_{i-1} jsou časy dvou nejbližších nalezených klíčových snímků.

Výpočet pozice kloubu v čase *T* se obvykle provádí obyčejnou lineární interpolací mezi pozicemi nejbližších klíčových snímků. K výpočtu interpolace se používá normalizovaný čas *Tn*.

$$Pozice = Pozice_{i-1} + Tn(Pozice_i - Pozice_{i-1}) \qquad (3.5)$$

Pozice je relativní pozice kloubu v čase *T*. $Pozice_i$ a $Pozice_{i-1}$ jsou pozice dvou nejbližších nalezených klíčových snímků. *Tn* je normalizovaný vstupní čas. Protože jsou v trojrozměrném prostoru pozice kloubů definované vektorem o třech hodnotách, musí se výpočet (3.5) provést buď vektorově, nebo po jednotlivých složkách.

Algoritmus výpočtu rotace záleží na tom, v jakém formátu je rotace uložena. Z hlediska přesnosti je nejlepší ukládat a následně interpolovat rotace ve formě kvaternionů. Kvaternion [9] (quaternion) je specifikován čtyřmi reálnými hodnotami. Hlavní výhoda použití kvaternionů spočívá právě ve výpočtu interpolace mezi dvěma kvaterniony. Známý vzorec sférické lineární interpolace [9] (slerp) provede interpolaci rotace po nejkratším možném povrchu koule.

$$Q = Q_{i-1}\frac{\sin(1-Tn)\Omega}{\sin\Omega} + Q_i\frac{\sin\ Tn\,\Omega}{\sin\Omega} \tag{3.6}$$

$$\cos\Omega = Q_{i-1}Q_i \tag{3.7}$$

Q je relativní rotace (kvaternion) kloubu v čase T. Q_i a Q_{i-1} jsou rotace dvou nejbližších nalezených klíčových snímků. Tn je normalizovaný vstupní čas.

Nakonec je nutné z vypočtených hodnot pozice a rotace sestrojit transformační matici daného kloubu v čase T. To lze udělat převedením kvaternionu na matici 3x3, která reprezentuje rotaci a následně z této matice a vektoru pozice sestavit transformační matici 4x4.

3.2.2 Deformace kůže

Jak už bylo uvedeno, kostra neboli kloubová soustava nemá ve skutečnosti žádnou fyzickou podobu (není vidět). Aby bylo možné kloubovou soustavu zobrazit, je nutné k ní přiřadit něco, co lze vykreslit na obrazovku. Tím „něco“ je v počítačové grafice zpravidla 3D model složený z trojúhelníků. Tento model je následně deformován onou kloubovou soustavou.

Historicky nejstarší a nejjednodušší deformace modelu předpokládala rozdělení kůže na části (objekty), které jsou přiřazeny jednotlivým kostem kloubové soustavy. Následně stačilo spočítat změny transformace kostí (kloubů) vůči jejich výchozí pozici (bind pose) a tyto změny aplikovat na objekty přiřazené jednotlivým kostem. Zjednodušeně řečeno: o kolik se změní poloha kostí, o tolik se změní poloha přiřazených objektů.

Jeli taková kloubová soustava animována, její pohyb připomíná robota (pevné části propojené klouby). Tento nejstarší způsob deformace modelu neposkytuje příliš kvalitní výsledky, je však nejrychlejší, protože výpočtů je relativně málo (pro každou kost jeden). V dnešní době by se tato deformace uplatnila maximálně při animaci pevných částí případně mechanických soustrojí. Pro pohyb organických charakterů je nutné použít nějaký lepší algoritmus deformace modelu.

$$\overline{M}\cdot T = M \tag{3.8}$$

$$T = \overline{M}^{-1}\cdot M \tag{3.9}$$

T je transformace mezi aktuální polohou kloubu a jeho výchozí pozicí, tato transformace se aplikuje na přiřazený objekt. M je aktuální transformace kloubu v globálních souřadnicích. $\overline{M}$ je globální transformace kloubu ve výchozí pozici – pozice kdy se kloubová soustava přiřazuje k modelu.

Transformace jednotlivých pevných částí modelu již dávno nestačí. K dispozici musí být algoritmy, které dokáží deformovat povrch modelu pružně. Těchto algoritmů se pak využívá k deformaci

kůže živých organizmů. Rozdíl oproti předcházejícímu způsobu deformace je v tom, že povrch živých bytostí se nedeformuje po částech, ale přirozeně se podle kostry natahuje, ohýbá, krčí atd.

Nástupcem transformace pevných částí je tzv. pružná deformace modelu. Pružná deformace již netransformuje podčásti modelu, protože model nemusí být na žádné další části rozdělen. Pružná deformace transformuje kůži, jako by byla z jednoho kusu. Tím dochází k jejímu ohýbání a natahování.

Podstatou pružné deformace je, že k jednotlivým kostem (kloubům) jsou přiřazeny určité vrcholy z modelu kůže, která se následně ohýbá podle kloubové soustavy. Protože se nepohybují celé objekty, ale jen vrcholy, nevznikají v deformovaném modelu díry nebo jiné nerovnosti.

V nejjednodušší variantě pružné deformace je každý vrchol kůže přiřazen právě k jedné kosti (kloubu). Aby tato deformace fungovala, je nutné znát výchozí polohu kloubové soustavy a modelu kůže (bind pose).

Postup výpočtu polohy jednotlivých bodů kůže, která je deformována kloubovou soustavou, je následující. Nejprve je nutné transformovat vrchol ze souřadného systému daného modelu do souřadného systému přiřazeného kloubu (3.10). Tato transformace se provede vynásobením polohy vrcholu inverzní transformační maticí přiřazeného kloubu. Nutno dodat, že se jedná o polohu ve výchozím modelu kůže a o transformační matici z výchozí polohy kloubové soustavy.

$$\bar{v}_i = \bar{v} \cdot \overline{M}_i^{-1} \tag{3.10}$$

$\bar{v}_i$ je poloha vrcholu z výchozího modelu kůže přepočítaná do souřadného systému přiřazené kosti i. $\bar{v}$ je pozice vrcholu ve výchozím modelu. $\overline{M}_i$ je výchozí globální transformační matice kloubu i, který je přiřazen k počítanému vrcholu.

Teď, když je poloha vrcholu vztažena k souřadnému systému přiřazeného kloubu (počátek souřadnic je v daném kloubu), se po aplikaci jakékoli transformace na daný kloub provede stejná operace i s transformovaným vrcholem. Protože tím, že je transformován kloub, transformuje se i jím tvořený souřadný systém a tím pádem i body, které jsou vztaženy k tomuto souřadnému systému.

I když to zní možná složitě, je to jednoduché. Po transformaci vrcholu do souřadného systému přiřazené kosti stačí vynásobit jeho novou polohu aktuální transformací přiřazeného kloubu (3.11). Transformace kloubu musí být opět v globálních souřadnicích. Tím se s vrcholem provede stejná transformace jako s daným kloubem. Zároveň se vrchol transformuje zpět do souřadného systému kůže, ale jeho poloha bude deformována aktuální polohou kloubové soustavy.

$$v_i = \bar{v}_i \cdot M_i \tag{3.11}$$

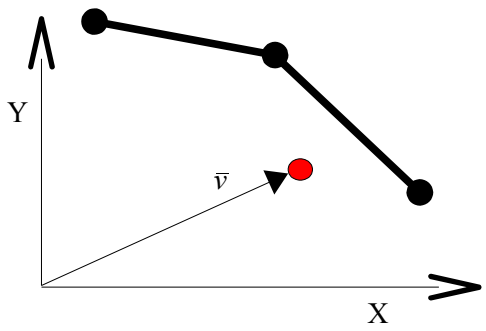
v_i je poloha vrcholu v souřadném systému kůže. Tento vrchol je již deformován aktuální transformací přiřazeného kloubu i. $\bar{v}_i$ je poloha vrcholu z výchozího modelu kůže přepočítaná do

souřadného systému přiřazené kosti i. M_i je globální transformační matice kloubu, který je přiřazen k vrcholu v_i.

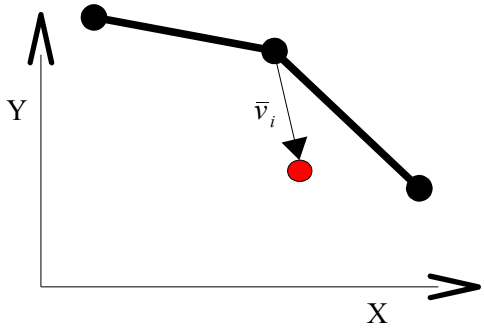
V rovnici (3.12) je uveden úplný výraz pro deformaci vrcholu, který je přiřazen k jednomu kloubu.

$$v_i = \bar{v} \cdot \overline{M}_i^{-1} \cdot M_i \qquad (3.12)$$

Jelikož se poloha vrcholů ve výchozím modelu (bind pose), ani transformace kloubů ve výchozí kloubové soustavě nemění, lze si předpočítat hodnoty $\bar{v}_i$ podle rovnice (3.10). Deformace kůže v každém snímku animace se tím zjednoduší na výpočet výrazu (3.11) pro každý vrchol modelu.

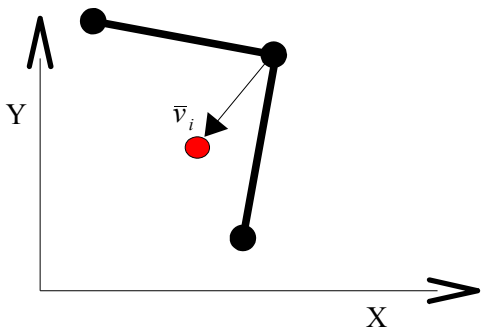


Obrázek 3.2: Výchozí poloha vrcholu

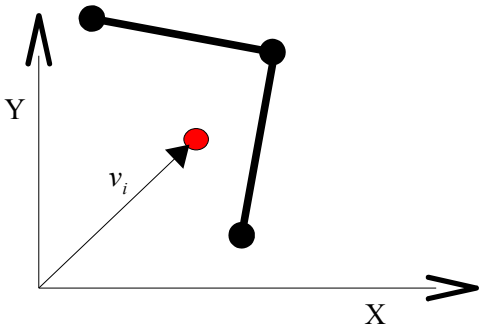


Obrázek 3.3: Po aplikaci vzorce (3.7)

Na horních dvou obrázcích (3.2 a 3.3) je vyznačena výchozí poloha kloubové soustavy (černé kolečka a černé čáry mezi nimi) a výchozí poloha vrcholu (červené kolečko), který je přiřazen k jednomu z kloubů. Je naznačeno, jak se po aplikaci vzorce (3.10) transformuje vrchol ze souřadného systému modelu do souřadného systému kloubu.



Obrázek 3.4: Změna kloub. soustavy



Obrázek 3.5: Deformovaný vrchol

Pozice vrcholu je následně ovlivněna transformací kloubové soustavy a tím i souřadných systémů jednotlivých kloubů (obrázek 3.4). Poslední obrázek (3.5) zobrazuje deformovaný vrchol po transformaci zpět do souřadného systému modelu.

Tato nejjednodušší varianta pružné deformace je vhodnou volbou mezi kvalitou výstupu a časovou náročností. I když se už v dnešní době téměř nepoužívá, je základem všech další metod pružné deformace.

V současnosti nejpoužívanější metodou pro real-time pružnou deformaci modelu je Skeletal Subspace Deformation [7]. Tento algoritmus je v literatuře a praxi rozšířena pod několika názvy např.: Linear Blend Skinning, Enveloping, Vertex Blending a další.

Rozdíl oproti předcházející metodě pružné deformace modelu je v tom, že v tomto algoritmu není každý vrchol přiřazen právě k jedné kosti, ale může být ovlivněn klidně všemi kostmi kloubové soustavy současně. Díky tomuto rozšíření je výsledkem daleko reálněji vypadající deformace kůže.

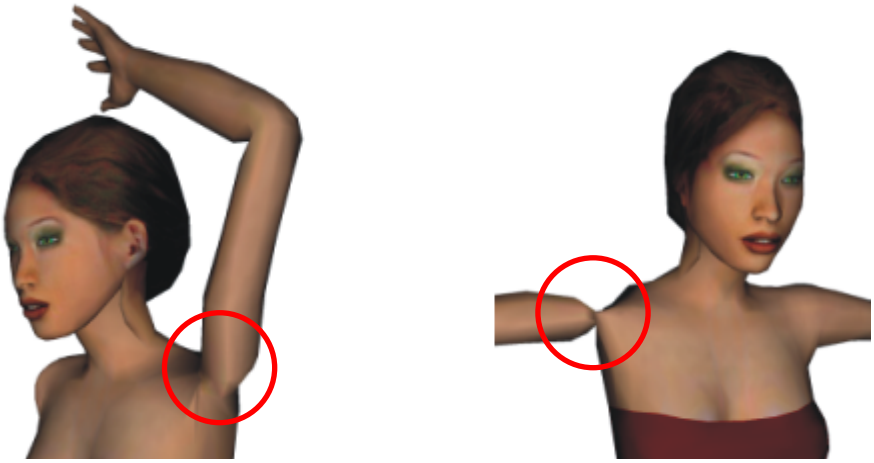
Skeletal Subspace Deformation vypočítá novou polohu vrcholu tak, že daný vrchol deformuje každou kostí zvlášť a pak z těchto dílčích poloh vrcholu spočítá výslednou pozici pomocí váženého průměru (3.13).

