VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

SEMESTRÁLNÍ PROJEKT

Jaroslav Přibyl **Brno 2005**

ZADÁNÍ

Nastudujte si problematiku zobrazování virtuálních scén, 3ds file formát a grafickou knihovnu Open Inventor. Na základě dřívějších zkušeností naimplementujte vlastní knihovnu pro načítání 3ds souborů. Načtená data zobrazte za použití knihovny Open Inventor. Proveďte detailní analýzu načítání modelů a na základě analýzy se pokuste knihovnu vylepšit.

PROHLÁŠENÍ

Čestně prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod odborným vedením Ing. Jana Pečivy, za což bych mu tímto způsobem velice rád poděkoval. Dále prohlašuji, že jsem uvedl všechny literární prameny a publikace, ze kterých jsem čerpal.

ABSTRAKT

Tématem tohoto semestrálního projektu je pokračovat v práci na ročníkovém projektu, ve kterém jsme se zabývali 3ds formátem 3D Studia. Pro načítání informací z 3ds souborů budeme využívat svoji vlastní knihovnu, jejíž vytvoření bylo jedním z úkolů tohoto semestrálního projektu. Pro zobrazování scény budeme používat grafickou knihovnu Coin 3D. Vytvořenou scénu exportujeme do formátu iv Open Inventoru. Současně s vytvářením knihovny budeme také analyzovat některé špatně načítané modely a budeme se snažit tyto modely zpracovat tak, aby byly zobrazeny správně.

ABSTRACT

Subject matter for this semestral project is to continue on project whitch we was working in last year. In this project we examined the 3ds file format and made a convertor from this format to the Open inventor iv file format. We are going to use my own 3ds load library named lib3ds. So the first objective of this semestral project is to create this 3ds load library. We are going to use a Coin3D library to view created scene from 3ds file. If the scene graph is created, we'll export it into the Open Inventor file format iv. When we are developing 3ds load library we deeply analyse 3ds file format structure at the same time and we try to improve the 3ds2iv convertor developed in last year. So this is a second objective for this project. We are expecting better conversion of some 3ds models.

KLÍČOVÁ SLOVA

3D Studio, 3ds file formát, lib3ds, Open Inventor, Coin3D, geometrie, materiály, normály, trojúhelníky, vrcholy, smoothing groups, two side lightning, back face culling, textury, bump mapping, animace, analýza, testování

KEY WORDS

3D Studio, 3ds file format, lib3ds, Open Inventor, Coin3D, geometry, material, normal, triangle, vertex, smoothing group, two side lightning, back face culling, texture, bump map, animation, analysis, testing

Obsah

1 Úvod

Na úvod bych Vám nastínil stručnou charakteristiku řešeného problému v širším kontextu a popsal náplň jednotlivých kapitol.

Renderování virtuálních scén je komplexní problém. Já se budu zabývat pouze určitými pasážemi. Vzhledem k různé obtížnosti jednotlivých částí, bude každé věnován jiný prostor. Prvním úkolem tohoto semestrálního projektu je vytvořit knihovnu pro načítání informací o scéně z 3ds souboru. Druhým úkolem je navázat na zkušenosti získané při řešení ročníkového projektu a pokusit se na základě analýzy špatně načítaných modelů vylepšit převodník (program) z formátu 3ds do formátu iv. Cíl zůstává stejný jako u ročníkového projektu, tedy vytvořit jednoduchý renderovací program, který umožní převod z formátu 3ds 3D Studia do formátu iv Open Inventoru a tento program následně poskytnout odborné veřejnosti.

3D Studio je nástroj přímo určený k modelování a následnému renderování virtuálních scén. Jsme tedy omezeni především vlastním exportním 3ds formátem, který obsahuje omezené množství informací o modelované scéně. S omezeními se setkáváme také u Open Inventoru, který nedisponuje dostatečnými prostředky pro plnohodnotné renderování virtuálních scén. O jaká omezení se jedná se dozvíme dalších kapitolách. Je však třeba podotknout, že renderování virtuálních scén, tak jak to umí 3D Studio, není posláním Open Inventoru a tedy ani knihovny Coin3D. Nakonec není ani naším cílem renderovat reálně vyhlížející scény, nehledě na to, že 3ds formát jako takový má značně omezené možnosti, co se týče ukládání detailních informací o scéně jako celku. Prostředky, které budeme používat, budou pro naše účely plně dostačující, ale s jejich popisem se seznámíme později.

V jednotlivých kapitolách se vyhodnotíme výsledky řešení předchozího ročníkového projektu, které jsou pro nás velmi důležité, a které tvoří základ tohoto semestrálního projektu. Také si stručně popíšeme prostředky realizace a vlastní implementaci programu, a především knihovny pro načítání informací z 3ds souboru (dále jen lib3ds). Důležitý bude také rozbor některých špatně načítaných modelů a detailní analýzy chování 3ds formátu ve vztahu k modelování v 3D Studiu. V závěrečných kapitolách se seznámíme s výsledky testování na různých modelech, s novými poznatky o 3ds formátu a tím, jak se podařilo tyto poznatky uplatnit při zdokonalování převodníku z formátu 3ds do Open Inventoru. Také se dozvíme, jakým způsobem lze pokračovat v práci na projektu a pokusit se tak o povýšení tohoto převodníku na jeden z nejúčinnějších dostupných nástrojů pro analýzu a převod 3ds souborů.

2 Teorie

V této kapitole se seznámíme s informacemi nezbytnými k tomu, abychom si udělali představu, co je potřeba znát pro úspěšné řešení projektu a nastíníme některé teoretické základy z oblasti 3D grafiky, použité při implementaci. Součástí této kapitoly bude rovněž stručný popis formátu, organizace 3ds souborů a způsob práce s tímto formátem.

2.1 Renderování virtuálních scén

Renderování virtuálních scén je obecně velice široký pojem a lze na něj nahlížet několika způsoby. Především je nutné si uvědomit, pro jakou aplikaci bude renderovaná scéna použita. Výstup renderovacího programu bude například jiný pro fotorealistickou fotografii, kde je kladen velký důraz na realističnost scény, na rozdíl od her, kde klademe důraz spíše na rychlost vykreslování (FPS).

Renderování virtuální scény se skládá z několika důležitých kroků. Jejich stručný popis bude obsahem následujících odstavců.

2.2 Geometrie

V první řadě je třeba vytvořit geometrii celé scény. Existuje množství modelovacích technik, které si zde nebudeme popisovat, ale každá z nich se vyznačuje specifickými vlastnostmi a hodí se pro modelování typických objektů scény. Například pro modelování stolu použijeme tzv. box modeling (modelování vychází ze základního tvaru, kterým je kvádr). Pro modelování sklenice pak zase využijeme spline křivky. Správný výběr modelovací techniky je důležitý jednak z hlediska času potřebného k vlastnímu vymodelování požadovaného tvaru, ale také z hlediska konečného počtu trojúhelníků. Se vzrůstajícím počtem trojúhelníků roste náročnost vykreslení scény a omezuje se tak použití jejího modelu. V praxi jsme tedy často nuceni volit kompromis mezi složitostí modelované scény (objektu) a rychlostí vykreslování. Existuje také řada programových prostředků (LOD, redukce počtu trojúhleníků), které dokáží urychlit vykreslování scény.

2.2.1 Základní geometrie objektů

Scéna je složená z mnoha objektů, které byly samostatně vymodelovány. Každý objekt je složen ze sítě trojúhelníků, která určuje jeho tvar (povrch). Takovému objektu složenému z trojúhleníků říkáme mesh. Existují však metody, pomocí kterých lze dosáhnout změny vzhledu objektu, aniž bychom měnili jeho základní geometrii. Mezi tyto techniky patří

například použití různých typů stínování (mesh smooth) nebo aplikace materiálu se specifickými vlastnostmi (bump mapping). V zásadě jde vždy o výpočet normál k trojúhelníkům, ze kterých je objekt složený. Trojúhelníkům, ze kterých je síť složená, říkáme faces. Abychom však mohli začít skládat objekt z trojúhelníků, musíme znát souřadnice jejich vrcholů. Navíc je také důležité pořadí vrcholů každého trojúhelníku, protože určuje směr normály. Velikost a orientace normály je pak důležitá z hlediska výpočtu osvětlení a pomáhá nám při řešení viditelnosti.

2.2.2 Normály a jejich výpočet

Normály pro objekt lze počítat několika odlišnými způsoby, přičemž výsledné zobrazení scény se podstatně liší v závislosti na použité metodě.

No shading

Jak již název napovídá, tak v tomto případě nebudou ve scéně spočítány žádné normály. Tuto metodu zde uvádím proto, abychom si uvědomili význam stínování ve vztahu k modelování 3d grafiky. Trojrozměrný objekt, který nebude mít spočítány normály, se bude jevit v prostoru jako plochý (dvourozměrný). Dochází tak k degradaci scény do 2D prostoru. Dojem 2D zobrazení je způsobem tím, že pro všechny pixely modelu je použita stejná barva.

Flat shading (konstantní stínování)

Konstantní stínování je použito tam, kde je třeba dosáhnout vysokých rychlostí při zobrazování, případně pro náhledové nebo pracovní modely. Pro každý trojúhelník modelu je spočítána jedna normála. Záleží přitom na pořadí vrcholů, které určují směr normály. Normála se spočítá pomocí trigonometrické operace nazvané cross product, což není nic jiného, než určení vektoru kolmého k ploše, která je zadána dvěma vektory.

Smooth shading (Gouraudovo stínování)

Tento způsob zajišťuje plynulé stínování nerovných povrchů tak, že aproximace povrchů ploškami (trojúhelníky) není zřetelná, na rodíl od konstantního stínování. Pro každý trojúhelník jsou spočítány tři normály, jedna v každém vrcholu. Normála je spočítána jako součet normál ploch všech trojúhelníků, kterých k danému vrcholu náleží.

Phong shading (Phongovo stínování)

Tato metoda poskytuje ještě lepší visuální výsledky než Gouraudovo stínování. Cena udávající poměr mezi kvalitou výstupu a výpočetními nároky je však mnohem vyšší, než u předcházejících tří metod. Phongovo stínování je založeno na interpolaci normálových vektorů. Normály jsou interpolovány současně při rasterizaci plochy a z nich je vyhodnocením osvětlovacího modelu určen odstín barvy každého pixelu. Oproti Gouraudovu stínování dochází při interpolaci normál u Phongova stínování k jejich postupnému naklánění a tedy i ke

správnému výpočtu jasu. Rozdíl mezi Gouraudovým a Phongovým stínování dále popisuje například [MoPoGr98].

2.3 Materiály a osvětlení

Dalším důležitým krokem v procesu renderování scény je definice vlastností materiálů aplikovaných na vytvářené objekty. Materiály úzce souvisí s osvětlovacím modelem a optickými vlastnostmi povrchu. Osvětlovací model mimo jiné definuje vlastnosti povrchu jako jsou barva, lesklost, apod. Zvolený osvětlovací model určuje výslednou kvalitu renderovaného modelu.

V reálném světě se určuje barva objektu na základě reflexních vlastností povrchu a množství dopadajícího světelného záření. Modelování vlastností materiálů pro celé světelné spektrum je velice náročné, a proto se v oblasti počítačové grafiky setkáme nejčastěji s modelem redukovaným na tři základní složky spektra, kterými jsou červená, zelená a modrá složka.

