České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačové grafiky a interakce

Bakalářská práce

Generátor modelů budov a interiérů

Denisa Hůlová

Vedoucí práce: Ing. Roman Berka, Ph.D.

Studijní program: Softwarové technologie a management, bakalářský

Obor: Web a multimedia

2011

Poděkování

Na tomto místě bych ráda poděkovala vedoucímu bakalářské práce Ing. Romanu Berkovi, Ph.D. za pozornost, kterou věnoval mé práci a za jeho odborné rady a pomoc při psaní.

Prohlášení

Prohlašuji,

že jsem tuto bakalářskou práci na téma „Generátor modelů budov a interiérů“ vypracovala zcela samostatně a veškerou použitou literaturu a další podkladové materiály, které jsem použila, uvádím v soupisu bibliografických citací.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č.121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne . . . . . .

Abstract

The aim of this thesis „The generator models of buildings and interior design“ was to create an application – using inputted data - that is able to create a simple 3D model using a repetition of the scene.

The first part of the thesis contains a brief introduction and looks at the theory, the documentation of the language used to create scene and a description of the application. The second section examines the implementation of Java applications.

Abstrakt

Cílem této bakalářské práce na téma „Generátor modelů budov a interiéru“ bylo vytvořit aplikaci, která na základě vstupních dat bude schopna vytvořit jednoduchý 3D model pomocí opakování jedné části celé scény.

První část je teoretická a obsahuje stručný úvod do problematiky, dokumentaci použitého jazyka pro tvorbu scén a popis aplikace. Druhou částí je samotná implementace aplikace v jazyce JAVA.

Obsah

Seznam obrázků

Seznam tabulek

1 ÚVOD

1.1 Motivace

Téma „Generátor modelů budov a interierů“ vzniklo na základě vyvíjeného projektu „Virtual Libra(ry“ skupiny umělců. Autorem tohoto projektu je Petr Šourek jehož cílem je zpřístupnit veřejnosti nedostupná historická místa pomocí virtualní technologe CAVE.

V dnešní době moderních technologií je tehdejší náplň pracovníků Ústřední evidence sociální pojišťovny pro mnoho z nás těžko představitelná. V přeneseném slova smyslu oni byli ti, kdo plnili dnes tak jednoduché SQL (=Structured Query Language) dotazy v rámci databáze, kterou zde představuje určitý sektor kartotéky, který měl každý z nich na starosti.

Náplní projektu Virtual Libra(ry je přiblížit široké veřejnosti pracoviště tehdejší Ústřední evidence sociální pojišťovny, nyní ústředí České správy sociálního zabezpečení v Praze na Smíchově *(obr. 3.1)*.

Unikátní kartoteční blok je dílem architektů Františka A. Libry a Jiřího Kana z let 1935-1936. Na první pohled obyčejná, nijak estetický zajímavá budova, je uvnitř tvořena 50 metry lískovnic otočené zády k sobě a z každé strany obsluhujících devíti posuvnými jeřáby. Tento naprosto dokonale promyšlený a bezchybně architektonicky navržený systém je v dnešní době, i přesto, že je veřejnosti nepřístupný, považován za světový unikát.

Jednou z možných technologií pro realizaci tohoto projektu je technologie CAVE (=Cave Automatic Virtual Environment). CAVE představuje virtuální prostředí, kde za pomoci několika projektorů se na 3-6 stěn promítá virtuální obraz a divák se tak může ponořit do virtuálního světa.

Projekt Virtual Libra(ry se snaží toto vrcholně modernistické dílo přiblížit návštěvníkovi a odhalit nenápadný půvab archivu. Za tímto účelem vznikne interaktivní scénář hry Virtual Libra(ry.

Obrázek 1.1: Pracoviště Ústřední evidence sociální pojišťovny

1.2 Současný stav

V současné době je velké množství projektů zabývajících se podobným problémem, tedy popsat realitu nějakou konkrétní gramatikou (vysvětlení později) a na základě ní vygenerovat 3D model. V této kapitola bych chtěla ukázat několik projektů zabývajících se automatickým generováním 3D staveb za pomoci gramatik.

1.2.1 Projekt „The LaHave House“

Projekt LaHave House je navazující prací na výzkumný projekt Fakulty architektury a počítačové vědy na Technické univerzitě Nova Scotia v Kanadě.

