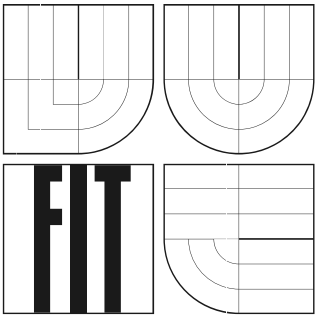
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



Modelování 3D scény a její vizualizace

Bakalářská práce

2006 Tomáš Seiner

Modelování 3D scény a její vizualizace

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Jana Pečivy.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Tomáš Seiner
27.4.2006

Abstrakt

Tento dokument pojednává o vypracování bakalářské práce na téma modelování 3D scény a její vizualizaci se zaměřením na animaci scény. Jsou tu popsané části programu, způsoby vytváření modelů, textur a celé scény. Také způsoby exportu modelů, a jejich stručný popis. Podrobně se tu popisují různé způsoby animací v Open Inventoru.

Klíčová slova

Modelování, model, modelovací prostředí, 3dsMax, export modelů, transformace, translace, rotace, Open Inventor, Coin3d, VRML, animace, rotor, interpolator, proximity sensor, pohyb kamery.

Poděkování

Chtěl bych poděkovat hlavně panu Ing. Janu Pečivovi, za pomoc a spousty času strávených nad řešením mých problémů.

Abstract

This document concerns about elaboration bachelor's work themed "Creation 3D scene and its visualization" with a directionality to scene animations. There are explained parts of program, the different ways of creating models, textures and whole scene. Also the ways of exporting models and their description. The different ways of scene animations in Open Inventor will be describe in detail.

Keywords

Modeling, model, modeling environment, 3dsMax, models exporting, transformation, translation, rotation, Open Inventor, Coin3d, VRML, animation, rotor, interpolator, proximity sensor, camera movement.

Obsah

Úvod

1.1 Úvod do problematiky

Počítačová grafika se od svých počátků v sedmdesátých letech vyvinula do samostatného, rozsáhlého a významného oboru. Ten se těší velkému zájmu profesionálů, programátorů a studentů coby vědní disciplína. Počítačová grafika byla ve svých prvních desetiletích využívána pouze v technických oborech, nyní s ní však pracuje vědomě i nevědomě každý uživatel počítače.

Pěknou grafiku ocení hlavně hráči počítačových her, ale v dnešní době proniká také hodně do filmového průmyslu, kde způsobila velký rozvoj. Počítačové efekty, kterých by se jinak nedalo dosáhnout, jsou v dnešní době natolik realistické, že je jen s těží dokážeme rozpoznat.

Jedna z nejdynamičtěji se rozvíjejících oblastí počítačové grafiky je trojrozměrná počítačová animace. Nejjednodušším příkladem počítačové animace je prosté zobrazování dynamické scény, nejčastěji metodou sledování paprsku. Objekty, které jsou součástí scény se stejně jako kamera pohybují. V diskrétních časových okamžicích sestavíme model scény a ten pak zobrazíme. Tento jednoduchý příklad je věrnou kopií toho, jak pracuje klasická animace, kde je dynamická scéna taktéž snímána v diskrétních časových krocích kamerou, Zjevnou výhodou počítačové animace je, že nemusíme mít žádnou scénu fyzicky k dispozici. Nemusíme stavět žádné kulisy ani chodit nikam do exteriérů, ale můžeme animovat v klidu a v teple.

1.2 Přehled dokumentu

Modelování scény

V této kapitole se probírá volba nejvhodnějšího software pro tvorbu modelů, jejich způsoby vytváření a export.

Vytváření scény

Tato část dokumentu se věnuje vytváření scény. Především třídám, pomocí kterých se jednotlivé objekty umísťují ve scéně.

Animace

Tato kapitola popisuje animační metody použité v bakalářské práci. Jejich podrobné vysvětlení a několik příkladů.

2 Modelování scény

2.1 Modelovací prostředí

Předtím než jsem začal pracovat bylo nutné se rozhodnout pro to, jaký modelovací software bude nejvhodnější pro vytvoření objektů ve scéně. Z těch dostupných a vhodných stojí za zmínku akorát 3ds Max, Cinema 4D a Maya.

Cinema 4D a Maya jsou vhodné, jak už z názvu vyplývá, pro filmovou tvorbu. Jednoduše se v nich pracuje s high-poly modely a nesmíme také zapomenout na kvalitní tvorbu materiálů. A i když by se v těchto programech scéna animovala velice dobře, tak bohužel neexistuje formát ve kterém by se z těchto programů dala celá scéna s animací uložit do souboru, který by zároveň podporovaly knihovny Open Inventoru.

3ds Max má velkou podporu v herním průmyslu a je proto specializovaný na práci s low-poly modely. Také mapovací nástroje jsou velmi dobře propracované, což ušetří spoustu času. Jediný z nedostatků tohoto studia je tvorba materiálů.

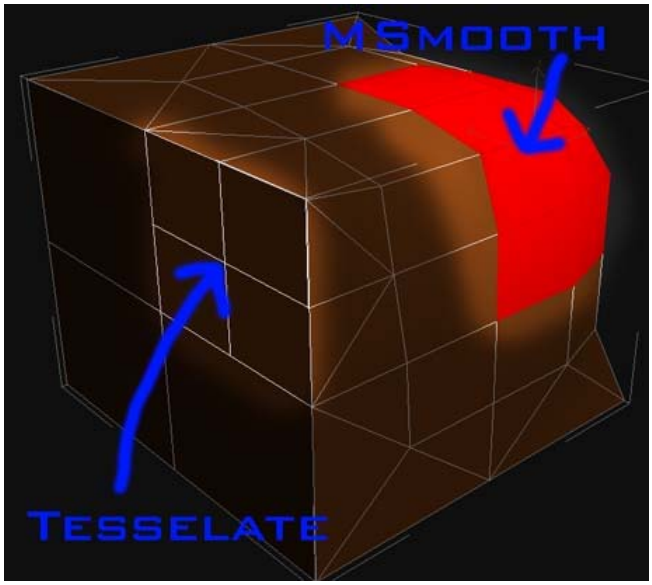
2.2 3dsMax

Nakonec jsem tedy zvolil studio 3dsMax 7, jelikož je specializovaný na práci s low-poly modely a to je přesně to co jsem potřeboval k vytvoření modelů ve scéně.Má velmi jednoduché uživatelské prostředí, rychlý přístup ke všem nástrojům a hlavně velkou podporu na internetu.

2.2.1 Vytváření modelů

Všechny modely jsem vytvářel nástrojem *Editable poly* [4]. Tento nástroj objekt rozdělí na jednotlivé body a hrany a v jednotlivých dalších režimech lze tento objekt měnit. Nejdůležitější je vždy zvolit správný prvotní tvar (kvádr, kužel, atd.), který se bude modifikovat.

Nejdříve se podíváme na ty nástroje, které můžete většinou použít bez ohledu na to, ve kterém Sub-Object režimu se nacházíte.