2.4 Struktura 3ds souboru

Struktura vlastního 3ds file formátu je poměrně jednoduchá. Soubor se skládá z datových bloků. Každý takový datový blok nazýváme *chunk*. Každý chunk (datový blok) má hlavičku, která obshauje dvě položky. První položku budeme nazývat *chunk identifier*. Ta nám podává informaci o tom, s jakým blokem budeme pracovat (mesh, material, keyframe, ...). Druhá položka pak určuje velikost celého datového bloku, včetně hlavičky. Každý datový blok má svou vlastní hlavičku, tedy dvojici *chunk identifier* a *chunk size*. To samozřejmě není celý princip uložení informací v 3ds souboru. Některé *chunky* mohou představovat rodiče jiných *chunků*. Potomka takového rodiče potom nazýváme *sub-chunk*. Rodičovský *chunk* může přistupovat ke koncovým informacím (souřadnice vrcholů, název materiálu, ...), ale může definovat také svoje potomky (*sub-chunky*), které mohou k těmto koncovým informacím přistupovat také. Tyto *sub*-chunky jsou zahrnuty ve velikosti datového bloku rodičovského chunku. Velikost datového bloku je tedy dána součtem velikostí všech sub-chunků a velikostí dat, která obsahuje daný chunk přímo. Tedy například obsahuje-li chunk dvě datové položky typu float a jeden sub-chunk, bude velikost tohoto chunku součet velikosti hlavičky + 2 x float + velikost sub-chunku. Jak jsme již uvedli, hlavička obsahuje dvě položky. První je identifikátor chunku a jeho velikost je 2 byte. Druhá položka představuje velikost datového bloku a její rozsah je 2^{32}, tedy 4 byte. Velikost hlavičky je tedy pro všechny chunky 6 byte. Pro

náš příklad se dvěma čísly float a jedním sub-chunkem bude tedy velikost datového bloku 6 byte + 2x4 byte + velikost datového bloku sub-chunku.

Struktura souboru s definovanou strukturou může vypadat například takto:

```
chunk 1
        sub-chunk 1
                data 1
                data 2
                data 3
                sub-chunk 1
                        data 1
                        data 2
                sub-chunk 2
                        data 1
                        data 2
                        data 3
chunk 2
        data 1
        data 2
.
.
.
```

Pokud to není z předchozího textu zřejmé, tak pro jistotu zopakujeme fakt, že k tomu abychom mohli přistoupit k nějakému sub-chunku, musíme nejdříve přečíst jeho rodiče. Nyní uvedu základní kostru 3ds souboru složenou z chunků:

```
MAIN CHUNK 0x4D4D
   3D EDITOR CHUNK 0x3D3D
      OBJECT CHUNK 0x4000
         TRIANGULAR MESH 0x4100
            VERTICES LIST 0x4110
            FACES DESCRIPTION 0x4120
            MAPPING COORDINATES LIST 0x4140
               SMOOTHING GROUP LIST 0x4150
            LOCAL COORDINATES SYSTEM 0x4160
         LIGHT 0x4600
            SPOTLIGHT 0x4610
         CAMERA 0x4700
      MATERIAL CHUNK 0xAFFF
         MATERIAL NAME 0xA000
         AMBIENT COLOR 0xA010
         DIFFUSE COLOR 0xA020
         SPECULAR COLOR 0xA030
         TEXTURE MAP 0xA200
         BUMP MAP 0xA230
      KEYFRAMER CHUNK 0xB000
         MESH INFORMATION BLOCK 0xB002
         FRAMES (START AND END) 0xB008
            OBJECT NAME 0xB010
            OBJECT PIVOT POINT 0xB013
            POSITION TRACK 0xB020
            ROTATION TRACK 0xB021
            SCALE TRACK 0xB022
            HIERARCHY POSITION 0xB030
```

V uvedené struktuře 3ds souboru vidíme, že k tomu, abychom získali například souřadnice vrcholů pro daný objekt, musíme nejprve přečíst MAIN_CHUNK, 3D_EDITOR_CHUNK, OBJECT_CHUNK, TRIANGLULAR_CHUNK a nakonec datový blok obsahující souřadnice vrcholů VERTICES_LIST. K tomu, abychom však mohli přečíst vrcholy, potřebujeme znát jejich přesný počet, abychom věděli, kolik floating point čísel máme přečíst. Počet vrcholů zjistíme také z 3ds souboru, ale z jiného sub-chunku, který obsahuje pouze jedno číslo, udávající počet vrcholů daného objektu.

Všechny OBJECT, MATERIAL a KEYFRAME CHUNK (a samozřejmě i některé další) se mohou v 3ds souboru objevit vícekrát. Tři uvedené chunky se můžou objevit tolikrát, kolik je definovano objektů, použitých materiálů a aplikovaných animací ve scéně.

K tomu, abychom mohli vytvořit knihovnu, umožňující načtení 3ds souboru, musíme znát přesnou strukturu 3ds file formátu, respektive přesný obsah jednotlivých chunků. Tento popis je dostupný v originální dokumentaci k 3ds formátu od jeho autorů z firmy Autodesk® [**odkaz na stranky s dokumentaci**].

3 Prostředky realizace

3.1 3D Studio MAX

3D Studio MAX je jeden z nejrozšířenějších programů pro vizualizaci a animaci virtuálních scén. Při řešení tohoto semestrálního projektu pro nás hraje 3D Studio důležitou úlohu. Umožňuje totiž vytvořenou scénu exportovat do formátu 3ds. 3D Studio budeme tedy využívat pro kreslení jednoduchých objektů, na kterých lze snadno testovat funkčnost programu. Důležitá pro nás bude především část týkající se animace, protože právě ona je jedním z hlavních důvodů, proč je 3ds formát považován za „tajemný" formát. Vytvořené modely se budeme snažit správně zobrazit s pomocí vylepšeného programu 3ds2iv, jehož základ byl vytvořen v rámci ročníkového projektu.

3.2 Open Inventor

Open Inventor je objektově orientovaná knihovna pro 3D grafiku vyvíjená firmou Silicon Graphics (SGI). Umožňuje řešení problémů při programování interaktivních grafických aplikací. Její programovací model je založen na vytvoření grafu scény. Díky tomuto programovacímu modelu se výrazně zjednodušuje programování aplikace. Existuje řada open source implementací Open Inventoru. My budeme používat implementaci Coin3D [Link3], která je plně kompatibilní s SGI Open Inventorem 2.1. Open Inventor je dnes standardem pro 3D vizualizační a simulační software pro vědeckou a inženýrskou komunitu. Více o Open Inventoru se lze dozvědět z [OpenIv] a [OpIvC] a [Link2].

3.3 Knihovna Lib3ds

Vytvořit jmenovanou knihovnu je jedním z úkolů, který je třeba splnit v rámci tohoto semestrálního projektu. Knihovna bude vycházet z knihovny 3dsftk (3D Studio File Toolkit) od Autodesku®, která byla upravena firmou Cyberloonies. Knihovna 3dsftk není k dispozici pod licencí public domain (nebo podobnou volnou licencí) a proto jsme se rozhodli si napsat knihovnu vlastní. Knihovna je pro nás sama o sobě důležitá, protože bude umožňovat načítání informací z vlastního 3ds binárního souboru. Pomocí funkcí této knihovny, pak jednoduše získáme informace o scéně a všech jejích částech. V budoucnu bychom chěli tuto knihovnu doplnit o další užitečné funkce pro práci s načtenými daty, týkající se především geometrie.

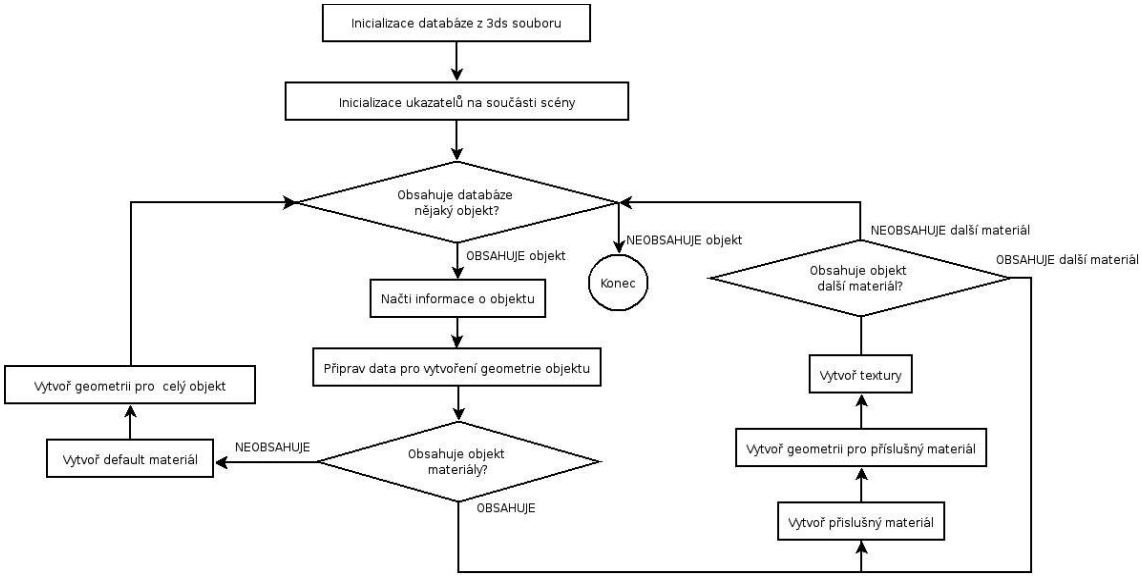
4 Implementace

4.1 Zobrazení scény

Před začátkem implementace programu pro zobrazení scény je nutné si uvědomit, jakým způsobem se vytváří scéna v Open Iventoru, ale především, které informace o scéně a v jaké podobě nám poskytuje vlastní 3ds model, ze kterého budeme scénu rekonstruovat. Tyto zkušenosti jsme získali při práci na ročníkovém projektu a právě z těchto zkušeností budeme dále vycházet. Není již tedy butné uvádět detailní postup při vytáření programu pro zobrazení scény. Uvedu pouze důležité klíčové kroky, po jejichž provedení získáme představu o tom, co se v rámci ročníkového projektu vytvořilo.

K implementaci budeme využívat knihovnu naši vlastní knihovnu Lib3ds, jejíž implementaci si popíšeme v jedné z následujících podkapitol.

Dovolíme si na začátek uvést vývojový diagram, který vystihuje princip funkce programu a umožňuje tak čtenáři abstraktní pohled na řešený problém. Podle tohoto modelu programu se postupovalo při vytváření programu. Dále v textu si připomeneme ty nejdůležitější kroky při implementaci programu. ZDE DOPLNIT TRANSFORMACE DO OBRAZKU!!!

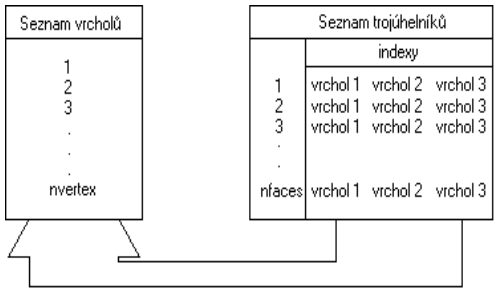


Obrázek 4.1 Zjednodušený vývojový diagram programu

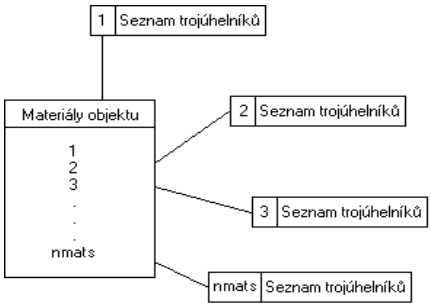
4.1.1 Geometrie objektů

Řešení geometrie objektů by se mohlo na první pohled jevit jako jednoduchý problém. Avšak vzhledem k tomu, jakým způsobem jsou uloženy informace o geometrii scény v 3ds

souboru se jedná o problém netriviální. K tomu, abychom vytvořili správnou geometrii objektu, budeme muset znát nejenom vrcholy, ale i to k jakému materiálu a do jakých smoothing group jednotlivé trojúhelníky (faces) patří. K tomuto účelu je 3ds soubor vybaven seznamy trojúhelníků, ze kterých lze zjistit vše potřebné. Situaci přibližují obrázky 4.2 a 4.3



Obrázek 4.2 Ilustarace mapování seznamu trojúhelníků do seznam vrcholů. Počet vrcholů je nvertex a počet trojúhelníků je nfaces.

