Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE

Vít Houska

Animace na GPU

Kabinet software a výuky informatiky
Vedoucí diplomové práce: **RNDr. Josef Pelikán**
Studijní program: **Informatika**

Poděkováni

Na tomto místě bych rád poděkoval osobám, bez kterých by vznik této diplomové práce byl mnohem obtížnější.

Na prvním místě bych rád poděkoval RNDr. Josefu Pelikánovi za věcné konzultace a připomínky k textu mé diplomové práce. Dále bych chtěl poděkovat Mgr. Ladislavu Kavanovi za přínosné a inspirativní diskuze na téma skinningu. Děkuji Andreji Svobodovi za poskytnutí modelu lidské postavy.

Nakonec bych rád poděkoval rodině, zejména mému otci, za podporu při studiu a při tvorbě diplomové práce.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 9. srpna 2005 Vít Houska

OBSAH

Název práce: Animace na GPU
Autor: Vít Houska
Katedra: Kabinet software a výuky informatiky
Vedoucí diplomové práce: RNDr. Josef Pelikán
e-mail vedoucího: Josef.Pelikan@mff.cuni.cz
Abstrakt: Práce popisuje výhody a nevýhody skeletální animace. Zabývá se problematikou výpočtu vrcholů povrchové sítě trojúhelníků podle poloh jednotlivých kostí (tzv. skinning) v reálném čase. Popisuje možnosti, jakými lze implementovat metodu vertex blending na dnešních grafických kartách za použití DirectX 9.0 se zaměřením na využití vertex shader jednotek verze 2.0. Jsou představeny nové přístupy, které se snaží řešit problematiku skinningu. Speciální pozornost je věnována metodám užívajících vzorové pózy a metodám, které používají interpolaci jednotlivých transformací. V práci je navržena a implementována nová metoda skinningu, která kombinuje přístupy obou metod.
Klíčová slova: skeletální animace, skinning, vertex blending, vertex shader

Title: Animation using GPU
Author: Vít Houska
Department: Department of software and computer science education
Supervisor: RNDr. Josef Pelikán
Supervisor's e-mail address: Josef.Pelikan@mff.cuni.cz
Abstract: This work describes advantages and disadvantages of skeletal animation. It examines the real-time skinning problem. The possibilities how to implement vertex blending on modern graphic cards using DirectX 9.0 (especially with vertex shader 2.0) are introduced. New approaches to problem of skinning are presented. Work focuses on methods using example poses and methods using interpolation of transformations more deeply. A new method that combines ideas of both techniques is introduced and implemented.
Keywords: skeletal animation, skinning, vertex blending, vertex shader

Kapitola 1. Úvod

Realtime počítačová grafika je jedním z dynamicky se rozvíjejících oborů této doby. Jedním z důvodů, proč tomu tak je, je tlak počítačového herního průmyslu. Neutichající poptávka a velká konkurence nutí vývojové týmy vytvářet stále dokonalejší virtuální prostředí. Velmi důležitou součástí virtuálních světů je využití počítačové animace, bez které si počítačové hry v dnešní době nelze představit.

Jsou to počítačové hry, kvůli kterým výrobci GPU každý rok produkují stále výkonnější procesory. Jejich složitost a výkon je již větší než u klasických CPU. Přitom ceny grafických karet se drží na přijatelné úrovni. V dnešní době jsou hitem programovatelná GPU, jejichž nasazení představuje menší revoluci v programování počítačové grafiky. Programovatelné jednotky grafického čipu umožňují vývojovým týmům implementovat netradiční a inovativní metody realtime počítačové grafiky bez zatížení CPU.

1.1 Cíl práce

Cílem této diplomové práce je popsat používané metody počítačové animace v realtime aplikacích (tj. aplikacích pracujících v reálném čase) se zaměřením na skeletální animaci a tzv. skinning.

Konkrétním úkolem je implementovat skeletální animaci, analyzovat několik způsobů implementace klasické metody vertex blending s využitím hardwarové podpory grafických karet (zejména za použití vertex shader jednotek verze 2.0), uvést nové přístupy k řešení skinningu, porovnat je s metodou vertex blending a pokusit se některou z nových přístupů implementovat za pomocí hardwarové podpory grafických karet.

Výsledkem práce má být také program který bude implementovat skeletální animaci a různé techniky skinningu.

Kapitola 2. Počítačová animace

Animaci lze souhrnně definovat jako sled obrázků či objektů, které se mění v čase. Počátky animace spadají do 19. století. V této době se staly populárními více či méně složitá optická zařízení, jako například projekční praxinoskop patentovaný Francouzem Charlesem Reynaudem [1]. Obor animace se naplno rozvinul s příchodem filmu. Průkopníkem na tomto poli byla studia Walta Disneye, která velkým dílem přispěla k technologickému rozvoji 2D filmové animace.

V dnešní době si většina lidí pod pojmem animace představí spíše speciální filmové efekty vytvořené počítačem a počítačové hry. V obou těchto případech je 2D animace vytlačována animací objektů v trojrozměrném prostoru (3D animací). Protože v každém filmu či počítačové hře se vyskytují (ať už více či méně reálné) postavy, je jedním z nejvíce studovaných typů animace tzv. animace postav (*character animation*). V této kapitole budou popsány techniky, které se převážně používají při animaci postav v 3D počítačových hrách.

2.1 Model

Nejdříve je nutné popsat jakou reprezentaci má objekt, jehož animace bude studována. V dnešní době se v realtime aplikacích pro svojí jednoduchost a masivní podporu v hardwaru používají jen trojúhelníkové reprezentace objektů. Model je tedy

- množina vrcholů, které mají svoji pozici v 3D prostoru
- množina normálových vektorů (normál), které jsou definovány ve vrcholech
- množina trojúhelníků, které jsou definovány pomocí tří vrcholů
- barva a souřadnice textur definované ve vrcholech
- textury

Obecně animace modelu spočívá v tom že se mění pozice vrcholů a hodnoty normál v čase. Barva a textury pro animaci nejsou nikterak podstatné. Pod pojmem *póza* modelu budeme rozumět konkrétní hodnoty všech pozic vrcholů a normál modelu. Všechny pózy stejného modelu mají stejný počet vrcholů a je známa korespondence mezi jednotlivými vrcholy.

2.2 Klíčování

Klíčování je pojem, který vznikl v tradiční 2D kreslené animaci. Zkušený animátor vytvořil *klíčové snímky* – obrázky postavy v některých extrémních pozicích v pevně

daném čase animace. Méně zkušení animátoři pak vytvářeli zbylé snímky pro vytvoření plynulé animace mezi klíčovými snímky.

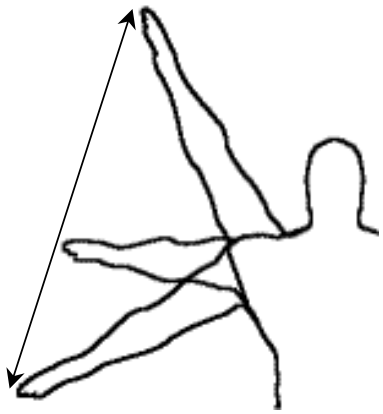
V počítačové animaci jsou obecně v klíčových snímcích uložena jakákoliv data potřebná pro správné animování objektu. O jaké informace se jedná, závisí na typu animace. Ve snímcích, které se nacházejí časově mezi klíčovými snímky, jsou tato data interpolována. Animační sekvencí budeme rozumět sled snímků (klíčových i interpolovaných) animace modelu.

2.3 Per vertex animace

Per vertex animací budeme nazývat to, čemu se v anglicky psané literatuře říká *vertex animation* nebo také *morphing*. Při této metodě jsou v každém klíčovém snímku uloženy vždy všechny pozice vrcholů modelu a hodnoty normál příslušné pózy modelu. Mezi klíčovými snímky jsou pak pozice vrcholu a hodnoty normál interpolovány.

Výhodou tohoto přístupu je, že animátor má úplnou svobodu při vytváření pohybu. Tímto způsobem může být zaznamenán jakkoliv složitá animace postavy včetně všech detailů, jako je například pohyb svalů. Další výhodou je jednoduchost implementace této metody a nenáročnost na výpočetní výkon.

obr. 1: Interpolace bodů při per vertex animaci. Při velkém odstupu klíčových snímků dochází k viditelnému zkracování končetin.



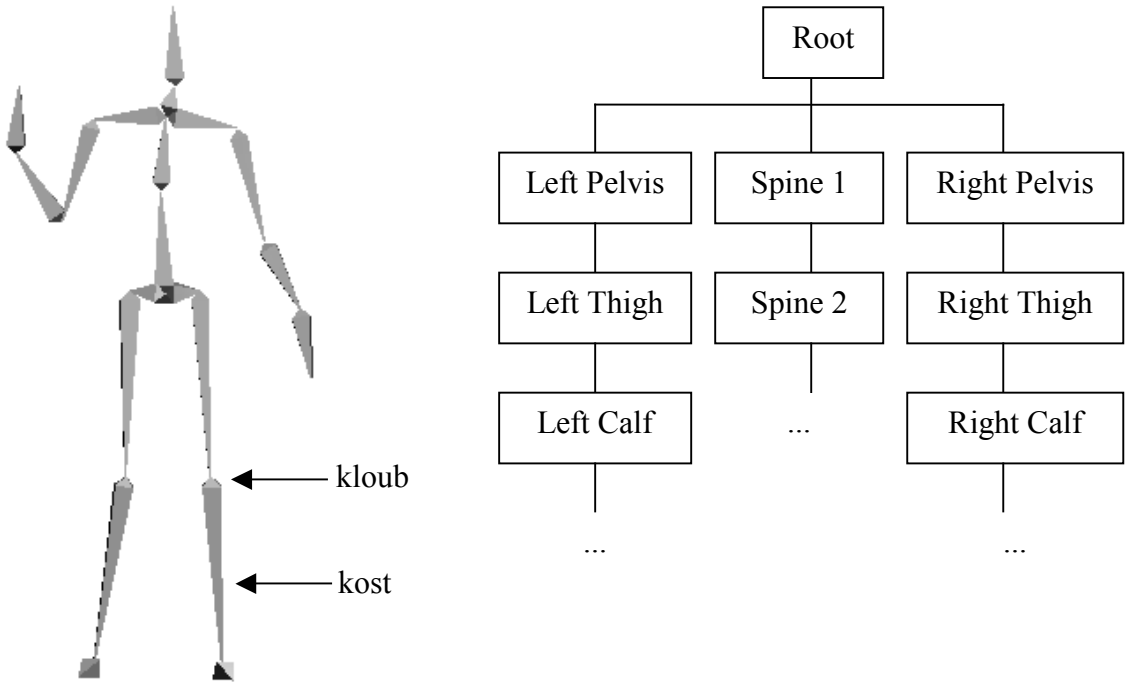
Avšak daní za univerzálnost tohoto přístupu je velká paměťová náročnost. Další nedostatek této metody je, že může docházet k nežádoucím deformacím při interpolování mezi klíčovými snímky. Tento neduh lze vidět na obr. 1. Per vertex animace neví nic o tom, že by vrcholy paže měly opisovat kruh. Aby geometrické deformace byly zanedbatelné, je potřeba mít v animační sekvenci dostatečnou hustotu klíčových snímků. Tím ale vzrůstá již zmiňovaná paměťová náročnost.

Ke geometrickým deformacím dochází také při interpolaci různých animačních sekvencí [2], [3]. Nežádoucí změny tvaru objektu v tomto případě nelze jednoduše odstranit.

2.4 Skeletální animace

Nedostatky per vertex animace jsou způsobené tím, že neexistuje jakýsi vyšší popis modelu. Jedinou informaci, kterou per vertex animace má k dispozici, jsou jen pozice vrcholů. Zavedením hierarchické vrstvy modelu předejdeme nevýhodám per vertex animace.

obr. 2: Vizualizace a hierarchie skeletonu.



Pro zjevnou paralelu s anatomickou kostrou, se hierarchické vrstvě modelu říká *skeleton*. Z matematického hlediska je skeleton stromová struktura. Uzly skeletonu se nazývají *kosti*. Všechny hrany mezi kostí a jejími přímými potomky tvoří *kloub*. Každá kost mimo kořenu představuje lokální transformaci souřadnic vzhledem ke svému přímému předku v hierarchii. Kořen obsahuje lokální transformaci souřadnic vzhledem k souřadné soustavě modelu. Příklad skeletonu lze spatřit na obr. 2. Skeletonu s konkrétními transformacemi kostí budeme říkat *konfigurace skeletonu*.

Skeletální animace spočívá v tom, že se v čase mění konfigurace skeletonu. V klíčových snímcích animační sekvence jsou tedy uloženy jednotlivé lokální transformace kostí. Vrcholy modelu jsou na kosti určitým způsobem navázány a tím kopírují pohyb skeletonu. Základní metody navázání vrcholů modelu na skeleton

popisují odstavce 2.4.1, 2.4.2 a 2.4.3. Vzhledem k tomu, že model může mít až několik desítek tisíc vrcholů, avšak počet kostí bývá v řádu desítek, maximálně stovek, je skeletální animace oproti per vertex animaci daleko méně náročná na paměť. Navíc stejnou hierarchickou strukturu může sdílet více modelů – například máme-li více modelů lidské postavy. Tyto modely pak mohou používat stejné animační sekvence.

Obvyklou podmínkou je, že lokální transformace spjatá s kostí k je omezena pouze na rotaci R_k okolo středu s_k. Střed rotace s_k si lze představit jako pozici kloubu, na který je zavěšena kost k v souřadné soustavě přímého předka. Další přirozenou podmínkou je, že hodnoty s_k zůstávají konstantní. Pokud by tato podmínka neplatila, mohla by se měnit vzdálenost mezi sousedními klouby. To by odpovídalo nerealistickému natahování a smršťování jednotlivých kostí. V průběhu celé animační sekvence se tedy omezíme jen na změnu rotace R_k. Tím, že se mezi klíčovými snímky interpolují rotace, nedochází k nežádoucím deformacím modelu jako u per vertex animace (obr. 1).

Jestliže budeme rotaci R_k chápat jako matici, lze lokální transformaci kosti k vyjádřit pomocí homogenní matice M_k, kde $\mathbf{0}^T = (0,0,0)$:

$$M_k = \begin{pmatrix} R_k & s_k \\ \mathbf{0}^T & 1 \end{pmatrix}$$

Přímého předka kosti k označíme $p(k)$. Transformační matici kosti k vzhledem k souřadné soustavě modelu vypočteme postupným vynásobením matic na cestě mezi kostí k a kořenem skeletonu[1].

$$A_k = M_{root} \ldots . M_{p(p(k))} M_{p(k)} M_k$$

Pozici vrcholu v v souřadné soustavě kosti k transformujeme do souřadné soustavy modelu vynásobením maticí A_k.

$$v' = A_k v$$

Normály ve vrcholu bychom obecně (bez omezení na rotace) museli transformovat transponovanou inverzní maticí k matici A_k. Pokud bychom normálu transformovali

[1] Nebude-li uvedeno jinak, jsou všechny matice, pozice vrcholů a normálové vektory vyjádřeny pomocí homogenních souřadnic.

maticí A_k, nezachovávala by se kolmost normály k povrchu modelu [5]. Avšak při omezení na rotace je tato vlastnost zachována.

Zavedení hierarchické vrstvy modelu přináší i další výhody než jen z pohledu čisté animace. Pomocí skeletonu lze zjistit, kde se nachází například ruka postavy a jaká je její pozice v prostoru. Jednoduchým způsobem lze „připojit“ postavě do ruky nějaký předmět. Hierarchická vrstva tedy usnadňuje interakci modelu se scénou.

Na skeleton se lze také dívat jako na určitý fyzikální popis modelu. U kostí lze definovat stupně volnosti a omezení pro rotace. Animace modelu pak může být řízena procedurálně – pomocí dopředné či inverzní kinematiky.