Aby bylo možné spočítat vážený průměr, musí být definovány nějaké váhy. Pojmem váha se označuje míra ovlivnění vrcholu určitou kostí. Kost, která má nastavenu nejvyšší váhu, bude mít na finální polohu daného vrcholu největší vliv. Kosti, které se na deformaci vrcholu nepodílejí, budou mít váhu 0. Váhy kloubů musí být pro každý vrchol nastaveny tak, aby splňovali rovnici (3.14).

$$v = \sum_{i=1}^{b} \bar{v} \cdot \overline{M}_i^{-1} \cdot M_i \cdot w_i \qquad (3.13)$$

$$\sum_{i=1}^{b} w_i = 1 \qquad (3.14)$$

v je výsledná poloha vrcholu po deformaci kloubovou soustavou. $\bar{v}$ je poloha počítaného vrcholu ve výchozím modelu kůže. M_i je globální transformační matice kosti (kloubu) s pořadovým číslem i. $\overline{M}_i$ je výchozí globální transformační matice kloubu i. w_i je míra ovlivnění počítaného vrcholu kostí i.



Obrázek 3.6: Ztráta objemu při extrémních rotacích kloubů [32]

Bohužel i tato metoda má několik zdokumentovaných nedostatků [1 – 6]. Nejhorším je asi to, že modely deformované touto technikou viditelně ztrácejí objem kolem kloubů, které jsou otočené do extrémních úhlů (obrázek 3.6). Tyto chyby vznikají, protože při výpočtu deformace vrcholu se transformační matice jednotlivých přiřazených kloubů lineárně interpolují. Lineární interpolace těchto matic ovšem neodpovídá lineární interpolaci jejich rotací.

Navzdory všem nedostatkům je SSD kvůli své jednoduchosti a výpočetní nenáročnosti stále velmi populární. Dokonce byly provedeny významné výzkumy v oblasti možného vylepšení tohoto široce používaného algoritmu. Jednou z cest je odstranění linearity obsažené v kombinaci transformačních matic kostí. Pánové Mohr a Gleicher navrhují použít v kloubech extra kosti, které rozdělí celkovou rotaci kloubu na dvě poloviny, více informací o této metodě zde [4]. Magnenat-Thalmann [30] použil speciální operátor pro blending matic. A neméně zajímavé jsou experimenty Kavana a Žáry [31] s lineární interpolací kvaternionů. Všechny zmíněné metody vylepšení algoritmu SSD jsou ovšem méně výpočetně efektivní. Kromě vylepšování stávající metody, ale vznikaly i úplně nové algoritmy, které se snaží obejít či minimalizovat chyby v animaci jiným způsobem. Mezi nejznámější patří Animation Space [3,7] a Multi-Weight Enveloping [6,7].

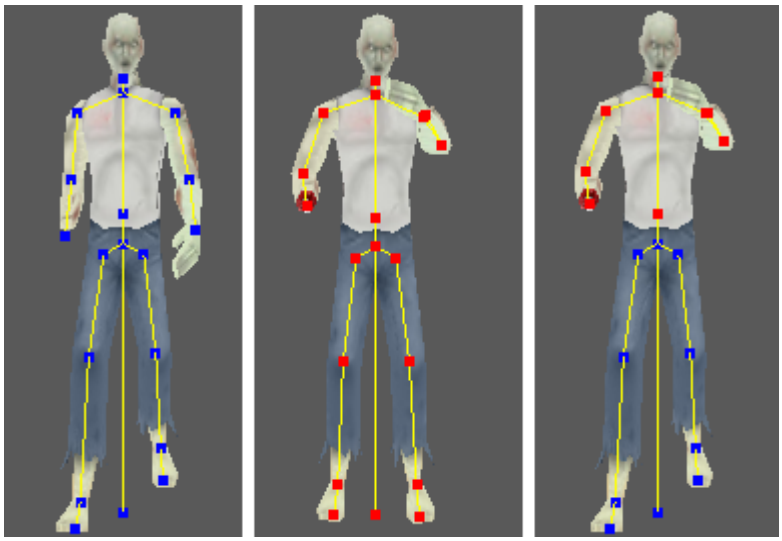
3.2.3 Plynulá změna animace

Klasická animace postavy, například v počítačové hře, je složena z velkého počtu dílčích animačních sekvencí, které se podle aktuálního dění ve hře respektive reakcí uživatele postupně přehrávají. Protože k přepnutí animační sekvence může dojít prakticky kdykoli v průběhu přehrávaní, je nemožné dopředu vytvořit plynulé přechody mezi jednotlivými sekvencemi. Takové přepnutí animace se pak jeví jako skokové. Skok je nejvíce patrný, když se polohy postav v přepínaných sekvencích výrazně liší. Třeba, když postava jde a najednou dojde ke změně animační sekvence na padání (např. když postavička stoupne do díry). Rozdíl mezi polohou modelu uprostřed chůze a prvním snímkem animace padání je velký, proto změna animační sekvence není plynulá a výsledek nevypadá příliš

dobře. Řešením tohoto problému je použití metody pro plynulou změnu animace respektive prolínání animací. Tato metoda funguje tak, že se mezi aktuální a cílovou sekvenci vloží krátká „přechodová“ animace. Vložená animace provede plynulou interpolací transformačních matic mezi kloubovou soustavou v aktuálním snímku a kloubovou soustavou z cílové sekvence (většinou z prvního snímku cílové sekvence). Po této vložené animaci následuje již klasické přehrávání cílové sekvence.

3.2.4 Kombinování animací

Kombinování animací respektive blending je velmi užitečná technika, při které dochází ke kombinaci dvou a více existujících animačních sekvenci s cílem vytvořit sekvenci novou. Příklad na obrázku 3.7 ukazuje dvě animační sekvence: chůzi (první obrázek) a útok rukama (druhý obrázek), které jsou zkombinovány do nové animace, kdy postava jde a zároveň útočí rukama (poslední obrázek).



Obrázek 3.7: Kombinace dvou animací

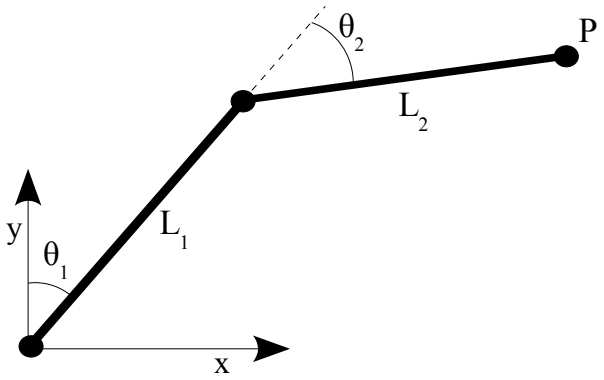
Princip této metody je velmi prostý. V prvním kroku se standardně vypočte nový snímek aktuální animace a aplikuje se na celou kloubovou soustavu. V následujících krocích se vždy vyhodnotí snímek (přesněji konfigurace kloubů) jiné animační sekvence s tím, že je výsledek aplikován jen na vybrané kosti kloubové soustavy. Tím je dosaženo smíchání dvou sekvencí. Obecně se dá smíchat libovolný počet různých animačních sekvencí, přičemž musí každá taková sekvence ovlivnit jen určité kosti, jinak by se navzájem přepsaly.

Blending animací má přímou podporu ve vytvořené animační knihovně, jež je výsledkem tohoto projektu. Metodě *Update(...)*, která slouží k výpočtu následujícího snímku v kloubové animaci, lze zadat parametr specifikující kloub z kloubové soustavy. Tento kloub je chápán jako kořen soustavy s tím, že vypočtené hodnoty následujícího snímku se aplikují jen na jeho potomky.

3.3 Inverzní kinematika

Zatímco přímá kinematika hledá polohu koncového členu kinematického řetězce ze známé konfigurace kloubové soustavy, inverzní kinematika dělá přesný opak. Inverzní kinematika [8] má oproti přímé zadánu polohu koncového členu (tzv. end effectoru) a hledá tomu odpovídající transformace jednotlivých kloubů. Tato úloha je mnohem složitější než úloha přímé kinematiky. Může mít dokonce nekonečně mnoho řešení.

Pro velmi jednoduché kinematické řetězce s nízkým počtem kloubů (většinou končetiny – ruce, nohy), lze aplikovat algebraické řešení (soustava algebraických rovnic), což je nejrychlejší známý způsob. Výhodou této metody je, že najde všechna možná řešení. Bohužel pro rozsáhlejší kloubové soustavy dochází k explozi počtu řešení a už dále není možné takové soustavy tímto způsobem řešit.



Obrázek 3.8: Jednoduchá soustava kostí ve 2D

Na obrázku 3.8 je jednoduchý kinematický řetězec složený pouze ze dvou kostí, vše je navíc jen ve 2D. Výpočet polohy koncového efektoru (P_x, P_y) ze známé konfigurace soustavy (jsou specifikovány hodnoty: θ_1, θ_2, L_1, L_2), by mohl vypadat nějak takto:

$$P_x = L_1 \sin(\theta_1) + L_2 \sin(\theta_1 + \theta_2) \tag{3.15}$$

$$P_y = L_1 \cos(\theta_1) + L_2 \cos(\theta_1 + \theta_2) \tag{3.16}$$

Jedná se tedy o přímou kinematiku velmi jednoduché soustavy o 2 DOF (stupně volnosti) a pouze ve 2D. Inverzní kinematika respektive výpočet úhlů θ_1 a θ_2 ze známé polohy koncového efektoru P řešená pomocí algebraické metody není ovšem triviální ani pro tento jednoduchý případ.

$$\theta_2 = \cos^{-1}\left(\frac{P_y{}^2 + P_x{}^2 - L_1{}^2 - L_2{}^2}{2 L_1 L_2}\right) \tag{3.17}$$

$$\theta_1 = \frac{-P_y L_2 \sin(\theta_2) + P_x(L_1 + L_2 \cos(\theta_2))}{P_x L_2 \sin(\theta_2) + P_y(L_1 + L_2 \cos(\theta_2))} \tag{3.18}$$

Další třídu algoritmů pro řešení úloh inverzní kinematiky tvoří skupina analytických metod. Sem patří například metody založené na pseudoinverzi jakobiánu, lineárním či kvadratickém programování.

Algoritmy vycházející z inverze jakobiánu využívají následující skutečnosti: pokud existuje závislost koncového efektoru na stavovém vektoru kloubové soustavy, musí existovat i závislost inverzní.

$$X = f(\vec{\Theta}) \tag{3.19}$$

$$\vec{\Theta} = f^{-1}(X) \tag{3.20}$$

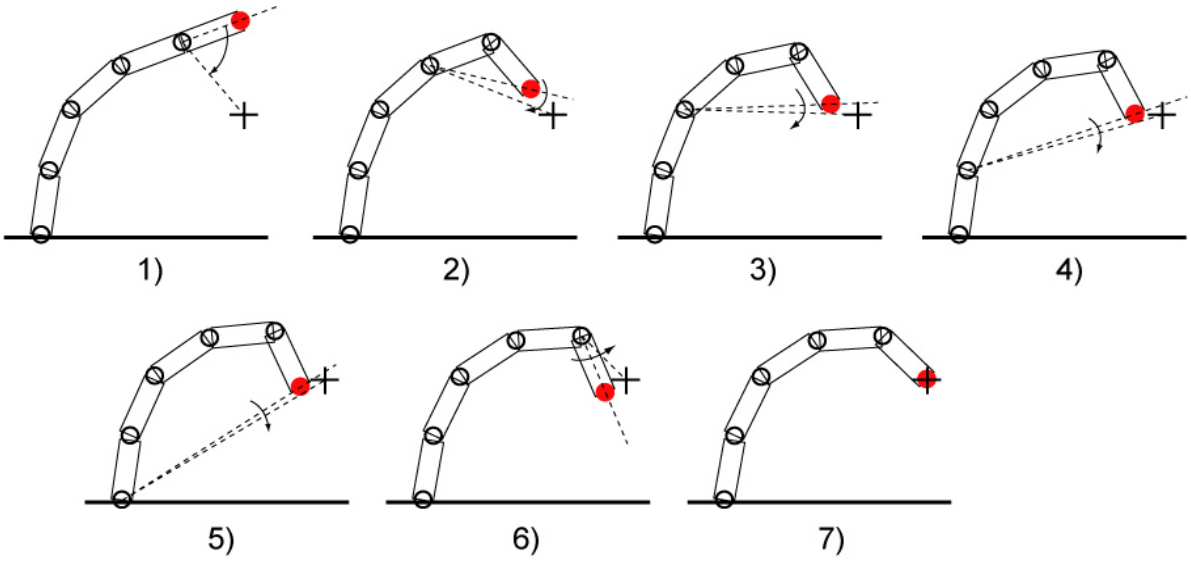
K výpočtu je využíváno diferenciálních rovnic. Klíčovou součástí těchto rovnic je invertovaná matice jakobiánu, která však nemusí vždy existovat kvůli redundancím ve stavovém prostoru. Jakobián je matice prvních parciálních derivací vektorové funkce. Jeho význam spočívá v tom, že je reprezentací nejlepší lineární aproximace k diferencovatelné funkci poblíž daného bodu. Způsobů jak vypočítat inverzi matice jakobiánu je několik: nahradit inverzi transponováním matice, pseudo-inverze jakobiánu, metoda nejmenších čtverců, inverze pomocí singulárního rozkladu aj. Více o zmíněných metodách zde [27, 28].