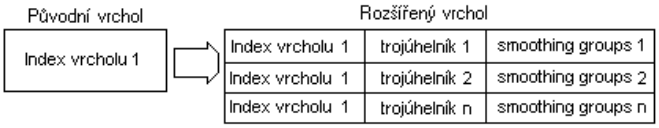


Obrázek 4.3 Ilustrace přiřazení seznamu vrcholů k materiálu. Počet použitých materiálů v objektu je nmats.

V případě, kdy jsou pro objekt definovány smoothing groups, nastává pro vytvoření scény nejkomlikovanější situace. Musíme totiž spočítat normály vrcholů s ohledem na smoothing groups a s ohledem na materiál, do kterého případný trojúhleník patří. Potřebujeme tedy získat informaci o tom, ke kterým trojúhelníkům zkoumaný vrchol patří a do kterých smoothing groups patří zpracovávané trojúhelníky. Řešením je rozšíření pole se seznamem vrcholů o dvě nové informace, což ukazuje obrázek 4.5. Pro každý vrchol tedy budeme vědět, do kterých trojúhelníků patří a do které smoothing groups patří tyto trojúhelníky.

Zde je vhodné místo, abych se zmínil o problému který vznikne při texturování. I když se texturování této kapitoly zdánlivě netýká, tak opak je pravdou. Pokud si v 3D Studiu vytvoříme model, který nebude mít definovány texturovací souřadnice, pak bude výše uvedený

Obrázek 4.4 Ukázka špatného výpočtu normál při vygenerování texturovacích souřadnic.



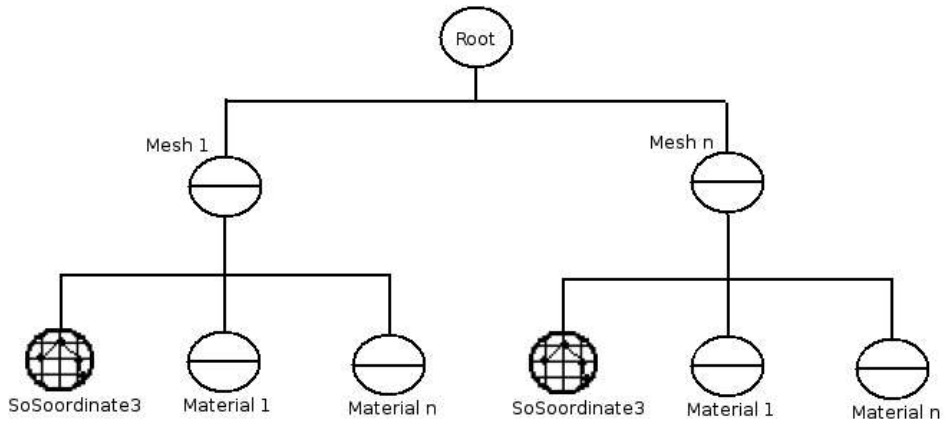
Obrázek 4.5 Rozšíření vrcholů. Pro každý vrchol objektu zjistíme, do kterých trojúhelníků patří. Pro trojúhelníky pak zjistíme, do kterých smoothing groups patří.

princip výpočtu normál pracovat správně. Jakmile však v 3D Studiu vygenerujeme pro objekt texturovací souřadnice, dojde ke změně v počtu vrcholů objektu, což má za následek špatný výpočet normál, jak je vidět na obrázku 4.4. K projevu změny počtu vrcholů dojde až při exportu do 3ds formátu. Je zřejmé, že hrany, kde dochází ke špatnému výpočtu normál, jsou místa, kam byly přidány nové vrcholy. Tento krok je zřejmě nezbytný pro správný wrapping textury na objekt. Nám to však přináší velké komplikace při výpočtu normál. Normály se v tomto místě počítají špatně, protože původní i nové vrcholy mají sice stejné souřadnice, ale nejsou vzájemně spojeny. Došlo tedy v podstatě k roztrhnutí modelu. K tomu, abychom správně spočítali normály, je nutné vrcholy se stejnými souřadnicemi umístit na stejný index v seznamu vrcholů. Toho lze docílit s využitím BSP stromu, který implementuje knihovna Coin3D a do kterého lze vkládat přímo vrcholy. Eliminují se tak kritické vrcholy se stejnými souřadnicemi a v podstatě dojde k opětovnému uzavření objektu, který byl přidáním nových vrcholů rozdělen.

Výše uvedený problém není ve skutečnosti chybou při zpracování 3ds souboru. Tento zásah do modelu vzniká při exportu do 3ds souboru a pokud zkusíte tento model importovat zpět do 3D Studia, zjistíte, že i zde se objevuje tato chyba. Výše uvedený problém se tedy netýká správnosti iterpretace a použití informací z 3ds souboru, ale týká se vlastního exportu 3D Studia do tohoto formátu. Evidentně je však možné upravit geometrii objektu tak, aby byla tato chyba eliminována, a proto tento problém nepovažují za omezení 3ds formátu, ale považuji jej za chybu (nedostatek) v exportním modulu 3D Studia.

4.1.2 Vytváření grafu scény v Open Inventoru

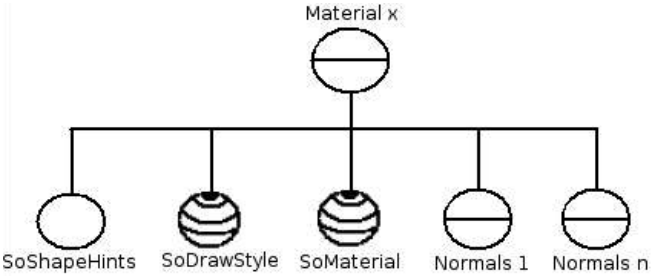
Při vytváření grafu scény v Open Inventoru budeme postupovat následovně. Vytvoříme kořen scény s názvem *root*, který zareferencujeme. Nyní pro každý samostatný



Obrázek 4.6 Základní graf scény rozdělený podle počtu objektů ve scéně Mesh 1 až Mesh n. Pro každý Mesh jsou přidány do grafu souřadnice vrcholů a třídy Material 1 až Material n, které budou rozdělovat objekt podle počtu aplikovaných materiálů.

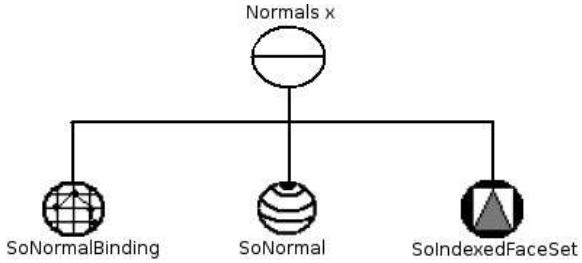
objekt ve scéně vytvoříme třídu *SoSeparator* (*Mesh1* až *Mesh n*). Do každé třídy *Mesh* vložíme souřadnice vrcholů *SoCoordinate3*. Třídu *Mesh* dále rozdělíme na tolik částí, kolik obsahuje materiálů. To znamená, že vytvoříme pro každý materiál jednu třídu *SoSeparator* s názvem *Material x*. Předem upozorňuji, že třída *Material* představuje třídu *SoSeparator,* a ne terminál *SoMaterial*, jak by se mohlo na první pohled zdát z názvu třídy *Material*. Čili graf scény bude prozatím vypadat podle obrázku 4.6.

Každá třída *Material 1* až *Material n* bude obsahovat uzly definující materiálové vlastnosti, informace o normálách a trojúhelnících. Vlastnosti normál a definice trojúhelníků budou uzavřeny ve společné třídě *Normals*. Materiálové vlastnosti jsou společné. Čili každá třída *Material* bude vypadat podle obrázku 4.7.



Obrázek 4.7 Rozvětvení separátoru Material 1 až Material n z obrázku 4.6

Na závěr rozepíšeme třídu *Normals* a získáme tak celkový graf geometrie scény.



Obrázek 4.8 Rozvětvení separátoru Normals1 až Normals n z obrázku 4.7.

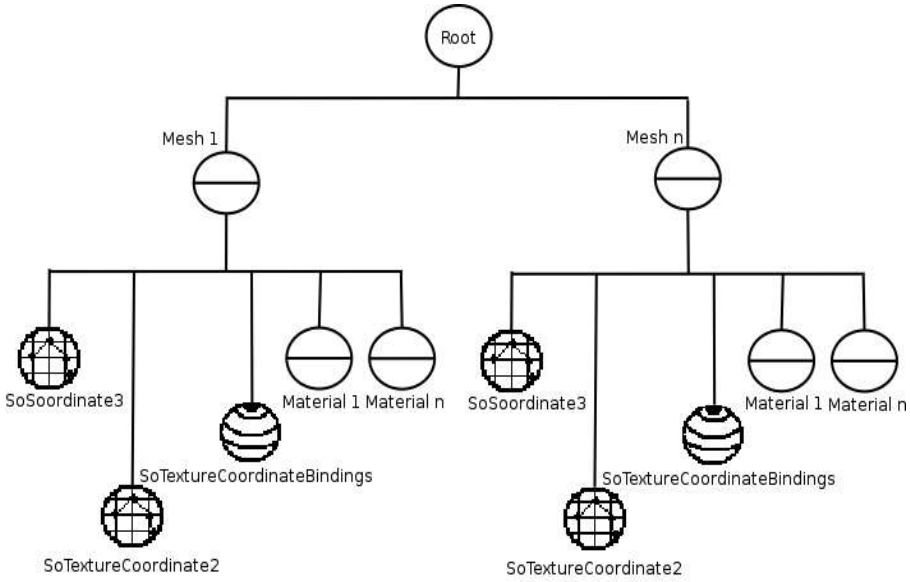
4.1.3 Aplikace materiálů

Hlavním uzlem, který je využíván pro definici vlastností materiálu je uzel *SoMaterial*. Z tohoto uzlu používáme všechny atributy, které nabízí. Pokud nemá objekt definován žádný materiál, použijeme vlastní nastavení nezbytně nutných atributů a takto vytvořený materiál objektu přiřadíme. Pokud má objekt definován jeden a více materiálů, pak zjistíme jejich vlastnosti z 3ds souboru a použijeme je jako atributy pro uzel *SoMaterial*. Materiály se do grafu scény přidávají v okamžiku vytvoření separátoru *Materials* (viz. obrázek 4.7). Na obrázku 4.7 vidíme, že jsme do grafu společně s uzlem *SoMaterial* přidali uzly *SoShapeHints* a

SoDrawStyle. Uzel *SoShapeHints* využíváme pro nastavení *twoSideLightning* a *backFaceCulling* pomocí parametrů *shapeType* a *vertexOrdering*. Dále bylo v rámci semestrálního projektu zjištěno, že bude potřeba některé objekty zrcadlově otočit. To se provádí tak, že se použije geometrická transformace záporného scale o velikosti -1.0 podle příslušné osy. To má však za následek to, že se provede také otočení normál a dojde tak k tomu, že objekt se stane neviditelný. Pro tento případ použití záporného scale tedy musíme nastavit vertexOrdering na CLOCKWISE, tedy pravotočivý, abychom mohli objekt (mesh) správně zobrazit.