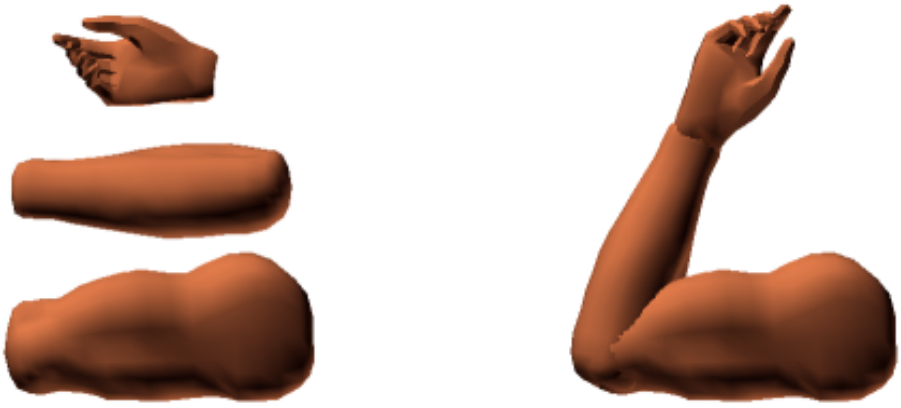
Další výhodou skeletální animace je jednodušší pořízení animačních dat. V dnešní době se používají tzv. *motion capture* systémy, které snímají pohyby reálné postavy a mapují tento pohyb na skeleton. Použití motion capture urychluje vytvoření realistických animačních sekvencí.

2.4.1 Komponentovaná skeletální animace

Jakým způsobem jsou svázány vrcholy modelu se skeletonem? Myšlenka první metody je mít celý model postavy složený z odděleně vymodelovaných částí těla - komponent. Nyní stačí ke kostem tyto části přiřadit.

Způsob renderování takového modelu je jednoduchý. Stačí projít celou hierarchií, pro každou kost k transformovat maticí A_k přiřazenou část modelu a poté ji renderovat. Části modelu musejí být uloženy v souřadných soustavách příslušných kostí.

obr. 3: Vymodelované části pravé paže a její složení typické pro komponentovanou skeletální animaci.



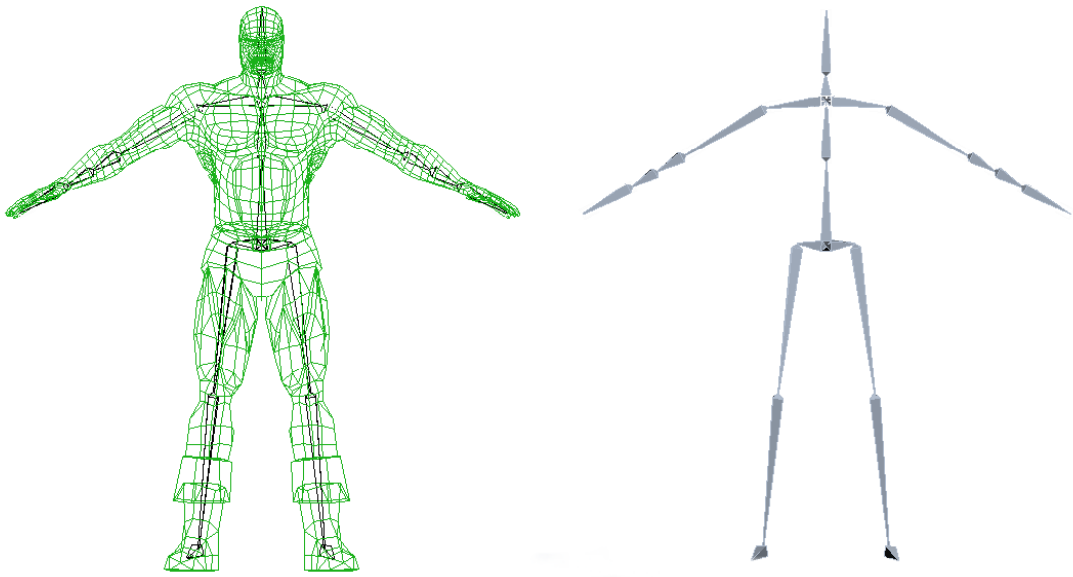
Slabinou této metody je vizuální kvalita. Lidské tělo je kompaktní, avšak tato metoda jej skládá z několika oddělených částí. Proto může model postavy působit nepřirozeně. Kusy těla se musí překrývat, aby složení postavy z několika částí nebylo tak nápadné. Části končetin se většinou modelují zakulaceně. Jinak by při velkých ohybech vznikaly nepřirozené hrany.

2.4.2 Základní skinning

Použijeme-li souvislý model pro skeletální animaci, odstraníme nedostatek výše popsané metody. Souvislému model se v anglicky psané literatuře říká *skin*. Skin je na skeleton určitým způsobem navázán, pohybem skeletonu je poté skin deformován. Všechny metody, které určují vztah mezi skeletonem a skinem, budou souhrnně nazývány termínem *skinning*. Idea základního skinningu spočívá v tom, že každý vrchol modelu je spjat (je transformován) s právě jednou kostí.

Při komponentované skeletální animaci jsou většinou modely částí těla uloženy v souřadných soustavách příslušných kostí. Důvod je ten, že při této metodě se nejdříve vytvoří skeleton a až poté se přiřadí připravené části modelu. Postup vytvoření modelu pro skeletální animaci je odlišný. Nejdříve se vymodeluje celá postava v souřadné soustavě modelu, až poté se navážou vrcholy na skeleton. Konfigurace skeletonu a póza modelu, v jaké je postava vytvořena, se nazývá *referenční póza*. Obvyklou referenční pózou je stojící rozpažená postava.

obr. 4: Model a skeleton v referenční póze.



Mějme konfiguraci skeletonu (označíme ji B), jejíž transformační matice kosti k vzhledem k souřadné soustavě modelu je B_k. Pozici vrcholu v, který je spjat s kostí k, při konfiguraci skeletonu B vypočítáme takto:

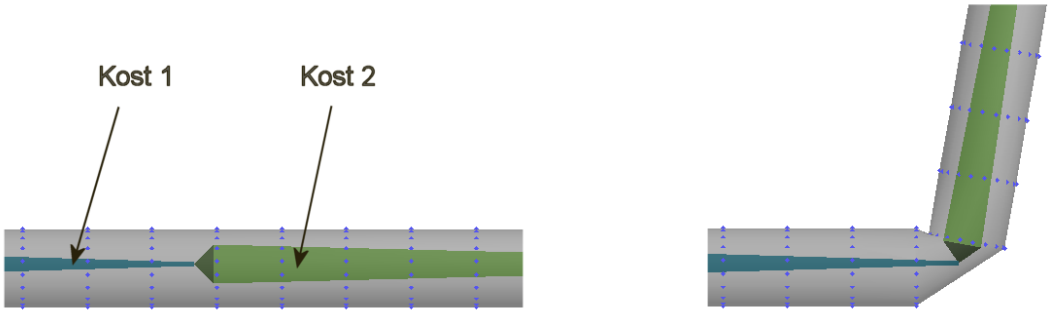
$$v' = B_k A_k^{-1} v$$

Matice A_k je transformační matice kosti k vzhledem k souřadné soustavě modelu v referenční póze. Vynásobením maticí A_k^{-1} transformujeme vrchol v do souřadné soustavy kosti k. Poté aplikujeme rotace a posunutí kosti k a jejích předchůdců pomocí matice B_k. Jestliže je konfigurace skeletonu B rovna konfiguraci v referenční póze, pak matice B_k je rovna matici A_k a vyjde nám předpokládaný výsledek $v' = v$.

Jak vypadá deformace modelu za použití základního skinningu, je patrné na obr. 5. Problém nastává na rozhraní kostí – u kloubu. Část modelu, která je ovlivňována kostí 1, přechází skokově v část modelu ovlivňovanou kostí 2. Deformovány jsou jen trojúhelníky na tomto rozhraní, neboť jsou tvořeny z vrcholů, které jsou ovlivňovány různými kostmi. Tato skutečnost má za následek vizuální nespojitost deformace modelu.

Trojúhelníky, které obsahují vrcholy spjaté s různými kostmi představují i další problém, chceme-li pro renderování využít hardwarově urychlované transformování vrcholů. Klasické zpracování vrcholů (bez vertex shader jednotek) neumožňuje jednoduše transformovat vrcholy ve stejném trojúhelníku různými maticemi. Tento požadavek lze simulovat pomocí níže popsané metody vertex blending, která bývá hardwarově podporovaná. Avšak v tomto případě je lepší přímo použít vertex blending, který dává lepší vizuální výsledky.

obr. 5: Příklad základního skinningu. Na levém obrázku je znázorněna referenční póza. Obrázek napravo znázorňuje deformaci modelu při ohybu kosti.



2.4.3 Vertex blending

Vertex blending je zobecnění základního skinningu, které se snaží odstranit deformační nespojitosti na modelu. Na rozdíl od výše popsané metody není vrchol spjat s právě jednou kostí, ale s celou množinou kostí. Výsledná pozice vrcholu v je počítána jako konvexní kombinace pozic získaných transformací vrcholu v jednotlivými kostmi. Podobný vzorec platí i pro normálu n ve vrcholu (stále platí omezení jen na rotace). Jestliže chceme zachovat jednotkovou délku normály, musíme ji však normalizovat. Matematicky zapsáno:

$$
\begin{aligned}
v' &= \sum_{i=1}^{n} w_i (B_i A_i^{-1} v) \\
n' &= \frac{\sum_{i=1}^{n} w_i (B_i A_i^{-1} n)}{\left\| \sum_{i=1}^{n} w_i (B_i A_i^{-1} n) \right\|} \\
\sum_{i=1}^{n} w_i &= 1, \;\; w_i \geq 0
\end{aligned}
\tag{2.1}
$$

Počet kostí ovlivňujících vrchol bývá zpravidla malý – v současných počítačových hrách se většinou nevyskytují modely, jejichž vrcholy jsou ovlivňovány více než čtyřmi kostmi. Každý vrchol má vlastní váhy w_i, které jsou neměnné. Číselně vyjadřují ovlivnění vrcholu jednotlivými kostmi. Jestliže pro každý vrchol existuje *k*, že $w_k = 1$, dostaneme rovnici základního skinningu.

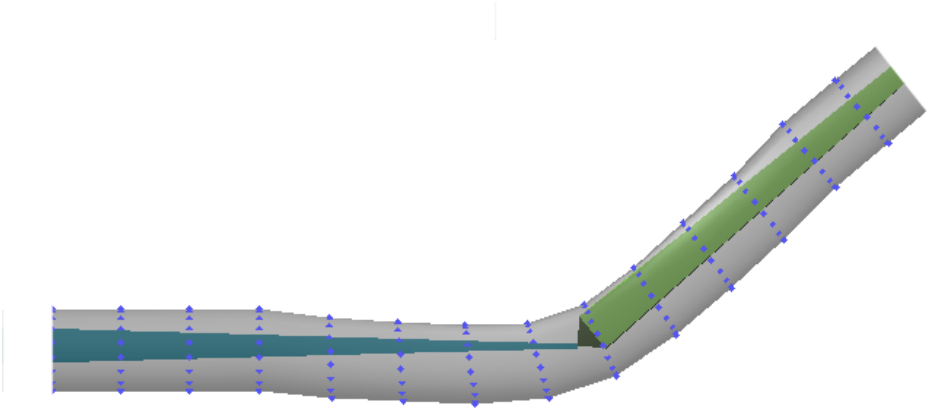
Deformaci patrnou na obr. 5 při ohybu kostí u základního skinningu vyhladíme správným nastavením vah u vrcholů. Vrcholy, které jsou od kloubu dostatečně vzdálené a nalevo, budou ovlivňovány jen kostí 1 – přiřadíme jim váhu rovnou jedné pro tuto kost. Postupujeme-li doprava ke kosti 2, budeme zvyšovat váhu ke kosti 2. Tím se bude snižovat váha ke kosti 1 (z podmínky konvexnosti). Vrcholy, které jsou v dostatečné vzdálenosti napravo od kloubu, budou ovlivňovány jen kostí 2. Výslednou deformaci při ohybu lze nahlédnout na obr. 6.

Vertex blending je velmi populární metoda, v dnešní době nejspíš neexistuje komerční 3D hra, která by ji nepoužívala. Za cenu mírně složitějšího výpočtu (ve srovnání se základním skinningem) dává kvalitní vizuální výsledky. I ona má však své nedostatky a omezení, které budou diskutovány v odstavci 4.1.

Chceme-li použít metodu vertex blending, musíme mít model, který bude mít přiřazeny váhy pro každý vrchol. Váhy pro jednotlivé vrcholy se nastavují v 3D modelovacích programech, které většinou obsahují nástroje pro zjednodušení tohoto procesu. Přesto může být nastavení vah velmi zdlouhavé a namáhavé pro docílení

optimálního výsledku [4] – zvláště pro méně zkušené 3D grafiky, kteří neznají omezení metody vertex blending.

obr. 6: Deformace modelu při použití metody vertex blending.



Kapitola 3. Implementace skeletální animace a metody vertex blending

V této kapitole bude popsáno, jakými způsoby lze implementovat vertex blending, a to hlavně za použití dnešních GPU. Budou zmíněny přednosti a nedostatky různých implementací, v závěru budou metody porovnány vzhledem k rychlost na několika konzumních grafických kartách. Všechny níže uvedené metody jsou implementovány v přiloženém programu.

Pro konkrétní implementace metod vertex blendingu jsem musel zvolit, jaké API bude použito – zda OpenGL nebo Microsoft DirectX. Po zralé úvaze jsem zvolil DirectX. Hlavním důvodem pro toto rozhodnutí byly mé větší zkušenosti s tímto API. Již jsem věděl, že DirectX umožňuje hardwarově urychlovaný vertex blending. Navíc DirectX obsahuje nativní podporu formátu .x file, který umožňuje ukládání skeletální animace a informací potřebných pro vertex blending. Jednoduchost nahrávání těchto souborů v DirectX a existence schopného exportéru mě přesvědčila tento formát používat.

3.1 Data modelu

V minulé kapitole byl definován model. Kvůli porovnání různých metod implementace vertex blendingu je třeba vysvětlit, jakým způsobem jsou data o modelu reprezentována a kde mohou být uložena.

Základním datovým prvkem modelu je struktura vrcholu. Struktura vrcholu kromě pozice obsahuje normálu. Dále musí obsahovat informace nutné pro vertex blending. O jaká data se jedná, závisí na konkrétní metodě implementace vertex blendingu. Struktura vrcholu může také obsahovat barvu a souřadnice textury, které nejsou pro vertex blending podstatné. Všechny vrcholy jsou uloženy v datovém poli (v*ertex buffer)*.

Trojúhelníky modelu jsou tvořeny z vrcholů pomocí indexace do vertex bufferu. Indexy jsou uloženy v tzv. *index bufferu*. Posloupnost sousedících indexů může tvořit samostatné trojúhelníky, nebo pásy trojúhelníků *(triangle strips*) či vějíře *(triangle fans*) [6].

Vertex buffery a index buffery mohou být uloženy ve video paměti nebo v systémové paměti. Renderování z bufferů, které se nacházejí ve video paměti, je rychlejší. Jestliže však potřebujeme měnit data nacházející se ve vertex bufferu či index bufferu, je nutné mít buffery uložené v systémové paměti.

Trojúhelníky modelu se renderují v tzv. dávce (*batch*). Každá dávka reprezentuje trojúhelníky, které využívají pro renderování stejně nakonfigurované renderovací stavy

(*render states*). Navíc trojúhelníky musejí tvořit v index bufferu souvislý blok. Používá-li model například více textur, musí být většinou renderován ve více dávkách. Avšak renderování většího množství dávek může zatěžovat CPU [7]. Proto se modely většinou optimalizují tak, aby byl počet dávek minimalizován.

Knihovna DirectX již obsahuje třídu reprezentující model. Umožňuje jeho nahrání z .x souboru, zakrývá přímé renderování z vertex bufferů a index bufferů a obsahuje funkce pro optimalizace modelu z hlediska dávek.

3.2 Reprezentace skeletonu a skeletální animace

Dalším krokem pro implementaci vertex blendingu je datový návrh skeletonu. Stavebním kamenem reprezentace skeletonu je struktura kosti. Základní datová struktura kosti je implementována v DirectX, neboť je pevně svázána s použitím .x souborů. Tato struktura se nazývá `D3DXFRAME`. Struktura kosti obsahuje mimo jiné lokální transformační matici (`TransformationMatrix`). Kosti musí tvořit hierarchickou strukturu, proto struktura kosti obsahuje odkaz na svého prvního potomka (`pFrameFirstChild`) a odkaz na svého sourozence v hierarchii (`pFrameSibling`).

```
struct D3DXFRAME
{
    LPSTR          Name;
    D3DXMATRIX     TransformationMatrix;
    D3DXFRAME      *pFrameSibling;
    D3DXFRAME      *pFrameFirstChild;
};
```

Pro renderování modelu potřebujeme výše popsanou strukturu kosti rozšířit o matici, která představuje transformaci z lokálních souřadnic kosti do souřadnic modelu – je to matice B_k v rovnici (2.1) (`CombinedTransformationMatrix`). Tyto transformační matice lze vypočítat pomocí prohledávání do hloubky v hierarchii skeletonu a postupným násobením lokálních transformačních matic.

```
struct D3DXFRAME_DERIVED : public D3DXFRAME
{
    D3DXMATRIX    CombinedTransformationMatrix;
};
```

Také je nutné mít uložené inverzní transformační matice kostí vzhledem k souřadné soustavě modelu v referenční póze (matice A_k^{-1} v rovnici (2.1), `OffsetMatrix`). Tyto matice nejsou uloženy ve strukturách kostí, ale ve speciálním objektu DirectX, který nese informace o skinningu.