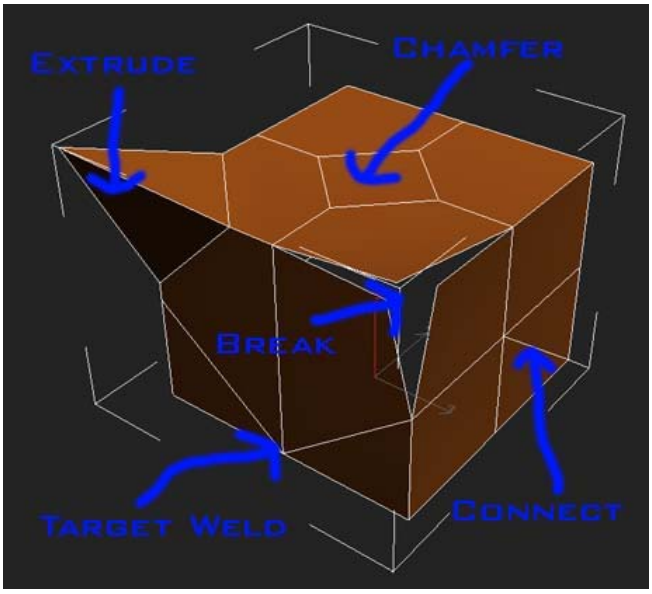
- **Attach** – připojí do sítě jakýkoliv jiný objekt ze scény
- **Quick** Slice – rozdělí hrany a polygony a vytvoří tak nové vrcholy
- **Cut** – dělá skoro to samé, jen můžeme síť rozdělit několika řezy
- **MSmooth** – zahustí a zaoblí vybrané objekty, nebo celou síť
- **Tesselate** – síť nebo vybrané podobjekty pravidelně rozdělí

Obr.1: Všeobecné nástroje

2.2.1.1 Sub-Object Vertex

Tento režim pracuje s vrcholy. Vlastnosti těch nejpoužívanějších nástrojů:

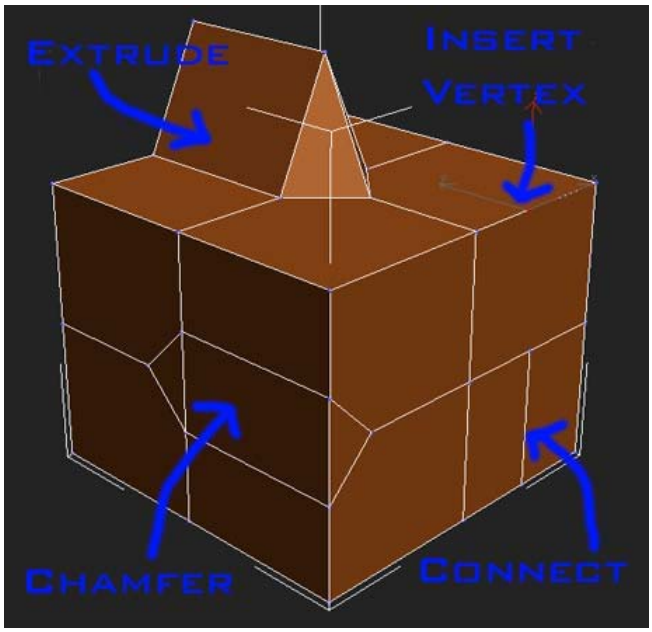
- **Remove** – odstranění vrcholu ze sítě, aniž by se smazaly okolní polygony
- **Break** – rozdělení vrcholu a oddělení tak k sobě připojené hrany a polygony
- **Extrude** – vytáhne vrchol
- **Chamfer** – rozdělí vrchol a vytvoří nový polygon
- **Weld** – spojí vrcholy blízko sebe
- **Connect** – vytvoří nové hrany mezi vrcholy
- **Colaps** – spojí všechny vybrané vrcholy



Obr.2: Nástroje Sub-Object Vertex

2.2.1.2 Sub-Object Edge

Režim pro práci s hranami je velmi podobný nástroji *Sub-Object Vertex*.



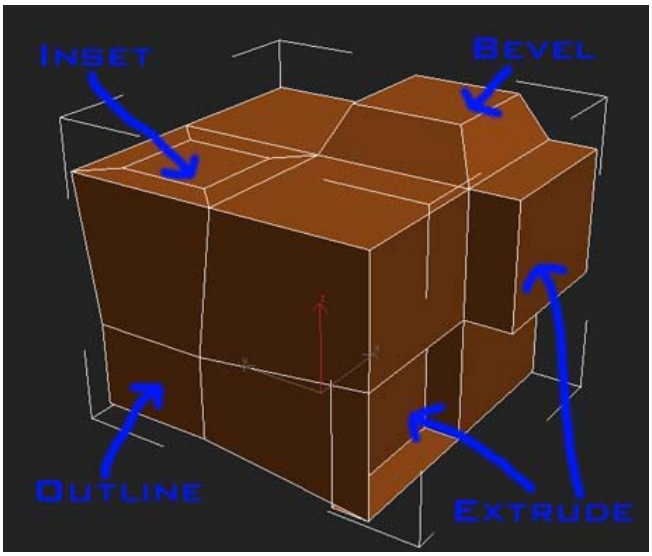
Obr.3: Nástroje Sub-Object Edge

- **Remove** – odstranění hrany ze sítě, aniž by se smazaly okolní polygony
- **Extrude** – vytáhne hranu
- **Chamfer** – rozdělí hranu a vytvoří nový polygon
- **Weld** – spojí hrany v selekci blízko sebe
- **Connect** – vytvoří nové hrany mezi hranami
- **Colaps** – spojí všechny vybrané hrany

2.2.1.3 Sub-Object Polygon

A asi nejvíce používaný režim je pro práci s polygony.

- **Extrude** – vytáhne/zatáhne polygon
- **Outline** – zvětší/zmenší polygon
- **Bevel** – spojení *Extrude* a *Outline*
- **Inset** – vloží další polygon
- **Retriangulate** – přehodí rozložení trojúhelníků určující tvar nerovinných polygonů
- **Flip** – přehodí normálu polygonu
- **Make** Planar – upraví polygony tak, aby byly ploché a ležely v jedné rovině



Obr.4: Nástroje Sub-Object Polygon

2.2.2 Vytváření textur

Jelikož 3dsMax nepodporuje tvorbu textur, tak jsem používal program TextureMaker Trial Version. V tomto programu lze vytvářet textury velice snadno. Bohužel pouze jen do určité velikosti obrázku. Spousta generátorů textur usnadní práci a všechny další funkce jsou velmi snadno použitelné. Výsledné textury se poté velice snadno importovali do 3dsMaxu.

2.2.3 Export modelů

Jelikož knihovny Open Inventoru podporují pouze formát *iv* nebo *wrl* a 3dsMax ukládá ve formátu max, je zapotřebí data exportovat, nebo zkonvertovat do příslušných formátů. Teď si popíšeme pár formátů.

2.2.3.1 Formát max

Formát *max* je standardní formát s kterým pracuje 3dsMax. Do tohoto formátu se ukládají veškeré informace, které jsou potřeba pro práci s modely, jejich animace atd. Jsou v něm uloženy i úrovně struktury modifikátorů, které umožňují zjednodušený přístup k různým fázím a detailům modelování. Jednoduše jsou v něm uloženy veškeré informace o celé scéně, kromě textur. Tento formát jsem využil pouze k ukládání modelů v 3dsMaxu při jejich modelování.

2.2.3.2 Formát 3ds

Tento formát je zjednodušená forma formátu *max* a používá se především k exportu modelů, takže o nich uchovává pouze základní informace. Jsou v něm uloženy informace o vrcholech, jejich seskupení, jednodušší animační schémata, jako je např. zrcadlení a také v sobě nese mapovací informace, které zjednodušují práci s texturami. Tento formát jsem využil při exportování složitějších modelů, které se chybně exportovali do *wrl*.