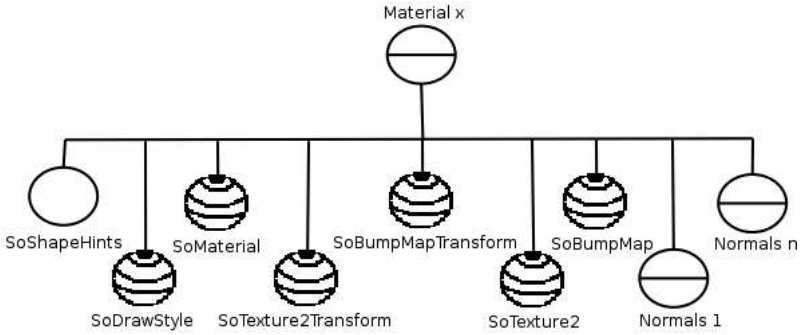
4.1.4 Texturování

Knihovna Coin 3D nám umožňuje poměrně dobrou práci se základními texturami. Využíváme zde uzlu *SoTexture2* a doplňující uzly *SoTextureCoordinate2, SoTextureCoordinateBinding* a *SoTexture2Transform*. Mimo uzel *SoTexture2* podporuje Coin 3D od verze 2.2 také bump mapping s pomocí uzlů *SoBumpMap, SoBumpMatCoordinate* a *SoBumpMapTransform*. Graf scény rozšířený o textury pak bude vypadat následovně.



Obrázek 4.9 Graf scény s přidanými texturovacími souřadnicemi a bindingem. Původní graf scény je na obrázku 4.6

Každý uzel *Material 1* až *Material n* je rozepsán tak, jak ukazuje obrázek 4.10.



Obrázek 4.10 Rozvětvení separátoru Material 1 až Material n z obrázku 4.9. Oproti obrázku 4.7 byly do grafu přidány texturovací uzly.

4.2 Knihovna Lib3ds

Při tvorbě této knihovny jsme vycházeli ze znalosti používání knihovny 3dsftk. Snažili jsme se tedy, aby princip používání byl podobný. Naše knihovna Lib3ds je implementována přímo na tělo programu pro zobrazení scény v Open Inventoru. Samozřejmě obsahuje některé části, které námi nejsou v programu využívány, ale jejich načtení bylo nezbytbé pro správnou funkci této knihovny. Abychom mohli začít vytvářet implementaci této knihovny, museli sme se nejprve seznámit s vlastní strukturou 3ds formátu. Jeho popis je uveden v teorii a příslušné literatuře [UVEST DOKUMENTACI 3DS FORMATU]. Podrobné informace o tom, jakými funkcemi a jakými typy disponuje naše knihovna Lib3ds, se dozvíte v její dokumentaci. Vlastní implementace knihovny Lib3ds se dá rozdělit na čtyři části (třídy), jejichž popis bude tématem následujících podkapitol.

4.2.1 Procházení 3ds souborem

V této třídě je implementováno procházení 3ds souborem po chuncích. Dále tato třída implementuje funkce, kterými knihovna disponuje, tedy implementuje rozhraní knihovny.

Nejprve si vytvoříme jednotlivé interní databáze pro meshe, materiály a pro keyframe informace. To zajistí konstruktory jednotlivých tříd (mesh, material, keyframe). Nyní nastává okamžik, kdy budeme postupně procházet 3ds souborem. Načteme ze 3ds souboru, který máme otevřený binárně dva byte, které nám určují *chunk id*. Bezprostředně poté načteme z 3ds souboru další 4 byte, které nám určují velikost aktuálního *chunku*. Nyní se musíme podívat do dokumentace 3ds formátu a zjistit, jestli je pro nás načtené *chunk id* důležité. Pokud je pro nás aktuální *chunk* určený svým *chunk id* nepotřebný, můžeme jej na základě jeho velikosti *chunk size* přeskočit. Pokud je pro nás aktuální chunk důležitý, budeme pokračovat v načítání podle toho, o jaký chunk se jedná. Vlastní ypracování chunku už zajišťují třídy starající se o jednotlivé databáze (mesh, material a keyfreame databáze).

4.2.2 Načtení geometrie objektů

Nejprve zjistíme z 3ds souboru informace o geometrii jednotlivých objektů scény a vytvoříme také seznam všech objektů ve scéně.

Z 3ds souboru zjistíme jméno aktuálního mesh objektu a přidáme jej do seznamu. Alokujeme potřebnou pamět a nastavíme všem parametrům pro mesh objekt výchozí hodnoty. Nyní načteme z 3ds souboru vrcholy pro daný mesh objekt. Přečteme nejprve číslo udávající počet vrcholů daného mesh objektu a na základě tohoto čísla pak přečteme přesný počet vrcholů. Dále v podobném duchu (musíme stále spolupracovat s dokumentací 3ds formátu) načítáme postupně *Flags array, Face array, MatGroup array, SmoothArray, Texture vertex array a Local matrix array*. V podstatě jde vždy o to, abychom alokovali místo pro dané pole s informacemi a správně načetli a interpretovali celý obsah zpracovávaného chunku. Důležité je vědět, že vytváříme seznam všech objektů ve scéně a pro každý takový objekt čteme informace, které ho definují.

4.2.3 Načtení materiálů

Zde je situace velmi podobná načítání geometrie (mesh objektů). Vytváříme zde seznam materiálů, které jsou definovány ve scéně. Při vytváření databáze mesh objektů jsme z 3ds souboru zjistili názvy materiálů, které jsou aplikovány na mesh objekt. Nyní vytváříme jejich seznam, ale především jejich definici. Každý materiál má při vytvoření všechny parametry definované knihovnou nastaveny na default hodnoty. Teprve při procházení 3ds souborem zjistíme, které marametry jsou pro daný materiál definovány a pak přečteme hodnoty těchto parametrů. Pro ostatní parametry zůstanou nastaveny default hodnoty. Popis výchozích (default) hodnot lze nalézt v dokumentaci knihovny Lib3ds.

Při vytváření materiálů postupujeme tak, že alokujeme paměť pro nový materiál a přidáme jej do jmenného seznamu. Dále pak čteme řadu parametrů (pokud byly v 3D Studiu aplikovány) a nastavujeme je v databázi pro aktuální materiál.

Současně s načítáním parametrů materiálů se zpracovávají textury. Pro každý materiál , pokud má definovánu texturu se pro ni alokuje místo a přečtou se její parametry. Opět je nutné velmi úzce spolupracovat s dokumentací 3ds formátu, protože bez ní bychom těžko binární 3ds soubor zpracovali.

4.2.4 Načtení keyframe částí (animace)

V této čáasti budeme načítat informace o animaci, ale také o hierarchické struktuře ve scéně. Ke každému význačnému objektu ve scéně (kamera, mesh, světlo, ...) může být definována aniamce (hierarchická struktura). Nás bude zajímat pouze hierarchická struktura mesh objektů a proto musíme ostatní objekty překočit.

Implementace této části pro nás byla nejsložitější, protože jsme s ní neměli žádné zkušenosti.

Navíc se objevil problém s knihovnou 3dsftk, která špatně pracovala s hierarchickou strukturou u modelů vytvořených v klasické verzi 3D Studia (bez MAX). Problém byl v tom, že podle nová verze 3ds formátu je hierarchická stuktura identifikována pomocí svého id čísla a názvu. Ve starších verzích 3ds formátu se však hierarchická struktura řídí podle názvu a podle pořadí výskytu objektu v části keyframe. Knihovna 3dsftk pravděpodobně pracuje pouze s názvy a s id čísly, které však nebyly u starších verzí definovány a knohovna tak nedokázala správně načíst informace o hierarchické struktuře. Vše demostruje následující příklad.

```
ChunkOBJECT_NODE_TAG (b002H)
  Length is 189 (bdH)

  Chunk NODE_ID (b030H)
  Length is 8 (8H)
   Node ID: 0

  Chunk NODE_HDR (b010H)
  Length is 17 (11H)
  Object name: roof
  Flags 1: 4000
  PRIMARY_NODE
  Flags 2: 0
  No Parent
```

```
Chunk OBJECT_NODE_TAG (b002H)
  Length is 191 (bfH)

  Chunk NODE_HDR (b010H)
  Length is 17 (11H)
  Object name: roof
  Flags 1: 4020
  ATKEY2
  ATKEYFLAGS
  PRIMARY_NODE
  Flags 2: 0
  No Parent
```

Vlevo je výstup programu 3D Studio MAX a vpravo výstup programu 3D Studio (výstup získán pomocí 3dsftk). Vidíme, že ve staré verzi opravdu chybí chunk NODE_ID. Řešením je takové, že NODE_ID je definováno jako pořadové číslo chunku OBJECT_NODE_TAG v keyframe části. Rodič (parent) je poté dohledán na základě pořadového čísla. U nové verze je hledání prováděno na základě shody *node_id* a *Parent*. S hledáním rodiče to však není tak jednoduché, jak jsme nyní naznačili. Nyní se pokusíme vysvětlit proč. V 3D Studiu máme možnost vytvářet nové objekty jako instance jiných. Tyto objekty mají naprosto shodnou geometrii jako jejich rodiče, ale můžou být jinak posunuté, pootočené, zvětšené nebo změnšené. Tyto objekty pak mají stejný název jako jejich rodiče, ale liší se parametrem *Instance name*. Tento parametr určuje jméno objektu vytvořeného jako instance jiného. V části vytváření databáze s geometrií, nejsou tyto objekty zahrnuty. Budeme-li tedy procházet objekty scény pomocí seznamu vytvořeného v databázi s mesh objekty, neuvidíme ty, které jsou vytvořeny jako instance. Vytvoříme-li objekt s názvem Box01, pak jeho instanci s názvem Box02 a nakonec instanci Box03 z instance Box02. Dále v 3D Studiu definujeme, že instance Box03 bude mít jako rodiče instanci Box02. Situace v 3ds souboru bude následující. Objekt Box01 je definován jako hlavní objekt (není to instance) a má tedy definovánu geometrii, materiál atd. Další objekt je definován opět jako Box01, ale má navíc definovánu instanci Box02. To znamená, že byl vytořen z objektu Box01. Má tedy stejnou geometrii jako objekt Box01, ale liší se posunutím/rotací/měřítkem. Další objekt je opět Box01, ale má instanci Box03. Navíc je u této instance Box03 definován rodič, kterým má název Box01.Box02. To znamená, že rodičem je instance objektu Box01 s názvem Box02. Situace může být ještě o něco složitější, protože 3D Studio umožňuje definovat specielní objekt s názvem DUMMY. Jde o objekt, který nemá žádnou geometrii. Jde o pomocný objekt, který může opisovat například nějakou dráhu a zároveň být rodičem nějakého objektu,

který pak tuto dráhu opisuje také. Na tuto vlastnost 3ds formátu si musíme dát při vytváření knihovny pozor. Popsanou situaci s objekty BOX01 až BOX03 ilustruje následující text.

```
Chunk OBJECT_NODE_TAG (b002H)
  Length is 168 (a8H)

  Chunk NODE_ID (b030H)
  Length is 8 (8H)
   Node ID: 0

  Chunk NODE_HDR (b010H)
  Length is 18 (12H)
  Object name: Box01
  Flags 1: 4000
  PRIMARY_NODE
  Flags 2: 0
  No Parent

Chunk OBJECT_NODE_TAG (b002H)
  Length is 180 (b4H)

  Chunk NODE_ID (b030H)
  Length is 8 (8H)
   Node ID: 1

  Chunk NODE_HDR (b010H)
  Length is 18 (12H)
  Object name: Box01
  Flags 1: 0
  Flags 2: 0
  No Parent
 Chunk INSTANCE_NAME (b011H)
  Length is 12 (cH)
  Instance name: Box02
```

```
Chunk OBJECT_NODE_TAG (b002H)
Length is 180 (b4H)

Chunk NODE_ID (b030H)
Length is 8 (8H)
 Node ID: 2

Chunk NODE_HDR (b010H)
Length is 18 (12H)
Object name: Box01
Flags 1: 0
Flags 2: 0
Parent 1

Chunk PARENT_NAME (80f0H)
 Length is 0 (0H)
 Parent name: Box01.Box02

 Chunk INSTANCE_NAME (b011H)
 Length is 12 (cH)
 Instance name: Box03
```

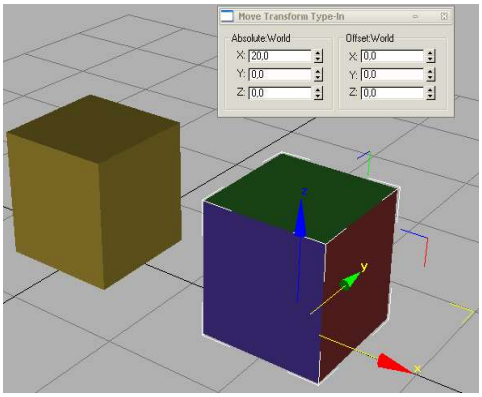
Povšimněte si prosím toho, jakým způsobem se skládá jméno rodiče, pokud je tento rodič instancí nějakého dalšího objektu (Box01.Box02). Tuto situaci musíme brít v úvahu při vytváření funkcí pro práci s animací. Pokud jakýmkoliv způsobem změníme geometrii objektu, ze kterého je odvozena nějaká instance, pak se tato instance stává samostatným objektem (už není instancí). Pro náš příklad by se instance Box02 stala samostatným objektem Box02. Rodičem instance Box03 by se potom stal objekt Box02.