DirectX zjednodušuje řízení skeletální animace. Implementuje tzv. animační kontroler, kterým lze ovládat animace na skeletonu. Animační kontroler obsahuje klíčové snímky různých animačních sekvencí. Pomocí animačního kontroleru můžeme nastavit, které animační sekvence se mají přehrávat. Objekt poté v závislosti na vstupním čase nastavuje lokální transformační matice jednotlivým kostem na skeletonu.

Celý postup přehrání animační sekvence lze shrnout v šesti bodech:

1. Aktualizace času *t*.
2. Nastavení `TransformationMatrix` v kostech pomocí animačního kontroleru podle času *t*.
3. Výpočet `CombinedTransformationMatrix` v kostech prohledáváním do hloubky hierarchií skeletonu.
4. Vynásobení `CombinedTransformationMatrix` a `OffsetMatrix` pro výpočet finálních transformačních matic kostí použitých ve vertex blendingu.
5. Správné nastavení renderovacích stavů a renderování modelu.
6. Zpět na krok 1.

Implementace bodu 5 je již závislá na použité metodě vertex blendingu.

3.3 Možnosti implementace vertex blendingu

DirectX nabízí několik způsobů implementace vertex blendingu. První možností je použití klasického zpracování vrcholů pomocí standardního grafického vykreslovacího řetězce (*fixed function pipeline*), který implementuje dva přístupy k vertex blendingu. Další možností je implementovat vertex blending přes tzv. vertex shader jednotky, které nahrazují klasické zpracovaní vrcholů. Ne však všechny možnosti jsou hardwarově implementovány na každé grafické kartě. Proto také bude diskutována hardwarová podpora různých implementací vertex blendingu na grafických čipech dvou předních výrobců – ATI a NVidia.

3.3.1 Fixed function non-indexed vertex blending (FFP1)

Technika neindexovaného vertex blendingu představuje myšlenkově nejjednodušší způsob, jakým lze vertex blending implementovat. Idea spočívá v tom, že lze přes rozhraní DirectX nastavit několik transformačních matic najednou. Při následném zpracování vrcholů se pozice a normály transformují všemi nastavenými maticemi. Výsledky se poté vynásobí vahami, které jsou uloženy ve struktuře vrcholu, a sečtou.

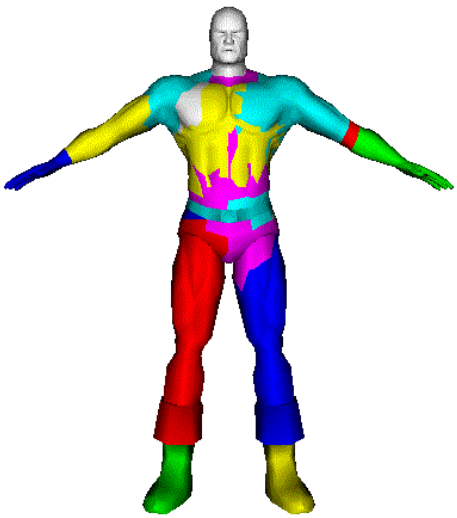
Struktura vrcholu pro vertex blending při nastavení čtyř transformačních matic, má tento tvar:

```
struct VERTEX
{
      D3DXVECTOR3 position;
      FLOAT       blend1, blend2, blend3;
      D3DXVECTOR3 normal;
};
```

Ve struktuře vrcholu jsou uloženy jen tři váhy. Poslední váha je vypočítána z podmínky konvexnosti.

DirectX umožňuje nastavit maximálně čtyři transformační matice najednou. Takový počet matic většinou nestačí pro renderování celého modelu. Proto je nutné model rozdělit na několik dávek. Trojúhelníky v každé dávce musejí používat stejnou sadu transformačních matic. Na obr. 7 lze vidět příklad modelu, který musí být renderován po dvaceti dávkách. To může při vykreslování více instancí tohoto modelu zbytečně zatěžovat CPU.

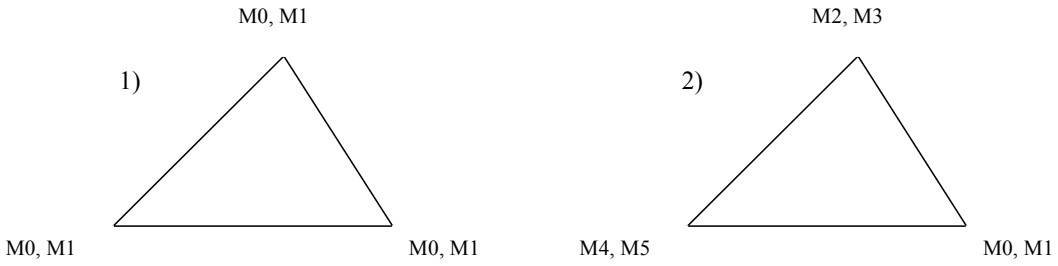
obr. 7: Rozdělení modelu na dávky při použití techniky neindexovaného vertex blendingu. Dávky tvoří trojúhelníky, které leží blízko sebe a mají stejné barevné označení



Neindexovaný vertex blending je navíc omezen faktem, že nelze měnit transformační matice při renderování jednoho trojúhelníku. Všechny vrcholy renderovaného trojúhelníku musejí být proto ovlivněny nastavenými transformačními maticemi. Příklad je uveden na obr. 8. Trojúhelník vlevo lze renderovat bez problémů. Avšak trojúhelník vpravo nelze za použití neindexovaného vertex blendingu vykreslit – i když se stále jedná o vertex blending se dvěma maticemi na vrchol. Občas se v tomto případě mluví o tzv. *per-triangle* vertex blendingu.

Neindexovaný vertex blending byl již hardwarově podporována grafickými kartami s čipem NVidia GeForce 256 (rok výroby 1999). Tyto karty umožňovaly nastavení dvou transformačních matic [8]. Dnešní čipy od výrobce NVidia podporují čtyři transformační matice. Firma ATI podporuje hardwarově urychlovaný neindexovaný vertex blending od uvedení čipu Radeon 7500 na trh (rok 2001). Již tento čip podporuje nastavení čtyř matic.

obr. 8: Ukázka 1) podporovaného, 2) nepodporovaného trojúhelníku za použití neindexovaného vertex blendingu se dvěma maticemi.



3.3.2 Fixed function indexed vertex blending (FFP2)

Indexovaný vertex blending odstraňuje téměř všechny nevýhody neindexovaného vertex blendingu. Na grafickou kartu se neposílají jen právě používané matice pro vertex blending, ale celé pole matic. Struktura vrcholu obsahuje indexy do tohoto pole, které označují jakými maticemi je vrchol ovlivňován (jsou uložené v proměnné `indices`).

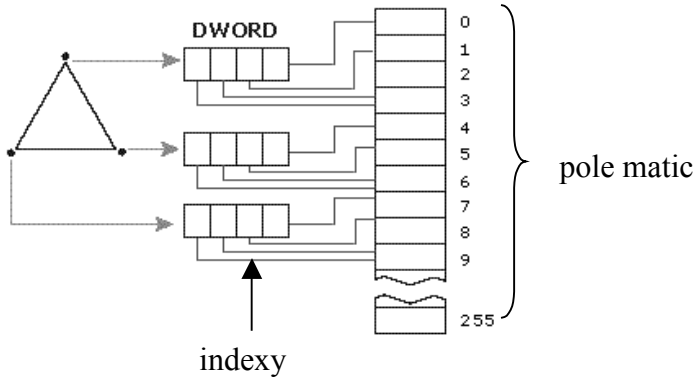
```
struct VERTEX
{
      D3DXVECTOR3 position;
      FLOAT       blend1, blend2, blend3;
      DWORD       indices;
      D3DXVECTOR3 normal;
};
```

Indexování razantním způsobem snižuje nutnost dělení modelu na dávky. I v tomto případě je však dělení modelu někdy potřeba. Zda je nutné model dělit, závisí na podporovaném počtu matic konkrétní karty a na velikosti skeletonu. Protože je indexování do pole matic na úrovni vrcholů, je odstraněno omezení per-triangle vertex blendingu.

Hardwarová podpora indexovaného vertex blendingu není tak rozšířená jako u neindexované verze. Tuto techniku podporují jen některé grafické čipy od výrobce ATI.

Prvním takovým čipem byl Radeon 8500 (rok 2001, podporuje 57 matic). Majitelé grafické karty s čipem od firmy NVidia musí vzít zavděk softwarovou emulací.

obr. 9: Schéma indexace vrcholů trojúhelníku do pole matic (převzato z [6]).



3.3.3 Software (SW)

Na první pohled se může zdát zbytečné zabývat se softwarovým řešením vertex blendingu, když existuje jeho hardwarová podpora. Avšak u starších karet vertex blending nemusí být podporován (nebo nemusí být podporován dostatečný počet matic). Navíc, jak bylo zmíněno výše, praktičtější indexovaný vertex blending není podporovám čipy výrobce NVidia, která drží podstatný podíl na trhu s grafickými čipy. Chceme-li výše popsané metody používat bez podpory hardwaru, stačí nastavit v DirectX softwarové zpracování vrcholů.

Dalším důvodem implementace vertex blendingu v softwaru může být i jiné použití vypočtených dat než jen na renderování modelu. Ačkoliv existují způsoby jak data zpracovaná na GPU přenést zpět na CPU, obvyklejším způsobem je výpočet provést přímo na CPU. Výpočet v softwaru se provádí, když potřebujeme například počítat přesné detekce kolizí nebo pro pozdější určení stínového tělesa k zobrazení ostrých stínů.

Nelze očekávat, že by softwarové řešení vertex blendingu mohlo rychlostně konkurovat hardwarovým implementacím. Proto by se mělo používat jen v nutných případech. Softwarové řešení také jako jediné ze všech uváděných implementací vertex blendingu potřebuje mít vertex buffer uložený v systémové paměti.

V programu je softwarový vertex blending implementovaný bez pomocí softwarového zpracování vrcholů v DirectX. Vrcholy jsou transformovány „ručně“.

3.3.4 Použití jednotky vertex shader

Poslední způsoby implementace vertex blendingu spočívají v použití vertex shader jednotky. Vertex shader je programovatelná jednotka GPU, která nahrazuje klasické

zpracování vrcholů. Použití vertex shader jednotek (spolu s jednotkami pixel shader) přináší nový pohled na programování grafiky. Klasické zpracování vrcholů za použití standardního vykreslovacího grafického řetězce spočívalo v nastavení několika renderovacích parametrů, jako například transformačních matic, barev či světel. Vertex shader jednotky umožňují napsat přímo program, který se pro každý vrchol provede.

Vstupním parametrem vertex shaderu je struktura vrcholu. Velikost této struktury je omezena na 16 vektorových registrů. Vertex shader navíc obsahuje registry, do kterých lze nahrát data, jež jsou konstantní pro všechny vrcholy. Klasické využití těchto registrů je uložení transformačních matic. Povinným výstupem vertex shaderu je homogenní pozice vrcholu v tzv. *clip space* (tj. vynásobená projektivní transformací). Používá-li se pixel shader, jsou ostatní data nepovinná a jejich případná sémantika závisí na využití těchto dat v pixel shaderu. Jestliže nebudou použity pixel shadery, musejí být výstupem vertex shaderu všechna data potřebná pro klasické zpracování pixelů ve standardním grafickém vykreslovacím řetězci.

Existuje několik verzí specifikace vertex shaderu, které se liší instrukčními sadami, délkou programu a počtem registrů. Nejjednodušší vertex shader verze 1.1 byl podporován již v DirectX 8.0 (rok 2001). Tato verze obsahuje jen aritmetické instrukce a délka programu je omezena na 128 instrukcí. Vertex shader verze 2.0 je podporován v DirectX 9.0 (rok 2002) a vyššími verzemi. Tyto shadery obsahují instrukce pro řízení toku kódu podle konstantních registrů. Délka programu se zvětšila na 256 instrukcí. Zatím nejnovější verze vertex shaderu 3.0 je podporovaná v DirectX 9.0c (rok 2004). Specifikace 3.0 umožňuje řízení toku kódu podle dat ve struktuře vrcholu. Navíc přibyly instrukce pro čtení dat z textury. Dolní hranice pro maximální délku programu je 512 instrukcí. Bližší informace o rozdílech mezi jednotlivými verzemi a počtech registrů lze nalézt v [6].

První programovatelný grafický čip byl GeForce 3 od výrobce NVidia.Tento čip podporoval vertex shadery verze 1.1. Podpora vertex shaderu 2.0 firmou NVidia přišla až s grafickými kartami postavenými na čipech GeForce FX 5xxx. GeForce FX 6xxx a nová GeForce FX 7800 podporují vertex shadery verze 3.0.

ATI podporuje vertex shadery verze 1.1 od vydání karet Radeon 8500. Vertex shadery 2.0 podporují karty Radeon 9500 a vyšší. ATI zatím nevyrábí čipy, které by podporovaly vertex shadery verze 3.0.

Programy pro vertex shader lze psát přímo v asembleru, který je nezávislý na grafickém čipu. Protože jsou GPU založena na architektuře SIMD, pracují aritmetické instrukce asembleru se čtyřsložkovými vektory. DirectX verze 9.0 již obsahuje vyšší programovací jazyk HLSL pro psaní shaderů, jenž se syntaxí podobá jazyku C. HLSL zjednodušuje programování shaderů. Pro psaní efektivního kódu v HLSL je však lepší znát instrukční sadu a využívat práci s vektory.

DirectX umožňuje kompilaci z programu psaném v HLSL do binárního kódu za běhu programu. Cílovou verzi vertex shaderu, do kterého se program přeloží, lze v tomto případě zvolit podle konkrétní grafické karty počítače, na němž program běží. Nevýhodou je však to, že je nutné spolu s programem distribuovat kód v HLSL. Praxe je spíše taková, že se zvolí, které verze shaderů budou podporovány. Do zvolených verzí jsou předem shader programy psané v HLSL zkompilovány.

Všechny zde níže popsané programy pro vertex shadery, které implementují vertex blending, jsou psány v HLSL. Kompilují se za běhu programu dle specifikace vertex shader verze 2.0. Vertex shader programy psané v HLSL jsou uloženy v tzv. efekt souborech. Každý efekt soubor může definovat několik kompilací stejného vertex shader programu v závislosti na parametrech. V tomto případě jsou vertex shader programy kompilovány několikrát podle různé hodnoty počtu kostí, které ovlivňují vrcholy.

Vertex shader program 1 (VSP1)

První implementace vertex blendingu pomocí vertex shaderu, kterou zde představím, lze najít v [6] – jediný rozdíl je v tom, že je překládán do verze 2.0 vertex shaderu. Implementace spočívá na myšlence indexovaného vertex blendingu, proto používá stejnou strukturu vrcholu.

Program vertex shaderu obsahuje pole, v němž jsou uloženy transformační matice (`mWorldMatrixArray`). Ty jsou indexovány jednotlivými vrcholy. Dále je potřeba matice, která obsahuje kombinovanou pohledovou a projekční transformaci (`mViewProj`). Poslední podstatný údaj je počet matic, které ovlivňují vrchol (`NumBones`). Ostatní údaje, jako difusní barva modelu či světla, jsou z hlediska vertex blendingu nepodstatné. Všechna tato data se v průběhu renderování nemění pro jednotlivé vrcholy modelu, proto mohou být uložena v konstantních registrech vertex shaderu. Nejzajímavější část programu vypadá takto:

```
for (int iBone = 0; iBone < NumBones-1; iBone++)
{
      LastWeight = LastWeight + BlendWeightsArray[iBone];
      Pos    += mul(i.Pos, mWorldMatrixArray[IndexArray[iBone]])*
                BlendWeightsArray[iBone];
      Normal += mul(i.Normal,mWorldMatrixArray[IndexArray[iBone]])*
                BlendWeightsArray[iBone];
}
LastWeight = 1.0f - LastWeight;
Pos    += mul(i.Pos, mWorldMatrixArray[IndexArray[NumBones-1]])*
          LastWeight;
Normal += mul(i.Normal,mWorldMatrixArray[IndexArray[NumBones-1]])*
          LastWeight;
```

V cyklu se postupně transformují pozice a normála vrcholu `i` prostřednictvím jednotlivých kostí, které ovlivňují tento vrchol. Indexy do pole matic jsou uloženy v poli `IndexArray`, váhy k jednotlivým kostem se nacházejí v `BlendWeightsArray`. Příspěvky jednotlivých kostí se postupně sčítají. Protože váha k poslední kosti není ve struktuře vrcholu obsažena, je spočítána ve vertex shader programu a příspěvky poslední kosti se započítají až po skončení cyklu. Tímto způsobem vypočteme pozici a normálu ve světových souřadnicích. Ve zbytku vertex shader programu je nutné pozici transformovat pohledovou a projekční maticí (`mViewProj`), případně normalizovat normálu a započítat příspěvky osvětlení.