2.2.3.3 Formát wrl

Formát wrl je odvozen od VRML [7]. VRML je v současné době jediný mezinárodně uznávaný formát pro popis virtuálních světů. Zkratka VRML vznikla ze slov Virtual Reality Modelling Language. VRML je jednak popisným jazykem a jednak formátem textového souboru. Je zajištěna kompatibilita při přenosu dat i export z jiných systémů (typicky CAD). Úsilí o vylepšení jazyka je koncentrováno. Je zajištěna dostupnost specifikace.

Zde je stručná charakteristika VRML:

- Podporuje se hraniční reprezentace objektů.
- Scéna je organizována do stromové struktury tzv. zhroucený strom (umožňuje dědění).
- Lze tvořit zcela nové parametrické objekty.
- Scénu můžou tvořit jak prvky umístěné v lokálních souborech, tak i na vzdálených počítačích v internetu.
- Virtuální světy lze vkládat do stránek HTML.
- Jsou podporovány prostředky pro popis animace objektů a interakce s uživatelem.
- Kromě prostorových objektů je možné vkládat i multimediální prvky (video,obraz,zvuk).
- Je textovým formátem nezávislým na výpočetní platformě.
- Obsahuje prostředky pro řízení rychlosti zobrazování v závislosti na konkrétním výkonu počítače.

Z historického hlediska byl předchůdcem VRML jazyk OpenInventor, vyvinutý pro počítače Silicon Graphics. Tento jazyk byl v roce 1995 doplněn o možnost využívat odkazy na internet a byl nazván VRML 1.0. Od této chvíle se datuje historie VRML. V roce 1995 byla také založena skupina programátorů VAG (VRML Architecture Group), která si určila následující základní cíle:

- Vytvořit prostředky pro popis statických světů.
- Vytvořit prostředky pro popis dynamických a interaktivních světů. Toto byl nejbližší cíl k plnění.
- Vytvořit prostředky umožňující spolupráci více uživatelů ve virtuálním světě.

Jako další milník lze považovat rok 1996, kdy byla vybrána z osmi různých návrhů společná specifikace firem Silicon Graphics a Sony s pracovním názvem Moving Worlds, která se stává základem jazyka VRML 2.0. Poté se v roce 1997 z neformální VAG stává oficiální sdružení VRML Consortium, Inc. Dále je také zahájena spolupráce s mezinárodní standardizační organizací ISO. Tato spolupráce vrcholí v dubnu roku 1997 vznikem VRML v podobě mezinárodní normy, která získává jméno VRML 97. Jako poslední důležitý okamžik v krátké historii VRML je rok 1999, kdy se VRML Consortium přejmenovává na Web3D Consortium. Hlavním cílem Web3D Consortia je vytvoření obecného formátu nazvaného X3D, určeného pro popis, přenos a prezentaci prostorových dat na WWW.

3 Vytváření scény

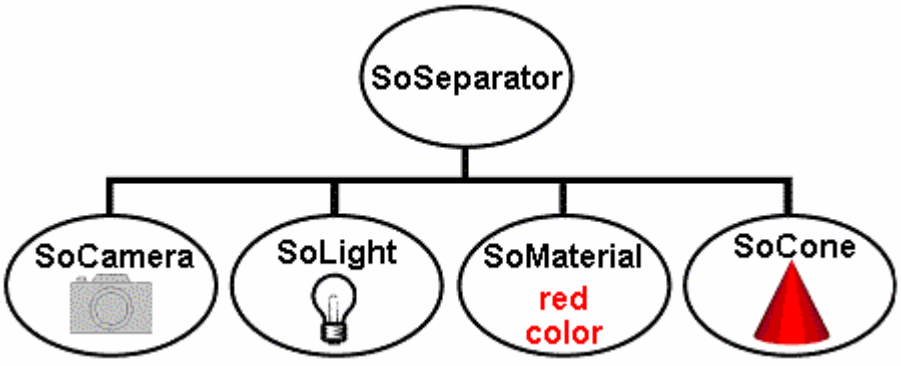
Jelikož jsou jednotlivé modely roztroušeny v samostatných souborech formátu wrl nebo iv, je zapotřebí je vložit do jedné scény. V každém souboru je model umístěn na pozici x = y = z = 0. Z toho vyplívá že se musí modely také posunout a také většinou natočit do správné pozice ve scéně. Téměř celá scéna je vytvořena v souboru „scene.iv“, jehož formát si lehce popíšeme.

3.1 Open Inventor

Open Inventor [6] je velmi populární knihovna pro tvorbu realtimové 3D grafiky, tedy i her. Programátorovi poskytuje rozsáhlou množinu C++ tříd, které skrývají před programátorem vlastní OpenGL API a posunují ho na mnohem vyšší úroveň. Tak může programátor mnohem rychleji vyvinout to, co potřebuje. Navíc, aplikace napsané v Open Inventoru jsou obyčejně rychlejší než ty přímo psané v OpenGL, protože před předáním zobrazené scény OpenGL knihovna provádí její optimalizaci.

Open Inventoru je knihovna napsaná v C++ a postavená nad OpenGL, která posunuje programátora od primitivního OpenGL rozhraní na vyšší úroveň a nabízí mu rozsáhlou množinu C++ tříd. Ta podstatně zjednodušuje práci programátora a dokonce často poskytuje vyšší výkon než přímá implementace v OpenGL. Vyšší výkon je možný díky jistým optimalizacím, které Open Inventor může provádět nad daty scény. Běžný programátor také obyčejně nemá čas provádět profilování a optimalizaci renderovacích algoritmů. Proto již vyprofilované rutiny Inventoru nejsou špatnou volbou.

Design Open Inventoru vychází z konceptu grafu scény. Tedy, scéna je složena z uzlů - anglicky nodes. Nody jsou různých typů. Jedny nesou informace o geometrii těles (krychle, kužel, model tělesa), další různé atributy (barva, textury, souřadnice objektu) a také existují speciální nody, které obsahují seznam jiných nodů, anglicky zvané groups. A právě tyto grupy umožňují organizovat ostatní nody do hierarchických struktur zvaných grafy. Takovýto graf nám pak reprezentuje naši scénu.



Obr.5: Příklad grafu v Open Inventoru

3.1.1 Třída Transform

Geometrická transformace je nejrozšířenější a nejčastěji používaná operace v počítačové grafice. Ty můžeme rozdělit na transformace lineární a nelineární. K lineárním patří např.: posunutí, otočení, zkosení, zvětšení, atd. K nelineárním transformacím patří např.: deformace. Speciální transformací je projekce. Je to převod vícerozměrných objektů na méněrozměrné. Nejčastější je převod trojrozměrného objektu na dvojrozměrný, aby se mohl např.: zobrazit na obrazovce nebo vytisknout.

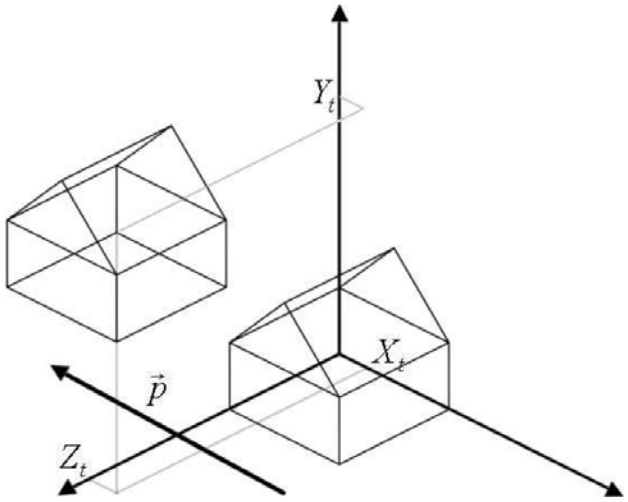
Objekty jsou v 3D reprezentovány vrcholy. Ty jsou definovány souřadnicemi, které se vztahují k určitému souřadnicovému systému. Geometrické transformace se můžou aplikovat na jednotlivé souřadnice, nebo určitou skupinu vrcholů. Při posunutí se transformují všechny vrcholy stejnou transformační maticí. Při otáčení jsou vrcholy ovlivňovány v závislosti na vzdálenosti od středu otáčení. A podobně jsou na tom i další operace lineární transformace.