5 Analýza, testování a řešení

Tato kapitola popisuje analýzu, testování a především způsob řešení nalezených problémů. Při práci na předcházejícím ročníkovém projektu bylo nalezeno množství modelů, které byly více či méně špatně zobrazeny. Ve většině případů šlo o problém s animací nebo-li definovanou hierarchickou strukturou v 3ds souborech. Dalším problémem je fakt, že řada modelů byla vytvořená ve starých verzích 3D Studia. Při importu/exportu těchto modelů do/z novější verze 3D Studia MAX došlo ke ztrátě některých informací, protože již v nových verzích nejsou podporovány. Takových modelů není zanedbatelné množství. Většina již nepodporovaných atributů je pro rekonstrukci scény skutečně nepotřebná, protože například souvisí s nastavením uživatelského rozhraní. Při práci na tomto semestrálním projektu byla zjišťěna řada zajímavých informací, které nám pomohou zcela odhalit roušku tajemství vznášející se nad formátem 3ds. Dále budeme rozlišovat tyto základní termíny.

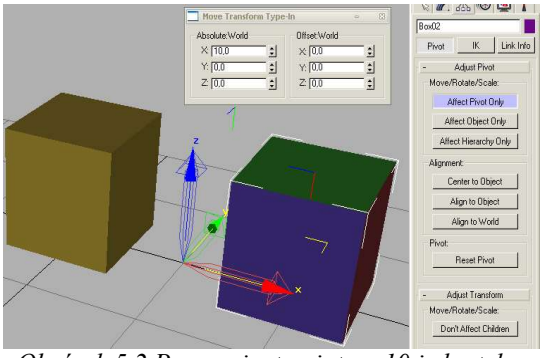
- **Pivot point** (PP) – střed lokálního souřadného systému, na který lze aplikovat geometrické transformace.
- **Lokální souřadný systém** (LSS) – lokální souř. systém objektu, na který lze aplikovat pouze transformaci posunu
- **Originální pivot point** (OPP) – lokální souřadný systém objektu, na který není aplikována žádný transformace. Je dán při vytvoření objektu interně 3D Studiem.

5.1 Základní principy

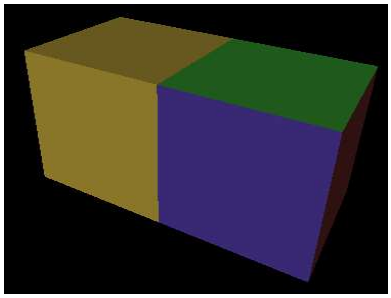


Obrázek 5.1 Základní scéna, ze které budeme vycházet při analýze animační části 3ds souboru

V této kapitole budeme postupovat v jednotlivých krocích tak, abychom dostatečně dobře pospali řešený problém a nastínili čtenáři princip jeho řešení. Výchozí scéna bude pro nás definována pomocí dvou kvádrů, přičemž jeden je umístěn v počátku souřadnic a druhý je posunutý o 20 jednotek ve směru osy x. Velikost hrany krychle je 10 jednotek. Základní scéna je vidět na obrázku 5.1. První a základní změnou týkající se mimo jiné animace, je posun lokálního souřadného systému (translace). Na obrázku 5.2 vidíme posun lokálního souřadného systému krychle Box02 (různobarevná) o deset jednotek ve směru osy x. Lze pozorovat, že v 3D Studiu zůstává model Box02 nezměněn (stále je na své původní pozici) a mění se pouze LSS. Pokud nebudeme uvažovat v programu s posunem tohoto



Obrázek 5.2 Posun pivot pointu o 10 jednotek ve směru osy x. Tato změna se ve scéně neprojeví.



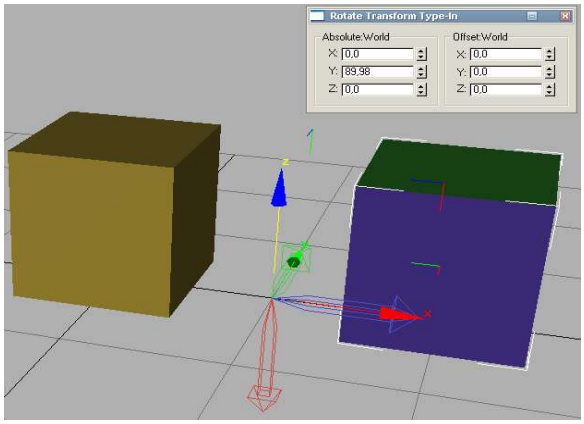
Obrázek 5.3 Výstup našeho programu, pokud nebudeme uvažovat pivot poitny.

lokálního souřadného systému, dostaneme csénu z obrázku 5.2 výstup stejný jako je ukázáno na obrázku 5.3.

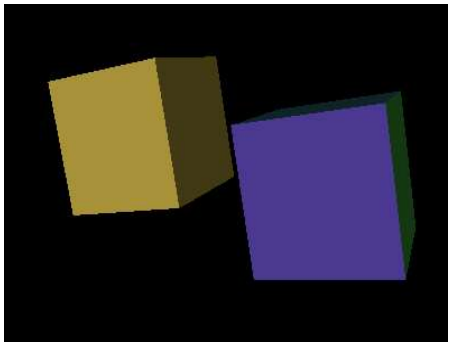
Nyní si položme otázku, jak se změní informace o scéně, pokud takto posuneme LSS. Analýzou 3ds souboru jsme zjistili, že při posunu LSS se automaticky posunuje také pivot, protože transformace posunutí aplikovaná na pivot má za následek také posunutí LSS. Posunutí LSS je zachyceno pomocí chunku PIVOT v části keyframe. Posunutí (transformační matice) pivot pointu je zachyceno v chunku MESH_MATRIX, která udává geometrickou transformaci pivot pointu. Problém při exportu do 3ds souboru je ten, že tato transformace (pivot pointu) se provede i s vrcholy objektu, který k tomuto souřadnému systému patří (v 3D Studiu se však s vrcholy nic neděje). Dostaneme potom výsledek, jaký je vidět obrázku 5.3. V 3ds souboru tedy dojde ke změně souřadnic vrcholů, psounutí lokálního souřadného systému, z toho plynoucího posunutív pivot pointu. Dále část týkající se keyframe obsahuje informace o pozici, rotaci a změně měřítka, které nám říkají, jak se objekt transformoval vzhledem ke svému lokálnímu souřadnému systému.

Dále situaci ještě o něco zkomplikujeme a provedeme navíc rotaci pivot pointu kolem osy y o 90°. Rotace je ukázána na obrázku 5.4 a výstup našeho programu je na obrázku 5.6.

Z obrázku 5.6 vidíme, co se vlastně stalo. Původní scéna z obrázku 5.4 byla změněna. Stalo se to, že operace s pivot pointem se provedly také s objektem. Tedy provedlo se posunutí o deset jednotek po ose x a poté otočení o 90° ve směru osy y. Výsledek je na obrázku 5.6. K tomu, aby bylo výsledné zobrazení správné, je potřeba provést jisté kroky, které si popíšeme později. Nejprve si musíme uvědomit, že se při exportu provedlo posunutí a otočení objektu Box02 podle pivot pointu. Transformace PP je uložena v transformační matici MESH_MATRIX objektu Box02. Spočítáme-li tedy inverzní matici a tuto inverzní matici aplikujeme na vrcholy objektu Box02, dostaneme tak původní scénu jako je na obrázku 5.4.

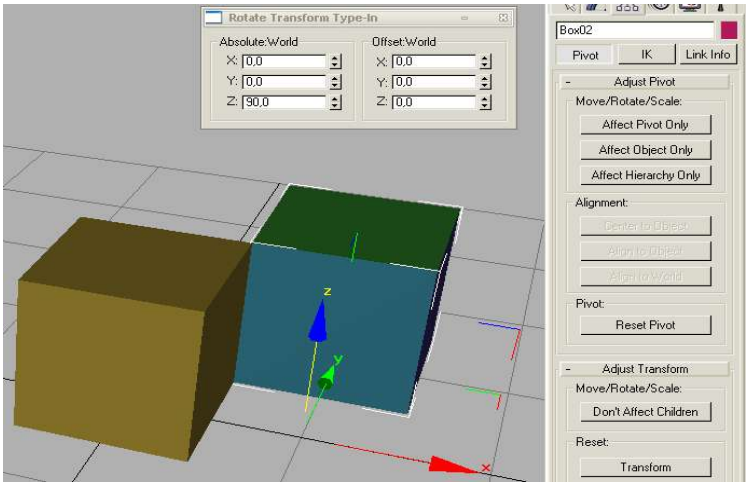


Obrázek 5.4 Rotace pivot pointu o 90° kolem osy y, přičemž je stále posunutý o 10 jednotek na ose x.



Obrázek 5.6 Výstup našeho programu. Vidíme, že rotace souřadného systému se promítla do vrcholů krychle, což je nežádoucí.