Důležitou otázkou je, jak velký počet světových matic kostí lze mít ve vertex shaderu. Jednu homogenní matici lze uložit do čtyř registrů (registry jsou vektorové a mají čtyři složky). Avšak v tomto případě není nutné ukládat celou homogenní matici. Do vertex shaderu stačí posílat podmatici 4x3, která představuje rotaci a posunutí. Vertex shader 2.0 obsahuje 256 konstantních čtyřsložkových registrů. Maximální možný počet matic je tedy 85 (zanedbáme-li požadavky ostatních parametrů). Takový počet matic je většinou dostatečný.

K tomu, abychom mohli lépe analyzovat vertex shader program, je nutné nahlédnout do přeloženého kódu. Podíváme-li se na posloupnost vygenerovaného kódu, zjistíme, že cyklus v HLSL je přeložen za pomocí instrukce pro řízení toku programu `REP`.

Vertex shader program 2 (VSP2)

Zjištění, že první vertex shader program využívá instrukce `REP`, mě přivedlo hned k další metodě. Pro implementaci vertex blendingu není použití instrukce pro cyklus nutně zapotřebí. Kompilátor, který je součástí [6], umožňuje pomocí přepínače zabránit použití instrukcí pro řízení toku programu. I s tímto přepínačem lze výše popsaný vertex shader program přeložit. Druhá implementace vertex blendingu pomocí vertex shaderu představuje stejný program v HLSL, avšak přeložený bez použití instrukcí pro řízení toku programu.

HLSL program pro vertex blending se čtyřmi maticemi je zkompilován do kódu v asembleru o délce 51 instrukcí. Pro podrobnější analýzu je nezbytné alespoň zhruba popsat, jak je podstatná část programu pro vertex blending přeložena.

Pro transformaci vektoru maticí v asembleru specifikace vertex shaderu 2.0 instrukce neexistuje. Násobení maticí je nutné implementovat pomocí tří instrukcí skalárního součinu (`DP`). K výpočtu příspěvku pozice jedné kosti jsou tedy potřeba tři instrukce `DP` plus jedna instrukce `MAD`, která příspěvek vynásobí vahou ke kosti a připočte do výsledné pozice. Zahrneme-li výpočet normály, lze počet instrukcí pro

důležitý úsek programu v závislosti na počtu kostí na vrchol (*NumBones*) vyjádřit takto:

$$NumBones(3.\text{DP} + \text{MAD} + 3.\text{DP} + \text{MAD})$$

Jestliže nebudeme rozlišovat typy instrukcí, je počet instrukcí roven $8.NumBones$. Poslední implementace vertex blendingu ukáže, jakým způsobem lze toto číslo zmenšit.

Vertex shader program 3 (VSP3)

Idea použitá v posledním vertex shader programu je velmi jednoduchá a vychází z asociativnosti a distributivnosti násobení. Jednoduchou úpravou výpočtu pozice vrcholu v rovnici (2.1) dostaneme

$$v' = \sum_{i=1}^{n} w_i(B_i A_i^{-1} v) = \sum_{i=1}^{n} (w_i B_i A_i^{-1}) v = (\sum_{i=1}^{n} w_i B_i A_i^{-1}) v$$

Výše popsané implementace počítají lineární kombinaci transformovaných pozic. Nyní nejdříve vypočteme výslednou matici, kterou vynásobíme pozici a normálu vrcholu. Tím podstatně snížíme počet instrukcí. Důležitá část shader programu vypadá takto:

```
for (int iBone = 0; iBone < NumBones-1; iBone++)
{
      LastWeight     = LastWeight + BlendWeightsArray[iBone];
      FinalMatrix    += mWorldMatrixArray[IndexArray[iBone]]*
                        BlendWeightsArray[iBone];
}
LastWeight     = 1.0f - LastWeight;
FinalMatrix    += mWorldMatrixArray[IndexArray[NumBones-1]]*
                  LastWeight;
Pos      = mul(i.Pos, FinalMatrix);
Normal   = mul(i.Normal, FinalMatrix);
```

Sčítání příspěvků matic se v asembleru provádí pro každý řádek matice zvlášť. Pro přičtení příspěvku jedné matice jsou nutné tři instrukce `MAD`. Nakonec je nutné transformovat pozici vrcholu vypočtenou maticí pomocí tří instrukcí `DP`. Stejně vypočteme i výslednou normálu. Výsledný počet instrukcí v závislosti na počtu kostí je

$$NumBones(3.\text{MAD}) + 3.\text{DP} + 3.\text{DP}$$

Nebudeme-li rozlišovat jednotlivé instrukce, lze počet instrukcí vyjádřit jako $3.NumBones + 6$, což je v porovnání s VSP2 výrazná úspora. Pro vertex blending se

čtyřmi maticemi ušetříme celých 14 instrukcí. Při vertex blendingu se dvěma maticemi lze ušetřit 4 instrukce.

3.4 Porovnání metod

Nyní je načase uvedené metody implementace vertex blendingu porovnat vzhledem k rychlosti. Všechny techniky (jsou-li podporovány) jsou porovnány na třech grafických kartách. Jediným zástupcem ATI je Radeon 9800 PRO, který představuje jednu z nejlepších karet starší generace. Firma NVidia je zastoupena dvěma kartami – starší GeForce FX 5700 a GeForce FX 6800, která jako jediná z uvedené trojice podporuje i specifikaci vertex shader 3.0. Testy byly prováděny na třech počítačích o různé konfiguraci. To snižuje vypovídající hodnotu porovnání jednotlivých karet. Primárním cílem však bylo změřit různé implementace vertex blendingu na konkrétní kartě, ne porovnávat rychlosti karet mezi sebou.

Implementace vertex blendingu byly měřeny na snímkovou frekvenci. Abychom mohli změřit rychlost jednotlivých metod, je nutné zaručit, aby úzké hrdlo (*bottleneck*) aplikace spočívalo ve zpracování vrcholů. Všechny ostatní možné příčiny zpomalení aplikace je nutné potlačit. V praxi to znamenalo snížit velikost frame bufferu na minimum (nebo alespoň do té míry, aby neovlivňoval snímkovou frekvenci), vyvarovat se textur a nepoužívat pixel shader. Náročnost aplikace na CPU byla minimální, testy tedy nebyly limitovány rychlostí CPU.

Model použitý pro testy se skládá z 13206 vrcholů a 4402 trojúhelníků. Důvod tak velkého počtu vrcholů je ten, že model nebyl nijak optimalizován. Nutné rozdělení modelu pro metodu FFP1 po optimalizaci totiž mění počet vrcholů. Dal jsem přednost neoptimalizovanému modelu, aby byl počet vrcholů pro všechny metody stejný.

Skeleton modelu se skládá z 23 kostí, model využívá čtyřmaticový vertex blending. Při měření snímkové frekvence byla přehrávána animační sekvence a model byl renderován v 36 kopiích. Výsledky měření jsou uvedeny v tabulce, hodnoty jsou uvedeny v počtu snímků za sekundu. Grafické karty jsou v tabulce řazeny vzestupně podle jejich udávané výkonnosti.

	SW	**FFP1**	**FFP2**	**VSP1**	**VSP2**	**VSP3**
GeForce FX 5700	4	**61**	-	15	27	40
Radeon 9800 PRO	6	59	46	35	54	**72**
GeForce FX 6800	5	69	-	34	73	**99**

V testech propadla metoda SW, což byl očekávaný výsledek. Softwarové řešení může přinést jiné výhody než je rychlost, jak již bylo zmíněno výše.

Druhou nejhorší implementací je podle výsledků testu VSP1. Je vidět, že se nevyplácí slepě kompilovat shader programy – implementace VSP2 bez použití instrukcí řízení toku programu dává daleko lepší výsledky. Z implementací využívajících vertex shader dopadla nejlépe VSP3. Redukce počtu instrukcí se výrazně projevila na zvýšení rychlosti.

Absolutní vítěz testu pro všechny karty neexistuje. Pro GeForce FX 5700 je jím FFP1. U dvojice Radeon 9800 PRO a GeForce FX 6800 již dominuje VSP3. U GeForce FX 6800 je dokonce i VSP2 rychlejší než FFP1. To nejspíš souvisí s kvantitou a kvalitou vertex shader jednotek. Počet a optimalizace těchto jednotek se bude i nadále zvětšovat, proto je použití vertex shader programů perspektivnější než využití klasického zpracování vrcholů.

Kapitola 4. Vylepšování skinningu

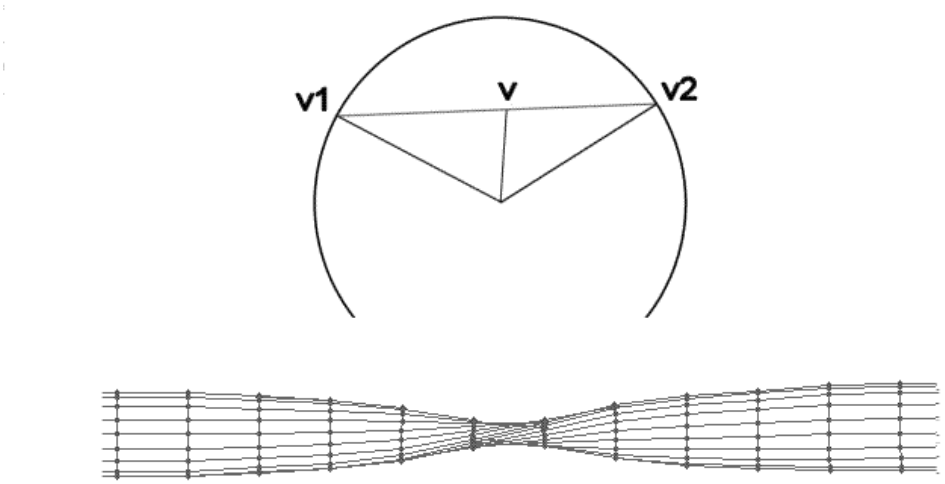
V této kapitole se budu vertex blendingu a obecně skinningu věnovat znovu více z teoretického hlediska. Jak jsem se již zmiňoval, vertex blending má i některé nedostatky. Nyní tyto nedostatky popíši a vysvětlím jejich příčiny. Také budou představeny některé nové přístupy k problému skinningu, které se snaží vyvarovat nedostatků vertex blendingu.

4.1 Nedostatky metody vertex blending

Deformace vertex blendingu vypadají velmi dobře pro nevelké změny v rotacích. Avšak při rotacích s velkým úhlem otočení tato metoda dává neuspokojivé výsledky. Klasickým příkladem je tzv. „candywrapper“ problém. Uvažujme model válce, jehož skeleton se skládá z kosti 1 a kosti 2, které leží v ose válce. Výsledek rotace kosti 2 okolo vlastní osy lze vidět na obr. 10. Vertex blending se tedy nechová tak, jak bychom potřebovali například pro rotaci předloktí modelu lidské postavy. Deformace vertex blendingu se proto přirovnávají k deformaci papírové trubičky.

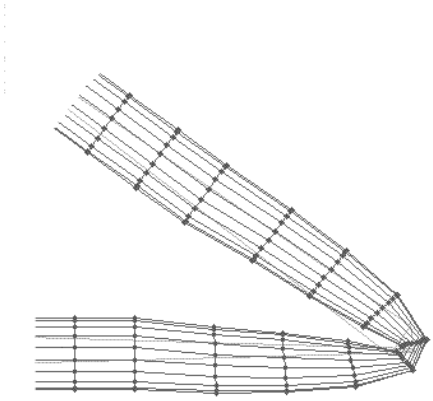
Důvod takového chování je patrný z rozboru výpočtu pozice vrcholu při vertex blendingu (2.1). Množina všech konvexních kombinací dvou pozic vrcholu $v1$ a $v2$, které představují příspěvky jednotlivých kostí, tvoří úsečku mezi těmito vrcholy. Výsledná pozice vrcholu v tedy musí ležet na této úsečce. Kruh na obr. 10 si lze představit jako průřez modelu válce. Z obrázku je patrné, že při velkých rotacích musí docházet ke kolapsu vrcholů do osy válce. Je důležité vědět, že tento kolaps není závislý na konkrétních vahách a nelze jej tedy odstranit jejich změnou.

obr. 10: Ukázka kolapsu při velké rotaci kolem osy kosti (tzv. „ candywrapper“ problém).



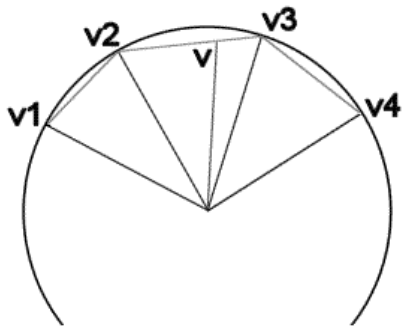
Neuspokojivé výsledky dává vertex blending také při velkém ohýbání. I v tomto případě dochází ke ztrátě objemu v bodě ohybu (obr. 11).

obr. 11: Ukázka kolapsu při velkém ohýbání válce.



3D grafik by měl znát tyto nepěkné vlastnosti vertex blendingu. Stejně tak by měl vědět, že pouhou úpravou vah toto chování nelze napravit. To ovšem neznamená, že nelze toto chování alespoň zmírnit. Jedním ze způsobů, jak toho docílit, je rozprostřít rotaci do více kostí. Tím zajistíme, aby rotace mezi dvěma kostmi nebyla příliš velká. Skeleton je tedy nutné rozšířit o malé kosti v kloubech, které mají velký rozsah rotací. Vrcholy blízko kloubu budou nyní využívat i nových kostí. Schéma zmírnění kolapsu pro rotaci kolem osy lze vidět na obr. 12.

obr. 12: Zmírnění kolapsu po přidání kostí.



Kosti lze přidat ručně při vytváření modelu. Přidávání kostí může být také zautomatizováno [9].

I přes výše popsané nepěkné vlastnosti je vertex blending velmi rozšířená metoda. Dá se říci, že vzhledem k jednoduchosti výpočtu je chování této metody při velkých ohybech tolerováno.

4.2 Nové metody

Problematika skinningu v realtime aplikacích není uzavřené téma. Svědčí o tom množství odborných článků, které se skinningem zabývají. S příchodem programovatelných vertex shader jednotek se navíc otevřela možnost implementace nových metod přímo na GPU. Vzhledem k velkému počtu odborných článků budou popsány jen některé práce, a to hlavně ty, které se zmiňují o hardwarové implementaci.

Většina nových přístupů vychází z metody vertex blending a určitým způsobem ji rozšiřují. Avšak žádná z nových metod není zatím natolik úspěšná či odzkoušená, aby mohla být masivně nasazena v praxi.

4.2.1 Multi-weight enveloping (MWE)

Multi-weight enveloping [10] zajímavým způsobem zobecňuje myšlenku vertex blendingu. Narozdíl od klasické metody vrchol neobsahuje jen jednu váhu ke každé kosti, ale celou matici vah. Výsledná pozice vrcholu je vypočtena následujícím vzorcem:

$$v' = \sum_{i=1}^{n} \begin{pmatrix} w_{00i}m_{00i} & w_{01i}m_{01i} & w_{02i}m_{02i} & w_{03i}m_{03i} \\ w_{10i}m_{10i} & w_{11i}m_{11i} & w_{12i}m_{12i} & w_{13i}m_{13i} \\ w_{20i}m_{20i} & w_{21i}m_{21i} & w_{22i}m_{22i} & w_{23i}m_{23i} \\ 0 & 0 & 0 & 1 \end{pmatrix} v$$

Váhy pro každý prvek transformační matice jistě rozšiřují množinu možných deformací.