Projekt se zabývá automatickým generováním domů na základě gramatik. Tyto gramatiky byly popsány v roce 1996 v knize *The lahave house project: Towards an automated architectural design service*. Andrewem Rau-Chaplinem, Brianem MacKay-Lyonsem a Peterem F. Spierenburgem, která se společně s architekturou v údolí řeky LaHave v Kanadě stala hlavní inspirací pro tento projekt. Gramatiky jsou založeny na předem dané knihovně architektonických prvků sloužících k vizualizaci 3D scény. Hlavní nevýhodou těchto gramatik je neschopnost vypořádat se s novými prvky, které nejsou v databázi definovány. I přesto se s její pomocí dá vytvořit velké množství dostatečně podrobných modelů.

Technologiemi pro realizaci projektu a vizualizaci 3D staveb se stal jazyk JAVA a VRML. Ukázky generovaných modelů jsou na obrázku 1.2.

Více o tomto projektu viz [1].

Obrázek 1.2: Modely generované systémem LaHave House

1.2.2 Projekt „Wall Grammar For Building Generation“

Jedním z dalších projektů, jehož základním stavebním kamenem je právě gramatika, nese název „Wall Grammar For Building Generation“ jehož autory jsou Mathieu Larive a Veronique Gaildrat. Tento projekt přináší novou techniku při tvorbě automaticky generovaných exteriérů budov. Vytvořit model budovy, který se co nejvíce podobá realitě vyžaduje mnoho hodin práce. Tento fakt byl možná jednou z motivací pro vznik tohoto projektu a schopnost rychle a věrohodně generovat modely budov je určitě obrovským přínosem pro tvorbu například celých takto generovaných měst.

Základem této práce je předdefinovaná stavební šablona, která určuje budovu a skládá se ze tří částí, a to samotného průčelí, střechy a půdy, na které budova stojí. Systém pak na základě informací ze strany uživatele vygeneruje 3D model konkrétní stavby. Projekt se zabývá hlavně generováním průčelí. Ostatní informace k vytvoření celé budovy jsou převzaty z jiných informačních zdrojů.

Hlavní inspirací tohoto projektu byla rozsáhlá teorie gramatik popsána v knize Instant Architecture Peterem Wonkou, M. Wimmerem, F. Sillionem a W. Ribarskym v roce 2003 (dále jen Wonka), která umožňuje popsat jakoukoliv geometrii obsaženou na průčelí budov. Wonka se zaměřuje na definování exteriéru stavby jen pomocí geometrických tvarů, což nemusí být vždy snadné a rozsáhlost gramatik tak, jak jí uvádí Wonka vyžaduje určitý čas k učení, aby byl uživatel vůbec schopný zvládnout složitost systému. Vytvoření „Wall Grammar“ sice využívá poznatků od Wonky, ale široká použitelnost gramatiky je pro tento projekt stále zbytečná. Proto je navržena tak, aby co nejvíce zjednodušila konkrétní podobu budovy, případně využila jejích opakujících se elementů. Také umožňuje zavedení již dříve vytvořených 3D objektů jako jsou např. balkony nebo římsy.

Systém, který takto generuje určité modely staveb, může být dobrým

nástrojem např. pro archeology nebo historiky umění k snadnému a efektivnímu vytvoření historických památek ve virtuální realitě. Více o tomto projektu viz [2].

Ukázka vygenerovaných staveb za pomoci tohoto systému je na obrázku 1.3.

Obrázek 1.3: Modely generované systémem projektu Wall Grammar

1.2.3 Shrnutí

Na základě tohoto průzkumu současného stavu podobně zaměřených aplikací vyplývá, že samozřejmě existuje celá řada gramatik popisujících skutečnou realitu a následné převedení do reality virtuální. Ve většíně případů jsou tyto projekty velmi rozsáhlé a také velmi detailní a opírají se o již existující gramatiky navržené pro tvorbu architektonických prvků. Snahou této bakalářské práce je vytvořit jednoduchý univerzální nástroj pro tvorbu 3D scén.

1.3 Cíl práce

Cílem této bakalářské práce je zjednodušit práci při tvorbě rozsáhlých scén určitého typu. V projektu Virtual Libra(ry, na základě kterého byla navržena, bude možné vytvořit rozsáhlou kartotéku nadefinováním jen jedné její části, která se zopakuje v prostoru. A právě pro tyto typy scén bude aplikace sloužit. Jedná se o ne příliš složité opakující se modely.