Doposud jsme v 3D Studiu prováděli pouze transformace samotného lokálního souřadnicového systému a pivot pointu. Zdůrazňuji samotného, tedy ovlivňovali jsme pouze vlastní počátek a transformaci (rotace) souřadného systému daného objektu, ale v neměnila se



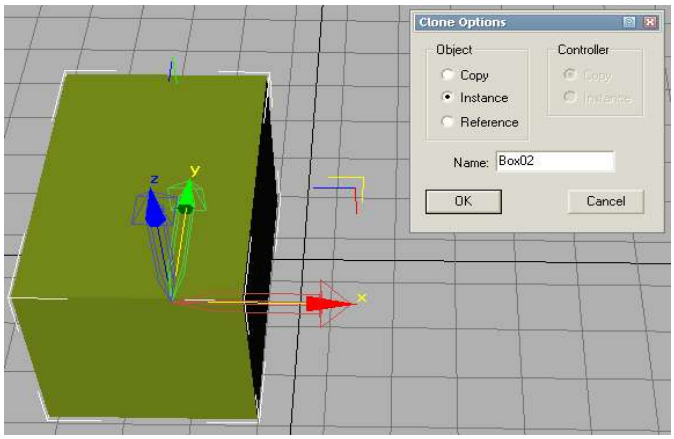
Obrázek 5.5 Scénu z obrázku 5.4 jsme modifikovali tak, že jsme otočili objektem Box02 kolem osy z o 90°. V tomto případě je však rotace prováděna z celou soustavou Box02+pivot point.

pozice ani rotace vlastního objektu. Pokud ale ve stejné scéně provedeme ještě například rotaci objektu Box02 podle lokálního souřadného systému (tedy měníme souřadnice vrcholů objektu i rotaci pivot pointu) nestačí již pouhá aplikace inverzní transformace. Problém je totiž v tom, že pokud transformujeme objekt podle lokálního souřadného systému, pak se transformuje i jeho pivot point. Uvažujme tedy, že model Box02 z obrázku 5.4, který ještě otočíme kolem osy z o 90°. Situaci s provedenými transformacemi ilustruje obrázek 5.5. Pak při pouhé aplikaci

inverzní transformace pivot pointu, dostaneme scénu z obrázku 5.4, což je špatně. Chybí totiž ještě onen posuv podle osy z o 90°, který jsme také eliminovali. Pro tento případ disponuje 3ds formát tzv. keyframerem a speciálně chunkem ROT_TRAG_TAG, ve kterém je uložena rotace objektu Box02 vzhledem k lokálnímu souřadnému systému. Ke správnému zobrazení scény v tomto případě je tedy nutné nejen aplikovat inverzní matici, ale také znovu provést ty transformace, které byly s objektem vzhledem k LSS prováděny (rotace, translace, změna měřítka). V našem případě to tedy bude nejprve provedení rotace pomocí ROT_TRACK_TAG a pak následná aplikace inverzní matice. Výsledkem pak bude scéna stejná jako na obrázku 5.5, což je již scéna shodná s tím, co jsme v 3D Studiu vymodelovali. Všechny transformace musí být prováděny se středem v pivot pointu. Nesmíme zapomenout po provedení inverzní transformace provést také posun objektu z lokálního souřadného syytému (posunutý o vektor PIVOT), zpět do originálního pivot pointu.

5.2 Instance objektů

Některé modely mohou být vyvořeny z jiných jako jejich instance. To znamená, že nemají vlastní geometrii, ale dědí geometrii na základě svého otce. Mohou pak být samozřejmě transformovány pomocí rotace, translace nebo změnou měřítka. Tato informace o transformaci je pak uložena pomocí POS, ROT a SCL_TRACK_TAGŮ a PIVOT chunku. Pokud jakkoliv změníme geometrii instance (změna tvaru), stává se z instance automaticky samostatný objekt. Následující obrázek 5.7 ukazuje dialog vytváření instance v 3D Studiu.



Obrázek 5.7 Ukázka dialogu pro vytváření instancí v 3D Studiu.

Objekty vytvořené jako instance tedy nemají vlastní geometrii, ani transformaci pivot pointu v 3ds souboru. Tedy jejich transformace pivot pointu (určená chunkem MESH_MATRIX) je stejná jako transformace objektu ze kterého byly vytvořeny. Mohou však

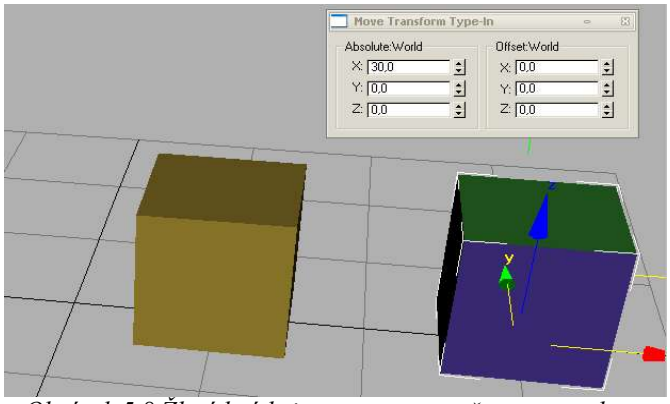
mít jinou transformaci pozice lokálního souřadnicového systému a jinout geometrickou transformaci vzhledem k LSS, která bude dána chunky PIVOT, POS_TRACK_TAG, ROT_TRACK_TAG a SCL_TRACK_TAG.

5.3 Hierarchická struktura

Hierarchická struktura je metoda práce se scénou, která je velice užitečná pro modelování v 3D prostoru, ale především nepostradatelná pro animaci. Hlavním principem hierarchické struktury je, že můžeme pro objekt určit jeho rodiče (jeden objekt může mít pouze jednoho rodiče, ten však může být synem dalšího objektu). Jedná se o stromovou strukturu. Pokud provedeme transformaci rodiče (translace, rotace, měřítko) jsou ovlivněny také všechny jeho děti.

Nás bude zajímat především, jak se změní scéna, přesněji obsah 3ds souboru, zavedeme-li v nějaké jednoduché scéně hierarchickou strukturu. Už teď můžeme prozradit, že v 3ds souboru se bude měnit pouze část, týkající se keyframeru. To znamená, že se nebude měnit MESH_MATRIX žádného objektu ve scéně. Musíme si uvědomit, že jsme prováděli inverzní transformaci nad objektem scény a znova prováděli transformace, které byly s objektem prováděny vzhledem k LSS, který byl zatím nezávislý (respektive byl závislý na globálním souřadném systému). Pak je zřejmé, že nelze k opětovné rekonstrukci scény použít stejný způsob získání informací z keyframe části (PIVOT, POS, ROT a SCL_TRACK_TAG), protože tyto informace se odvíjejí od vytvořené hierarchické struktury.

Vytvoříme tedy scénu se dvěma kvádry. První kvádr Box01 bude posunutý o deset jednotek doprava. Druhý kvádr Box02 bude od své výchozí pozice o 30 jednotek doprava. Scéna tedy bude vypadat jako na obrázku 5.8. Přičemž žlutý kvádr Box01 je rodičem



Obrázek 5.8 Žlutý kvádr je posunut ve směru osy x o deset jednotek. Druhý kvádr je posunut o 30 jednotek ve směru osy x. Žlutý kvádr je rodičem druhého kvádru.

vícebarevného kvádru Box02. Pokud bychom tuto informaci neznali, tak hierarchickou strukturu na první pohled nepoznáme. Pokud nebude definována hierarchická struktura (žádné rodiče a žádné děti), pak budou chunky v 3ds souboru definovány takto:

```
Box01       POS_TRACK_TAG    10,0,0
Box02       POS_TRACK_TAG    30,0,0
```

Pokud však bude Box01 rodičem kvádru Box02, pak budou v 3ds souboru definovány uvedené chunky takto:

```
Box01       POS_TRACK_TAG    10,0,0
Box02       POS_TRACK_TAG    20,0,0
```

Vidíme, že pozice objektu Box02 se změnila v závislosti na objektu Box01, který je jeho otcem. Tedy transformace posuvu je nyní složená. Skládá se z transformace rodiče Box01 a z transformace syna Box02. Z tohoto příkladu lze dále odvozovat, jak bude vypadat situace pro ROT a SCL chunky. Takže při rekonstrukci scény po provedení inverzní transformace musíme projít postupně od zpracovávaného chunku, přes všechny jeho rodiče. Výsledkem je tedy složená transformace. Keyframe informace (POS, ROT, SCALE) se tedy mění na základě keyframe části rodiče. S tímto musíme při interpretaci informací z keyframe části počítat.

5.4 Shrnutí významu chunků v key frameru

Jméno chunku	***Význam chunku***
MESH_MATRIX	Jedná se o transformační matici, která slouží k uložení informace o transformaci pivot pointu (PP) každého objektu ve scéně. Je zde uložena informace o posuvu, rotaci a měřítku.
PIVOT	Jedná se o tříprvkový vektor, který nese informaci o tom, jak je lokální souřadnicový systém posunut oproti originálnímu pivot pointu, který je určen při vytváření objektu interně 3D Studiem.

Jméno chunku	*Význam chunku*
POS_TRACK_TAG	Opět jde o tříprvkový vektor (může jich být pro jeden objekt více, záleží na počtu klíčových snímků a na použitých transformacích pozice v těchto snímcích). Nese informaci o tom, jak je objekt posunutý vzhledem ke svému rodiči nebo globálnímu souřadnému systému. Je zde započítán i posun lokálního souřadného systému vůči originální poloze pivot pointu.
ROT_TRACK_TAG	Jde o čtyřprvkový vektor (může jich být pro jeden objekt více, záleží na počtu klíčových snímků a na použitých transformacích rotace v těchto snímcích). Nese informaci o tom, jakým směrem a o kolik stupňů se rotuje objekt vůči svému lokálnímu souřadnému systému.
SCL_TRACK_TAG	Jde o tříprvkový vektor (může jich být pro jeden objekt více, záleží na počtu klíčových snímků a na použitých transformacích měřítka v těchto snímcích). Nese informaci o tom, jak se mění měřítko objektu v jednotlivých osách vůči lokálnímu souřadému systému.

5.5 Popis implementace

Vzhledem k tomu, že situace s načítáním geometrie 3ds souborů je poměrně dosti komplikovaná, dovolíme si na tomto místě uvést část implementace, která snad úplně osvětlí pohled na její vytváření v 3ds souboru a ukáže řešení jejího správné interpretace.

5.5.1 Řešení geometrie modelů bez uvažování hierarchické struktury

Pokud nebude v 3ds souboru definována žádná hierarchická struktura, tak následující komentovaný kód je řešením správného načítání geometrie všech objektů. Předpokládáme, že máme načtenou geometrii (vrcholy) tak, jak byly do 3ds souboru exportovány. Pak musíme provést následujících 5 kroků ke správné transformaci těchto načtených vrcholů.

1. Pozice instance

Instance objektu dědí geometrii ze svého otce. Má ale vlastní keyframe část. Pro objekty, které jsou autonomní (nejsou instancí jiného objektu) je následující posun nulový. Pro instance je zde však rozdíl, protože mohou mít jiné posunutí, než je uložené v matici otce, a které používají v části keyframe. Posunutí se však liší nejen pro instance, ale i pro všechny

objekty, které mají s nějakou instancí hierarchickou souvislost.

```
SoTransform *transform4 = new SoTransform();
transform4->translation.setValue(SbVec3f(pos.operator[](0) - mesh->locmatrix[9],
                                         pos.operator[](1) - mesh->locmatrix[10],
                                         pos.operator[](2) - mesh->locmatrix[11]));
transform4->center.setValue(mesh->locmatrix[9], mesh->locmatrix[10], mesh->locmatrix[11]);
meshSet->addChild(transform4);
```

2. Rotace

Nyní přichází na řadu rotace objektu. Rotace se provádí kolem lokálního souřadného systému, jehož poloha je určena v LOC_MATRIX prvky 9, 10 a 11.

```
// rotace
if( rot.operator[](0) != 0.0 || rot.operator[](1) != 0.0 || rot.operator[](2) != 0.0 ) {
  SoTransform *transform1 = new SoTransform();
  transform1->rotation.setValue(SbVec3f(rot.operator[](0),
                                        rot.operator[](1),
                                        rot.operator[](2)), -rot.operator[](3));
  transform1->center.setValue(mesh->locmatrix[9], mesh->locmatrix[10],mesh->locmatrix[11]);
  meshSet->addChild(transform1);
}
```

3. Odstranění bugu v 3ds souboru

Pokud je na objekt aplikována změna měřítka a počet záporných změn měřítek v jednotlivých osách je lichý (např. záporné měřítko v ose z nebo záporné měřítko ve všech třech osách), je v 3ds souboru špatně vytvořená geometrie. Tuto chybu lze odstranit provedením záporného scale kolem osy x. Upozorňujeme, že se nejedná o otočení o 180 stupňů, ale skutečně o scale kolem osy x!

```
// meritko - odstraneni bugu v 3ds souboru
if( (scl.operator[](0) * scl.operator[](1) * scl.operator[](2)) < 0.0 )
{
  *clockshape = 1;
  SoTransform *transform3 = new SoTransform();
  transform3->scaleFactor.setValue(SbVec3f(-1,1,1));
  transform3->center.setValue(mesh->locmatrix[9], mesh->locmatrix[10],mesh->locmatrix[11]);
  meshSet->addChild(transform3);
}
```