Inverzní kinematiku lze řešit také pomocí heuristických algoritmů. Nejznámějším zástupcem této třídy je bezesporu metoda CCD (cyclic coordinate descent). Algoritmus CCD postupně minimalizuje chybu každé kosti, dokud není dosaženo požadované polohy s dostatečnou přesností. Výhodou této metody je: rychlost, relativně malá výpočetní náročnost a schopnost akceptovat více podmínek (lze zadat pozice několika koncových efektorů). Protože je metoda CCD použita i v jednom z příkladů využití implementované knihovny (viz. příloha), následuje její podrobnější popis.

3.3.1 Cyclic Coordinate Descent

Tato metoda je kompromisem mezi rychlostí algebraických metod a přesností analytických metod. Poprvé tuto metodu představili Wang a Chen v [25]. CCD algoritmus odstraňuje nejvíce časově náročnou operaci analytických metod – pseudoinverzi jakobiánu, tím je výpočet výsledného řešení mnohem rychlejší. Další výhoda, vzhledem k algebraickým metodám, je možnost spočítat konfiguraci libovolně rozsáhlých kloubových soustav. Hlavním nedostatkem může být nutnost přepočítávat polohy všech kloubů v každé jednotlivé iteraci a obecně pomalejší konvergence k cílovému řešení.

CCD je heuristická iterativní metoda, která hledá nejvhodnější natočení kloubů v každé iteraci tak, aby bylo dosaženo co nejmenší odchylky od požadované polohy. Pro aktuální kloub se vyjádří úhel mezi koncovým efektorem a polohou cíle, načež je aplikována odpovídající rotace. Tento postup je opakovaně použit na každý kloub směrem nahoru (nebo dolů) v kinematickém řetězci, dokud nejsou splněny definované podmínky.



Obrázek 3.9: Iterativní řešení IK pomocí algoritmu CCD

Ve srovnání s metodami, které jsou založeny na inverzi jakobiánu, je iterativní postup mnohem rychlejší, přičemž je algoritmus předčasně ukončen, kdykoli je odchylka od cílové polohy menší než nastavený treshold. Rotace jednotlivých kloubů se řeší nezávisle, což není vždy vítané. Analytické algoritmy hledají odpovídající natočení jednotlivých kloubů souběžně, proto mají (oproti CCD) větší pravděpodobnost nalezení optimálních respektive minimálních rotací. Z pohledu animátora je tento fakt dosti nepříjemný, protože jednotlivé rotace nejsou rovnoměrně rozloženy do celého kinematického řetězce, navíc pokud dojde k minimalizaci odchylky moc brzy, bude ovlivněno jen prvních *n* kostí. Výsledná animace tak může působit poněkud nepřirozeně. Nicméně velkou výhodou algoritmu CCD je relativně rychlý výpočet a v případě menších odchylek od cílové polohy i velmi rychlá konvergence. Kromě toho je CCD imunní na vznik singularit.

Většinou lze v literatuře najít popis metody CCD pouze pro jednu cílovou podmínku a jeden kinematický řetězec [26]. Níže popsaná metoda umožňuje počítat s libovolným počtem podmínek a navíc můžou mít jednotlivé klouby nastavenu tuhost (η).

Je definovaná poloha cíle (target) P_t a vybrán nějaký kloub z kinematického řetězce, který reprezentuje koncový efektor, jeho poloha je P_i. Cílová podmínka je specifikována jako $\|P_t - P_i\| = 0$. Nyní se postupuje od koncového efektoru směrem vzhůru v kloubové soustavě a s každým kloubem se provedou následující operace. Poloha aktuálního kloubu je P_{i-n}. Sestrojí se dva vektory – jeden od aktuálního kloubu směrem ke koncovému efektoru a druhý od aktuálního kloubu směrem k cíli viz. rovnice (3.21).

$$v_1 = \frac{P_i - P_{i-n}}{\|P_i - P_{i-n}\|} \qquad v_2 = \frac{P_t - P_{i-n}}{\|P_t - P_{i-n}\|} \tag{3.21}$$

Pokud jsou vyjádřeny oba vektory, lze poměrně jednoduše zkonstruovat kvaternion q_{i-n}, který reprezentuje rotaci nutnou k zarovnání v_1 a v_2.

$$v_{i-n} = v_1 \times v_2 \tag{3.22}$$

$$\alpha = \frac{1}{2} \sin^{-1}(\|v_{i-n}\|) \cdot \eta \tag{3.23}$$

$$q_{i-n} = (\cos(\alpha), \sin(\alpha) \cdot \frac{v_{i-n}}{\|v_{i-n}\|}) \tag{3.24}$$

Vypočtený kvaternion q_{i-n} reprezentující rotaci je následně převeden na matici, kterou je vynásobena relativní transformační matice aktuálního kloubu R_{i-n}. Tím se celý podstrom od aktuálního kloubu otočí o úhel α a koncový efektor se přiblíží pozici cíle. Aby se změny v kinematickém řetězci projevily a algoritmus mohl pokračovat dalším kloubem ve směru nahoru od aktuální kosti, musí se přepočítat globální transformační matice všech kloubů v podstromu. Je důležité poznamenat, že pořadí v jakém se přepočítávají transformace jednotlivých kostí, ovlivní celkovou konfiguraci kloubové soustavy. Jak už bylo zmíněno dříve, iterační proces se zastaví, jakmile jsou splněny požadované podmínky, případně pokud byl překročen maximální nastavený počet iterací. Popsaný algoritmus upravuje kinematický řetězec směrem od listů ke kořeni. Tento způsob zaručuje, že kratší kosti, které se vyskytují hlavně v končetinách, mají vyšší prioritu než kosti v trupu.

3.4 Ragdoll

V počítačové grafice je ragdoll označení pro procedurální animaci, která se nejčastěji používá jako náhrada za tradiční statické animace smrti či různých pádů.

V prvních počítačových hrách se používaly ručně předvytvořené animační sekvence pro hráčovu smrt. Výhoda tohoto přístupu spočívala v nízkých nárocích na hardware, protože jediné co bylo třeba k animaci umírajícího charakteru, bylo vybrat tu nejvhodnější sekvenci z předpřipravených animací.

Postupné vylepšování výpočetní techniky umožnilo omezené využití real-time fyzikálních simulací. Ragdoll je tedy struktura rigidních objektů spojených dohromady systémem kloubů, které mají nastaveny limity omezující vzájemný pohyb jimy spojených objektů. Jednotlivé rigidní objekty

jsou přiřazeny ke kostem z kloubové soustavy modelu, přičemž je jejich velikost nastavena tak, aby co nejlépe aproximovaly povrch modelu postavy. Když pak postava např. padá na zem, dostane se ke slovu fyzikální engine, který počítá polohy rigidních objektů v jednotlivých snímcích. Podle poloh rigidních objektů je následně upravena kloubová soustava a tím pádem i celý model postavy. Díky použití fyzikálního enginu (gravitace, kolize s terénem a ostatními objekty ve scéně) vypadá výsledná animace velmi reálně.

Rigidní objekt (rigid body) je ideální tuhé těleso určité velikosti a váhy, jehož deformace se zanedbává tzn. vzdálenost mezi dvěma body tohoto tělesa je konstantní bez ohledu na vnější síly, které na těleso působí. Rigidní objekty se používají proto, aby byl výpočet fyzikální simulace co nejrychlejší (většinou v reálném čase), což by nebylo možné přímo s objekty scény, které mají často i několik tisíc polygonů.

Termín ragdoll vznikl podle problému, kterým trpí respektive trpěla většina fyzikálním enginem vytvořených animací kloubové soustavy. Kvůli hardwarovým omezením není nebo je, ale jen minimálně, simulována tuhost kloubů (svaly), což vede k animaci, která připomíná více dětskou hračku – hadrovou panenku (angl. rag doll) než živého člověka. Nezřídka skončí simulovaná postava v nereálné, komické nebo kompromitující poloze.

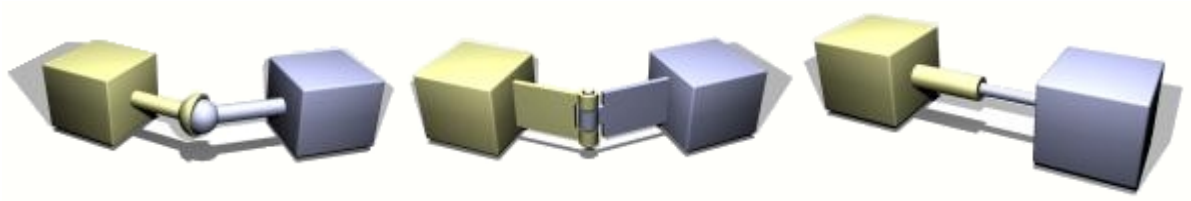
Kromě klasického ragdollu pomocí rigidních objektů a fyzikálního enginu existují i zajímavé pseudo-ragdoll techniky. Jednou z nich je verletova integrace, která byla použita ve hře Hitman: Codename 47. V této metodě je každá kost modelována jako bod, který je propojen s libovolným počtem podobných bodů pomocí jednoduchých vazeb. Verletovy vazby jsou mnohem jednodušší a tím i rychleji simulovatelné než klouby v klasickém ragdollu, což vede k menšímu zatížení CPU.

Další technika využívá inverzní kinematiky v kombinaci s post-procesingem a byla použita např. ve hře Halo. V této metodě se k animaci smrti používají předvytvořené sekvence, které jsou následně pomocí inverzní kinematiky umístěny do konkrétního prostředí. To znamená, že během animace se může postava dostat pod po povrch terénu nebo projít skrz okolní geometrii, ale jakmile animační sekvence skončí, vrátí se kloubová soustava do správné pozice pomocí inverzní kinematiky.

Poslední hojně využívanou technikou je spojení statické animace a ragdollu. Zatímco je přehrávána předpočítaná sekvence dochází ke korekci kloubové soustavy pomocí ragdollu, fyzikální engine kontroluje kolizi s okolními předměty a terénem. I když je přehrávání normální animace se současnou fyzikální simulací v podobě ragdollu náročnější na hardware počítače, zdá se, že přínos této techniky převáží nad nutností vlastnit výkonné CPU.

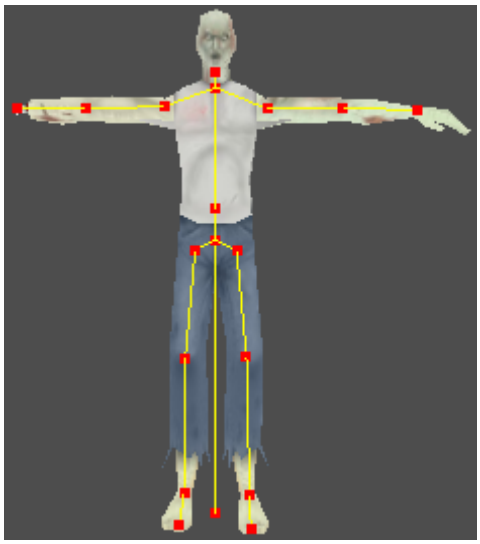
Konkrétní implementace ragdollu je do značné míry ovlivněna volbou fyzikálního enginu. Po funkční stránce jsou všechny enginy v podstatě stejné, každý poskytuje detekci kolizí a výpočet fyzikální simulace, rozdíl je v jejich programové struktuře. Některé enginy jsou objektově orientované a napsané v C++, jiné sází na volání jednotlivých funkcí v jazyku C. Samozřejmě se liší i v jiných věcech jako např. rychlost simulace, zatížení CPU případně GPU, kvalita dokumentace, cena atd.