Uživatel by měl mít k dispozici jednoduchý přehledný jazyk, s jehož pomocí bude schopný vytvořit scénu dle svých požadavků, aniž by musel mít jakékoliv pokročilé technické vzdělání. Tento jazyk by měl ze strany uživatele vyžadovat jen nezbytné minimum informací o použitých objektech a o vše ostatní už se postará samotná aplikace. Ta bude provádět především výpočty hodnot udávajících rozmístění jednotlivých objektů ve scéně, případně jejich velikost.

2 POŽADAVKY NA APLIKACI

Dle metodiky Unified Proces, která určuje proces vývoje SW, je vhodné rozdělit požadavky na skupinu funkčních a skupinu nefunkčních požadavků. Do první skupiny spadají požadavky, které definují co má systém dělat. Do skupiny nefunkčních požadavků poté, jak to má systém dělat, případně definují různá omezení.

2.1 Funkční požadavky

1. výběr ze základních geometrických tvarů VRML jazyka

Uživatel má na výběr z několika klíčových slov zastupujících příslušný geometrický tvar, tzn. Box – krychle, Cylinder – válec, Sphere – koule a Cone - kužel

2. definování základních vlastností pro každý objekt
3. určení pozice objektu
4. určení pozice pomocí klíčových slov (top/bottom/right/left/front/back)

Další možnost zadání transformace posunutí bude pomocí klíčových slov, tzn. top – nahoře, bottom – dole, right – vpravo, left – vlevo, front – vpředu, back – vzadu.

5. určení barvy objektu pomocí klíčových slov

Barva objektu je zadána pomocí předdefinovaných klíčových slov zastupujících konkrétní barvy. Je také možné si nadefinovat vlastní barvy.

6. určení opakování po lokálních osách x, y, z
7. definování viditelnosti konkrétního objektu

Objektu lze přiřadit způsob zobrazování při opakování, kdy se objekt buď zobrazí v každém zopakovaném prototypu nebo bude jeho viditelnost náhodná.

8. pojmenování objektu
9. nadefinování iterací pro vytvoření scény

2.2 Nefunkční požadavky

Samotná aplikace bude za pomoci této XML struktury realizovat:

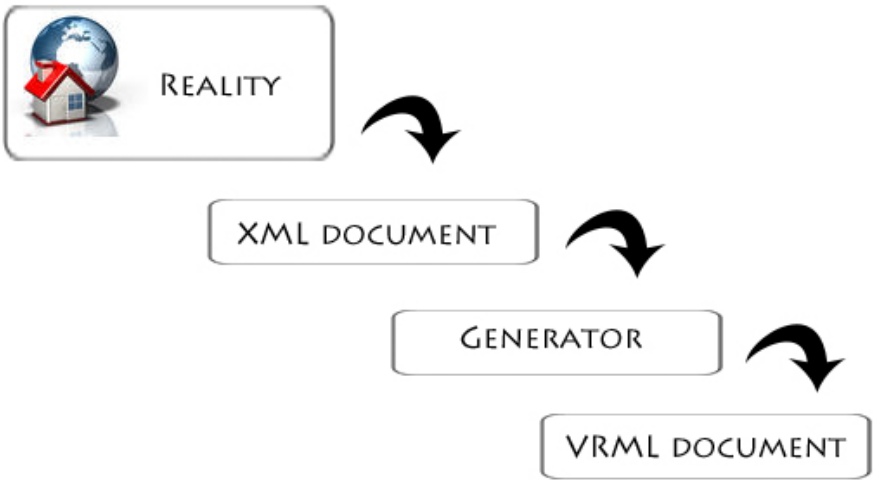
1. Platformní nezávislost systému.
2. Systém umožní načtení scény ve formátu XML.

3. Systém bude umět ziterovat model a tím vytvořit scénu.
4. Systém bude umět sám vygenerovat pozici objektu na základě klíčových slov (top/bottom/right/left/front/back)
5. Systém bude umět vypočítat náhodnou velikost objektu na základě vstupních informací od uživatele.
6. Systém bude umět určit viditelnost konkrétního objektu na základě vstupních informací od uživatele.
7. Systém bude umět vygenerovat konstrukci LOD mechanismu tak, jak je definována specifikací VRML.
8. Systém bude umět uložit vygenerovanou scénu ve formátu VRML

Jako součást této práce bude několik vzorových scén jako ukázka a návod na používání aplikace spolu s podrobnou dokumentací.