4. Měřítko

Dále musíme aplikovat změnu měřítka, tentokráte už podle pravidel. To znamená že zjistíme měřítko z keyframe části a aplikujeme jej na vrcholy objektu.

```
// meritko
SoTransform *transform2 = new SoTransform();
transform2->scaleFactor.setValue(scl.operator[](0), scl.operator[](1), scl.operator[](2));
transform2->center.setValue(mesh->locmatrix[9], mesh->locmatrix[10], mesh->locmatrix[11]);
meshSet->addChild(transform2);
```

5. Inverzní matice

Posledním krokem je aplikace inverzní matice lokálního souřadného systému. Toto je nezbytná akce, protože 3D Studio při exportu aplikuje transformace pivot pointu na vrcholy objektu, ale při práci ve vlastním 3D Studiu se tomu tak neděje. Následuje tedy ono provedení inverzní transformace. Dostaneme tak původní scénu, ovšem posunutou do lokálního souřadného systému (chunk PIVOT). Musíme tedy ještě provést posunutí zpět do originálního souřadného systému, abychom dostali scénu, která je shodná s tím, jak jsme ji vymodelovali ve

3D Studiu.

```
// inverzni matice k transformaci pivot pointu, jeste se musi pridat zaporne posunuti pivot
// pointu
SbMatrix matrix(
  mesh->locmatrix[0], mesh->locmatrix[1], mesh->locmatrix[2], 1,
  mesh->locmatrix[3], mesh->locmatrix[4], mesh->locmatrix[5], 1,
  mesh->locmatrix[6], mesh->locmatrix[7], mesh->locmatrix[8], 1,
  0, 0, 0, 1
);

matrix = matrix.inverse();
SoTransform *transform = new SoTransform();
transform->setMatrix(matrix);
transform->translation.setValue(-kfmesh->pivot.x, -kfmesh->pivot.y, -kfmesh->pivot.z);
transform->center.setValue(mesh->locmatrix[9], mesh->locmatrix[10], mesh->locmatrix[11]);
meshSet->addChild(transform);
```

5.5.2 Řešení geometrie včetně hierarchické struktury

V předchozí podkapitole jsme si ukázali řešení modelů, u kterých není definována hierarchická struktura. To znamená, že všechny objekty jsou samostatné a jejich lokální souřadnicové systémy jsou vzájemně nezávislé. V praxi ale existuje řada modelů, které obsahují hierarchickou strukturu, takže toto řešení je sice správné, ale nedostačující. Nás bude především zajímat, jak se změní předchozí uvedený postup při zpracování geometrie. U modelů bez hierarchické struktury platilo, že keyframe informace byly vztaženy k lokálnímu souřadnicovému systému (pozice ke globálnímu souřadnému systému). U hierarchické struktury platí, že keyframe informace jistého objektu jsou vztaženy ke keyframe informacím a tedy také lokálnímu souřadnicovému systému otce tohoto objektu. Předchozí postup zůstane zachován, ale musíme provést transformaci keyframe informací podle rodičů tak, abychom získali takové keyframe informace, které jsou stejné, jako kdyby scéna nebyla hierarchická. Musíme tedy zjisti, zda-li zpracovávaný objekt nemá nějakého rodiče. Pokud ano, pak se musíme zeptat, zdali tento rodič teké nemá nějakého rodiče, až dojdeme k objektu, který žádného rodiče nemá. Keyframe část tohoto objektu pro nás bude tvořit základní transformace, ke kterým budeme postupně přidávat keyframe transformace synů, až se dostaneme k našemu původnímu objektu, kde získáme takové keyframe informace, které budou stejné, jako kdyby celá struktura byla bez hierarchie.

K hledání rodiče jsme využili rekurze a jednotlivé keyframe informace (pozice, rotace a scale) předáváme pomocí parametrů rekurzivní funkce. Opět si dovolíme uvést komentovaný zdrojový který by měl dostatečně osvětlit pohled na provádění transformací mezi keyframe informacemi. V této části musíme provést XXX kroků.

1. Nalezení kořene objektu

Pokud zjistíme, že objekt má nějakého rodiče, najdeme tohoto rodiče. Pak se musíme zpetat, zda-li tento nalezený rodič také nemá rodiče. Pokud má také rodiče, musíme jej najít. Tento postup opakujeme rekurzivně, dokud nenajdeme rodiče, který je kořenem, to znamená, že

již žádného jiného rodiče nemá. Pak ukončíme rekurzi a vrátíme informace z keyframe části tohoto kořene.

```
p_pos->setValue(kfmesh->pos[0].x, kfmesh->pos[0].y, kfmesh->pos[0].z);
p_rot->setValue(kfmesh->rot[0].x,kfmesh->rot[0].y, kfmesh->rot[0].z, kfmesh->rot[0].angle);
p_scl->setValue(kfmesh->scale[0].x, kfmesh->scale[0].y, kfmesh->scale[0].z);
```

2. Zpětný chod rekurze (zpracování synů)

Na základě informací z kořene (rodičů) provádíme transformace s informacemi ve všech synech, až dojdeme k původnímu objektu, jehož kořen jsme hledali. Musíme opět provést několik kroků, ke správné transformaci keyframe části syna podle rodiče.

transformace pozice

Ke správné transformaci pozice syna podle rodiče, musíme na POS_TRACK_TAG (syna) aplikovat postupně transformace změny měřítka rodiče, rotace rodiče a pozice rodiče (p_pos, p_rot a p_scl).

```
// pozice rodice
SbMatrix posMatrix;
posMatrix.setTranslate(SbVec3f(p_pos->operator[](0),
                               p_pos->operator[](1),
                               p_pos->operator[](2)));
// rotace rodice
SbMatrix rotMatrix;
rotMatrix.makeIdentity();
if( p_rot->operator[](0) != 0.0 ||
    p_rot->operator[](1) != 0.0 ||
    p_rot->operator[](2) != 0.0 )
  rotMatrix.setRotate(SbRotation(SbVec3f(p_rot->operator[](0),
                                         p_rot->operator[](1),
                                         p_rot->operator[](2)),
                                 -p_rot->operator[](3)));
// scale rodice
SbMatrix sclMatrix;
sclMatrix.setScale(SbVec3f(p_scl->operator[](0),
                           p_scl->operator[](1),
                           p_scl->operator[](2)));
// postupne provedeni transformaci
SbVec3f c_pos;
sclMatrix.multVecMatrix(SbVec3f(kfmesh->pos[0].x, kfmesh->pos[0].y, kfmesh->pos[0].z),
                        c_pos);
rotMatrix.multVecMatrix(c_pos, c_pos);
posMatrix.multVecMatrix(c_pos, *p_pos);
```

transformace rotace

Při transformaci rotace syna podle rotace rodiče stačí rotační matici rodiče vynásobit rotační maticí syna a s takto získané rotační matice získat jednotlivé složky (vektor a úhel)

```
// rotacni matice syna
SbMatrix aux_matrix;
aux_matrix.makeIdentity();
if( kfmesh->rot[0].x != 0.0 || kfmesh->rot[0].y != 0.0 || kfmesh->rot[0].z != 0.0 )
  aux_matrix.setRotate(SbRotation(SbVec3f(kfmesh->rot[0].x,
                                          kfmesh->rot[0].y,
                                          kfmesh->rot[0].z),
                                  kfmesh->rot[0].angle));
```

```
// rotacni matice rodice
rotMatrix.makeIdentity();
if( p_rot->operator[](0) != 0.0 ||
    p_rot->operator[](1) != 0.0 ||
    p_rot->operator[](2) != 0.0)
rotMatrix.setRotate(SbRotation(SbVec3f(p_rot->operator[](0),
                                       p_rot->operator[](1),
                                       p_rot->operator[](2)),
                               p_rot->operator[](3)));
//slozeni rotaci
rotMatrix.multRight(aux_matrix);


// ziskani rotace z matice
rotMatrix.getTransform(pos, r, scl, so);
r.getValue(rot2, angle);
p_rot->setValue(rot2.operator[](0), rot2.operator[](1), rot2.operator[](2), angle);
```

transformace měřítka

Pro správnou transformaci scale syna potřebuje znát rotační matici syna a scale matici rodiče. Princip spočívá v tom, že scale rodiče transformujeme pomocí rotace syna tak, abychom mohli provést jednoduché vynásobnení transformovaného scale rodiče a scale syna.

```
// child rotation matrix
rotMatrix.makeIdentity();
if( kfmesh->rot[0].x != 0.0 || kfmesh->rot[0].y != 0.0 || kfmesh->rot[0].z != 0.0 )
  rotMatrix.setRotate(SbRotation(SbVec3f(kfmesh->rot[0].x,
                                         kfmesh->rot[0].y,
                                         kfmesh->rot[0].z),
                                 -kfmesh->rot[0].angle));
// slozeni rotace syna a scale rodice
rotMatrix.multRight(sclMatrix);
rotMatrix.getTransform(pos,r,*p_scl,so);
// slozeni scale syna a transformovaneho scale rodice
p_scl->setValue(p_scl->operator[](0)*kfmesh->scale[0].x,
                p_scl->operator[](1)*kfmesh->scale[0].y,
                p_scl->operator[](2)*kfmesh->scale[0].z);
```

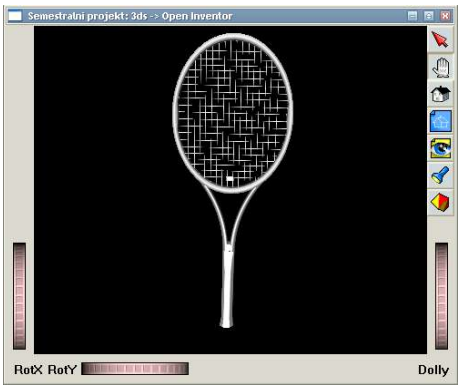
5.6 Testování

V této kapitole bychom rádi ukázali modely, které byly načítány programem vytvořeným v rámci ročníkového modelu a které byly načítány špatně nebo neúplně. Stručně popíšeme, v čem jaký byl u těchto modelů problém a jak vypadá ten stejný model, který je pomocí našeho vylepšeného programu, načtený správně.

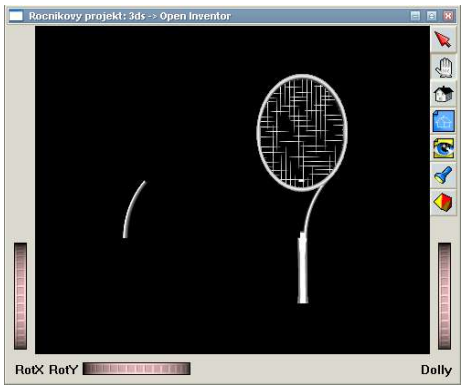
Test 1.

V tomto testu je ukázáno implementace opravení chyby v 3ds souboru. Jde o chybu v aplikaci záporného scale. Část rukojeti rakety má aplikováno v ose x záporné scale a proto je zobrazena špatně. Pokud zkusíme použít záporné scale ještě v jedné ose, pak je model zobrazen správně. Na obrázku 5.10 vidíme zmiňovaný bug v 3ds souboru – model je zobrazen programem implementovanýmv rámci ročníkového projektu. Na obrázku 5.9 je výstup programu vytvořeného na základě tototo semestrálního projektu a toto zobrazení je již správné. Dále bude vždy vlevo uvedený obrázek z programu z tohoto semestrálního projektu a vpravo

obrázek z programu z ročníkového projektu.