Důležitým faktorem při výběru fyzikálního enginu je počet různých typů rigid objektů, které daný engine nabízí a které mohou být použity při detekci kolizí mezi ragdoll modelem a okolím. Několik základních tvarů rigid objektů však poskytuje každý engine. Mezi tyto základní tvary patří: kvádr, koule, válec a kapsle (válec se zaoblenými podstavami). Kromě rigid objektů umožňuje fyzikální engine vytvářet i tzv. klouby (angl. joints, constraints). Kloub definuje vztah mezi dvěma rigid objekty ve smyslu vymezení poloh a orientací, které mohou spojené objekty vůči sobě zaujímat. Základní typy kloubů jsou zobrazeny na obrázku 3.10.

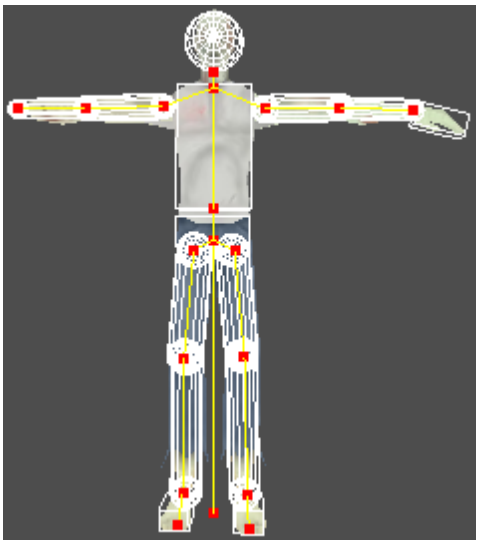


Obrázek 3.10: Základní typy kloubů ve fyzikálním enginu (ball and socket, hinge, slider)

Obecně se tedy ragdoll implementuje přibližně takto. Nejdříve je nutné vytvořit hierarchii rigid objektů podle modelu postavy. Většinou se předtím model transformuje do nějaké výchozí polohy, například do pozice T (viz. obrázek 3.11), aby se daly rigid objekty co nejjednodušeji a co nejpřesněji umístit. Rozmístění rigid objektů se řídí kloubovou soustavou modelu, přičemž velikost jednotlivých objektů se volí tak, aby co nejlépe aproximovaly tvar postavy (obrázek 3.12).



Obrázek 3.11: Model postavy v poloze T

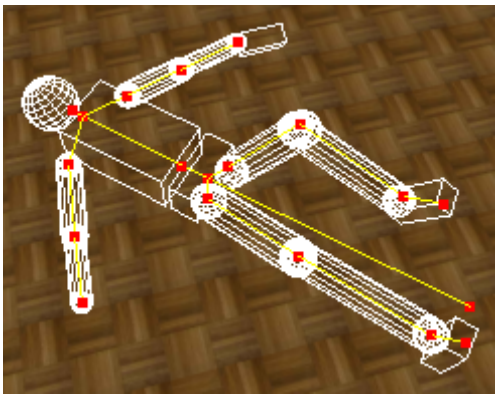


Obrázek 3.12: Rozmístění rigid objektů

Jakmile jsou všechny rigid objekty na svých místech, je nutné je pospojovat dohromady pomocí kloubů poskytnutých fyz. enginem. Rozmístění jednotlivých kloubů odpovídá kostře modelu respektive polohám kloubů v kostře modelu. Typ kloubu se volí podle odpovídajícího kloubu

v kloubové soustavě, konkrétně podle DOF a možností pohybu tohoto kloubu. Například pro kloub v koleni se použije typ hinge (umožňuje rotaci podle jedné osy), pro kloub v rameni se použije typ ball and socket (pohyb kloubu je vymezen kuželem) atd. Bližší popis jednotlivých typů kloubů lze nalézt v dokumentaci použitého fyzikálního enginu.

Dále je důležité uchovat informaci o vzdálenosti rigidních objektů od jim přiřazených kostí a to hned ze dvou důvodů: předně se této informace využije při aktivaci ragdollu, kdy může být kloubová soustava v jakékoli poloze a je nutné této poloze přizpůsobit i rigidní objekty. Ale hlavně je znalost této vzdálenosti nezbytná pro zpětnou úpravu kloubové soustavy podle aktuálních poloh rigidních objektů, což se dělá během samotné fyzikální simulace.



Obrázek 3.13: Fyzikální simulace

Obrázek 3.14: Modifikace polohy modelu

Fyzikální engine simuluje působení sil a kolize rigidních objektů, tím se mění jejich poloha a rotace. Následně jsou podle aktuálních transformací rigidních objektů upraveny pozice a rotace kloubů v kloubové soustavě (obrázek 3.13). Zároveň s modifikací kloubové soustavy, dochází k modifikaci celého modelu (obrázek 3.14).

3.5 Animace pomocí shaderu

I když jsou algoritmy 3D animace docela optimalizované a lze je bez větších problémů provozovat na libovolném hardwaru (záleží na počtu modelů ve scéně), existuje způsob jak tyto výpočty ještě urychlit. Řešením je grafický akcelerátor podporující technologii shaderů.

Shader [18] je krátký počítačový program určený pro zpracování přímo na grafické kartě. Tento program se je napsán nejčastěji v jazycích Cg, HLSL nebo GLSL a později překladačem přeložen do assembleru pro danou grafickou kartu. I když práci shaderů nelze dále zrychlit tím, že budou využívat již jednou vypočtené hodnoty (výstupy shaderů nelze uložit, DX10 toto částečně obchází), je přesto výpočet animace velmi rychlý, protože probíhá paralelně pro několik vrcholů a je akcelerován pevným vybavením počítače. Konkrétní příklad shaderu pro skeletální animaci lze najít v příloze práce.

4 Implementace knihovny

Knihovna (angl. library) je v programování funkční logický celek, který poskytuje služby pro programy. Většinou se jedná o sbírku procedur, funkcí a datových typů, či při objektově orientovaném přístupu o sadu tříd, uložených v jednom diskovém souboru.

Knihovna poskytuje aplikační programátorské rozhraní (API), které umožňuje programu volat funkce poskytované touto knihovnou. Existuje mnoho knihoven pro různé účely, např. pro využívání služeb operačního systému, grafické funkce, řízení periferií, vědeckotechnické výpočty atp.
Podle způsobu propojení knihovny a programu, který knihovnu využívá, se knihovny dělí na statické a dynamické. Podrobnější popis obou typů knihoven lze najít zde [21].

Při návrhu knihovny je nutné hledat optimum mezi komplexností knihovny a její jednoduchostí. Ne každý uživatel bude chtít studovat rozsáhlý manuál, aby byl schopen knihovnu použít, případně přidružovat složitou strukturu funkcí knihovny kvůli jedné metodě, kterou zrovna potřebuje. Komplexní funkčnost knihovny by měla být dostupná přes jednoduché a snadno použitelné rozhraní. Na hotovou knihovnu se pak dá dívat jako na černou skříňku (tzv. zapouzdřenost).

Dalším důležitým kritériem při návrhu knihovny je její znovupoužitelnost. Kvalitně navržená knihovna může dobře posloužit při vývoji budoucích projektů, kdy bude potřeba stejná nebo podobná funkčnost.

4.1 Datové struktury

Knihovna zhotovená v rámci této diplomové práce obsahuje, z ryze praktických důvodů, jen základní funkce pro kloubovou a snímkovou animaci. Díky tomu se velmi jednoduše používá, což by měl zvládnout i začínající programátor. Součástí projektu jsou dále jednoduché dobře komentované programy, které demonstrují možnosti této knihovny. Bližší popis jednotlivých demonstračních aplikací a ostatního software z přiloženého CD, lze najít v příloze této dokumentace.

Protože implementovaná knihovna zapouzdřuje funkce pro skeletální i snímkovou animaci, je důležité vhodně navrhnout datové struktury. S tím souvisí i hlavní problém: specifikace datových struktur, které jsou sice stejně pojmenované, ale obsahují odlišná data. Například struktura pro uložení vrcholu se musí nacházet jak ve snímkové, tak ve skeletální animaci. Zádrhel je v tom, že zatímco vrchol ve snímkové animaci obsahuje pouze polohu a normálu daného bodu, v kloubové animaci obsahuje vrchol navíc údaje o přiřazených kostech a jejich vahách.

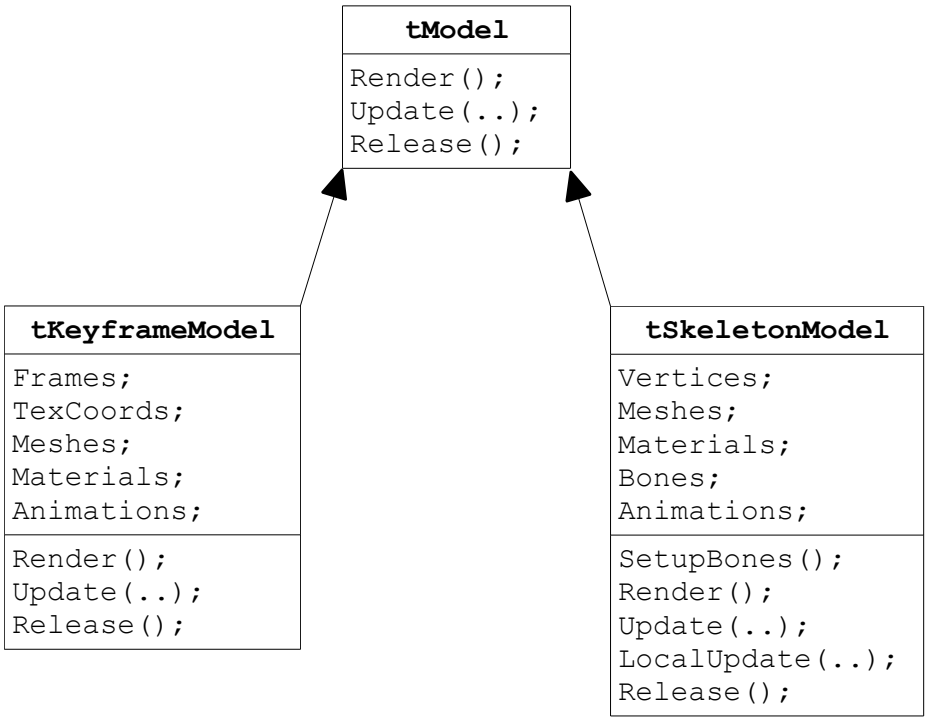
Nejjednodušším řešením je uložit třídy a datové struktury pro jednotlivé animace v samostatných souborech. Problém by nejspíš nastal v případě, kdy by bylo nutné použít zároveň oba typy animace, existovalo by více struktur se stejným názvem a překladač by zahlásil chybu.

Další možností je definovat datové struktury v rámci vytvořených tříd, např. struktury pro kloubovou animace definovat ve třídě pro kloubovou animaci a naopak. Nedostatkem tohoto řešení je velká nepřehlednost výsledného kódu, hlavně zmíněných tříd. Navíc jsem chtěl mít všechny struktury jednoduše přístupné z kteréhokoli místa programu, bez nutnosti použít operátoru '.' pro přístup k vlastnostem dané třídy.

Jako nejpřijatelnější řešení se nakonec ukázalo použití prostorů jmen (namespace) pro oddělení datových struktur snímkové a kloubové animace. Toto řešení je maximálně přehledné a struktury lze použít kdekoli v programu.

Implementovaná knihovna tedy definuje dva prostory jmen: *Keyframe* a *Skeleton* pro uložení struktur jednotlivých typů animace. Konkrétní datové struktury jsou vypsány v příloze této dokumentace.

4.2 Diagram tříd



tModel je abstraktní třída (nelze přímo z ní vytvářet objekty), která poskytuje základní metody pro práci s animovaným modelem v obecném formátu. Z této třídy jsou následně odvozeny dvě podtřídy, které jsou již specializované na konkrétní typy animace – *tKeyframeModel* (snímková animace) a *tSkeletonModel* (kloubová animace). Každá podtřída používá specifické datové typy definované v příslušných jmenných prostorech (proto mohou být použity stejné názvy struktur, i když tyto obsahují rozdílná data). Díky tomu, že jsou obě podtřídy potomky jedné globální třídy, která

specifikuje určité metody, které musí každý její potomek bezpodmínečně implementovat, lze pro práci s jednotlivými typy animací používat stejně nazvané funkce se stejnými parametry.

Základní rozhraní knihovny (lze vyčíst z diagramu tříd) je velmi jednoduché a osvojí si ho velmi rychle i začínající programátor. Knihovna poskytuje jen nejdůležitější funkce pro načítání, zpracování a zobrazení animací, ostatní pokročilé techniky jsou navrženy jako externí metody využívající jádro knihovny (viz. příklady použití animační knihovny v příloze tohoto dokumentu).