2.3 Workflow diagram

Následující obrázek znázorňuje workflow celého procesu od popisu objektu tak, jak existuje v reálném světě až po vygenerování VRML dokumentu představující konečný 3D model. Více o jednotlivých krocích tohoto diagramu v následujících kapitolách.



Obrázek 2.1: Workflow diagram

3 ANALÝZA A NÁVRH ŘEŠENÍ

Tato kapitola se zabývá především analýzou všech teoreticky možných řešení pro konečnou implementaci a udává obecné informace o použitých technologiích. Následně zde popíšu podobu XML struktury a také rozhraní použité pro vytvoření parseru.

3.1 Úvod do gramatik

V prvopočátcích vývoje bylo třeba vymyslet gramatiku, která by za pomoci předdefinovaných přepisovacích pravidel vytvořila požadovaný výsledek, tedy VRML model. V tuto chvíli bych tedy ráda přiblížila co to gramatika je, jak je definovaná a který typ je vhodný pro tuto aplikaci.

3.1.1 Základní pojmy

Nejdříve si vysvětlíme základní pojmy, které gramatiky běžně využívají.

Každý jazyk je vymezen dvěma základními pojmy – abeceda a řetězec.

Definice

Každá neprázdná množina prvků se nazývá abeceda a jednotlivé prvky množiny nazýváme symboly této abecedy.

Definice

Nechť x je nějaká konečná posloupnost symbolů nad abecedou Σ. Potom posloupnost x nazýváme řetězec (také slovo) abecedy Σ.

Posloupnost, která neobsahuje žádný symbol, nazýváme prázdná posloupnost, neboli prázdný řetězec a značíme ε.

Nechť Σ je abeceda. Potom Σ^* je množina všech řetězců nad abecedou Σ včetně prázdného řetězce ε. Symbolem Σ^+ označíme množinu všech řetězců nad abecedou Σ kromě prázdného řetězce ε. Potom platí $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$.

Definice

Konkatenací rozumíme zřetězení dvou řetězců x a y nad abecedou Σ. Konkatenací tedy vznikne nový řetězec xy. Operace konkatenace je vždy asociativní, tj. x(yz) = (xy)z, nikoliv však komutativní, tj. xy ≠ yx.

Gramatika

Každá gramatika je definovaná množinou gramatických pravidel, které slouží pro generování řetězců daného jazyka, který příslušná gramatika popisuje.

Gramatika G je definovaná jako čtveřice symbolů G = (Π, Σ, P, S), kde:

Π – je konečná množina neterminálních symbolů, určující pomocné proměnné jako syntaktické celky.

Σ – je konečná množina terminálních symbolů, které představují abecedu, nad kterou je konkrétní jazyk definován.

Sjednocením těchto dvou množin dostáváme slovník gramatiky.

P – je konečná množina přepisovacích pravidel typu A → α, kde A je neterminál a α je řetězec terminálů a/nebo neterminálů

S – představuje počáteční symbol patřící do množiny neterminálů Π

3.1.2 Chomského hierarchie

Chomského hierarchie formálních jazyků byla zavedena roku 1956 americkým matematikem Noamem Chomskym. Formálním jazykem označíme takovou množinu, kde každé slovo je konečné. Tzn. má konečnou délku nad určitou abecedou.

Gramatiky v Chomského hierarchie existují ve čtyřech typech – typ 0, typ 1, typ 2 a typ 3. Následující tabulka udává přehled jazyků a nejjednodušší možný automat, který rozpozná daný typ.

Typ v Chomského hierarchii	Jazyk	Automat
Typ 0 (frázová gramatika)	Rekurzivní	Turingův stroj
Typ 1 (kontextová gr.)	Kontextový	Lineárně ohraničený Turingův stroj
Typ 2 (bezkontextová gr.)	Bezkontextový	Zásobníkový automat
Typ 3 (regulární)	Regulární	Konečný automat

Tabulka 3.1: Chomského hierarchie gramatik

Typ 0

Gramatiky typu 0 nazýváme také gramatikami neomezenými, jelikož pravidla množiny P jsou v nejobecnějším možném tvaru a neklademe na ně žádná omezení.