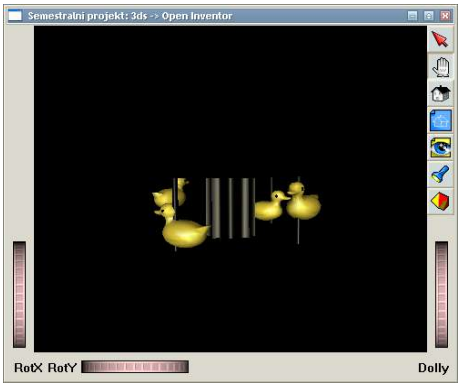
Obrázek 5.9 Model z opravou chyby v 3ds souboru (chyba v 3D Studiu)



Drawing 5.10 Model zobrazený programem, který neumí tuto chybu odstranit

Test 2.

Původní model vytvořený ve staré verzi 3D Studia. Je zobrazen správně v obou programech. Importujeme-li však tento model do 3D Studia MAX a opět exportujeme, bude již zobrazen špatně. Je to způsobeno posunutými pivot pointy. Nový program již model zobrazí správně.



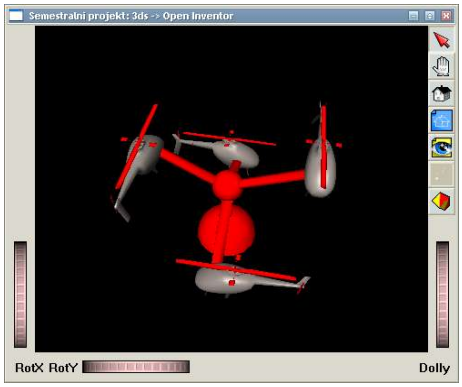
Obrázek 5.11 Správně zobrazený model kolotoče.

Obrázek 5.12 Problém s pivot pointy u staré verze programu 3ds2iv.

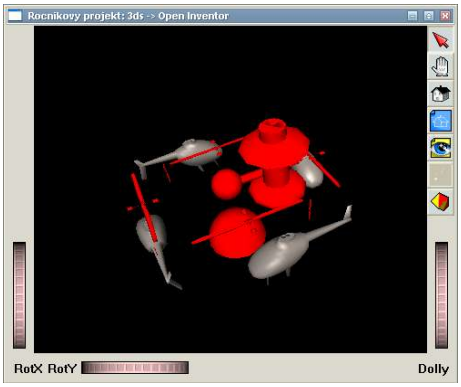
Test 3.

Model použitý v tomto testu je model, který obsahuje všechny problémy, které se podařilo vyřešit (kromě scale bugu v 3D Studiu). Jednak bylo potřeba zde vyřešit problém s herarchií, s posunem pivot pointů, problém se starou verzí 3D Studia a problém s instancemi objektů. Zde je mimo jiné vhodné místo, abychom upozornili na to, že knihovna 3dsftk od cyberloonies obsahuje chybu při načítání starých verzí 3ds souborů. Nedokáže totiž rozpoznat hierarchickou strukturu v souborech, ve kterých je vytvořená. Tato „maličkost“ je velice

důležitá a bez ní jsme nemohli nikdy modely vytvořené ve staré verzi 3D Studia zobrazit správně.



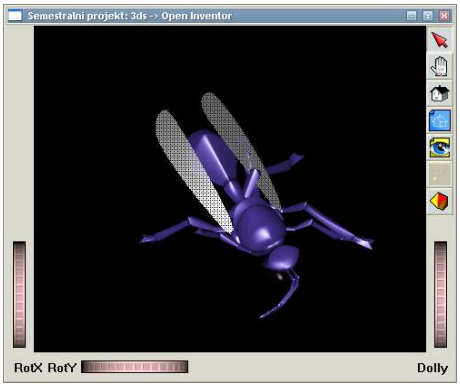
Obrázek 5.13 Správně načtený model. Implementována byla hierarchie, instance, transformace pivot pointů.



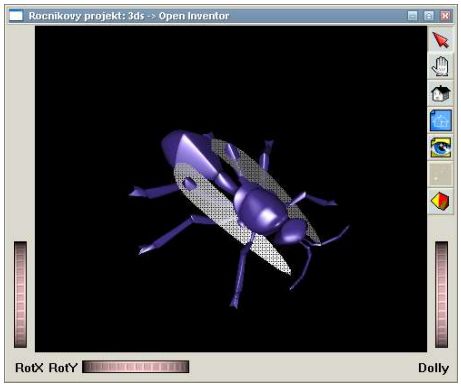
Obrázek 5.14 Model zobrazený ve staré verzi 3D Studia je zobrazen špatně.

Test 4.

Poslední test ukazuje, že ne vždy se může zdát načtený model zobrazený správně, ale opak je často pravdou. Mnoho modelů jejich autoři vytvořili a v dobré víře exportovali do 3ds formátu. Neuvědomili si však, že některé informace se mohou tímto exportem ztratit. To je případ například následujícího obrázku. Model vlevo, ač to tak nevypadá, je zobrazen správně. Model vpravo vypadá také vypadý, že je zobrazen špatně a také tomu tak je.



Obrázek 5.15 Model mravence vypadá, že je zobrazen špatně. Při importu do 3D Studia však dostaneme stejný model.



Obrázek 5.16 Model mravence zobrazený ve staré verzi 3D Studia. Mohlo by se zdát, že je model načten správně, ale ve skutečnosti je model zobrazen špatně. Správné zobrazené geometrie je na obrázku vlevo.

6 Závěr a pokračování projektu

Na tomto místě se očekává zhodnocení dosažených výsledků a případné možnosti pokračování práce na tomto projektu. Nemohu však začít jinak, než kritikou 3ds formátu. V prvé řadě musím konstatovat, že dokumentace tohoto formátu je pro potenciální vývojáře, kteří chtějí pracovat s modely v 3ds formátu, více než stručná. V podstatě jde pouze o výčet nezbytně nutných informací o binární struktuře v 3ds souboru. S tímto bych však byl ochoten se smířit, vždyť bylo úkolem předchozího ročníkového projektu se v této dokumentaci zorientovat. Ale s faktem, že 3ds soubor obsahuje chybu, kdy při použití záporného scale buď v jedné nebo ve všech třech osách (lichý počet), vede k nepochopitelnému zápornému scale kolem osy x, už se ochoten smířit nejsem. Vývojáři 3D Studia pravděpodobně o této chybě vědí, ale její odstranění není možné mimo jiné také z důvodu, protože by vznikla nekompatibilita mezi 3ds soubory. Tento fakt je také jednou z příčin, proč je 3ds formát považován za tajemný.

Dalším problémem, který přispívá k oné tajemnosti 3ds formátu, je hierarchická struktura. Bez tohoto nástroje konstrukce scény se při vytváření animace v 3D Studiu neobejdeme. Řešení tohoto problému bylo klíčové pro zobrazení mnoha modelů.

V rámci ročníkového projektu byl řešen problém s texturovacími souřadnicemi. Šlo v podstatě o to, že pokud jsme například pro kouli vygenerovali texturovací souřadnice, došlo k roztržení tohoto objektu (fyzickému roztržení) a tím pádem ke špatné práci se smoothing groups. Považoval jsem to za problém interpretace práce se smoothing groups. Pravda však byla jiná. Toto fyzické roztržení modelu se projeví i při importu této koule zpět do 3ds souboru. Není to tedy chyba špatného načtení a interpretace dat z 3ds souboru, ale je to vlastnost 3ds formátu. Tento problém byl vyřešen a programově dojde k opětovnému spojení roztrženého objektu. Tuto vlastnost programu lze tedy považovat za rozšíření načítání 3ds souboru.

Oproti původnímu ročníkovému projektu byl program vylepšen po stránce správy paměti. Nyní by měla knihovna i převodník uvolňovat všechnu zabranou paměť (knihovna po zrušení objektu metodou releaseModel() a převodník po dereferencování kořene scény). Celková paměťová náročnost se znížila cca o 30 procent.

Poslední důležitá informace se týká knihovny 3dsftk. Dovoluji si tvrdit, že tato knihovna obsahuje chybu při práci se starými verzemi 3ds souborů. Nedokáže totiž nalézt rodiče těch objektů, které je mají. Odhalení této chyby lze připsat tomu, že zadáním tohoto semestrálního projektu byla mimo jiné implementace vlastní knihovny. To považuji za přínos. Musím však ocenit i knihovnu 3dsftk, bez které by byla práce na projektu mnohem obtížnější. Celkový dojem z této knihovny však kazí ona zmíněná chyba, která byla zdrojem mnoha nejsaností a problémů.

Přes všechny překážky, které se v průběhu práce na projektu objevovaly, se nám podařilo vytvořit program, jehož parametry jsou, troufnu si říci, jedinečné. Jsou mi známy pouze dva programy (tedy spíše jejich pluginy), ale bude jich zajisté více, které dokážou načíst

3ds soubor stejným způsobem jako ten náš. Je to program PolyTrans a program 3D Studio. Oba jsou vsou však komerční a přístup k jejich zdrojovým kódům je tedy nemožný. A právě proto jsme se pustili do analýzy tohoto formátu, abychom umožňili široké vývojářské komunitě pracovat efektivně a hlavně správně s tímto formátem. S tímto faktem souvisí také směr další práce na projektu, kterým bude vypracování diplomové práce a napsání odborného článku, ve kterém se pokusíme shrnout informace získané o 3ds formátu. Vhodné by bylo dokončit načítání dalších objektů (kamera, světlo) z 3ds souboru. Také bude pokračovat vývoj knihovny tak, aby umožňovala práci s objekty, které již budou transformované podle hierarchické struktury a zbavené všech problémů objevujících se v 3ds formátu. Tedy aby se dala knihovna využít například pro napsání jiného převodníku z formátu 3ds. Také zůstáváme trochu dlužni podrobnější vysvětlení transformací, které se dějí v rámci keyframe části a hierarchické struktury a dokumentaci naší knihovny lib3ds. Náplní diplomové práce tedy bude odstranit tyto nedostatky a vytvořit tak ucelený dokument pojednávající o 3ds formátu a sloužící zároveň jako jeho podrobná dokumentace. Už nyní však můžeme říci, že tajemství 3ds formátu je odhaleno!

7 Přílohy

7.1 Seznam použité literatury

[MoPoGr98] Jiří Žára, Bedřich Beneš, Petr Felkel. Moderní počítačová grafika. Computer Press, 1998

[3ds497] David Řeháček. 3D Studio 4.0/MAX. Computer Press, 1997

[3ds604] Jan Kříž. 3ds max 6 Praktické postupy. Computer Press, 2004

[OpenIv] The Inventor Mentor: Programming Object Oriented 3D graphics with Open Inventor, Release 2.

[OpIvC] Open Inventor Architecture Group. Open Inventor C++ Reference Manual. Addison-Wesley, 1994

[Link1] http://www.paulsprojects.net/tutorials/simplebump/simplebump.html
Odkaz na úvod do bump mappingu a normálových map.

[Link2] http://www.root.cz/clanky/open-inventor/
Seriál článků o Open Invetoru od Ing. Jana Pečivy.

[Link3] http://www.coin3d.org/
Coin 3D developers page

[Link4] http://www.ati.com/developer/tools.html
Utility pro práci s texturami a pro generování normál od firmy ATI.

[Link5] http://www.opengl.org/
Sobelův filtr pro výpočet významu (magnitudy) a orientace hran v obraze.

[Link6] http://www.cyberloonies.com/3dsftk.html
Zdrojové soubory knihovny 3D Studio File Toolkit a oficiální dokumentace 3ds formátu od Autodesku.

7.2 Popis ovládání programu

```
"Usage: 3ds2iv [options] infile [outfile]
  -e      : Use Examine viewer to show the model
  -f      : Compute normals using flat shading
  -h      : Print this message (help)
  -s      : Force two sided materials
  -t      : Do not apply textures
  -v      : Do not create outfile, only use Examine viewer to
            show the model

   If no output file name is specified, stdout will be used.
```