Jestliže se o vertex blendingu občas říká, že je obtížné pro 3D grafiky nastavit váhy, v tomto případě nemůže být o ručním nastavení ani řeč. Autoři této metody váhy počítají automaticky z několika vzorových póz modelu. Analýzou hlavních komponent (*principal component analysis*) odstraňují korelace jednotlivých vah, metodou nejmenších čtverců vypočtou samotné váhy.

Výsledky prezentované v [10] ukazují, že touto metodou lze odstranit „candywrapper“ problém. Autoři také poukazují na fakt jednoduché implementace samotného výpočtu pozice vrcholu v hardwaru. Problémem by však mohl být ve velkém objemu dat spojených s vrcholem. Limit na velikost struktury vrcholu ve vertex shaderu povoluje maximálně vliv čtyř kostí na jeden vrchol.

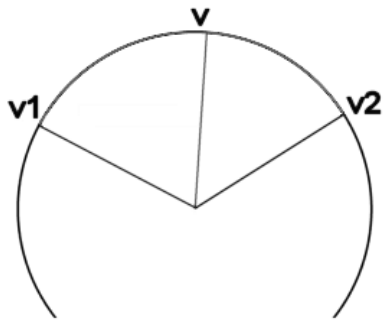
4.2.2 Metody využívající interpolaci transformací

V poslední době se objevily dvě práce, které se místo interpolace pozice jednotlivých vrcholů snaží interpolovat jednotlivé transformace kostí [11], [13]. Výsledná transformace je poté použita pro výpočet pozice vrcholu. Upravená rovnice (2.1) k této myšlence přímo vybízí:

$$v' = \sum_{i=1}^{n} w_i (B_i A_i^{-1} v) = (\sum_{i=1}^{n} w_i B_i A_i^{-1}) v = (\sum_{i=1}^{n} w_i C_i) v$$

Výraz $\sum_{i=1}^{n} w_i C_i$ lze chápat jako jakousi interpolaci transformací. Pro lepší výsledky je však nutné použít jiný způsob interpolace. Obě zmiňované práce používají tzv. sférickou interpolaci, která přirozeným způsobem interpoluje rotace. Podívejme se na obr. 13. Narozdíl od vertex blendingu, který vrcholy interpoluje lineárně, sférická interpolace zachovává vzdálenost od středu rotace. Proto také nedochází ke ztrátě objemu a nepříjemnému „candywrapper“ problému.

obr. 13: Sférická interpolace.



Pro sférickou interpolaci však není maticová reprezentace rotace vhodná. *Bones blending* L. Kavana a J. Žáry [11] používá reprezentaci osa-úhel (*axis-angle representation*) [14]. J. Hejl [13] ve své metodě s*pherical joint blending* využívá pro reprezentaci rotací kvaterniony (viz Dodatek A).

S translačními částmi transformací se zachází poněkud jinak než ve vertex blendingu. Obě práce se zmiňují o tom, že vrchol ovlivňovaný více kostmi má být posunut jen podle jedné z těchto kostí. Metoda bones blendingu byla představena jen na dvou spojených kostech a vrchol má být posunut podle kloubu mezi těmito dvěma kostmi. To odpovídá posunutí druhé kosti v hierarchii. J. Hejl se o volbě kosti nezmiňuje.

Práce J. Hejla dokonce ukazuje, jakým způsobem lze jeho metodu implementovat pomocí vertex shader programu, v němž počítá aproximaci sférické lineární interpolace pomocí lineární interpolaci kvaternionů. Svoji implementaci porovnává s konkrétní implementací vertex blendingu [15] a ukazuje, že je rychlejší. Je však nutné podotknout, že porovnávaná implementace vertex blendingu není optimální. I tato práce prezentuje výsledky jen na jednoduchém modelu válce se dvěma kostmi.

Dá se předpokládat, že obě práce mají velmi podobné výsledky a liší se jen v implementaci.

Metody interpolace transformací mají velkou přednost v tom, že používají shodná data jako vertex blending (narozdíl např. od metody MWE). Způsob editace vah u vrcholů může zůstat naprosto stejný. Vzhledem k rozšířenosti vertex blendingu je to nesporná výhoda.

4.2.3 Metody využívající interpolace vzorových póz

Poslední skupina metod používá vzorové pózy modelu, mezi kterými se při animaci interpoluje. Dá se tedy říci, že se jedná o kombinaci skeletální animace s per vertex animací. Tyto metody mají výhodu v tom, že lze pomocí vzorových póz simulovat například i pohyb svalů. Nevýhodou je však potřeba vytvoření vzorových póz.

Pose space deformation (PSD)

První takovou prací je *pose space deformation* [4]. Autoři navrhují obecný postup interpolace dat v prostoru póz. V případě skeletální animace zahrnuje prostor póz všechny konfigurace skeletonu a jeho dimenze je dána počtem kostí. Prostor póz tvoří definiční obor interpolace. Oborem hodnot jsou jednotlivé pozice všech vrcholů modelu. Vzorové pózy (neboli dvojice konfigurace skeletonu a příslušné pozice vrcholů, případně hodnoty normál) tvoří známé body v tomto prostoru. Interpolace pozic vrcholů se provádí pomocí radiálních bázových funkcí (viz Dodatek B). Celý proces této metody je popsán v několika krocích:

1. Vytvoření vzorových póz pro několik konfigurací skeletonu.
2. Výpočet vektorů posunutí. Pro každou vzorovou pózu a každý vrchol je vypočten vektor, o který byl vrchol posunut vzhledem k referenční póze. Vektor posunutí je počítán v souřadné soustavě kosti, ke které je vrchol spjat (tato metoda rozšiřuje základní skinning).
3. Výpočet vah nutných pro interpolaci pomocí radiálních bázových funkcích. Váhy jsou vypočítány pro každý vrchol zvlášť. Pro výpočet se používají vzorové pózy a vektory posunutí.
4. Syntéza pózy modelu v nové konfiguraci skeletonu. Pomocí radiálních bázových funkcí a vah vypočtených v kroku 3 jsou vypočítány

interpolované vektory posunutí. Pomocí nich již lehce vypočteme výslednou pozici vrcholu. Neodpovídá-li výsledek představě 3D grafika, je nutné přidat novou vzorovou pózu v této konfiguraci skeletonu.

Pro interpolaci pomocí radiálních bázových funkcí je nutné definovat metriku v prostoru póz (mezi jednotlivými konfiguracemi skeletonu). Jakým způsobem zvolit tuto metriku autoři neuvádějí.

Proces syntézy výsledné pózy v libovolné konfiguraci skeletonu je dle autorů jen o trochu náročnější než vertex blending. S jakým úspěchem lze však touto metodou odstranit „candywrapper“ problém autoři nepopisují.

Shape by example (SBE)

Metoda *shape by example* [16] je velmi podobná předchozí práci. V PSD je pozice každého vrcholu brána jako samostatný interpolační problém. SBE naproti tomu interpoluje vzorové pózy jako nedílné celky. Tímto způsobem je zjednodušen krok 3 v PSD. Váhy nutné pro interpolaci pomocí radiálních bázových funkcích se tedy počítají pro celou vzorovou pózu, ne pro každý vrchol zvlášť.

Jakým způsobem lze touto metodou rozšířit vertex blending, autoři ukazují na modelu paže se dvěma kostmi. V tomto případě používají jednodušší interpolační schéma než je interpolace pomocí radiálních bázových funkcí. Rovnici vertex blendingu se dvěma kostmi lze zapsat v tomto tvaru:

$$v' = w_1 C_1^{\theta} v_0 + w_0 C_0 v_0$$

Matice C_1^{θ} je transformace dolní kosti v hierarchii pro úhel ohnutí θ vzhledem ke kosti předka. Úhel θ je mapován na interval $\langle 0,1 \rangle$, v_0 je pozice vrcholu v referenční póze modelu. Paže je vymodelována ve dvou pózách. Natažená paže určuje referenční pózu (θ je rovno 0). Druhou pózu představuje ohnutá paže v lokti. V tomto případě je úhel θ roven 1, pozici vrcholu v této póze budeme značit v^1. Tato pozice je promítnuta do referenční pózy (obr. 14):

$$v_0^1 = (w_1 C_1^1 + w_0 C_0)^{-1} v^1$$

Nyní máme dvě pozice stejného vrcholu v referenční póze pro různé vzorové pozice kosti. Mezi nimi můžeme interpolovat podle úhlu ohnutí kosti θ:

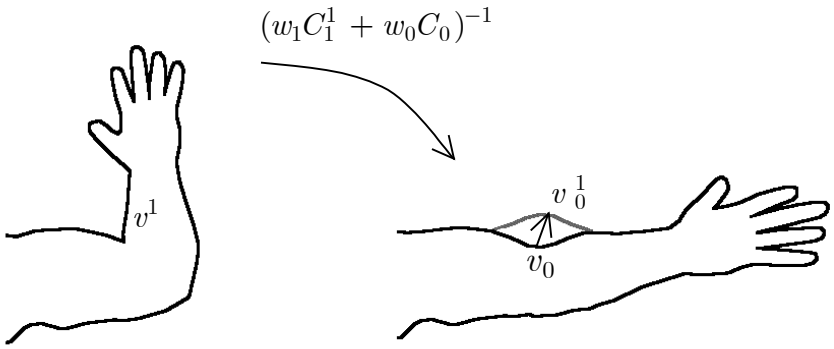
$$v_0^{\theta} = \theta v_0^1 + (1-\theta) v_0$$

Výsledná pozice vrcholu je počítána pomocí vertex blendingu za použití interpolované pozice vrcholu v referenční póze:

$$v' = w_1 C_1^{\theta} v_0^{\theta} + w_0 C_0 v_0^{\theta}$$

Stejně jako u PSD, také u této metody se autoři nezmiňují, zda lze tímto způsobem řešit problémy vertex blendingu při rotaci kosti okolo své osy.

obr. 14: Schéma metody SBE. Pozice vrcholu v^1 je ze vzorové pózy transformována do referenční pózy. Při zobrazení pózy, která leží mezi referenční pózou a vzorovou pózou je pozice vrcholu interpolována z hodnot v_0 a v_0^1, až poté je transformována. Na druhém obrázku je zvýrazněn rozdíl transformované vzorové pózy od referenční pózy.



Eigenskin

Výše popsané práce spíše nastiňují, jakým způsobem lze také skeletální animaci řešit. Práce popisující metodu *eigenskin* [17] představuje konkrétní implementaci. Základní idea metody je naprosto stejná jako v SBE – interpolace několika pozic vrcholu v referenční póze. Vzorové pózy pro eigenskin jsou tvořeny ohýbáním vždy jen jedné kosti. Z těchto dat se zjistí tzv. *support* pro konkrétní kost. Support kosti *k* je množina vrcholů, které po promítnutí do referenční pózy z nějaké vzorové pózy pro kost *k* mají nenulový vektor posunutí. Pomocí analýzy hlavních komponent jsou tyto vektory posunutí pro každou kost dekolerovány a transformovány do nové báze.

Interpolace mezi pózami je prováděna pomocí radiálních bázových funkcí. Vzdálenost v prostoru póz (mezi dvěma pózami jedné kosti) je definována jako velikost úhlu v reprezentaci rotace osa-úhel.

Implementace metody eigenskin využívá vertex shaderů. Struktura vrcholu obsahuje vektory posunutí pro různé pózy, interpolace je tedy přesunuta na GPU.

Eigenskin byl představen na detailním modelu lidské ruky. Ani tato práce neukazuje chování při rotaci kosti okolo své osy.

Kapitola 5. Implementace nové metody skinningu

Po prostudování nových přístupů k řešení skinningu jsem byl rozhodnut jednu z nových metod implementovat (za použití vertex shader specifikace 2.0). V této kapitole popíši, jakou metodu jsem se rozhodl implementovat a proč. Zvolená technika bude podrobněji zkoumána z vizuálního hlediska, také se podrobněji zmíním o jejích nedostatcích. Nakonec bude navrženo a implementováno rozšíření této metody.

5.1 Výběr metody

Většina nových přístupů využívá určitým způsobem vzorové pózy. Metody MWE a PSD je přímo potřebují pro správné fungování. Vytvoření vzorových póz pro všechny možné polohy kostí může být značně náročné. Proti tomu metodou SBE a metodou eigenskin lze pomocí póz řešit jen nepřesnosti vertex blendingu. Pózy mohou být vytvořeny jen pro ty polohy kostí, kdy vertex blending dává špatné výsledky. Avšak ani autoři SBE, ani autoři metody eigenskin se nezmiňují o řešení „candywrapper“ problému. I kdyby bylo možno tento problém řešit, bylo by nutné pro potřebné kosti vzorové pózy vytvořit.

Mým favoritem mezi popsanými novými přístupy byly jednoznačně metody využívající interpolaci transformací. Tyto metody nevyžadují vytvoření vzorových póz, navíc - jak již bylo řečeno - velmi přirozeně řeší „candywrapper“ problém. Kromě toho jejich způsob použití a implementace se zásadně neliší od vertex blendingu. Kvůli těmto přednostem jsem se rozhodl pro implementaci metody *spherical joint blending*.

5.2 Metoda spherical joint blending

Dříve než bude popsána implementace SJB, je nutné popsat matematické odvození této metody. Z matematického odvození bude patrné jisté omezení kladené na tuto metodu, které není výslovně uvedeno ani v [13], ani v [11]. Autor [11] však toto omezení potvrdil.

Nahrazení rovnice vertex blendingu interpolací transformací není tak přímočaré, jak by se mohlo na první pohled zdát. Nelze jednoduše nahradit konvexní kombinaci rotačních matic kombinací kvaternionů. Důvod je zřejmý po rozepsání rovnice vertex blendingu[1]:

$$\sum_{i=1}^{n} w_i B_i {A_i}^{-1} v \equiv \sum_{i=1}^{n} w_i B_{Ri} {A_{Ri}}^{-1} \left(v + A_{Ri} A_{Ti}^{-1} \right) + \sum_{i=1}^{n} w_i B_{Ti}$$

[1] Nyní opouštíme homogenní vyjádření matic, pozic vrcholů a normálových vektorů.

Rotační část homogenní matice je indexována písmenem R, translační písmenem T. Konvexní kombinaci rotačních matic $\sum_{i=1}^{n} w_i B_{Ri} {A_{Ri}}^{-1}$ nelze jednoduše nahradit konvexní kombinací kvaternionů. Možné by to bylo jen v případě, kdyby výraz $A_{Ri} A_{Ti}^{-1}$ byl pro každé i konstantní pro vrchol v. Jednoduše lze ukázat, že výraz $(-A_{Ri} A_{Ti}^{-1})$ je roven vektoru A_{Ti}. A_{Ti} odpovídá pozici počátku kosti i v referenční póze v souřadné soustavě modelu. Uvedený požadavek tedy odpovídá podmínce, aby vrchol v byl ovlivňován jen kostmi, které začínají ve stejném bodě – kloubu. Tyto kosti jsou sourozenci v hierarchii skeletonu. Z omezení ještě můžeme vyloučit kost, která v kloubu končí – je tedy předkem kostí, které v kloubu začínají. Rotace všech těchto kostí budou interpolovány. Výsledný vzorec (již za pomocí kvaternionů) pro pozici vrcholu a normálu lze zapsat takto[1]:

$$
\begin{aligned}
v' &= \frac{\sum_{i=1}^{n} w_i \,\mathrm{Quat}({B_{Ri}} {A_{Ri}}^{-1})}{\left\| \sum_{i=1}^{n} w_i \,\mathrm{Quat}({B_{Ri}} {A_{Ri}}^{-1}) \right\|} (v - A_{Tv}) + B_{Tv} \\
n' &= \frac{\sum_{i=1}^{n} w_i \,\mathrm{Quat}({B_{Ri}} {A_{Ri}}^{-1})}{\left\| \sum_{i=1}^{n} w_i \,\mathrm{Quat}({B_{Ri}} {A_{Ri}}^{-1}) \right\|} (n)
\end{aligned}
\tag{5.1}
$$

Po odečtení pozice kloubu je vrchol v otočen interpolovanou rotací. B_{Tv} představuje novou pozici kloubu v určité konfiguraci skeletonu. Normála je pouze otočena interpolovanou rotací. Pro kombinaci rotací je také nutný správný výběr kvaternionů. Kvaternion reprezentující rotaci předka je brán za řídící. Pro ostatní kvaterniony musí platit, že kosinus úhlu, který svírají s řídícím kvaternionem je nezáporný (viz Dodatek A).