Tyto jazyky jsou rozpoznatelné tzv. Turingovým strojem. Pojem Turingův Stroj zavedl v roce 1936 Alan Turing. Turingův stroj T je definován jako šestice T = (Q, A, q, e, F, δ) kde:

Q – je konečná množina vnitřních stavů

A – je konečná pásková abeceda, tzn. konečná množina symbolů a znaků

q – je počáteční stav patřící do množiny Q

e – určuje prázdný symbol

F – je podmnožinou množiny Q a vyjadřuje množinu koncových stavů

δ – značí přechodovou funkci

Turingův stroj je složen z pásky a z čtecí/zapisovací hlavy. Je to jednoduché abstraktní výpočetní zařízení, které rozpozná, zda daný problém se za jeho pomoci dá nebo nedá vyřešit. Při výpočetním procesu dochází k přechodu ze stavu do stavu a nahrazování symbolů, pokud v přechodové funkci existuje přechod pro aktuální stav. Takto stroj pokračuje, dokud nedojde k přechodu na konečný stav a zjištění výsledku nebo je ve stavu, kde není definován přechod. V takovém případě výpočet skončí.

Gramatiky typu 0 definují tzv. rekurzivní jazyky. Rekurzivní jazyk je takový formální jazyk, pro který existuje Turingův stroj, který slova tohoto jazyka buď zamítá nebo akceptuje, tzn. nikdy se nezacyklí.

Typ 1

Tyto gramatiky nazýváme gramatikami kontextovými. Kontextová gramatika je taková gramatika, kde nonterminál A může být nahrazen řetězcem δ pouze tehdy, když jeho pravým kontextem je řetězec α a levým kontextem řetězec β.

Jsou rozpoznatelné lineárně ohraničeným Turingovým strojem. Od běžného Turingova stroje se liší způsobem zápisu, kdy smí zapisovat pouze na prvních n buněk pásky v závislosti na délce vstupního slova.

Typ 2

Neboli bezkontextové gramatiky. Tyto gramatiky umožňují využívat přepisovacích pravidel bez ohledu na kontext, ve kterém se neterminál nachází.

Pro rozpoznání gramatik typu 2 se využívá zásobníkový automat. Jde o rozšíření konečného automatu, jelikož využívá zásobníku jako paměti. Zásobníkový automat [4] Z je definován jako sedmice Z = (Q, A, Z, δ, q_0, z_0, F), kde:

Q – je konečná množina stavů

A – je konečná vstupní abeceda

Z – je konečná zásobníková abeceda

δ – je přechodová funkce, která popisuje činnost automatu, tj. jednotlivé přechody mezi stavy

q_0 – značí počáteční stav

z_0 – značí počáteční symboly zásobníku

F – je podmnožinou množiny Q a značí množinu přijímajících stavů

Výpočetní proces probíhá podobně jako u Turingova stroje, kdy dochází k přechodu ze stavu do stavu, s tím rozdílem, že přechod může kromě vstupního symbolu ovlivnit také symbol uložený v zásobníku. Proces skončí v případě, že se automat nachází ve stavu, který patří do množiny přijímajících stavů nebo je zásobník prázdný.

Typ 3

Generují tzv. regulární jazyky. Formální jazyk L je regulární, pokud je rozpoznatelný konečným automatem. Můžeme říct, že jazyk L je regulární, když existuje konečný automat, který po přečtení každého vstupního slova jazyka L skončí v koncovém stavu. Konečný automat [5] K je definován jako pětice K = (S, A, δ, s, F), kde:

S – je konečná množina stavů

A – označuje abecedu neboli konečnou množinu vstupních symbolů

δ – je přechodová funkce, která popisuje činnost automatu, tj. jednotlivé přechody mezi stavy

s – značí počáteční stav

F – je podmnožinou množiny S a značí množinu přijímajících stavů

Činnost konečného automatu se v mnohém neliší od automatu zásobníkového. Princip výpočetního procesu je stejný, s rozdílem, že přechody mezi stavy se řídí pouze aktuálním stavem a vstupním symbolem. Výpočet skončí v případě, že je dosaženo stavu, který se nachází v množině přijímajících stavů. V případě, že je řetězec tímto automatem přijat, patří do množiny, která tvoří regulární jazyk.