3.1.1.1 Translation – Posunutí

Posunutí je v 3D určené vektorem posunutí $\vec{p} = (X_t, Y_t, Z_t)$, který se dá v OpenInventoru vyjádřit pouze zadáním souřadnic x, y, a z, které určují kam se má objekt posunout.

Příklad formátu zápisu a výchozí hodnota:

```
Transform {
        translation 0 0 0
    }
```



Obr.6: Posunutí

3.1.1.2 Rotation – otočení

Otáčení je v OpenInventoru dáno vektorem $\vec{r} = (x, y, z)$ a úhlem otočení α. Hodnoty ve vektoru nabývají hodnot od 0 až po 1 a určují o kolik a kolem které osy bude objekt natočen.

Příklad formátu zápisu a výchozí hodnota:

```
Transform {
        rotation 0 0 1  0
    }
```

4 Animace

A teď konečně k hlavnímu tématu této bakalářské práce. Rozpohybování objektů ve scéně se dá naprogramovat spousty způsoby. Jako první a nejjednodušší animaci pomocí třídy rotor, která umožňuje nepřetržitou rotaci tělesa. Další pomocí třídy Interpolator. Její podtřídy dovolují nastavit transformaci nebo rotaci v určitý časový okamžik. A jako poslední je animace pomocí třídy ProximitySensor, která spouští animaci na určitý podmět, jako může být třeba pohyb kamery do blízkosti dveří.

4.1 Rotor

Tato třída se používá k animaci rotace. Zadává se jednotkový vektor, úhel a rychlost. Z vektoru se pozná, jestli se bude objekt otáčet klem osy x, y nebo z a z úhlu se pozná jakým směrem se bude otáčet a v jaké pozici začne. Pokud je úhel větší než nula tak se bude rotovat proti směru hodinových ručiček a pokud bude úhel menší než nula, tak se bude otáčet po směru hodinových ručiček. Rychlost se zadává v sekundách a určuje za jak dlouho se objekt otočí kolem své osy.
Příklad formátu zápisu a výchozí hodnota:

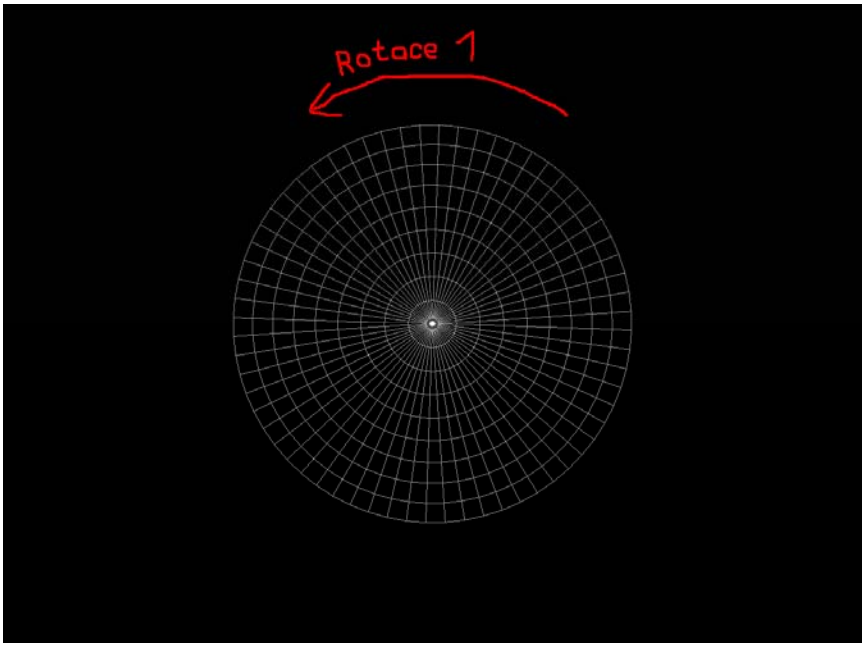
```
Rotor {
        rotation 0 0 1  0
        speed 1
        on TRUE
      }
```

4.1.1 Příklad

Toto je jeden příklad z programu. Jde o satelit obíhající kolem stanice. Ve výsledné scéně jsou tři satelity a fázově posunuté.

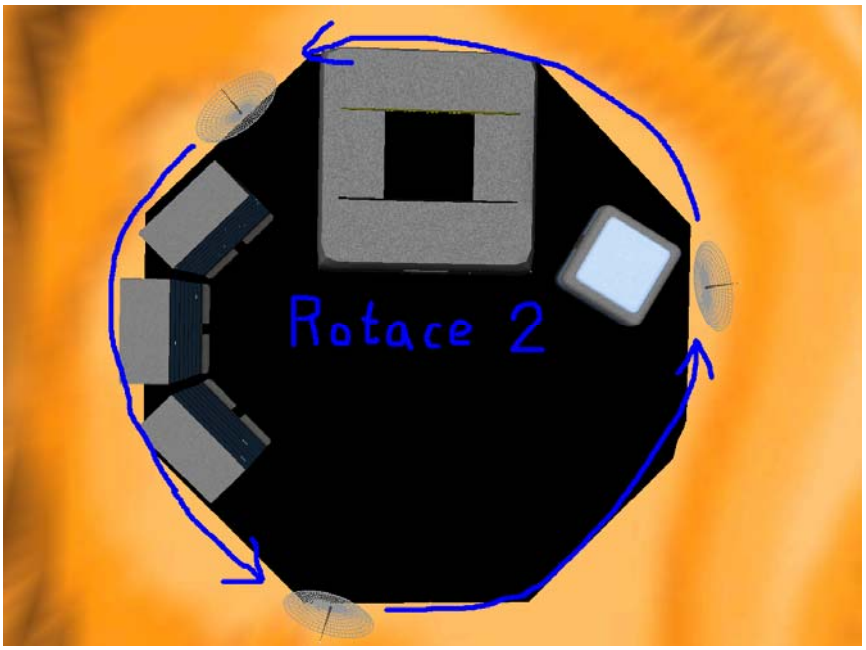
```
Separator{
      Rotor { rotation 0 1 0  2.0944 speed 0.02 on TRUE }
      Separator{
            Transform { translation 120 70 0 rotation 0 0 1  -0.7854 }
            Rotor { rotation 0 1 0  3.14159 speed 1 on TRUE }
            File{ name "satelit.wrl" }
      }
}
```

První animace rotace je zvýrazněná červeně. Jde se o jednoduché otáčení podél vlastní osy. Ze zápisu vyplívá, že se bude otáčet kolem osy y (viz. obr.) a že se otočí jednou za sekundu.



Obr.7:Animace rotace č.1

Druhá animace je zvýrazněna modře a jejím výsledkem je obíhání satelitu kolem celé základny. Jelikož v poduzlu je satelit posunut pomocí třídy transform, tak se tentokrát nebude otáčet kolem vlastní osy y, ale kolem osy, která je v centru souřadnicového systému. A podle rychlosti poznáme, že satelit základnu oběhne za $0.02^{-1} = 50$ s.