Popsané omezení na ovlivňované kosti může být velice nepříjemné. Vertex blending žádné takové omezení nemá a některé modely toho využívají. Metodou SJB nelze zobrazovat velmi flexibilní části modelu, pro které by toto omezení bylo nepřijatelné. Avšak například pro lidské končetiny je tato metoda přípustná.

V době vzniku této práce byla publikována metoda, která se pokouší toto omezení odstranit [18].

[1] Funkce Quat(M) převádí matici M na kvaternion.

5.2.1 Implementace

Pro implementaci budeme předpokládat, že model splňuje podmínku popsanou výše. Implementace SJB je velmi podobná implementaci vertex blendingu. Místo matic však jsou do vertex shader programu předávány rotace kostí v podobě kvaternionů (`WorldQuatArray`). Jejich hodnoty jsou počítány na CPU. Vedle kvaternionů musejí být předávány pozice kloubů B_{Tv}. Ty jsou obsaženy v poli posunutí všech kostí, které je do vertex shader programu předáváno (`WorldTrans`). Hodnoty A_{Tv} jsou pro každý vrchol konstantní po celou dobu animační sekvence. Model je proto předzpracován a od pozice vrcholu v je příslušný vektor A_{Tv} odečten.

Nejdůležitější část vertex shader programu vypadá takto:

```
for (int iBone = 0; iBone < NumBones-1; iBone++)
{
      LastWeight  =  LastWeight + BlendWeightsArray[iBone];
      FinalQuat  +=  WorldQuatArray[IndexArray[iBone]]*
                     BlendWeightsArray[iBone];
}
LastWeight  =  1.0f - LastWeight;
FinalQuat  +=  WorldQuatArray[IndexArray[NumBones-1]]*LastWeight;
FinalQuat   =  normalize(FinalQuat);
FinalMatrix =  QuaternionToMatrix(FinalQuat);
Pos      = mul(i.Pos,FinalMatrix);
Pos     += WorldTrans[IndexArray[0]];
Normal   = mul(i.Normal, FinalMatrix);
```

Postupně jsou sčítány jednotlivé příspěvky kvaternionů. Výsledný kvaternion je pak nutné normalizovat. Potom je kvaternion převeden na rotační matici, kterou je vynásobená předzpracovaná pozice vrcholu. Nakonec je přičtena pozice kloubu – v předzpracování je postaráno o to, aby se jednalo o pozici první kosti, která ovlivňuje vrchol. Při této metodě není nutné normalizovat normálový vektor, protože transformace výslednou interpolovanou rotací zachovává jednotkovou délku vektoru.

Nejsložitějším výpočtem je převod kvaternionu na matici. Jakým způsobem lze převézt kvaternion na matici pomocí 12 instrukcí assembleru je popsáno v [13]. Je použitý stejný postup, avšak přepsán v HLSL.

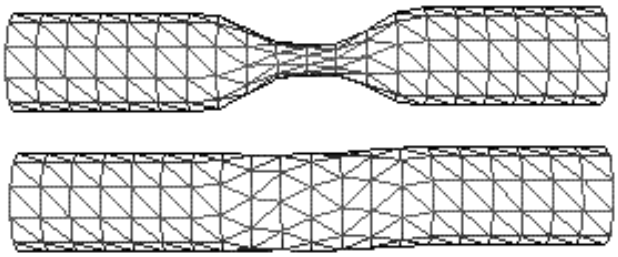
Užití kvaternionů namísto matic také umožňuje použití většího počtu kostí. Každá kost zabírá dva čtyřsložkové vektory – pro posunutí kosti a rotaci reprezentovanou kvaternionem. Maximální možný počet kostí je tedy 128 (zanedbáme-li požadavky ostatních parametrů nesouvisejících se skinningem).

5.2.2 Vizuální výsledky

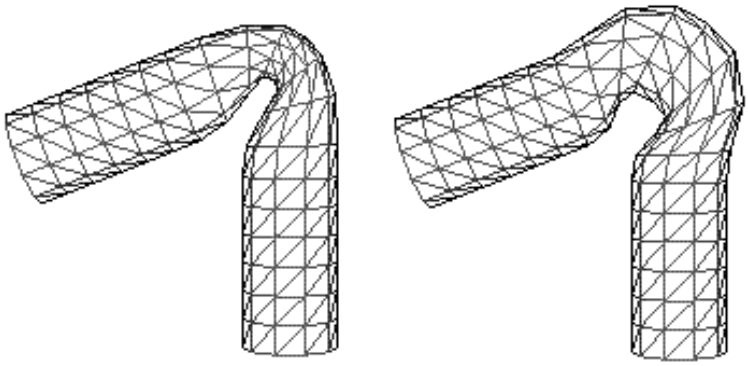
Vizuální vlastnosti SJB lze ukázat na jednoduchém modelu válce se dvěma kostmi. Rotaci okolo vlastní osy kosti lze vidět na obr. 15. Jak již bylo řečeno, SJB se v tomto případě chová velmi pěkně.

Poněkud horší výsledky dává SJB při ohybu. Válec sice zachovává svůj objem, ale kolem kloubu se vytvoří nepěkné „kolínko“ (obr. 16) . Na obr. 17 lze vidět porovnání vertex blendingu s metodou SJB v určité póze lidské paže. V tomto případě chování v kloubu není tak výrazné. Z obrázku je patrné, že SJB dává daleko lepší výsledky.

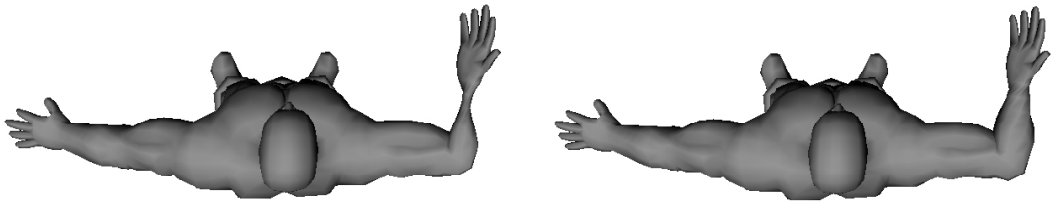
obr. 15: Porovnání vertex blendingu a SJB při rotaci okolo osy kosti.



obr. 16: Porovnání vertex blendingu a SJB při ohybu.



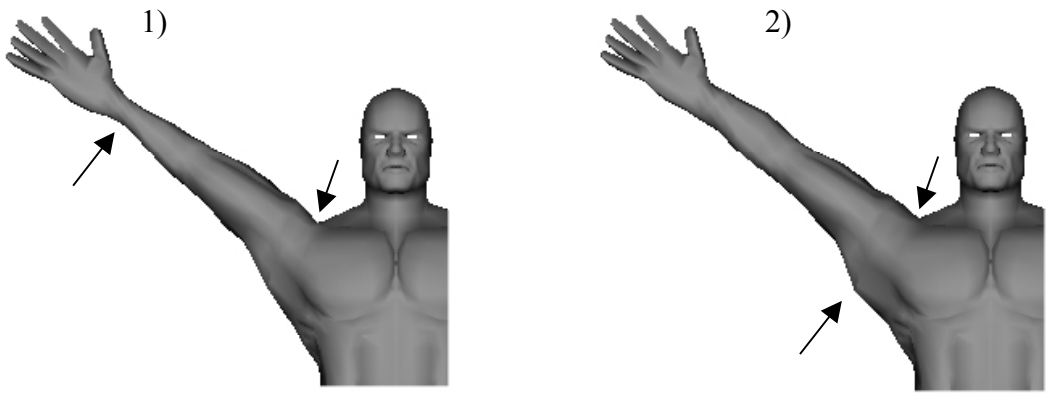
obr. 17: Ukázka vertex blendingu a SJB na reálném modelu.



5.3 Rozšíření metody SJB – spherical joint blending by example

I když dává metoda SJB zajímavé výsledky, nedá se očekávat že pomocí ní lze dobře napodobit chování všech kloubů tak, jak je potřeba. Například velmi komplikovaným kloubem je rameno. Některé problémy tohoto kloubu lze vidět na obr. 18. V tomto směru se zdají být ideální techniky využívající interpolaci vzorových póz, neboť tak lze vytvořit přesný vzhled modelu v jisté konfiguraci skeletonu. Právě o vzorové pózy bude metoda SJB rozšířena. Výslednou metodu lze nazvat *spherical joint blending by example* (SJBBE). Idea této metody bude stejná jako u SBE či eigenskin – vzorové pózy budou potřeba jen pro ty polohy kostí, kdy SJB dává nedostačující výsledky.

obr. 18: Problémy s ramenem a rotací předloktí 1) v jisté póze za použití vertex blendingu, 2) ve stejné póze za použití metody SJB. Šipkami jsou označeny nepěkné deformace metod.

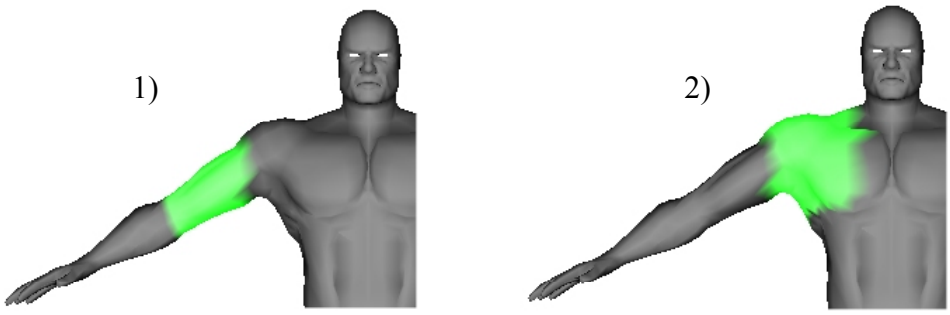


5.3.1 Vzorové pózy

V nové metodě se omezíme jen na případy, kdy z kloubu vychází jen jedna kost. Vzorové pózy budou modelovány jen pro ty případy, ve kterých se konfigurace skeletonu liší od konfigurace skeletonu v referenční póze transformací jedné kosti.

Dále je nutné omezit množinu vrcholů, kterou bude možno ovlivňovat vzorovými pózami pro určitou kost *k* (stejně jako v metodě eigenskin bude používán termín support kosti *k*). Dá se předpokládat, že opravy ve vzorových pózách pro kost *k* vzhledem k výsledkům SJB budou jen lokální v blízkosti kloubu. Kromě toho, pokud je ohýbáno například předloktí, dochází k pohybu svalů v nadloktí. Zvolíme-li support kosti *k* jako množinu vrcholů, které jsou ovlivňovány předchůdcem kosti *k* (ale už ne předchůdcem předka kosti *k*), budeme moci modelovat i stahy svalů při ohybu kostí. Takto zvolený support také zaručuje prázdný průnik jednotlivých supportů různých kostí. Support pro kost předloktí a nadloktí lze spatřit na obr. 19.

obr. 19: Zeleně zvýrazněný support 1) předloktí, 2) nadloktí.



Podobně jako v metodě SBE promítneme pozice vrcholů ve vzorové póze do referenční pózy. Tím získáme vektory posunutí jednotlivých vrcholů.

Úpravou rovnice (5.1) transformujeme pozici vrcholu v^p ze vzorové pózy p do pozice v_0^p v konfiguraci skeletonu v referenční póze:

$$v_0^p = \left(\frac{\sum_{i=1}^{n} w_i \, \mathrm{Quat}(B_{Ri} {A_{Ri}}^{-1})}{\left\| \sum_{i=1}^{n} w_i \, \mathrm{Quat}(B_{Ri} {A_{Ri}}^{-1}) \right\|} \right)^{-1} (v^p - B_{Tv}) + A_{Tv}$$

Vypočteme posunutí v_{ofset}^p vzhledem k původní pozici vrcholu v v referenční póze:

$$v_{ofset}^p = v_0^p - v$$

Výsledná pozice vrcholu v' je rovna transformaci pozice vrcholu v s přičtenými příspěvky posunutí různých vzorových póz. Váhy příspěvků α_i jsou závislé na pozici kosti, jejíž support obsahuje vrchol v. Jakým způsobem jsou váhy α_i počítány popisuje odstavec 5.3.2.

$$v' = \frac{\sum_{i=1}^{n} w_i \, \mathrm{Quat}(B_{Ri} {A_{Ri}}^{-1})}{\left\| \sum_{i=1}^{n} w_i \, \mathrm{Quat}(B_{Ri} {A_{Ri}}^{-1}) \right\|} \left(v - A_{Tv} + \sum_{i=1}^{m} \alpha_i v_{ofset}^i \right) + B_{Tv}$$

Podobným způsobem dospějeme ke vzorci pro výpočet normály:

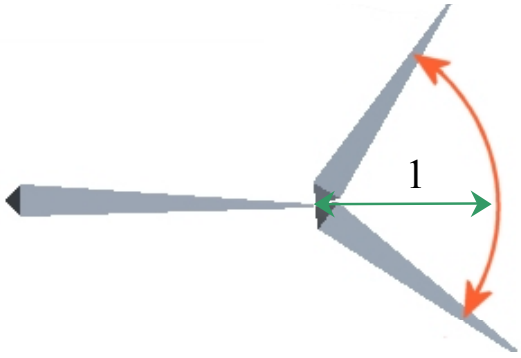
$$n' = \frac{\sum_{i=1}^{n} w_i \, \mathrm{Quat}(B_{Ri} {A_{Ri}}^{-1})}{\left\| \sum_{i=1}^{n} w_i \, \mathrm{Quat}(B_{Ri} {A_{Ri}}^{-1}) \right\|} \left(n + \sum_{i=1}^{m} \alpha_i n_{ofset}^{i} \right)$$

5.3.2 Interpolace

Přirozený požadavek na vektor α je, aby v referenční póze byly všechny jeho složky nulové. Stejně tak, jestliže se kost k nachází ve vzorové póze p, pak α_p musí být rovno 1, ostatní složky zůstanou nulové. V ostatních pózách je třeba hodnoty α_i interpolovat. Stejně jako v metodě eigenskin byla zvolena interpolace užívající radiální bázové funkce (viz Dodatek B). Za bázové funkce jsou zvoleny Gaussovy křivky s konstantním rozptylem. Váhy nutné pro interpolaci pomocí radiálních bázových funkcí jsou vypočítány z popsaných požadavků na vektor α.

Pro výpočet hodnot α_i je nutné zvolit metriku mezi pózami kosti. V metodě eigenskin je metrika definovaná jako velikost úhlu v reprezentaci rotace osa-úhel. V našem případě bude definována jako vzdálenost pozic kostí normalizované délky po sféře (obr. 20). Myšlenka je založena na faktu, že SJB dává dobré výsledky pro rotaci okolo osy kosti. Proto jsou rozlišovány pózy jen pro ohyby kostí.

obr. 20: Vzdálenost dvou póz jedné kosti použitá pro interpolaci je dána velikostí oblouku mezi jednotlivými pozicemi kosti o poloměru 1.



5.3.3 Implementace

Metoda SJBBE používá odlišnou strukturu vrcholu narozdíl od SJB. Do vertex shader programu je nutné předávat posunutí pozic vrcholů a normál pro jednotlivé vzorové pózy. Protože je velikost struktury vrcholu omezena na 16 čtyřsložkových vektorů, je maximální počet vzorových póz na jednu kost roven šesti – pokud model nepoužívá texturu nebo barvu pro každý vrchol. V naší implementaci se omezíme na čtyři vzorové pózy. Struktura vrcholu vypadá takto:

```
struct VERTEX
{
      D3DXVECTOR3   position;
      FLOAT         blend1, blend2, blend3;
      DWORD         indices;
      D3DXVECTOR3   normal;
      D3DXVECTOR4   position1;
      D3DXVECTOR3   position2;
      D3DXVECTOR3   position3;
      D3DXVECTOR3   position4;
      D3DXVECTOR3   normal1;
      D3DXVECTOR3   normal2;
      D3DXVECTOR3   normal3;
      D3DXVECTOR3   normal4;
};
```

Čtvrtá složka posunutí pro první vzorovou pózu je rezervována pro index kosti, jejíž support obsahuje tento vrchol. Číslo kosti je použito pro indexaci do pole vah póz, reprezentující vektor α pro každou kost. Hodnoty vah α jsou počítány na CPU a předávány do vertex shader programu (`ExampleWeights`).