Gramatiky v chomského hierarchii jsou gramatiky, jejichž účelem je rozpoznat, zda nějaké slovo patří do daného jazyka. Tedy kontrolují, zda je vstupní řetězec syntakticky správně a tudíž je podmnožinou konkrétního jazyka, který daná gramatika definuje.

3.1.3 Překladové gramatiky

Dalším typem gramatik jsou tzv. překladové gramatiky. U tohoto typu gramatik je množina terminálních symbolů Σ rozdělena na konečnou množinu vstupní abecedy Σ a konečnou množinu výstupní abecedy Δ. Tyto gramatiky využívají překladu mezi dvěma jazyky, kdy na vstupu je symbol ze zdrojového jazyka a na výstupu příslušný symbol z jazyka cílového. Tyto vztahy určují nadefinovaná přepisovací pravidla.

Tento typ gramatiky je očividně vhodným typem pro generátor modelů, který využívá jako vstup jeden typ programovacího jazyka, konkrétně XML a výstupem je jiný jazyk, VRML.

Podrobné vysvětlení teorie gramatik je možné nastudovat např. v učebních skriptech Gramatiky a jazyky od Prof. RNDr. Milana Češka, Csc [3].

Na základě těchto znalostí byla v počátcích vývoje navržena gramatika pro další postup, která se však společně s vývojem aplikace měnila a upravovala. V příloze D přikládám podobu této gramatiky.

3.2 Vstup vs Výstup

Nyní jsme se seznámili se základními typy gramatik a určili vhodnou variantu pro naší aplikací, kterou je právě překladová gramatika definující přepisovací pravidla mezi dvěma programovacími jazyky. V našem případě je jedním z nich jazyk XML a druhým, výstupním jazykem jazyk VRML definující 3D model. V následujících kapitolách si řekneme základní informace a specifikace obou z nich.

3.2.1 Vstupní jazyk

Jedno z vhodných řešení vstupního jazyka pro tuto aplikaci je jazyk XML, který je dle mého názoru pro člověka dobře čitelný a proto s ním i laický uživatel může dobře pracovat. Také je nezávislý na operačním systému, což pokrývá nefunkční požadavek č.1 (kapitola 2 Požadavky na aplikaci). Nejdříve si tedy přiblížíme co to vůbec XML je (použitý zdroj viz [6]).

XML, neboli Extensible Markup Language je značkovací jazyk určený pro výměnu dat mezi různými typy aplikací. Patří do podmnožiny jazyka SGML, která umožňuje uživateli definovat vlastní DTD jazyk, pro tvorbu sad značek a jejich vzájemných vztahů. Tímto způsobem je XML použitelný v mnoha oblastech určitých typů dokumentů.

Jednou z výhod XML může být využití znakové sady ISO 10646, což je 32bitová znaková sada podporující všechny dnes používané jazyky. Díky ní má uživatel možnost psát dokumenty v kterémkoliv světovém jazyce a tím se stává mnohem dostupnější.

Význam jednotlivých částí v dokumentu určují XML značky a efektivita jazyka je na této struktuře silně závislá. Abychom mohli říci, že dokument je tzv. well-formed, neboli dobře strukturovaný musí obsahovat alespoň minimální vlastnosti, kterými jsou:

- právě jeden kořenový element
- neprázdné elementy musí začínat startovací značkou a končit značkou ukončovací
- hodnoty atributů se uzavírají do uvozovek (jednoduchých nebo dvojitých)

Pro zpracování takovýchto dokumentů se využívají nejčastěji dva přístupy. Prvním z nich je tzv. DOM (= Document Object Model) parser, který zpracuje XML data a na základě nich následně vytvoří příslušnou datovou strukturu v podobě stromu.

Druhým přístupem je SAX (= Simple API for XML) parser, který přistupuje k datům sériově. Jedná se o tzv. proudové zpracování, kdy se postupně čte celý dokument a vyvolává příslušné události. Tato metoda zpracování je výhodnější v několika hlediscích. Především co se týče rychlosti zpracování a menší paměťové náročnosti oproti DOM parseru, který celý dokument ukládá jako obraz v paměti a až poté s ním pracuje.