Obr.8:Animace rotace č.2

4.2 Interpolator

V této třídě jde hlavně o dvě podtřídy a to CoordinateInterpolator a OrientationInterpolator. Pomocí těchto tříd je naprogramována většina animací. Jde o složitější animování, kde se zadávají jednotlivé časy a zároveň pozice nebo rotace. Tudíž jde o časově a programově delší operaci.

Definice CoordinateInterpolator:

```
CoordinateInterpolator {
  eventIn      SFFloat set_fraction        # (-∞,∞)
  exposedField MFFloat key           []    # (-∞,∞)
  exposedField MFVec3f keyValue      []    # (-∞,∞)
  eventOut     MFVec3f value_changed
}
```

Do pole key se zadávají jednotlivé časy za sebou oddělené čárkou. Jde o pole floatů. Do pole keyValue se zadávají vektory oddělené čárkou. Vektory se zadávají jako 3 floaty oddělené mezerou.

Definice OrientationInterpolator:

```
OrientationInterpolator {
  eventIn      SFFloat    set_fraction      # (-∞,∞)
  exposedField MFFloat    key           []  # (-∞,∞)
  exposedField MFRotation keyValue      []  # [-1,1],(-∞,∞)
  eventOut     SFRotation value_changed
}
```

Pole key je stejné jako u CoordinateInterpolator, do keyValue se zadává vektor a úhel, tzn. 4 floaty oddělené mezerou.

Float set_fraction definuji pomocí třídy TimeSensor a to jen jednou pomocí DEF. Dále všude kde je potřeba použít TimeSensor stačí použít USE.

Definice TimeSensor a výchozí hodnota:

```
TimeSensor {
    exposedField SFTime   cycleInterval 1       # (0,inf)
    exposedField SFBool   enabled       TRUE
    exposedField SFBool   loop          FALSE
    exposedField SFTime   startTime     0       # (-inf,inf)
    exposedField SFTime   stopTime      0       # (-inf,inf)
    eventOut     SFTime   cycleTime
    eventOut     SFFloat  fraction_changed      # [0, 1]
    eventOut     SFBool   isActive
    eventOut     SFTime   time
  }
```

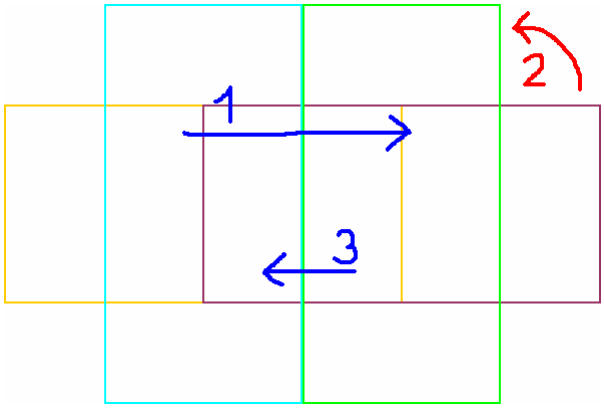
CycleInterval je doba, jak dlouho se bude stopovat, loop se nastavuje pokud chceme, aby se stopovalo stále dokola.

4.2.1 Příklad

Na tomto zjednodušeném příkladu si ukážeme jak lze vytvořit animaci pomocí Interpolátoru.

```
Separator{
     Transform {
          translation = SelectOne {
               type SoMFVec3f input = VRMLCoordinateInterpolator {
                    set_fraction = DEF timeSrc VRMLTimeSensor {
                         cycleInterval 100 loop TRUE}.fraction_changed
                    key [ 0, 0.25, 0.65, 0.9 ]
                    keyValue [ 0 0 0, 0 0 5, 0 0 5, 0 0 0 ]
               }.value_changed
          }.output
         rotation = SelectOne {
               type SoMFRotation input = VRMLOrientationInterpolator {
                    set_fraction = USE timeSrc.fraction_changed
                    key [ 0, 0.25, 0.65,0.9 ]
                    keyValue [ 0 1 0 0, 0 1 0 0, 0 1 0 1.57, 0 1 0 1.57]
               }.value_changed
          }.output
     }
     File{name "plosina.wrl"}
}
```

Následující obrázek ukazuje, jaké jsou jednotlivé kroky animace (vychází z předchozího kódu).



Obr.9: Synchronizace posunu a otočení

Nejprve bych vysvětlil nastavení TimeSensoru. Pomocí příkazu DEF timeSrc se definuje proměnná timeSrc s nastaveným intervalem na 100 sekund. To je schválně, protože hodnoty v key nejsou v sekundách, ale v procentech, tudíž při nastavení intervalu na 100 sekund se jedna sekunda rovná jednomu procentu. Dále je nastaveno neustálé opakování pomocí loop. Pokud chceme použít toto nastavení časovače pro další modely, stačí použít příkaz USE timeSrc.

A teď k průběhu animace.

- 0 s Na začátku je model na pozici 0 0 0
- 0 – 25s Lineární posun na pozici 0 0 5
- 25 – 65s Otáčení podle osy y o π/2
- 65 – 90s Lineární posun zpět na pozici 0 0 0

4.3 ProximitySensor

Tato třída není přímo na animaci, ale používá se ke generování událostí když kamera vstoupí, nebo vystoupí z dané oblasti.

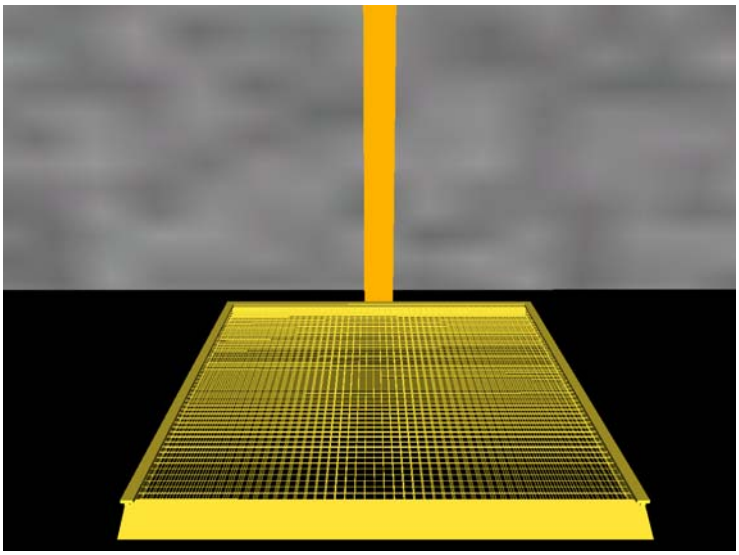
```
ProximitySensor {
    exposedField SFVec3f    center      0 0 0    # (-,)
    exposedField SFVec3f    size        0 0 0    # [0,)
    exposedField SFBool     enabled     TRUE
    eventOut     SFBool     isActive
    eventOut     SFVec3f    position_changed
    eventOut     SFRotation orientation_changed
    eventOut     SFTime     enterTime
    eventOut     SFTime     exitTime
  }
```

Pokud chceme, aby byl ProximitySensor aktivní, musí být enabled nastaveno na TRUE. Pro nastavení aktivního pole se používají proměnné center a size. Size je vektor, který určuje velikost snímacího pole. Je definován jako kvádr s šířkou x, výškou y a hloubkou z. Polohu této oblasti určuje vektor center.