Základ vertex shader programu vypadá takto:

```
iBoneEx = (int) i.Pos1.w;
for (int iBone = 0; iBone < NumBones-1; iBone++)
{
      LastWeight  =  LastWeight + BlendWeightsArray[iBone];
      FinalQuat  +=  WorldQuatArray[IndexArray[iBone]]*
                     BlendWeightsArray[iBone];
}
LastWeight  =  1.0f - LastWeight;
FinalQuat  +=  WorldQuatArray[IndexArray[NumBones-1]]*LastWeight;
FinalQuat   =  normalize(FinalQuat);
FinalMatrix =  QuaternionToMatrix(FinalQuat);

i.Pos.xyz += i.Pos1.xyz*ExampleWeights[iBoneEx].x;
i.Pos.xyz += i.Pos2.xyz*ExampleWeights[iBoneEx].y;
i.Pos.xyz += i.Pos3.xyz*ExampleWeights[iBoneEx].z;
i.Pos.xyz += i.Pos4.xyz*ExampleWeights[iBoneEx].w;

i.Normal.xyz += i.Norm1.xyz*ExampleWeights[iBoneEx].x;
i.Normal.xyz += i.Norm2.xyz*ExampleWeights[iBoneEx].y;
i.Normal.xyz += i.Norm3.xyz*ExampleWeights[iBoneEx].z;
i.Normal.xyz += i.Norm4.xyz*ExampleWeights[iBoneEx].w;

Pos  = mul(i.Pos,FinalMatrix);
Pos += WorldTrans[IndexArray[0]];
Normal  = mul(i.Normal, FinalMatrix);
Normal  = normalize(Normal);
```

Postupně se sčítají příspěvky jednotlivých vzorových póz, výsledná pozice a normála jsou transformovány ve stejném smyslu jako u metody SJB. V tomto případě je již nutné výslednou normálu znormalizovat na jednotkový vektor.

Náročnost na počet konstantních registrů pro vertex shader je u této metody stejná jako u implementace vertex blendingu. Jsou nutné tři registry na jednu kost – pro kvaternion, posunutí kosti a váhy jednotlivých vzorových póz.

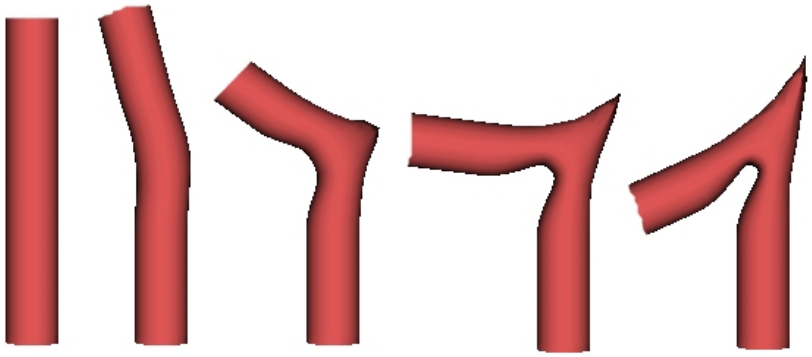
5.3.4 Vizuální výsledky

Výsledky budou nejdříve představeny na „nereálném“ příkladu válce. Chceme-li, aby se deformace válce při ohybu chovala odlišně od SJB, stačí vytvořit vzorovou pózu. Při ohybu se poté bude hladce přecházet mezi referenční pózou a vzorovou pózou. Vzorovou pózu a několik interpolovaných póz lze nahlédnout na obr. 21.

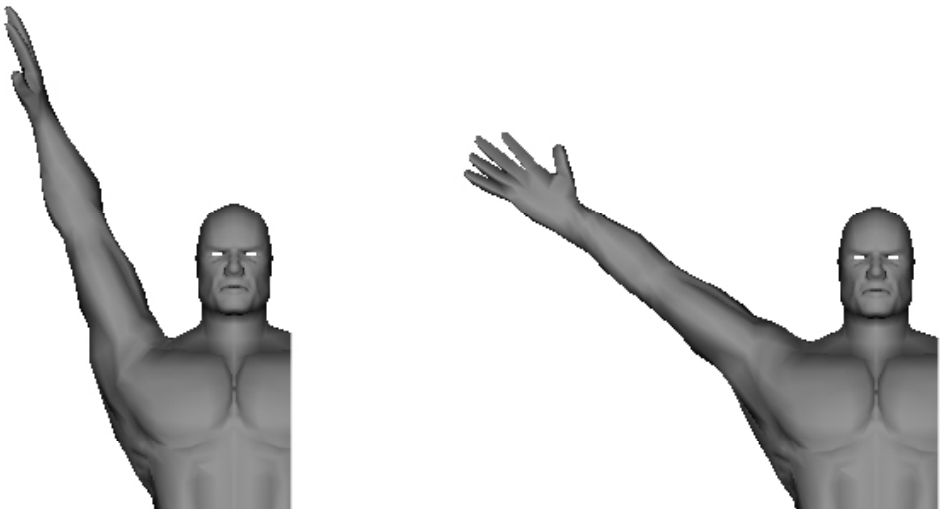
Pomocí metody SJBBE můžeme vyřešit problém s ramenním kloubem patrným na obr. 18. V limitní pozici kloubu je vytvořena vzorová póza. Při animaci vzpažení budou referenční póza a vzorová póza interpolovány. Vzorová póza je spolu s výslednou interpolovanou pózou zobrazena na obr. 22. Na rotaci předloktí je také patrná výhoda interpolace rotací převzatá z metody SJB.

Stejně tak můžeme například vytvořit vzorovou pózu pro ohnutou paži v loketním kloubu. V této póze můžeme vymodelovat zatnutý biceps. Při animaci pak bude vznikat dojem zatínání svalů.

obr. 21: Postupný přechod z referenční pózy do vzorové pózy.



obr. 22: 1) Vzorová póza pro kost nadloktí, 2) stejná póza jako na obr. 18 za použití metody SJBBE.



5.4 Porovnání

Vizuální výsledky metod SJB a SJBBE již byly představeny. Nyní budou tyto metody porovnány vzhledem k rychlosti. Testy byly vykonány se stejnými podmínkami jako v odstavci 3.4.

	VSP1	VSP2	VSP3	SJB	SJBBE
GeForce FX 5700	15	27	40	36	24
Radeon 9800 PRO	35	54	72	50	33
GeForce FX 6800	34	73	99	65	46

V tabulce jsou pro porovnání uvedeny také snímkové frekvence jednotlivých implementací vertex blendingu používajících vertex shader.

Implementace metod SJB a SJBBE v rychlosti zaostávají za nejrychlejší implementací vertex blendingu VSP3. Metoda SJBBE je přibližně o polovinu pomalejší. Je však nutné podotknout, že se jedná o test, který měří jen rychlost zpracování vrcholů. V dnešní době úzké hrdlo aplikací ve zpracování vrcholů nebývá. Dá se proto předpokládat, že případné nasazení v „reálné“ aplikaci s několika modely využívajících metodu SJB či SJBBE nebude tolik ovlivňovat snímkovou frekvenci.

Kapitola 6. Závěr

Programovatelné vertex shader jednotky grafických karet nabízejí nové způsoby zpracování vrcholů modelu. Lze pomocí nich implementovat jak metodu vertex blending, tak i nové techniky skinningu bez zbytečného zatížení CPU.

V této práci byla popsána implementace skeletální animace a analyzovány různé způsoby implementace metody vertex blending pro dnešní konzumní grafické karty pomocí knihovny DirectX 9.0, a to jak za použití klasického zpracování vrcholů ve standardním grafickém vykreslovacím řetězci, tak s použitím programovatelných vertex shader jednotek. Implementace využívající vertex shader jednotky vycházely z příkladu v [6]. Vertex shader program byl jednoduchou myšlenkou výrazným způsobem optimalizován (odstavec 3.3.4).

Metoda vertex blending má však určité nedostatky, které jsou příčinou hledání nových technik k řešení problému skinningu. Proto byly představeny některé nové metody (odstavec 4.2) se zaměřením na metody využívající vzorové pózy a metody, které používají interpolaci rotací. Pro implementaci byla vybrána metoda SJB, která řeší tzv. „candywrapper“ problém typický pro vertex blending (odstavec 5.2). Tato metoda byla rozšířena o vzorové pózy podobným způsobem, jakým metoda SBE rozšiřuje vertex blending. Výsledná metoda SJBBE těží z výhod jak interpolace transformací, tak použití vzorových póz (odstavec 5.3).

Metoda SJBBE byla představena jen na kloubu, ze kterého vychází jen jedna kost. Možnosti rozšíření této metody na složitější klouby nebyly studovány. Stejně tak nebyl ve větší míře studován vliv volby rozptylu u Gaussových funkcí na vizuální výsledky. Možné rozšíření a vliv rozptylu na kvalitu výsledků mohou být předmětem budoucího výzkumu.

Ačkoliv jsou vizuální výsledky metod SJB a SJBBE zajímavé, nedá se očekávat, že by podobné metody v blízké době vytlačily klasický vertex blending v komerčních projektech. Jsou pro to dva hlavní důvody. První důvod spočívá v omezení kladeném na model (odstavec 5.2). Je velmi obtížné vytvořit celý animovaný model lidské postavy, který by tuto podmínku splňoval. Druhý důvod spočívá v masivním použití techniky vertex blending. Ta má sice své chyby, ale je již řadu let prověřena užitím v mnoha komerčních počítačových hrách.

Kombinace skeletální animace s per vertex animací ve smyslu použití vzorových póz má však v sobě veliký potenciál, neboť využívá výhody obou přístupů. I tato práce je důkazem toho, že lze spojení obou metod s jistým omezením implementovat za použití vertex shader jednotek, tedy s malým zatížením CPU.

Dodatek A. Kvaterniony

Kvaternion je čtyřsložková veličina, kterou lze chápat jako rozšíření komplexních čísel.

$$q = s + xi + yj + zk = (s, \mathbf{v})$$

Pro prvky i, j, k platí následující vztahy: $i^2 = j^2 = k^2 = ijk = -1$, $ij = k$, $ji = -k$. Pomocí těchto vztahů lze odvodit pravidla pro sčítání a násobení kvaternionů:

$$q + q' = \left(s + s', \mathbf{v} + \mathbf{v}'\right)$$
$$qq' = \left(ss' - \mathbf{v}.\mathbf{v}', \mathbf{v} \times \mathbf{v}' + s\mathbf{v}' + s'\mathbf{v}\right)$$

Konjugovaný kvaternion ke kvaternionu $q = (s, \mathbf{v})$ je $\overline{q} = (s, -\mathbf{v})$. Absolutní hodnota (velikost) kvaternionu $q = (s, \mathbf{v})$ je definována jako $\|q\| = \sqrt{s^2 + x^2 + y^2 + z^2}$. Inverzní kvaternion ke kvaternionu q je definován jako $q^{-1} = \dfrac{\overline{q}}{\|q\|}$.

Kvaternion $q = (s, \mathbf{v})$, pro který platí, že $s = 0$, reprezentuje vektor $\mathbf{v} \in R^3$. Kvaternion, jehož velikost je rovna 1, určuje jednoznačně rotaci v prostoru. Jestliže $\|q\| = 1$, pak existuje $\theta \in \langle 0, 2\pi)$ a $\mathbf{n} \in R^3, \|\mathbf{n}\| = 1$, že platí $q = \left(\cos\dfrac{\theta}{2}, \mathbf{n}\sin\dfrac{\theta}{2}\right)$. Definujme operátor $\mathrm{R}_q(p) = qpq^{-1}$, kde kvaternion $p = (0, \mathbf{r})$. Výsledkem tohoto operátoru je kvaternion $\left(0, \mathbf{r}'\right)$. Vektor $\mathbf{r}'$ reprezentuje vektor $\mathbf{r}$ otočený o úhel θ okolo osy rotace $\mathbf{n}$.

$$\mathrm{R}_q(p) = \left(0, (s^2 - \mathbf{v}.\mathbf{v})\mathbf{r} + 2\mathbf{v}(\mathbf{v}.\mathbf{r}) + 2s(\mathbf{v} \times \mathbf{r})\right)$$
$$\mathrm{R}_q(p) = (0, \mathbf{r}\cos\theta + (1 - \cos\theta)\mathbf{n}(\mathbf{n}.\mathbf{r}) + (\mathbf{n} \times \mathbf{r})\sin\theta)$$

Správnost výpočtu vektoru $\mathbf{r}'$ lze dokázat pomocí přímého odvození vzorce pro rotaci okolo osy $\mathbf{n}$ o úhel θ [2]. Všechny kvaterniony, které reprezentují rotace tvoří povrch koule ve čtyřrozměrném prostoru. Ze vzorce operátoru $\mathrm{R}_q(p)$ také vyplývá, že kvaterniony q a $-q$ reprezentují stejnou rotaci.

Pro zjednodušení zápisu je rotace vektoru $\mathbf{r}$ pomocí kvaternionu q v této práci zapisována jako $q(\mathbf{r})$.

V praxi se většinou kvaternion převádí na matici, kterou je potřebný vektor $\mathbf{r}$ transformován. Důvod je ten, že převod kvaternionu na matici a transformace touto maticí dává menší počet operací, než přímá transformace pomocí kvaternionu. Převod kvaternionu na matici lze nalézt například v [2], [14]. Proč jsou tedy kvaterniony používány? Rotaci reprezentují menším počtem prvků než matice. Také skládání rotací je s pomocí kvaternionů rychlejší. Hlavní důvod však spočívá v jednoduchosti interpolace rotací.

Interpolace rotací

Interpolaci mezi dvěma rotacemi R_0 a R_1 lze vyjádřit obecně bez konkrétní reprezentace jako $\mathrm{R}_t = \mathrm{R}_0(\mathrm{R}_0{}^{-1}\mathrm{R}_1)^t$, kde $t \in \langle 0,1 \rangle$. Používáme-li jako reprezentaci rotací matice, je problém v operaci umocňování reálným číslem t. Proto se v tomto případě matice $\mathrm{R}_0{}^{-1}\mathrm{R}_1$ převádí na reprezentaci osa-úhel, na které je operace umocňování provedena (je převedena na násobení úhlu rotace), a výsledek je nazpátek převeden na matici [14].

Používáme-li pro reprezentaci rotací kvaterniony, je postup interpolace jednodušší. Jak již bylo řečeno, kvaterniony jsou rozšířením komplexních čísel. Stejně jako komplexní číslo, lze jednotkový kvaternion $q = (s, \mathbf{v})$ vyjádřit v exponenciálním tvaru:

$$q = e^{\mathbf{v}\theta}$$

Interpolaci rotací pomocí kvaternionů lze vyjádřit jako $q_0(q_0{}^{-1}q_1)^t$. Definujeme-li $p = q_0{}^{-1}q_1$, pak lze interpolaci vyjádřit jako

$$q_t = q_0 p^t = q_0 e^{\mathbf{v}t\varphi} = q_0(\cos t\varphi + \mathbf{v}\sin t\varphi)$$

Tento výraz lze dále upravit. Dá se dokázat [12], že

$$q_t = q_0 \frac{\sin(1-t)\varphi + p\sin t\varphi}{\sin\varphi}$$

Dosazením za p dostaneme výsledný vzorec sférické lineární interpolace (SLERP).

$$q_t = q_0 \frac{\sin(1-t)\varphi}{\sin\varphi} + q_1 \frac{\sin t\varphi}{\sin\varphi}$$

SLERP vytyčuje oblouk na 4D kouli mezi vektory q_0 a q_1. Výpočtem reálné části kvaternionu p vynásobením kvaternionů ${q_0}^{-1}$ a q_1 lze ukázat, že kosinus úhlu φ je roven skalárnímu součinu vektorů q_0 a q_1 v R^4.

Jak již bylo řečeno, konkrétní rotace je reprezentována dvěma kvaterniony q a $-q$. Při interpolaci SLERP hraje roli, s jakými kvaterniony budeme počítat. Volbou kvaternionů q_0 a q_1 je jednoznačně určen kosinus úhlu φ, který svírají. Jestliže jsou zvoleny kvaterniony q_0 a q_1 tak, že $\cos\varphi < 0$, pak je rotace kolem osy, která je určena kvaternionem, interpolována po delším oblouku. Většinou je však požadováno, aby interpolace probíhala po kratším oblouku. Tento požadavek je zaručen, pokud $\cos\varphi \geq 0$. Jestliže tedy platí, že $\cos\varphi < 0$, stačí nahradit kvaternion q_1 kvaternionem $-q_1$.

Interpolace SLERP je však celkem náročná na výpočet (obzvlášť má-li se provádět ve vertex shader programu). Proto je často nahrazována jen aproximací. Ta spočívá v lineární interpolaci kvaternionů a následné normalizaci:

$$q_t = \frac{q_0\,(1-t) + q_1 t}{\|q_0\,(1-t) + q_1 t\|}$$

Tato aproximace vytyčuje stejnou křivku v R^4 jako SLERP, avšak nemá konstantní krok. Práce [18] ukazuje, že horní chyba aproximace činí přibližně 8 úhlových stupňů.