Pro tento způsob implementace jsem se rozhodl ze dvou důvodů: předně jsem chtěl zachovat maximální jednoduchost samotné knihovny, aby ji dokázal použít každý, aniž by musel věnoval nadměrné úsilí studiu dokumentace. A zadruhé jsem chtěl vytvořit knihovnu, kterou lze jednoduše integrovat do již hotových systémů, bez nutnosti zdlouhavě upravovat kód knihovny či daného systému. Navíc tento způsob dovoluje uživateli vybrat si jen ty techniky animace, které hodlá ve svém projektu využít. Uživatel jednoduše přilinkuje základní knihovnu do svého programu a následně podle frameworku a demonstračních aplikací přidá jen ty algoritmy, které skutečně potřebuje. Kromě toho jsou některé implementované metody závislé na konkrétních zdrojích, třeba ragdoll fyzika potřebuje fyzikální engine. Kdyby byl ragdoll implementovaný přímo v knihovně, musel by uživatel kromě knihovny připojit ke své aplikaci i zmíněný fyzikální engine. Pokud by ovšem uživatel ve svém projektu používal vlastní simulaci fyziky nebo engine jiný než animační knihovna, nastal by problém, buď by musel připojit další fyz. knihovnu nebo by bylo nutné složitě přepisovat jádro animační knihovny. Ve stávající podobě animační knihovny lze opět přilinkovat jen základní rozhraní a toto následně rozšířit o metody pro výpočet ragdollu. Upravení daných metod pro přítomný fyz. engine už není problém.

4.3 Použité prostředky

V této kapitole jsou krátce popsány nástroje a prostředky, které byly použity při práci na tomto projektu tzn. při implementaci knihovny pro animaci 3D postav a ostatních přiložených aplikací.

4.3.1 Code::Blocks

Code::Blocks (http://www.codeblocks.org/) je volně dostupné multiplatformní grafické rozhraní nad překladačem MinGW. Toto IDE je navrženo tak, aby uspokojilo potřeby co největšího počtu vývojářů (v podstatě každý se na jeho tvorbě může podílet). Code::Blocks je vytvořen tak, aby byl maximálně rozšiřitelný (pluginy) a zároveň si ho mohl každý upravit podle svých aktuálních potřeb (téměř vše se dá neomezeně nastavovat).

4.3.2 FreeGlut

Knihovna OpenGL byla vytvořena tak, aby byla nezávislá na operačním systému, grafických ovladačích a správci oken. Proto v ní také nejsou obsaženy žádné funkce pro práci s okny (otevírání, rušení, změna velikosti), pro vytváření grafického uživatelského rozhraní (GUI) nebo pro zpracování událostí. Tyto funkce, které jsou systémově závislé, se dříve musely naprogramovat pro každý operační systém zvlášť, což od vývojáře aplikace vyžadovalo podrobnou znalost funkcí daného operačního systému, grafické nadstavby a správce oken.

Knihovna FreeGlut (http://freeglut.sourceforge.net/), případně její starší varianta s názvem GLUT [33], částečně řeší problémy naznačené v předchozím odstavci. Definuje a implementuje aplikační rozhraní pro tvorbu oken a jednoduchého grafického uživatelského rozhraní, přičemž je systémově nezávislá, tj. pro práci s okny se na všech systémech používají vždy stejné funkce, které mají stejné parametry.

Nezávislost na operačním systému i platformě jde dokonce tak daleko, že se ve všech funkcích knihovny FreeGlut používají pouze základní datové typy jazyka C. Například při vytvoření okna je vrácen identifikátor okna jako kladné číslo typu int a ne HWND či ukazatel, jak je tomu u systémových funkcí Xlib nebo WinAPI. V současné době je knihovna FreeGlut používána především na Linuxu, BSD Unixech, IRIXu, OS/2 a na platformách Microsoft Windows.

4.3.3 Glui

Glui (http://sourceforge.net/projects/glui) je knihovna, která slouží k tvorbě uživatelského rozhraní. Jedná se o platformně nezávislou knihovna napsanou v jazyku C++. Glui (user interface library) je postavena nad knihovnou GLUT a neobsahuje žádný systémově závislý kód. Proto je program používající Glui stejný na všech platformách (knihovnu lze použít na veškerých systémech podporujících GLUT). Mimo jiné jsou k dispozici tyto ovládací prvky: buttony, checkboxy, radio buttony, panely, spinnery, textová pole a další.

4.3.4 Newton Game Dynamics

NGD (http://www.newtondynamics.com/) je fyzikální engine pro interaktivní simulaci rigidních těles. Knihovna poskytuje detekci kolizí, správu scény a kompletní řešení pro real-time simulaci realistického chování objektů v daném prostředí. Výpočet fyziky je prováděn deterministickým algoritmem, jenž není založen na tradičních LCP nebo iterativních metodách, ale poskytuje stabilitu a rychlost obou těchto přístupů. I když je NGD primárně určen pro použití v počítačových hrách, osvědčí se i v jiných real-time fyzikálních simulacích.

4.3.5 Cg

Cg (http://developer.nvidia.com/object/cg_toolkit.html) je vysokoúrovňový multiplatformní programovací jazyk pro grafické akcelerátory. Syntaxí je tento jazyk podobný programovacímu jazyku C. Jazyk Cg byl vyvinut firmou nVidia ve spolupráci s firmou Microsoft. V současnosti je možné v tomto poměrně jednoduchém a přitom mocném programovacím jazyce vytvářet programy pro vertex shader, pixel shader i geometry shader, přičemž jsou podporovány dvě v současnosti nejrozšířenější grafické knihovny – OpenGL a DirectX. Program v jazyce Cg je možné předkompilovat a v aplikaci použít pouze výsledný kód v jazyce nižší úrovně nebo lze použít run-time překladače, kdy kompilace programu ze zdrojového kódu probíhá až při spuštění programu. Druhá možnost má tu výhodu, že je umožněn překlad na téměř jakémkoli současném nebo v budoucnu vyrobeném grafickém akcelerátoru. Programátor aplikace může při inicializaci otestovat typ grafické karty a podle výsledku testu nastavit požadovaný výstup překladače. Tímto postupem je také zajištěna optimalizace kódu pro aktuální grafický akcelerátor.

4.4 Popis implementace

Hlavním implementačním jazykem bylo zvoleno C++, protože patří mezi nejrozšířenější jazyky a mám s ním dlouholeté zkušenosti. Navíc je knihovna primárně určena pro herní průmysl, kde se nejvíce uplatňuje právě C++.

Jako vývojové prostředí posloužilo IDE Code::Blocks. Jedná se o volně dostupné multiplatformní rozhraní nad překladačem MinGW, ale může být použito klidně s libovolným překladačem. Všechny vytvořené zdrojové kódy respektive zdrojové projekty, jsou tedy uloženy ve formátu Code::Blocks projekt (*.cbp). Ale s případným převodem třeba do Visual Studia by neměli být větší problémy. Pro zájemce, kteří by chtěli popsané vývojové prostředí vyzkoušet, je Code::Blocks umístěn na přiloženém CD.

První důležitější volba se týkala výběru grafického API. Vzhledem k tomu, že je vytvořená knihovna určena pro real-time interaktivní grafiku a jako implementační jazyk bylo zvoleno C++, přicházely reálně v úvahu pouze dvě grafické knihovny – Direct3D a OpenGL. Nakonec bylo vybráno OpenGL, protože je důležité, aby byla knihovna multiplatformní. Pokud by chtěl někdo využít knihovnu i v prostředí DirectX, neměla by být úprava zdrojových kódů příliš obtížná, protože v samotné knihovně se OpenGL používá jen minimálně. Horší by bylo přepsání doprovodných příkladů použití.

Aby bylo možné využít funkcí knihovny OpenGL, je nejdříve nutné inicializovat zařízení, do kterého se bude vykreslovat. Tímto zařízením je typicky okno aplikace. Bohužel otevření a správa okna se liší téměř na každé platformě. Abych zachoval multiplatformnost projektu použil jsem

knihovnu FreeGlut, což je nástupce velmi známé knihovny GLUT (OpenGL Utility Toolkit). Tyto knihovny zajišťují společné rozhraní pro správu oken. K dispozici jsou také funkce pro práci se vstupem z klávesnice, myši a jiných periferních zařízení.

S tvorbou stále složitějších programů (příkladů použití knihovny) bylo nutné vybrat knihovnu, která by umožňovala vytvořit alespoň základní uživatelské rozhraní a zároveň splňovala podmínku multiplatformnosti. Po vyzkoušení několika známějších knihoven typu FLTK (http://www.fltk.org/) a wxWidgets (http://www.wxwidgets.org/) jsem narazil projekt Glui. Jedná se o knihovnu pro tvorbu uživatelského rozhraní, která je vytvořena nad knihovnou GLUT respektive FreeGlut. Hlavní výhodou oproti ostatním knihovnám je maximální jednoduchost, s jakou lze vytvořit i poměrně složité ovládací panely či jiné prvky UI. Kromě toho, že většinu ovládacích prvků lze vytvořit pouhým jedním řádkem kódu, zavádí Glui i tzv. živé proměnné (live variables). Taková proměnná je přilinkována k určitému prvku UI a reflektuje jeho aktuální hodnotu (netřeba složitě reagovat na události a následně nastavovat hodnotu proměnné, stačí normálně pracovat s danou proměnnou, přičemž tato obsahuje stále aktuální hodnotu přilinkovaného prvku – hodnota se sama aktualizuje).

Nemalé potíže vznikly při výběru vhodného fyzikálního enginu, jenž je potřeba v příkladu demonstrujícím ragdoll fyziku. Nejprve jsem vyzkoušel volně dostupnou (včetně zdrojového kódu) knihovna Tokamak (http://www.tokamakphysics.com/), která se ale docela špatně ovládala, zvláště nastavení omezení jednotlivých typů kloubů mi zůstalo docela záhadou. Následně jsem otestoval relativně známou knihovnu ODE (http://www.ode.org/), která mne překvapila velmi dobrou dokumentací. Bohužel ani s tímto enginem jsem nebyl stoprocentně spokojen. Nakonec jsem použil volně dostupnou knihovnu NGD (Newton Game Dynamics), jež mi vyhovovala po stránce syntaxe i nabízené funkcionality. Navíc má tato knihovna velmi dobré forum, kde lze najít odpovědi na jakékoli otázky ohledně použití knihovny. V budoucnu bych chtěl přejít na jeden ze dvou profesionálních fyzikálních enginů, které jsou však zdarma k dispozici pro nekomerční použití. Jednou z těchto knihoven je Havok (http://www.havok.com/), který byl nasazen již v desítkách různých aplikací (Halo 3, BioShock, Half-Life), filmů (Matrix, Troja) a za kterým stojí firmy jako Microsoft, Sony, EA, Ubisoft, Activision a další. Druhou knihovou je bývalý fyz. engine Novodex, dnes známý jako PhysX (http://developer.nvidia.com/object/physx.htm), který byl rovněž použit již v mnoha aplikacích včetně UT3, Age of Empires III, Gears Of War, Gothic 3 atd.

5 Závěr

Po prostudování dostupných materiálů, byly vybrány ty nejzajímavější respektive nejpoužívanější algoritmy, které byly následně prezentovány v této práci. Na základě popsaných metod byly navrženy datové struktury a vytvořen objektový návrh animační knihovny. Po úspěšné implementaci byla vytvořena sada demonstračních aplikací, které prezentují možnosti knihovny. Obrázky z demonstračních programů včetně podrobnějšího popisu každé jednotlivé aplikace, lze nalézt v příloze této práce.

Díky tomuto projektu vznikl kvalitní framework, který umožňuje jednoduchou práci s libovolným typem animace (je podporována kloubová i snímková animace). Samotná knihovna nabízí velmi jednoduché rozhraní a použít ji dokáže i začínající programátor. Knihovna poskytuje jen nejdůležitější funkce pro načtení, zpracování a zobrazení animace, ostatní pokročilé techniky jsou součástí již zmíněných demonstračních aplikací. K tomuto kroku jsem se odhodlal ze dvou důvodů: předně jsem chtěl zachovat maximální jednoduchost samotné knihovny, aby ji dokázal používat každý, aniž by věnoval nadměrné úsilí studiu dokumentace. A zadruhé jsem chtěl vytvořit knihovnu, kterou lze jednoduše integrovat do již hotových systémů, bez nutnosti zdlouhavě upravovat kód knihovny či daného systému. Navíc tento způsob dovoluje uživateli vybrat si jen ty techniky 3D animace, které hodlá ve svém projektu využít. Uživatel jednoduše přilinkuje základní knihovnu do svého programu a následně podle frameworku a demonstračních aplikací přidá jen ty algoritmy, které skutečně potřebuje.

Součástí frameworku jsou kromě samotné knihovny a demonstračních aplikací i nástroje pro export hotových animací z populárních animačních systémů. Konkrétně se jedná skript, kterým lze exportovat snímkovou animaci z 3D Studia Max a uložit ji ve formátu, kterému rozumí animační knihovna. Druhým nástrojem je dynamická knihovna, která po nahrání do kořenového adresáře programu MilkShape 3D umožní ukládat kloubové animace vytvořené tímto programem do formátu animační knihovny.