Pokud jsme v tomto poli, je generována událost isActive = TRUE, jinak FALSE. Událost enterTime je generována pokaždé, když do tohoto pole vstoupíme a událost exitTime je generována pokaždé, když z tohoto pole odejdeme. Události position_changed a orientation_changed jsou generovány při změně pozice či orientace uvnitř pole. Tyto události jsou do stromové hierarchie vloženy pouze pokud je isActive TRUE a poté jsou zase odebrány.

4.3.1 Výtahy

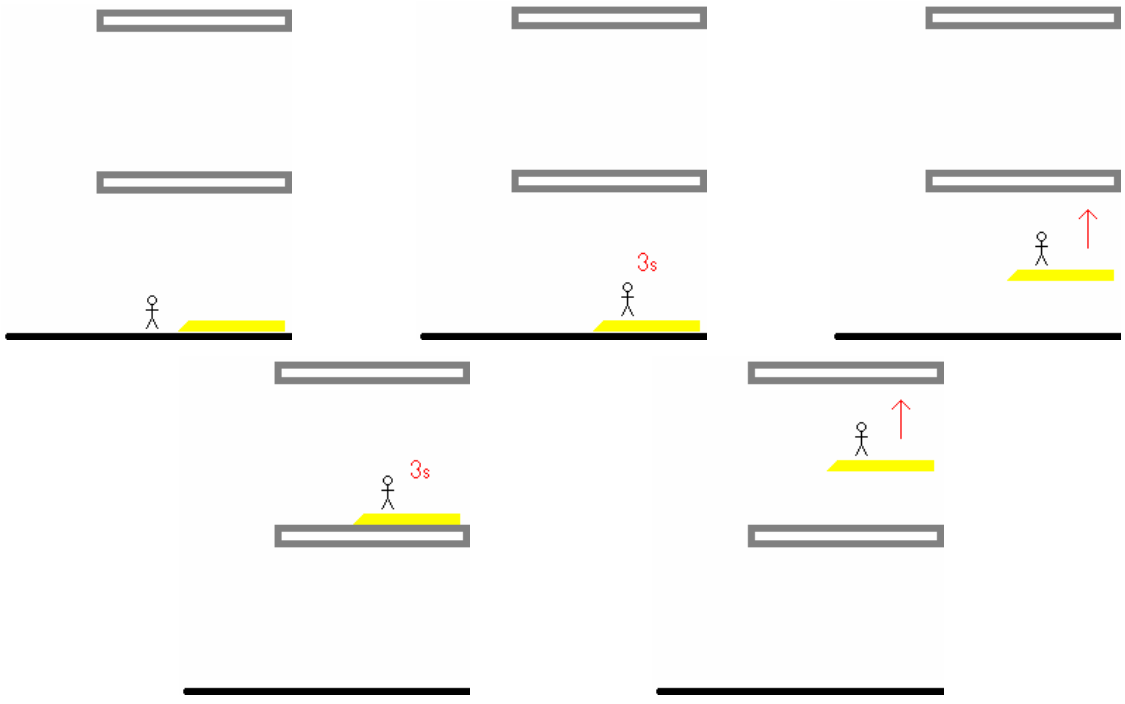
V hlavním hangáru jsou umístěny 4 výtahy. Na každé straně je jeden takový, jako vidíte na obrázku.



Obr.10: Výtah

Samozřejmě je nad každým výtahem umístěn ProximitySensor, který nad ním kontroluje pohyb. Pokud nad výtahem počkáme 3 sekundy, tak se rozjede a my s ním. Výtah má tři zastávky. Pokud na další zastávce vystoupíme výtah počká a když v něm zůstaneme, tak se rozjede dál.Když nastupujeme na prostřední zastávce, tak se výtah rozjede tím směrem, jakým jel naposled. Pokud z výtahu vystoupíme za jízdy, tak se zastavíme, ale výtah pokračuje až do další zastávky.

Na následujícím obrázku je postup, jak se dostat do 2. patra.



Obr.11: Jak se dostat do druhého patra

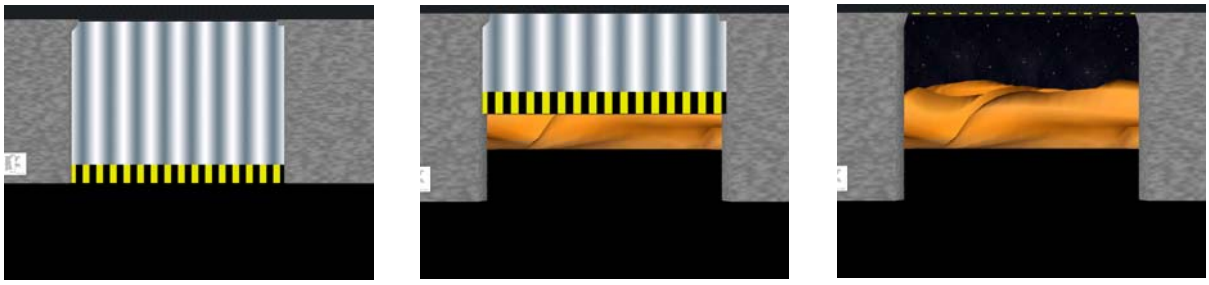
4.3.2 Hlavní brána

Tento zjednodušený příklad naznačuje, jak naprogramovat otevírání dveří. Je vytažen z větší části zdrojového kódu, takže v něm nejsou definovány všechny proměnné, ale na vysvětlení principu je to postačující.

```
SoVRMLProximitySensor *GlVrataSensor = new SoVRMLProximitySensor();
GlVrataSensor->center.setValue(-0.0f, 7.5f, -29.5f);
GlVrataSensor->size.setValue( 20, 15, 60);
root->addChild(GlVrataSensor);


static void timeStep(void *data, SoSensor *sensor)
{
     if(GlVrataSensor->isActive.getValue()==TRUE){
          if(BVrataMove){
               GlVrata->translation.getValue().getValue(x,y,z);
               if(y<=15)
                    GlVrata->translation.setValue(x,y+0.05f,z);
          }
          else
               BVrataMove = true;
     }
     else if(BVrataMove){
          GlVrata->translation.getValue().getValue(x,y,z);
          if(y>=0)
               GlVrata->translation.setValue(x,y-0.05f,z);
          else
               BVrataMove = false;
     }
}
```

Nejprve se musí vytvořit a nastavit ProximitySensor. Nová proměnná GlVrataSensor tohoto typu je vytvořená a nadefinovaná na prvních čtyřech řádcích. Funkce timeStep se provádí každých několik setin. Nejdříve se zjistí, jestli jsme v dané oblasti před dveřmi. Pokud ano, tak se vrata budou zvedat až do určité výšky. Nejprve se musí získat jejich aktuální pozice a teprve poté posunout. Pokud z této oblasti odejdeme a vrata jsou v pohybu, tak se zase začnou zavírat. Postup je stejný, jako při otevírání vrat.



Obr.12: otevírání hlavní brány

4.4 Kamera

Aby bylo možné si prohlédnout celou scénu, bylo zapotřebí naprogramovat pohyb a otáčení kamery. Na pohyb i otáčení jsou nastaveny události na některé z tlačítek na klávesnici.