Další řešením pro definování vstupních dat je vytvoření nějakého proprietárního jazyka v podobě značek a klíčových slov, na základě nichž by se řídil příslušný parser. Vstupní data v této podobě, by se buď přeparsovala na podobu XML zápisu, pro který by byl vytvořen příslušný SAX parser a vytvořil by model stejně tak, jako ze zápisu značkovacího jazyka XML. Další možností je vytvoření speciálního parseru pro tento typ jazyka. Tudíž by byly implementovány dvě aplikace pro různé vstupy, ale generující stejný výstup.

Tato varianta by definovala vstupní jazyk formou, která je pro uživatele co nejpřirozenější. Tedy by používala klíčová slova takovým způsobem, jak je uživatel zvyklý z reálného života, to znamená např. krabice je na polici. Obě dvě zde uvedené věci, tzn. krabice i police, jsou z hlediska VRML základní geometrické tvary – Box. Sousloví „je na“ vypočítá pozici druhého objektu tak, aby odpovídal požadovanému výsledku. Toto definování je možná trochu nadnesené a i v tomto případě by jazyk musel definovat nějakou konkrétní strukturu pro vygenerování konečné scény, ale uvádím jí pro přiblížení smyslu a síly tohoto typu řešení.

3.2.2 Výstupní jazyk

Výstupem bude vždy jazyk VRML. V tuto chvíli bych ráda stručně popsala co to VRML je, kdy, kde a proč vznikl a na co se používá. Veškeré zde uvedené informace jsou ze zdrojů, které uvádím v seznamu literatury ([7], [8], [9]).

VRML, neboli Virtual Reality Modeling Languge, je grafický formát navržen pro popis trojrozměrných scén využívajících jak aktivní tak i pasivní prvky. Umožňuje mnoho funkcí ať už měnit objekty, jejich barvu, tvar, přemístění, tak dokáže reagovat i na přítomnost virtuálního návštěvníka, na časovač nebo na událost vyvolanou jiným objektem. Pro potřeby této práce však aktivní prvky jazyka nebudou potřeba.

Formát je založený na deklarativním programovacím jazyce pro využití na internetu. Uživatel potřebuje mít jen nainstalovaný plugin pro svůj internetový prohlížeč a může se jednoduše procházet virtuálními světy. V posledních letech ho plnohodnotně zastupuje formát X3D, který spojuje VRML97 a značkovací jazyk XML.

Historie VRML

Koncem 80. let programátoři ze Silicon Graphics navrhli grafickou knihovnu Inventor, určenou právě pro tvorbu virtuálních scén. Tímto se začíná vyvíjet jazyk VRML. Inventor se stal jakousi nadstavbou pro existující knihovnu GL, která se na počátku 90.let rozšířila na dnes velmi známou OpenGL. Spolu s ní vzniká také knihovna OpenInventor, která byla základem pro vytvoření jazyka VRML.

VRML 1.0

První stavební kámen jazyka VRML byl položen na jaře roku 1994 v Ženevě na první výroční konferenci WWW. Na podzim téhož roku byla poté představena jeho první pracovní verze.

Výsledná oficiální podoba specifikace VRML 1.0 spatřila světlo světa 26. května 1995. Ve stejnou dobu vzniká také skupina zvaná VAG (Vrml Architecture Group) sestávají se z nezávislých programátorů a návrhářů oslovující odborníky s výzvou k vytvoření specifikace pro budoucí verzi jazyka VRML 2.0.

VRML 2.0

V dubnu roku 1996 bylo představeno celkem osm návrhů, ze kterých byl vybrán společný návrh firem Sony a Silicon Graphics, jehož pracovní název byl Moving Worlds. Tento návrh je základem pro vytvoření verze VRML 2.0. Po více než ročním úsilí při tvorbě VRML 2.0 se z dosud neformální skupiny VAG stává VRML Consorcium, Inc. a zahajuje okamžitou spolupráci s mezinárodní standardizační

organizací ISO na vznik VRML jako normy. V roce 1997 je toto úsilí odměněno a norma dostává název VRML 97 (č. n. ISO/EC 14772-1:1997).

Jazyk VRML je velice komplexní nástroj pro modelování 3D scén. Díky své rozmanitosti umožňuje vytvořit velké množství různých modelů a scén s aktivními i pasivními prvky. Pro potřeby této práce bylo zapotřebí jen úzké části této specifikace, která nám pomůže k požadovanému výsledku.