Samotná knihovna je navržena s ohledem na možná budoucí rozšíření. V další verzi knihovny by tak mohly být zdokonaleny algoritmy inverzní kinematiky, které v současnosti zažívají nebývalý rozvoj. Kromě vylepšení stávajících metod by mohla být knihovna rozšířena i o úplně nové techniky inverzní kinematiky. Například o některou z hojně používaných analytických metod, které se mohou pyšnit velmi dobrými výsledky, i když za cenu větších nároků na hardware počítače. Dalšího vylepšení by se mohlo dostat kloubové animaci respektive algoritmům pro deformaci modelů. V současné době dosahuje nejlepších výsledků tzv. nelineární skinnování. Například metoda deformace pomocí dvou kvaternionů (Dual quaternion skinning [32]) by určitě stála za implementaci. V neposlední řadě by mohla práce pokračovat ve vývoji shaderů, které se stávají s nástupem nových grafických karet stále mocnějším nástrojem. Jako příklad lze uvést implementaci dokonalejších

osvětlovacích modelů a hlavně vržených stínů, které nejsou v současné verzi knihovny podporovány vůbec.

Myslím, že tento projekt (dokumentaci i výsledný framework) uvítají zejména začínající a mírně pokročilí uživatelé, kteří by rádi ve svých programech využili možnost jednoduše zobrazit různé typy animace, případně se chtějí dozvědět něco více o algoritmech zpracování animací respektive dalších technikách používaných v současném herním průmyslu (ragdoll, inverzní kinematika, blending animací atd.). Doufám, že podobně jako jsou NeHe tutoriály odrazovým můstkem pro všechny začínající OpenGL vývojáře, budou i mé demonstrační aplikace, spolu s implementovanou knihovnou, cenným zdrojem informací pro programátory tápající nad základními technikami 3D animace.

Literatura

[1] Kry, Paul G., James, Doug L., Pai, Dinesh K.: Eigenskin: real time large deformation character skinning in hardware. In Proceedings of the ACM SIGGRAPH Symposium on Computer Animation, ACM Press, 2002.

[2] Lewis, J. P., Cordner, M., Fong, N.: Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In Proceedings of the 27th annual conference on Computer graphics and interactive techniques, ACM Press, 2000.

[3] Merry, B., Marais, P., Gain, J.: Animation space: A truly linear framework for character animation. ACM Trans. Graphics, 2006.

[4] Mohr, A., Gleicher, M.: Building efficient, accurate character skins from examples. ACM Trans. Graphics, 2003.

[5] Sloan, P. J., Rose, C. F., Cohen, M. F.: Shape by example. In Proceedings of the 2001 symposium on Interactive 3D graphics, ACM Press, 2001.

[6] Wang, X. C., Phillips, C.: Multi-weight enveloping: least-squares approximation techniques for skin animation. In Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation, ACM Press, 2002.

[7] Jacka, D., Reid, A., Merry, B., Gain, J.: A comparison of linear skinning techniques for character animation. In Proceedings of the 5th international Conference on Computer Graphics, Virtual Reality, Visualisation and interaction in Africa, ACM Press, 2007.

[8] Kolda, P.: Knihovna pro pokročilou animaci kloubových soustav, diplomová práce, FIT VUT v Brně. Brno, 2006.

[9] Dam, E., Koch, M., Lillholm, M.: Quaternions, interpolation and animation, Technical Report DIKU-TR-98/5, University of Copenhagen, 1998.

[10] Eulerovy úhly. [online], [cit. 2008-01-02].
URL <http://en.wikipedia.org/wiki/Euler_angles>

[11] Přímá kinematika. [online], [cit. 2008-01-02].
URL <http://en.wikipedia.org/wiki/Forward_kinematic_animation>

[12] Alexa, M.: Linear combination of transformations. In SIGGRAPH 02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques, ACM Press, 2002.

[13] Animace kloubové soustavy. [online], [cit. 2008-01-02].
URL <http://en.wikipedia.org/wiki/Skeletal_animation>

[14] Tišinovský, P.: Grafická knihovna OpenGL. [online], [cit. 2008-04-18]
URL <http://www.root.cz/clanky/graficka-knihovna-opengl-1/>

[15] Kilgard, M. J.: The OpenGL Utility Toolkit (GLUT) Programming Interface, API Version 3. Silicon Graphics Inc., November 13, 1996.

[16] Wright, R. S.: OpenGL SuperBible. Waite Group Press, Macmillan Computer Publishing, 1999.

[17] Gary, J. K.: Morphing and animation. [online], [cit. 2008-04-22].
URL <http://www.seas.upenn.edu/~cis665/Animation Morphing.ppt>

[18] Shader. [online], [cit. 2008-04-22].
URL <http://cs.wikipedia.org/wiki/Shader>

[19] Tišinovský, P.: Grafické karty a grafické akcelerátory (21). [online], [cit. 2008-04-22].
URL <http://www.root.cz/clanky/graficke-karty-a-graficke-akceleratory-21/>

[20] Mrkvička, T.: Informace o Direct3D 10, projekt do GZN, FIT VUT v Brně, Brno, 2007

[21] Encyklopedie - Knihovna (programování). [online], [cit. 2008-04-28].
URL <http://encyklopedie.seznam.cz/heslo/134725-knihovna-programovani>

[22] Embedded systém. [online], [cit. 2008-04-28].
URL <http://cs.wikipedia.org/wiki/Embedded_syst%C3%A9m>

[23] Direct3D. [online], [cit. 2008-04-28].
URL <http://en.wikipedia.org/wiki/Direct3D>

[24] Fědor, M.: Application of inverse kinematics for skeleton manipulation in real-time. In Proceedings of the 19th Spring Conference on Computer Graphics, ACM Press, 2003

[25] Wang, L. C., Chen, C. C.: A Combined Optimization Method for Solving the Inverse Kinematics Problem of Mechanical Manipulators. IEEE Trans, 1991

[26] Welman, C.: Inverse Kinematics and Geometric Constraints for Articulated Figure Manipulation. Master thesis, Simon Fraser University, 1993

[27] Buss, R. B., Kim, J.: Selectively Damped Least Squares for Inverse Kinematics. In Journal of Graphics Tools, vol. 10, no. 3, 2005

[28] Buss, R. B.: Introduction to Inverse Kinematics with Jacobian Transpose, Pseudoinverse and Damped Least Squares methods. University of California, 2005

[29] Rigid body. [online], [cit. 2008-05-08].
URL <http://en.wikipedia.org/wiki/Rigid_body>

[30] Magnenat-Thalmann, N., Cordier, F., Seo, H., Papagianakis, G.: Modeling of bodies and clothes for virtual environments. In Third International Conference on Cyberworlds , 2004

[31] Kavan, L., Žára, J.: Spherical blend skinning: a real-time deformation of articulated models. In Proceedings of the 2005 symposium on Interactive 3D graphics and games, ACM Press, 2005

[32] Kavan, L., Collins, S., O'Sullivan, C. Žára, J.: Skinning with Dual Quaternions. ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, ACM Press, 2007

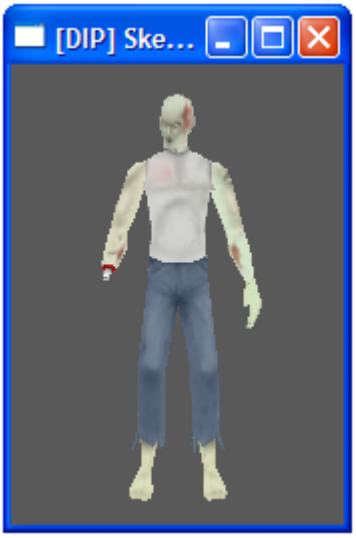
[33] Tišinovský, P.: Seriál o knihovně GLUT. [online], [cit. 2008-05-08].
URL <http://www.root.cz/clanky/glut-1/>

Seznam příloh

Př. 1. Příklady použití animační knihovny

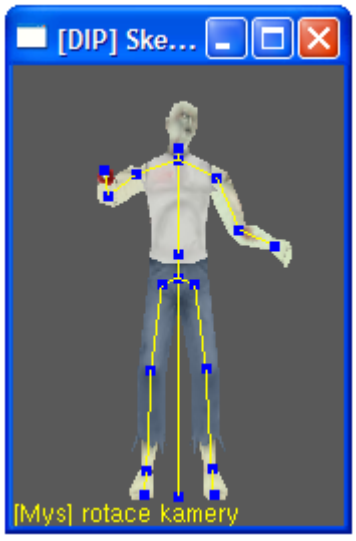
Bližší popis jednotlivých příkladů použití implementované animační knihovny. Všechny příklady lze nalézt na přiloženém CD a to včetně zdrojových kódů.

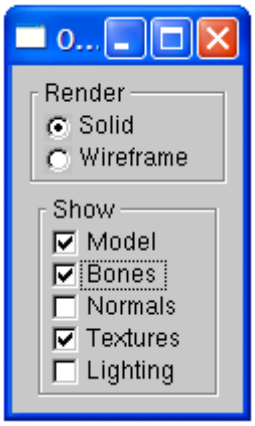
Příklad 01: Kloubová animace – základní aplikace



Ukázka nejjednodušší aplikace s použitím implementované knihovny. Konkrétně se jedná o kloubovou animaci. Nejprve se načte soubor s 3D modelem, kloubovou soustavou a s animací této soustavy. Následně se projdou všechny materiály modelu a do grafické paměti se načtou potřebné textury. O zobrazení modelu se postará metoda Render implementovaná knihovnou. K tomu, aby se postava hýbala, je nutné mezi jednotlivými snímky volat metodu Update. Tato metoda má tři parametry: krok animace, index animační sekvence a příznak opakování. Krok animace je reálné číslo, které se automaticky přičítá k aktuálnímu snímku v animaci. Index animační sekvence vybírá úsek animace, který se bude přehrávat. Protože v tomto příkladu nebyly žádné sekvence definovány, bude se přehrávat sekvence s indexem 0, která je vytvořena automaticky a obsahuje všechny snímky animace. Příznak opakování specifikuje, jestli se má vybraná sekvence opakovat nebo jestli se přehraje jen jedenkrát.

Příklad 01_ui: Kloubová animace – základní aplikace + UI

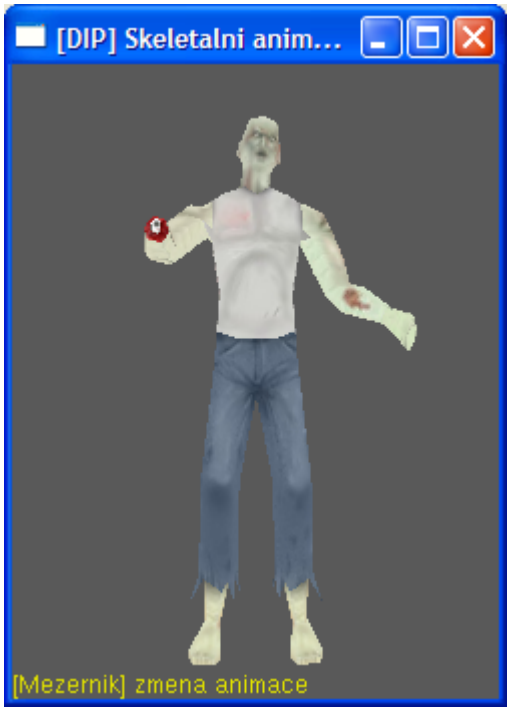




Tento příklad je v podstatě totožný s předcházející ukázkou, jen je navíc přidáno jednoduché uživatelské rozhraní a možnost otáčení s modelem. Uživatelské rozhraní je vytvořeno pomocí knihovny glui, což je nadstavba nad knihovnou glut respektive freeglut.

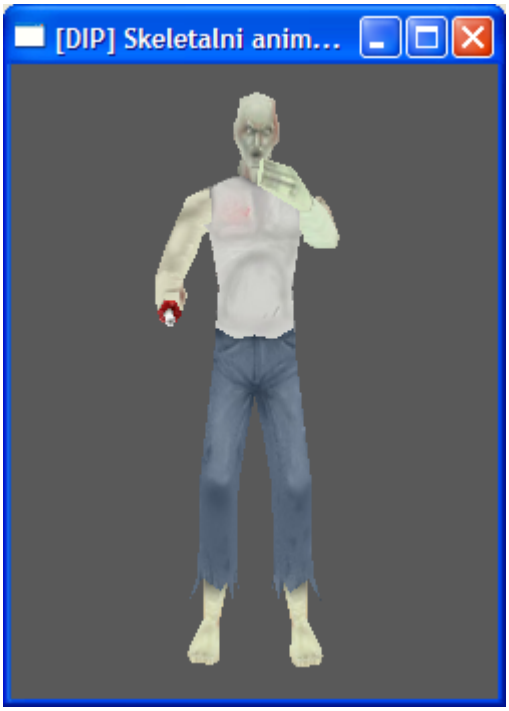
Příklad dále demonstruje, jak lze přistupovat k datovým strukturám z animační knihovny. Informace z těchto struktur jsou následně využity k zobrazení normál, kostry a dalších užitečných věcí, které nelze vykreslit přímo použitím metod knihovny. Samozřejmě nic nebrání tomu, aby si uživatel tyto metody přidal přímo do knihovny, stačí vytvořit novou třídu, která bude potomkem některé z knihovních tříd a nové metody jednoduše doimplementovat.