4.4.1 Pohyb

Pro pohyb byli vybrány tlačítka šipka nahoru a dolů, pro pohyb dopředu a dozadu a Insert a Delete pro úkroky doprava a doleva a pro pohyb nahoru a dolů jsou tlačítka Home a End. Pro zrychlení nebo zpomalení pohybu jsou určena tlačítka '+' a '-'. Nejdůležitější bylo najít správnou funkci pro chození, aby když zmáčknete šipku dopředu, aby jste šli vždy tam kam se koukáme, i když kameru otočíme jinam.
Následující zdrojový kód ukazuje, jakým způsobem je vyřešeno chození dopředu.

```
if(SoKeyboardEvent::isKeyPressEvent(event, klavesy[0].klavesa)){
     GlCamera->orientation.getValue().getValue(vektor,alpha);
     vektor.getValue(x,y,z);
     alpha *= y;
     GlCamera->position.getValue().getValue(x,y,z);
     x -= RP * sin(alpha);
     z -= RP * cos(alpha);
     GlCamera->position.setValue(x,y,z);
}
```

Nejprve si zjistíme otočení kamery a vypočítáme se odchylku α. Poté si načteme pozici a od souřadnice x se odečte sin(α) a od souřadnice z se odečte cos(α). Obě rovnice jsou násobeny proměnou RP, která určuje rychlost pohybu.

Ostatní směry jsou řešeny podobným způsobem, akorát se mění znaménka.

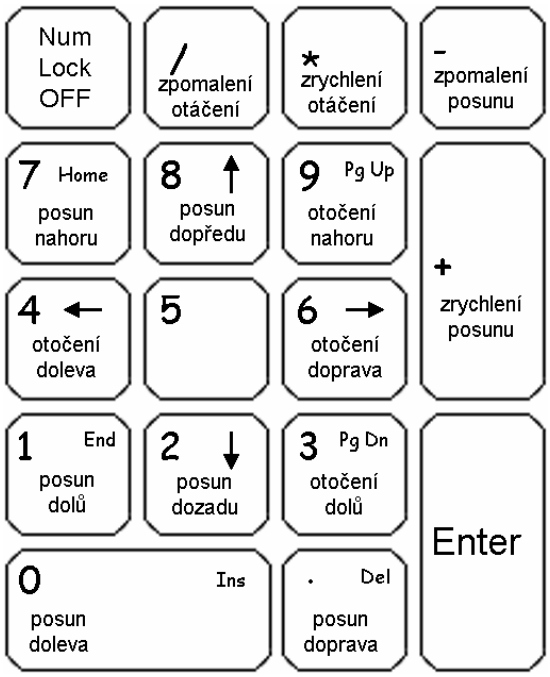
4.4.2 Otáčení

Otáčení doleva a doprava je realizováno pomocí šipek doleva a doprava. Nahoru a dolů pomoci PageUp a PageDown. Pro zrychlení či zpomalení otáčení lze použít tlačítka '*' a '/'. Pokud chceme kameru otočit do výchozí pozice, stačí stisknout 'r'.

```
if(SoKeyboardEvent::isKeyPressEvent(event, klavesy[8].klavesa)){
     GlCamera->orientation.getValue().getValue(vektor,alpha);
     vektor.getValue(x,y,z);
     beta = y*alpha;
     GlCamera->orientation.setValue(SbRotation(vektor,alpha) *
}
```

4.4.3 Ovládání

Kamera je ovládána klasickým způsobem pomocí šipek.Na obrázku je vidět, jak se dá jednoduše pohybovat za pomocí numerické klávesnice.



Obr.13:Ovládání kamery

Další užitečné klávesnice:

R…….Reset kamery – nastavení natočení na 0, tzn. pohled po záporné straně osy z.

I……...Info – přesune a natočí kameru k informační tabuli.

Esc……..Přepínání mezi režimy prohlížení scény v prohlížeči.

5 Závěr

V této bakalářské práci jsem se toho spoustu naučil, protože to byla moje první práce s grafickými knihovnami. Vymodeloval jsem spoustu modelů, a vyzkoušel si různé modelovací techniky. U vytváření animací scény jsem strávil spoustu času, protože je to časově náročné a bohužel jsem ani nemohl naplnit své představy, kvůli slabé hardwarové podpoře. Tak jsem se zaměřil na zjednodušování modelů, a snižování náročnosti pomocí knihoven LOD, ale bohužel už nebylo moc času, a tak je tato část projektu pouze rozpracovaná. V budoucnu bych se rád pokusil snížit náročnost této aplikace a trochu ji ještě vylepšit přidáním dalších modelů.

Literatura

[1] Pečiva, J., Open Inventor: Knihovna pro realtimovou 3D grafiku. *Root.cz*. dostupné z www: http://www.root.cz/clanky/open-inventor/

[2] Ondřej, Š., Principy virtuální reality, VRML. dostupné z www: http://netra.felk.cvut.cz/~apg/apg-tutorials02/ch09s220.html

[3] Palguta, M., Animated Models, dostupné z www: http://merlin.fit.vutbr.cz/upload/IvProjects/2005/AnimatedModel-doc.pdf

[4] Rohr, P., Modelování pomocí Editable Poly. g*rafika.cz.* dostupné z www: http://www.grafika.cz/art/3dscena/3dsmax-editable-poly.html

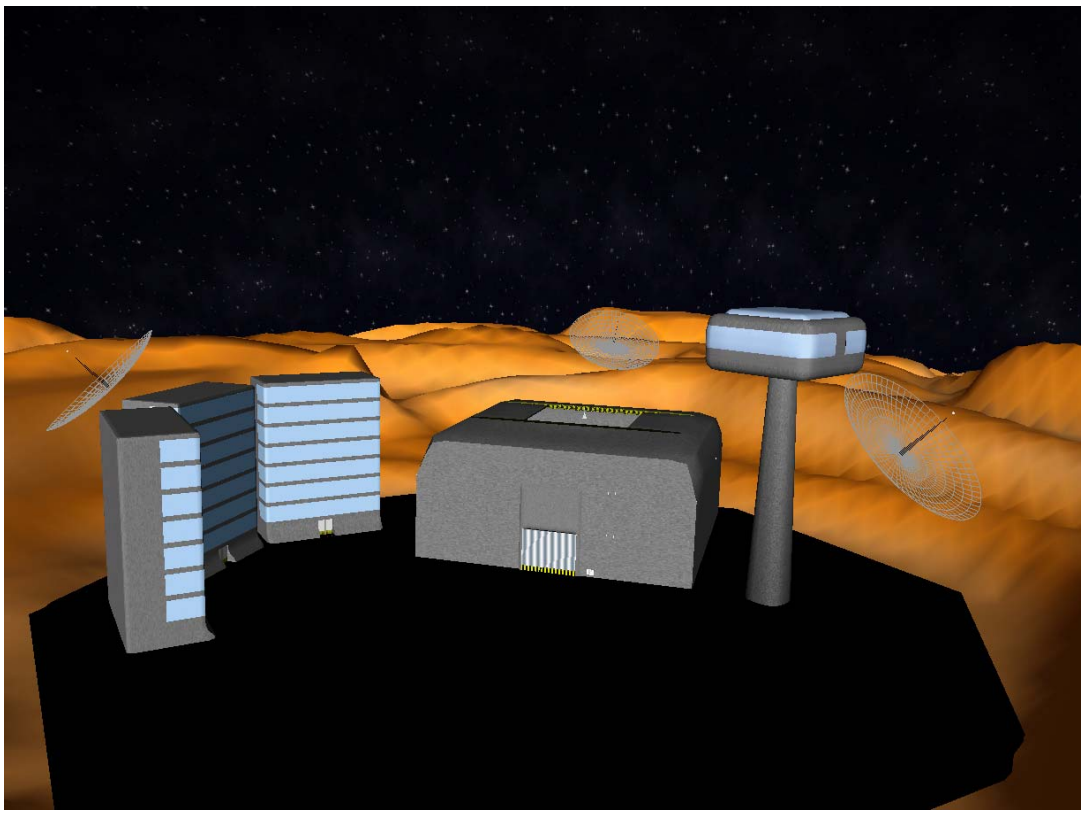
[5] Coin Documentation. dostupné z www: http://doc.coin3d.org/Coin/