Aproximaci SLERP lze velmi jednoduše rozšířit i pro více kvaternionů (rozšíření je potřebné pro popsané metody skinningu SJB a SJBBE):

$$p = \frac{\sum_{i=1}^{n} w_i q_i}{\left\| \sum_{i=1}^{n} w_i q_i \right\|}$$

$$\sum_{i=1}^{n} w_i = 1,\ w_i \geq 0$$

Dodatek B. Interpolace pomocí radiálních bázových funkcí

Interpolace využívající radiální bázové funkce se používá na rozptýlených datech. Mějme definovaných N bodů $[x_i, f_i]$, kde $x_i \in R^d$, $x_j \neq x_k$ pro $j \neq k$, $f_i \in R$. Potřebujeme definovat spojitou funkci $s(x): R^d \to R$, pro kterou bude platit $s(x_i) = f_i$. Funkce $s(x)$ je volena ve tvaru

$$s(x) = \sum_{i=1}^{N} \lambda_i \, \mathrm{g}(x - x_i)$$

kde

- λ_i jsou reálná čísla – váhy,
- $\mathrm{g} : R^d \to R$ je radiálně symetrická funkce, neboli existuje $\phi : \langle 0, \infty) \to R$ tak, že $\phi(|x - x_i|) = \mathrm{g}(x - x_i)$.

Jako ϕ je často volena Gaussova křivka $\phi(r) = e^{\frac{-r^2}{\sigma^2}}$. Neznámé váhy λ_i jsou vypočítány z daných bodů $[x_i, f_i]$. Nechť $\mathbf{f} = (f_1, ..., f_N)^{\mathrm{T}}$, $\mathbf{l} = (\lambda_1, ..., \lambda_N)^{\mathrm{T}}$, $\mathbf{G} = \mathbf{G}_{ij} = \phi(|x_i - x_j|)$ pro $i, j = 1, ...N$, potom

$$\mathbf{f} = \mathbf{G}\mathbf{l}$$
$$\mathbf{l} = \mathbf{G}^{-1}\mathbf{f}$$

Použití Gaussových funkcí zaručuje existenci $\mathbf{G}^{-1}$. Bližší informace o interpolaci pomocí radiálních bázových funkcí a další volby funkce ϕ lze najít například v [19].

Pro metodu SJBBE představuje x_i normalizovanou osu kosti v souřadné soustavě kosti. Vzdálenost je definována jako délka oblouku o jednotkovém poloměru mezi dvěma pozicemi kosti x_i a x_j. Hodnota rozptylu je zvolena $\sigma = 1$. Váhy jednotlivých póz α_i odpovídají funkci $s(x)$. Výpočty složek vektoru α jsou brány jako jednotlivé interpolační problémy.

Dodatek C. Uživatelská dokumentace

Experimentální program má podobu jednoduchého prohlížeče modelu, ve kterém lze přepínat mezi jednotlivými metodami skinningu. Byly implementovány tyto metody: SW, FFP1, FFP2, VSP1, VSP2, VSP3, SJB, SJBBE.

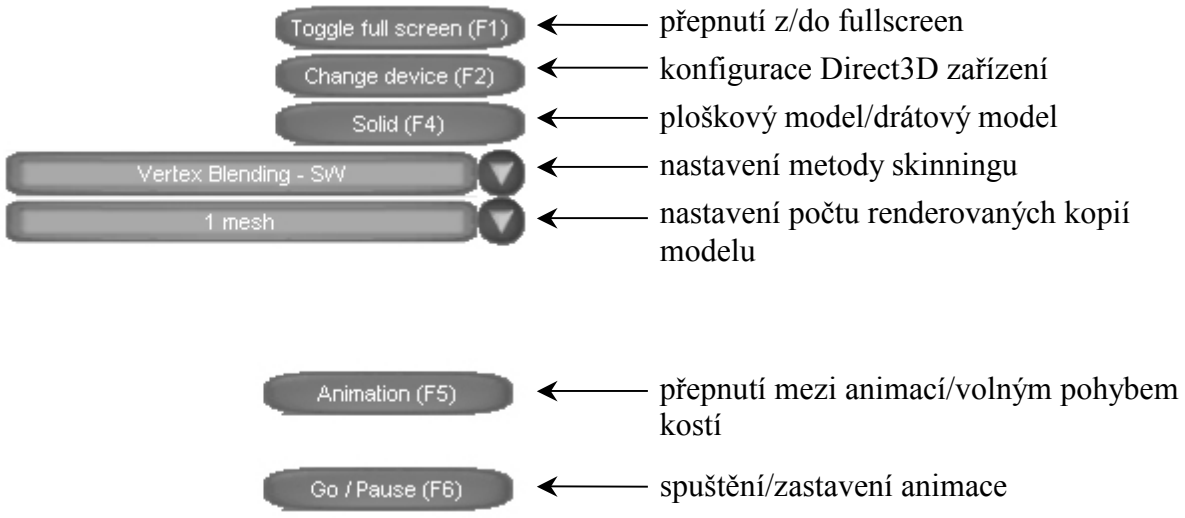
Pro správný chod experimentálního programu je vyžadována grafická karta, která podporuje vertex shader specifikaci 2.0. Je potřeba mít nainstalovánu knihovnu DirectX 9.0c, která se nachází na přiloženém CD.

Spustitelný soubor aplikace `SkinnedMesh.exe` přebírá na příkazové řádce parametry. Ty určují, jaký model má být zobrazen, zda má být model optimalizován a také barvu pozadí. Pro usnadnění je připraveno několik dávkových souborů:

`run_man.bat`	- spustí aplikaci s modelem lidské postavy
`run_cylinder.bat`	- spustí aplikaci s modelem válce
`run_tests.bat`	- spustí aplikaci s modelem lidské postavy pro účely měření rychlosti

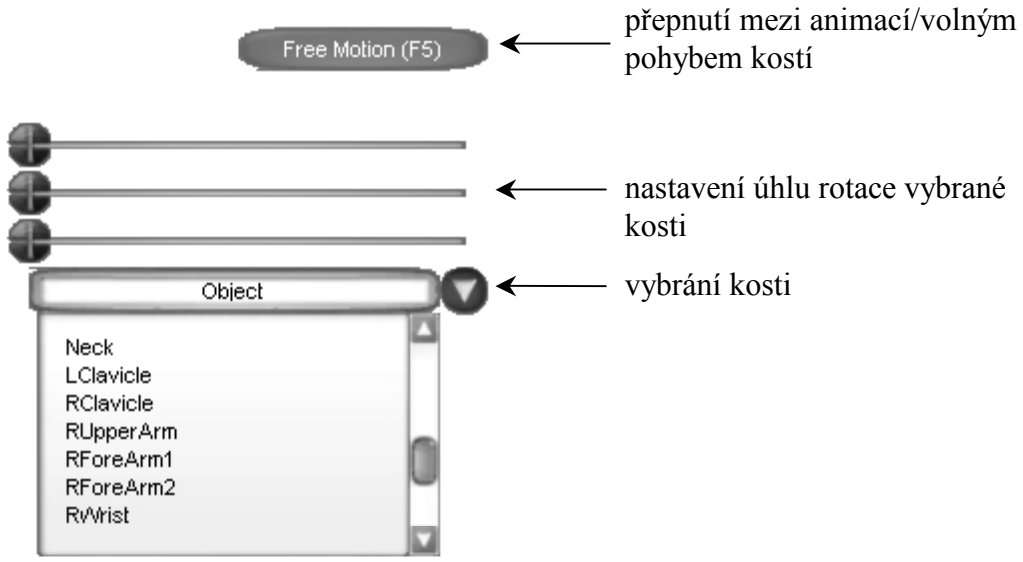
V jediném okně aplikace je zobrazen model. Pomocí levého tlačítka myši lze modelem rotovat, pravým posouvat a prostředním přibližovat. Kromě modelu lze spatřit v okně aplikace základní nabídku, jejíž tlačítka jsou popsána na obr. 23.

obr. 23: Základní nabídka.



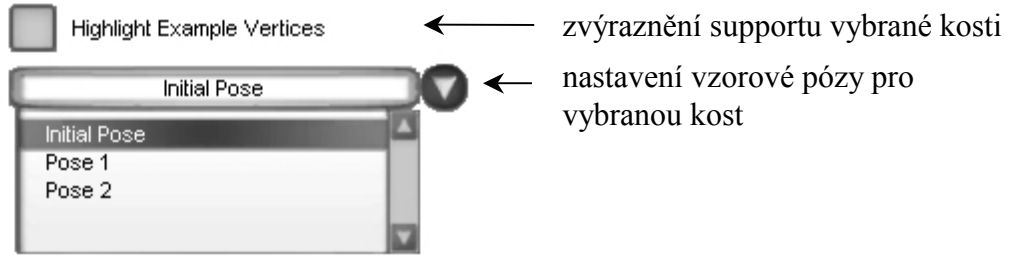
Stisknutím klávesy F5 nebo zmáčknutím příslušného tlačítka je otevřena nabídka pro volný pohyb s kostmi. Popis jednotlivých uživatelských prvků této nabídky je na obr. 24.

obr. 24: Nabídka pro volný pohyb kostí.



Jestliže je vybrána metoda SJBBE, je nabídka pro volný pohyb kostí rozšířena o speciální funkce. Lze zvýraznit support vybrané kosti nebo nastavit vzorovou pózu pro vybranou kost (obr. 25).

obr. 25: Speciální nabídka pro metodu SJBBE.



Aplikaci lze ukončit klávesou ESC.

S aplikací jsou také dodány modely, na kterých byly metody skinningu testovány (hlavně SJBBE). Model válce má vzorové pózy modelovány pro kosti `Bone02` a `Bone03`. Model lidské postavy má vzorové pózy vymodelovány pro kosti `LUpperArm`, které řeší některé problémy s ramenem, a `LForeArm1`, jež má ve vzorové póze zatnutý biceps.

Dodatek D. Základní popis programu

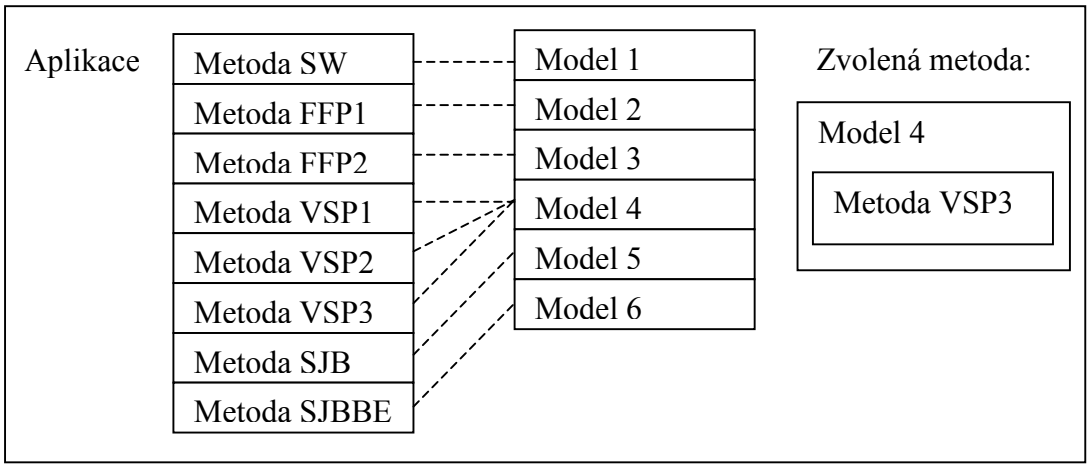
Program je napsán v Microsoft Visual C++ .NET za použití knihovny Microsoft DirectX 9.0c (aktualizace duben 2005).

Základní třídou aplikace je `CApplication`. Tato třída obsahuje prvky uživatelského rozhraní. Pro reprezentaci prvků uživatelského rozhraní je použit framework demo programů z [6]. Základní třída aplikace zpracovává události vyvolané stisknutím prvku uživatelského rozhraní, události klávesnice a události myši. V základní třídě aplikace jsou registrovány objekty modelů a objekty reprezentující metody skinningu. Hlavními metodami třídy aplikace jsou `OnFrameMove` a `OnFrameRender`. Metoda `OnFrameMove` přepočítá reprezentaci scény podle aktuálního času (v našem případě nastaví konfiguraci skeletonu podle času a animační sekvence). Metoda `OnFrameRender` model spolu s prvky uživatelského rozhraní vykreslí. Tyto dvě metody jsou volány v hlavní smyčce aplikace.

Základní třída aplikace komunikuje s objektem modelu přes rozhraní `CVirtualMesh`. Z třídy `CVirtualMesh` jsou odvozeny třídy, které reprezentují model pro různé metody skinningu (neboť různé metody skinningu mají na model různé požadavky).

Metody skinningu jsou reprezentovány třídou `CSkinningMethod`. Tato třída zakrývá práci s efekt souborem. Každá instance třídy `CSkinningMethod` (tj. metoda skinningu) ví, jaký objekt reprezentující model má používat. Jestliže uživatel zvolí konkrétní metodu skinningu, je nastaven na vykreslování odpovídající model. Navíc je objekt reprezentující skinning registrován do tohoto objektu modelu – při vykreslování modelu se přes něj nastavují konstantní parametry vertex shader programu typické pro zvolenou metodu skinningu.

obr. 26: Schéma aplikace. Aplikace obsahuje objekty reprezentující metody skinningu a objekty reprezentující modely. Čárkovaně je vyznačeno, jaká metoda skinningu používá jaký model. Model zvolené metody je vykreslován.



Dodatek E. Obsah CD

- **RUN** - přeložený program
- **SOURCE** - zdrojový kód programu
- **VIDEO** - ukázkové video
- **DOC** - programová dokumentace vygenerovaná programem Doxygen
- **DIRECTX9** - instalace DirectX 9.0c runtime
- **readme.txt** - stručné informace o diplomové práci a obsah CD
- **dp.pdf** - text diplomové práce

Seznam literatury

[1] Laura Hayes and John Howard Wileman Exhibit of optical toys, http://courses.ncssm.edu/gallery/collections/toys/opticaltoys.htm

[2] Watt A., Policarpo F. (2003): 3D games: animation and advanced real-time rendering. Addison-Wesley.

[3] Jeppson D. (2000): Realtime character animation blending using weighted skeleton hierarchies. Diplomová práce.

[4] Lewis, J.P. a kol. (2000): Pose space deformation: a unified approach to shape interpolation and skeleton driven deformation. *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co. 165-172.

[5] Ken Turkowski a kol. (1990): Transformation of surface normal vectors with applications to three dimensional computer graphics. Apple Computer, Inc. http://www.worldserver.com/turk/computergraphics/NormalTransformations.pdf

[6] Microsoft DirectX 9.0 SDK Update April 2005. Microsoft Corporation. http://msdn.microsoft.com/directx/

[7] Wloka M. (2003): "Batch, batch, batch:"What does it really mean? GDC 2003 Presentation. http://developer.nvidia.com/docs/IO/8230/BatchBatchBatch.pdf

[8] Dietrich S. (2000) Vertex blending under DirectX 7 for the GeForce 256, http://developer.nvidia.com/object/Vertex_Blending_GeForce_256.html

[9] Weber J. (2000): Run-time skin deformation. *Proceedings of Game Developer Conference*.

[10] Wang C., Philips C. (2002): Multi-weight enveloping: least-squares approximation techniques for skin animation. *Proceeding of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM Press.

[11] Kavan L., Žára J. (2003): Real time skin deformation with bones blending. *WSCG Short Papers Proceedings*.

[12] Kavan. L. (2003): Simulation of fencing in virtual reality. Diplomová práce.

[13] Hejl J. (2004): Hardware skinning with quaternions. *Game Programming Gems 4*. Charles River Media.

[14] Eberly D. (2002): Rotation representation and performance issues. Magic Software, Inc.

[15] Domine S. (2003): Mesh skinning. NVIDIA Corporation. http://developer.nvidia.com/object/skinning.html

[16] Sloan P. a kol. (2001): Shape by example. *Proceedings of the 2001 symposium on Interactive 3D graphics*, ACM Press.

[17] Kry P. G. a kol. (2002): Eigenskin: real time large deformation character skinning in hardware. *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM Press.

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

[19] Baxter, B.J.C. (1992): The interpolation theory of radial basis functions. Disertační práce.