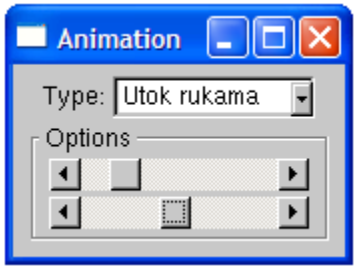
Příklad 02: Kloubová animace – přepínání animací



Tato ukázka použití animační knihovny vychází z příkladu 01. Navíc je zde přidána možnost přepínat mezerníkem animační sekvence. Změna animační sekvence je v podstatě základní úlohou všech animačních knihoven respektive programů přehrávajících 3D animaci. V animovaném 3D souboru je obvykle uložena jen jediná dlouhá animace, která v obsahuje všechny dílčí sekvence jako např. chůzi, běh, mávání rukou, otáčení hlavou atd. K danému souboru jsou pak specifikovány parametry jednotlivých sekvencí (počáteční snímek, koncový snímek, rychlost přehrávání apod.). Díky znalostí těchto parametrů lze následně přehrávat vždy jen určitou animaci. Třeba v počítačové hře se bude dokola přehrávat animace chůze zatímco bude uživatel držet šipku dopředu. V okamžiku, kdy uživatel stiskne šipku doleva, přehraje se jiná sekvence např. úkrok stranou.

Příklad 02_ui: Kloubová animace – přepínání animací + UI

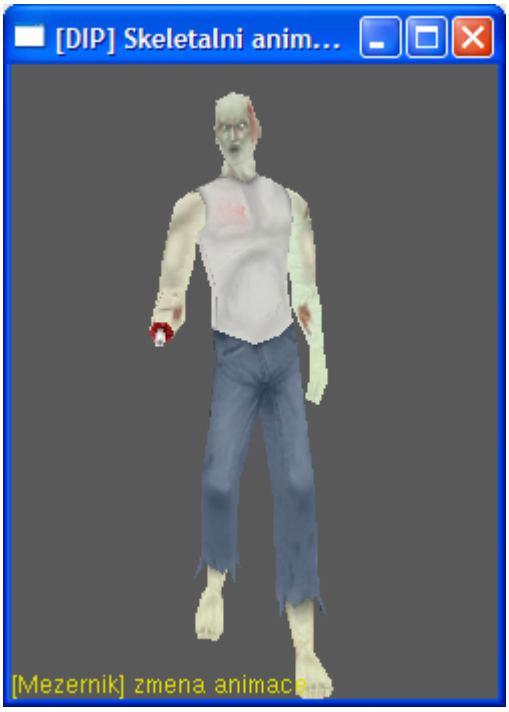




Další příklad na přepínání animačních sekvencí, tentokrát i s jednoduchým uživatelským rozhraním. V roletovém výběru nazvaném *Type*, lze měnit aktuálně přehrávanou animační sekvenci.

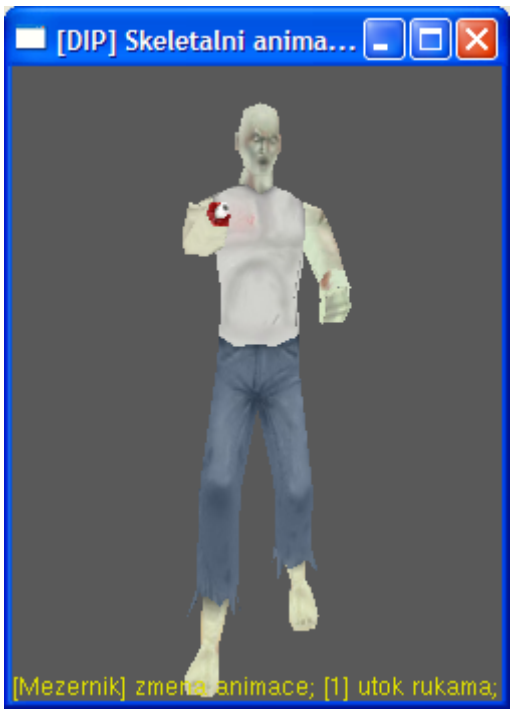
Následují dva scroll bary, kterými se dají upravovat parametry právě přehrávané animace. Prvním scroll barem lze měnit rychlost animace. Druhý scroll bar zastaví přehrávání a umožní uživateli posun po statických snímcích z aktuální sekvence. Pokud tato obsahuje například snímky 88 až 120, pak je druhý scroll bar omezen právě na tento rozsah. Opětovné spuštění animace se provede scroll barem pro změnu rychlosti aktuální sekvence.

Příklad 03: Kloubová animace – plynulé přepínání animací



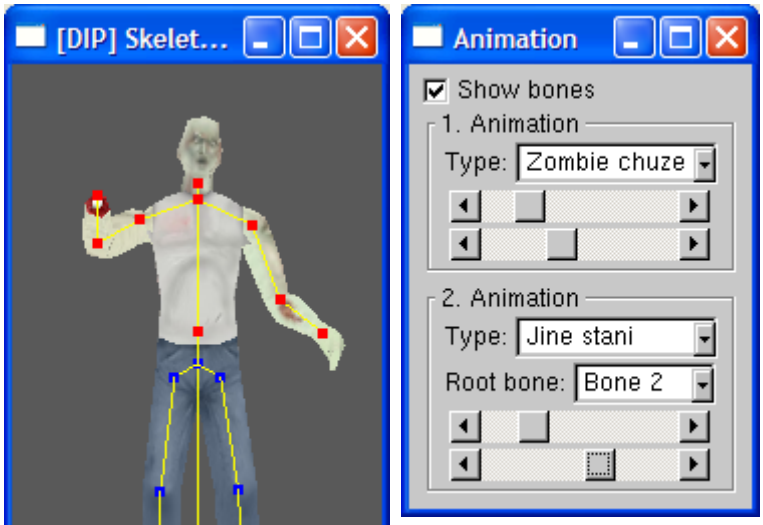
Program na plynulé přepínání animací navazuje na příklad 02 (přepínání animací). Stejně jako v příkladu 02 se i zde mění animační sekvence stiskem mezerníku. Tím ovšem veškerá podobnost končí. Zatímco v ukázce 02 docházelo k přepínání animací skokově, v tomto příkladu se animace mění plynule. Skoková změna animační sekvence je nejvíce patrná v sekvencích, kde se transformace postavy výrazně liší. Třeba, když se přehrává animace výskoku a uprostřed této animace dojde ke změně animační sekvence např. na chůzi. Rozdíl mezi polohou modelu uprostřed skoku a prvním snímkem animace chůze je výrazný, proto je přepnutí animace skokové a nevypadá příliš dobře. Plynulá změna animace funguje tak, že se mezi aktuální a cílovou sekvenci vloží krátká animace. Tato animace je vlastně plynulou interpolací transformačních matic mezi kloubovou soustavou v aktuálním snímku a kloubovou soustavou z prvního snímku cílové sekvence. Po této krátké animaci následuje již klasické přehrávání cílové sekvence.

Příklad 04: Kloubová animace – blending animací



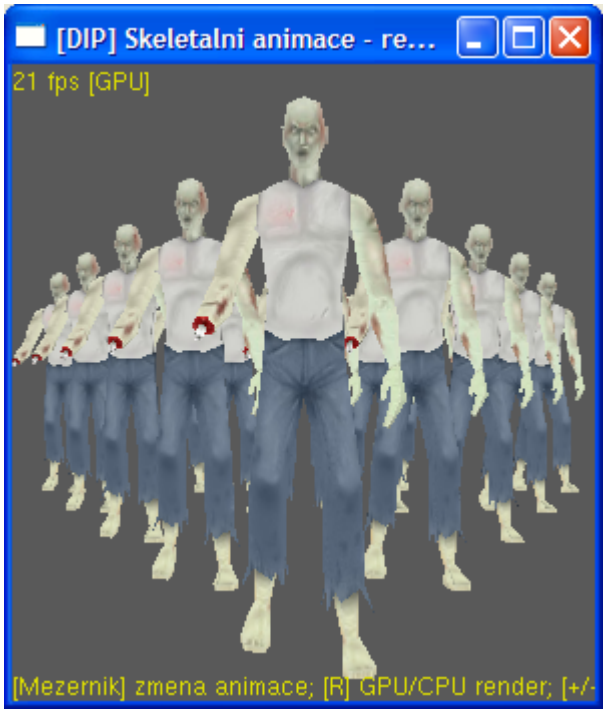
Blending animací je velmi užitečná technika, při které dochází ke kombinaci dvou a více animačních sekvencí do jednoho výstupu. Ovládání tohoto příkladu je následující: klávesou mezerník se přepínají animace celého modelu (např. stání na místě, běh, chůze, apod.) a klávesami '1' a '2' lze zaútočit rukama a hlavou (dojde k přehrání daných animačních sekvencí). Trik je v tom, že sekvence pro celý model, ty které se přepínají mezerníkem, se aplikují na všechny kosti v kloubové soustavě. Zatímco animace útoků modifikují jen vybrané kosti. Tím se dosáhne smíchání dvou animací. Máme-li k dispozici např. animace běhu, skákání a držení zbraně. Lze smícháním těchto animací jednoduše vytvořit běh se zbraní respektive skákání se zbraní.

Příklad 04_ui: Kloubová animace – blending animací + UI



Tato aplikace je vylepšením předcházejícího příkladu, navíc bylo přidáno jednoduché uživatelské rozhraní. Zaškrtnutím *Show bones* lze zapnout vykreslování kloubové soustavy jako součásti modelu. Následují dvě sekce – *1. Animation* a *2. Animation*, kde se dají upravovat parametry dvou animačních sekvencí, které se následně zkombinují dohromady. První sekce je stejná jako v příkladu 02_ui, umožňuje vybrat animační sekvenci a upravovat její rychlost, případně lze vybranou sekvenci zastavit a posouvat se po jednotlivých snímcích. Druhá sekce umí stejné funkce jako ta první, ale navíc dovoluje specifikovat kořenový kloub (*Root bone*). Vybraný kloub a všichni jeho potomci jsou následně ovlivněni vybranou animací. Tím se docílí smíchání dvou animací – první sekvence se aplikuje na celý model, zatímco druhá modifikuje jen vybrané kosti. Zvoleným sekvencím lze nezávisle upravovat parametry.

Příklad 05: Kloubová animace – render pomocí vertex shaderu



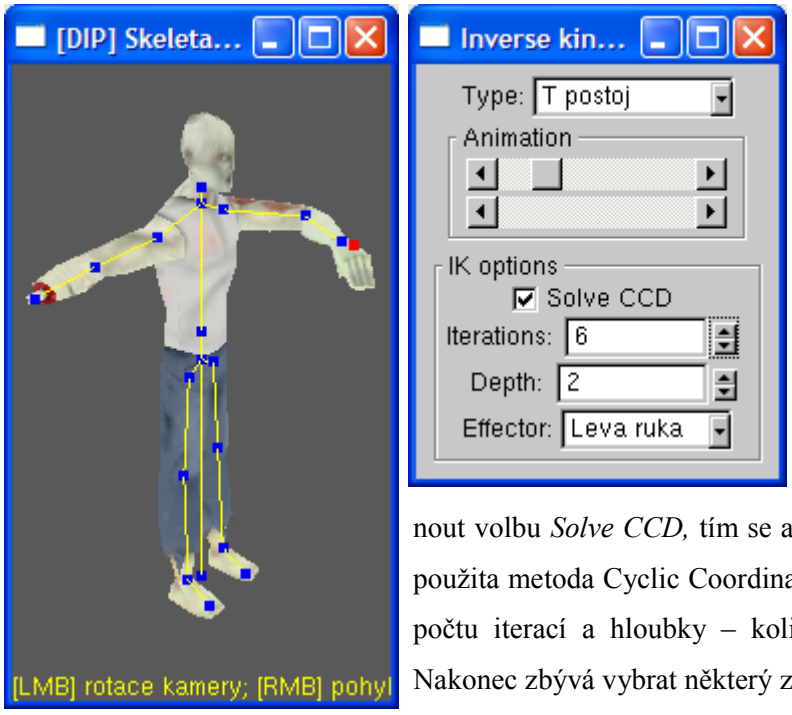
Příklad demonstruje využití vertex shaderu při výpočtu skeletální animace. Mezerníkem lze měnit animační sekvence, klávesa 'R' přepíná mezi výpočtem deformace kůže pomocí shaderu (GPU) nebo standardně s použitím CPU. Aby byl nárůst výkonu patrný, lze klávesami '+' a '-' přidávat respektive ubírat modely ze scény. Výkon je měřen ve snímcích za sekundu (fps) a zobrazen v levém horním rohu aplikace.