[6] Open Inventor. dostupné z www: http://oss.sgi.com/projects/inventor/

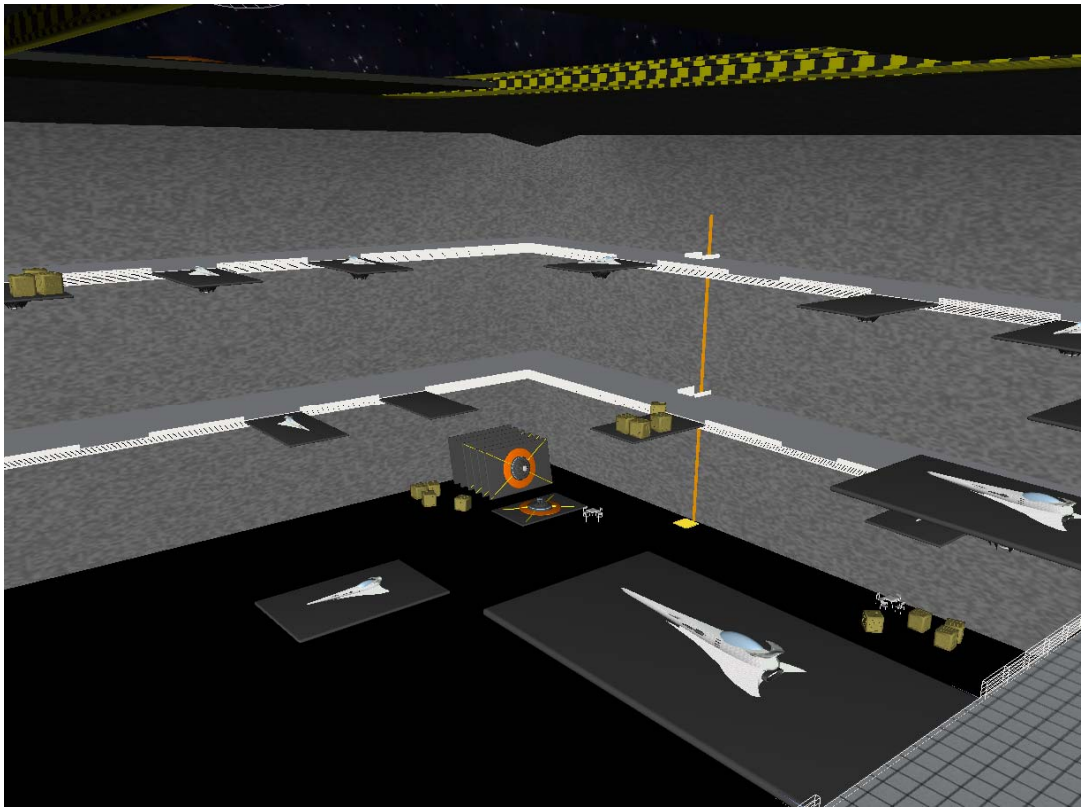
[7] VRML97 Documentation. dostupné z www: http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-VRML97/

[8] Žára, J., Beneš, B., Felkel, P., *Moderní počítačová grafika*. 1. vydání. Computer Press 2004

Obrazová příloha 1

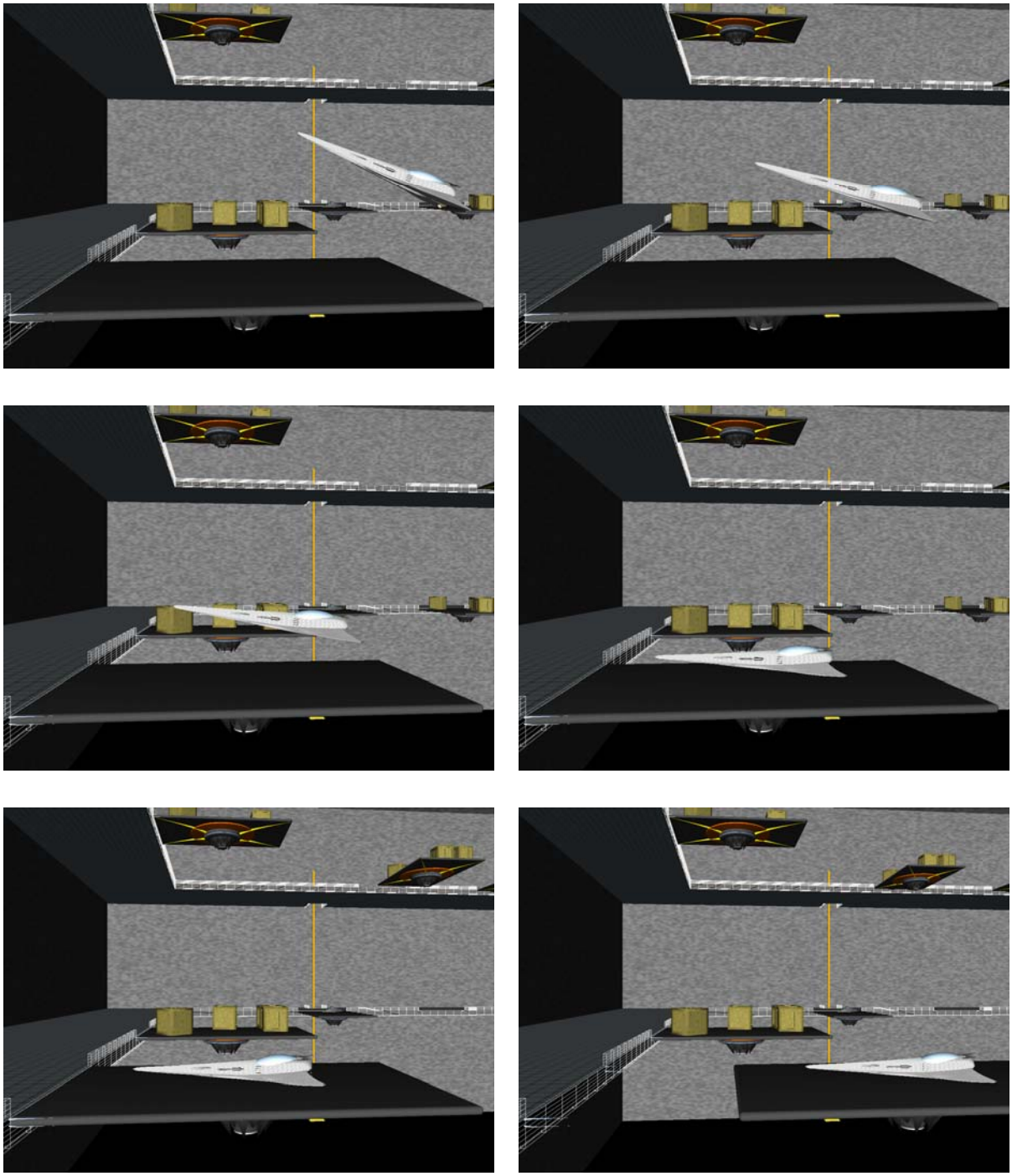


Obr. 14: Celá scéna



Obr. 15: Hangár

Obrazová příloha 2



Obr.16: Přistání raketky