Zajímavé je, že na všech testovaných počítačích bylo renderování pomocí shaderu pomalejší než standardní CPU render. Může to být způsobeno tím, že využívám jen vertex shader, ale už ne pixel shader nebo bude chyba někde jinde.

Příklad 06_ui: Kloubová animace – ragdoll + UI

Hadrová panenka angl. ragdoll je animace kloubové soustavy, která je ovlivněna fyzikálními zákony. V podstatě se programuje tak, že se jednotlivé kosti obalí do tzv. rigidních těles (ideální těleso jehož deformace se zanedbává, těleso nemění svůj tvar v závislosti na působení vnějších sil). Velikost těchto těles se volí tak, aby co nejlépe odpovídala 3D modelu postavy. Následně se tyto tělesa propojí klouby, kterým se nastaví limity. S takto vytvořenou hierarchií lze provádět různé simulace. K simulaci fyzikálních zákonů se většinou používá externí fyz. engine, protože udělat kvalitní simulátor není jen tak. V tomto příkladu je použit Newton Game Dynamics. První část UI je totožná s příkladem 02_ui (výběr sekvence a nastavení parametrů). Tlačítkem *Ragdoll/Reset* se pouští/resetuje fyzikální simulace a úplně dole lze povolit zobrazení rigidních těles.

Příklad 07_ui: Kloubová animace – inverzní kinematika + UI



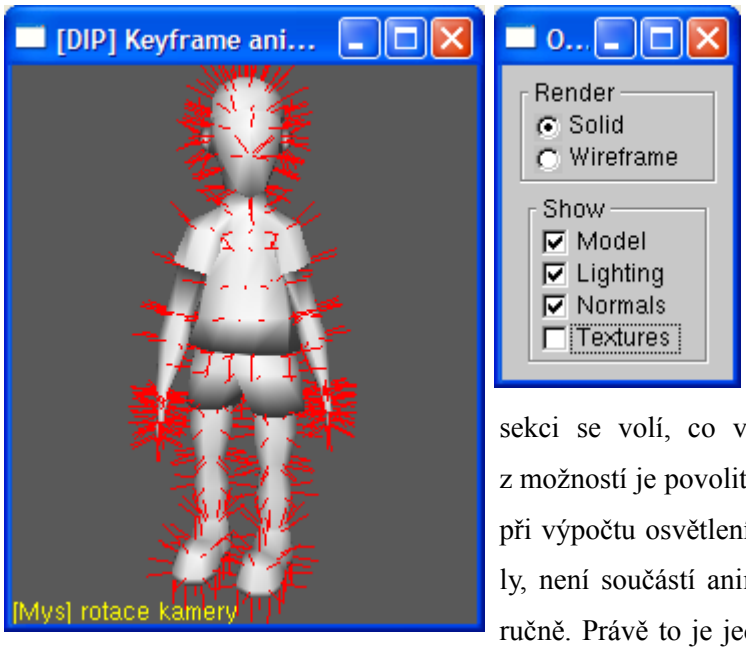
Inverzní kinematika je v dnešní době velmi populární, ať už v grafických editorech nebo třeba v počítačových hrách.

Popis ovládání: první polovina uživatelského rozhraní je opět tradiční (výběr animace, nastavení rychlosti atd.), druhá sekce se týká přímo inverzní kinematiky. Nejdříve je třeba zaškrtnout volbu *Solve CCD,* tím se aktivuje výpočet IK (konkrétně je použita metoda Cyclic Coordinate Descent). Následuje nastavení počtu iterací a hloubky – kolik kloubů se výpočtem ovlivní. Nakonec zbývá vybrat některý z koncových efektorů (kloubů).

Příklad 08: Snímková animace – základní aplikace

Nejjednodušší možná aplikace s použitím implementované knihovny, tentokrát pro snímkovou animaci. Rozhraní knihovny je navrženo tak, aby bylo jedno, jestli se pracuje se snímkovou nebo s kloubovou animací. V obou případech se používají v podstatě stejné názvy metod. Nejprve je tedy nutné načíst samotný soubor s 3D modelem a jeho snímkovou animací. Následně se postupně projdou všechny materiály modelu a do grafické paměti se načtou potřebné textury. Zobrazení modelu se stejně jako u kloubové animace provede pomocí metody Render. Aby se vykreslený model hýbal, je nezbytné mezi jednotlivými snímky volat metodu Update. Tato metoda má tři parametry: krok animace, index animační sekvence a příznak opakování. Všechny tři parametry mají stejný význam jako v případě kloubové soustavy (viz. příklad 01).

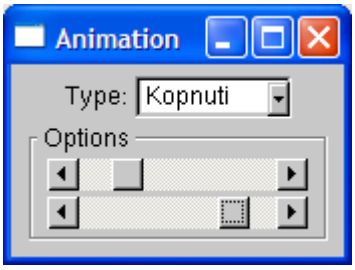
Příklad 08_ui: Snímková animace – základní aplikace + UI



Tento příklad je v zásadě stejný jako příklad předcházející, jen je navíc doplněn jednoduchým uživatelským rozhraním a možností otáčet s modelem pomocí myši.

V první sekci ovládacího panelu lze zvolit způsob zobrazení – pevný model nebo drátový model. V druhé sekci se volí, co všechno se má vykreslovat. Jednou z možností je povolit zobrazení normál, které se používají při výpočtu osvětlení. Metoda, která by zobrazila normály, není součástí animační knihovny, je nutné ji vytvořit ručně. Právě to je jednou z věcí, které demonstruje tento příklad – přístup k datovým strukturám z animační knihovny. Informace z těchto struktur jsou využívány ve všech dalších příkladech. Protože samotná knihovna poskytuje jen nejnutnější metody pro snímkovou a kloubovou animaci, je nutné si všechny ostatní funkce doimplementovat, ostatně to je předmětem právě těchto příkladů použití.

Příklad 09_ui: Snímková animace – přepínání animací + UI



Tento program vychází z příkladu 01, navíc je zde možnost přepínat animační sekvence a je přidáno i jednoduché uživatelské rozhraní. V roletovém výběru nazvaném *Type* lze vybírat z nabídky dostupných animačních sekvencí.

Následují dva scroll bary, kterými lze upravovat parametry právě přehrávané sekvence. Prvním scroll barem se dá měnit rychlost aktuální animace. Druhý scroll bar zastaví přehrávání a umožní uživateli posun po statických snímcích z aktuální sekvence. Pokud tato obsahuje například snímky 88 až 120, pak je druhý scroll bar omezen právě na tento rozsah. Opětovné spuštění animace se provede scroll barem pro změnu rychlosti aktuální sekvence.

Příklad 10: Kloubová a snímková animace

Poslední ukázka použití animační knihovny vychází z příkladů 01 a 08, lépe řečeno kombinuje oba tyto příklady dohromady.

Díky tomu, že třídy pro jednotlivé typy animace mají společnou bázovou třídu, lze oba modely uložit do jednoho pole. Na toto pole se dají následně v cyklu volat metody bázové třídy a tím jednotlivé modely animovat.

Snažil jsem se, aby byl příklad co nejjednodušší, proto zde není žádné uživatelské rozhraní. Kromě výchozí animační sekvence, která obsahuje všechny snímky načtené animace, nejsou definovány žádné další sekvence.

Kombinace skeletální a snímkové animace se používá relativně často. K animování těla postavy je ideální využít kloubovou soustavu, zatímco mimika obličeje se lépe animuje snímkovou animací (animátoři mají nad snímkovou animací větší kontrolu, navíc je dost obtížné navázat svaly v obličeji na nějakou kostru respektive kloubovou soustavu).

Př. 2. Datové struktury snímkové animace

```
struct tVertex                    /*** vrchol 3D modelu ***/
{
  tVector Position;               // poloha vrcholu (x,y,z)
  tVector Normal;                 // normála vrcholu (x,y,z)
};


struct tTexCoord                  /** texturovací koordináty **/
{
  float u,v;
};


struct tFrame                     /*** snímek animace ***/
{
  vector<tVertex> Vertices;       // vrcholy modelu
};


struct tMesh                      /*** mesh (objekt) ***/
{
  vector<unsigned> Vertices;      // indexy vrcholů
  vector<unsigned> TexCoords;     // indexy tex. souřadnic
  char MaterialIndex;             // index materiálu (-1 = není)
};


struct tMaterial                  /*** materiál ***/
{
  float Ambient[4];
  float Diffuse[4];
  float Specular[4];
  float Emissive[4];
  float Shininess;
  float Transparency;

  char TextureFileName[..];       // soubor s texturou
  char AlphaMapFileName[..];      // soubor s alfa mapou

  unsigned int Texture;           // id textury v paměti
  unsigned int AlphaMap;          // id alfa mapy v paměti
};


struct tAnimation                 /*** animační sekvence ***/
{
  float Speed;                    // rychlost animace
  float StartFrame;               // počáteční snímek
  float StopFrame;                // koncový snímek
  float Frame;                    // aktuální snímek
};
```

Př. 3. Datové struktury kloubové animace

```
struct tVertex                    /*** vrchol 3D modelu ***/
{
  tVector Position;               // poloha vrcholu (x,y,z)
  tVector Normal;                 // normála vrcholu (x,y,z)
  float TexCoords[2];             // texturovací souřadnice (u,v)
  float BoneWeight[4];            // váha přiřazených kostí
  char BoneIndex[4];              // indexy přiřazených kostí
};


struct tMaterial                  /*** materiál ***/
{
  float Ambient[4];
  float Diffuse[4];
  float Specular[4];
  float Emissive[4];
  float Shininess;
  float Transparency;

  char TextureFileName[..];       // soubor s texturou
  char AlphaMapFileName[..];      // soubor s alfa mapou

  unsigned int Texture;           // id textury v paměti
  unsigned int AlphaMap;          // id alfa mapy v paměti
};


struct tPositionKey               /*** poziční klíč ***/
{
  float Frame;                    // číslo snímku
  tVector Position;               // poloha kloubu (x,y,z)
};

                                  /*** rotační klíč ***/
struct tRotationKey
{                                 // číslo snímku
  float Frame;                    // rotace kloubu (x,y,z,w)
  tQuaternion Rotation;
};


struct tAnimation                 /*** animační sekvence ***/
{
  float Speed;                    // rychlost animace
  float StartFrame;               // počáteční snímek
  float StopFrame;                // koncový snímek
  float Frame;                    // aktuální snímek
};
```

```
struct tMesh                        /*** mesh (objekt) ***/
{
  vector<unsigned> Vertices;        // indexy vrcholů
  char MaterialIndex;               // index materiálu (-1 = není)
};


struct tBone                        /*** kloub (kost) ***/
{
  char Name[..];                    // jméno kosti

  tBone* pParentBone;               // ukazatel na rodiče
  vector<tBone*> pChilds;           // ukazatele na potomky

  tMatrix StartLocalMatrix;         // počáteční transformace
  tMatrix CurrentLocalMatrix;       // aktuální transformace
  tMatrix CurrentGlobalMatrix;      // aktuální globální trans.

  vector<tPositionKey> PosKeys;     // poziční klíče (snímky)
  vector<tRotationKey> RotKeys;     // rotační klíče (snímky)

  void GetPosition(..);             // vrátí interpolovanou polohu
  void GetRotation(..);             // vrátí interpolovanou rotaci
};
```

Př. 4. Kloubová animace – vertex shader

Ukázka velmi jednoduchého vertex shaderu pro výpočet kloubové animace. Jako implementační jazyk byl použit Cg od firmy nVidia.

```
void main(float3 Position : POSITION,       // vstupní poloha
          float2 TexCoord : TEXCOORD0,      // vstupní koordináty
          float4 BWeight : TEXCOORD1,       // váhy kostí
          float4 BIndex : TEXCOORD2,        // indexy kostí
          out float4 oPosition : POSITION,  // výstupní poloha
          out float2 oTexCoord : TEXCOORD0, // výstupní koordináty
          out float4 oColor : COLOR0,       // výstupní barva
          uniform float4x4 BMatrix[20],     // max. 20 kostí
          uniform float4x4 ModelViewProj)   // pohledová matice
{
  float4 NewPosition = 0;

  // až 4 kosti ovlivňují každý vrchol
  for(int i = 0; i < 4; i++)
  {
    if(BIndex[i] >= 0)   // -1 pokud není nastavena kost
    {
      // rovnice deformace kůže
     float4 BPosition = mul(BMatrix[BIndex[i]],float4(Position,1));
      NewPosition += BWeight[i] * BPosition;
    }
  }

  // nastavení výstupních hodnot
  oPosition = mul(ModelViewProj,NewPosition);
  oTexCoord = TexCoord;
  oColor = float4(1,1,1,0);  // bez osvětlení, pouze textura
}
```