Struktura samotného VRML jazyka musí být vygenerována tak, aby se dala lehce modifikovat a byla navržena co nejjednodušeji. Tedy nejlépe jen za pomoci základních geometrických tvarů, případně využití tzv. PROTO uzlu. PROTO uzel je příkaz jazyka VRML, který umožňuje definovat nové tvary a následně je jednoduše modifikovat díky nadefinovaným atributům.

Uživatel tedy bude mít na výběr z několika typů objektů, jako je kvádr, koule, kužel nebo válec, díky nimž by měl být schopen vytvořit scénu podle jeho představ.

Jazyk umožní uživateli vybraný typ objektu měnit, co se týče velikosti a posunutí a rozhodnout zda se objekt ve vygenerované scéně zobrazí vždy nebo bude jeho zobrazení náhodné. Také bude mít možnost objekty pojmenovávat, měnit barvy a jednoduše nadefinovaný objekt zopakovat po lokálních osách x, y a z. Pomocí těchto geometrických tvarů uživatel nadefinuje model, tzn. opakující se část celé scény, ten se převede na PROTO uzel a podle zadaných hodnot zopakuje po jednotlivých osách s příslušnou modifikací. Za zvážení stojí, zda zahrnout i přiřazování textur, kdy by uživatel místo klíčového slova určující konkrétní barvu napsal cestu k příslušné textuře. Případně by měl možnost si jí nadefinovat stejně jako barvu a přiřadit jí klíčové slovo, které by se dále používalo ve scéně stejným způsobem jako klíčové slovo udávající barvu.

3.2 Rozbor aplikace

3.2.1 Rozhraní ContentHandler

V první řadě při načtení XML souboru je nutné převést informace z XML souboru do podoby použitelné pro aplikaci. O tuto část se postará tzv. parser.

Aplikace bude implementovat základní rozhraní specifikace SAX - rozhraní ContentHandler (použité zdroje viz [10]), což je základní rozhraní pro zpracování

XML dokumentů. ContentHandler definuje několik základních metod pro načtení a zpracování XML souboru.

void setDocumentLocator(Locator locator) - poskytuje informaci o tom, kde v dokumentu událost vznikla. Zavolá se vždy jako první, ještě před metodou startDocument(). Při parsování je poté možné pomocí metod getLineNumber() a getColumntNumber() dotazovat na pozici.

př. implementace:

```
Locator locator;
   public void setDocumentLocator(Locator locator) {
      this.locator = locator;
   }
```

void startDocument() - definuje začátek dokumentu

void endDocument() - definuje konec dokumentu. Udává, že veškeré informace XML dokumentu jsou již zpracované.

void startElement(String uri, String localName, String qName, Attributes atts) – definuje začátek elementu.

Vstupní atributy:

String uri – identifikátor jmenného prostoru

String localName – lokální jméno

String qName – jméno ze zdrojového XML dokumentu, tzv. kvalifikované jméno

Attributs atts – množina atributů v otevíracím tagu elementu

void endElement(String uri, String localName, String qName) – definuje konec elementu. Udává, že veškeré informace daného elementu jsou již zpracované. Všechna volání mezi metodou startElement a endElement udávají obsah elementu.

void characters(char[] ch, int start, int length) – znaková data. Udává obsah elementu.

Vstupní atributy:

Char[] ch – pole znaků

int start – startovní pozice v poli znaků

int lenght – počet všech znaků

void startPrefixMapping(String prefix, String uri) – udává informaci o zahájení mapování jmenného prostoru
Vstupní atributy:
String prefix – prefix jmenného prostoru
String uri – identifikátor jmenného prostoru

void endPrefixMapping(String prefix) – udává informaci o ukončení mapování jmenného prostoru

void ignorableWhitespace(char[] ch, int start, int length) – udává znaky, které se mohou ignorovat
Vstupní atributy:
char[] ch – pole znaků
int start – startovní pozice v poli znaků
int length – počet všech znaků

void processingInstruction(String target, String data) – definuje instrukce ke zpracování.
Vstupní atributy:
String target – řetězec od znaku <? až k prvnímu bílému znaku
String data – všechny další informace za prvním bílým znakem

void skippedEntity(String name) – definuje entitu, kterou má parser přeskočit
Vstupní atributy:
String name – název přeskakované entity

3.2.2 Funkčnost aplikace

Samotná aplikace bude implementovat několik základních algoritmů pro vygenerování scény. Chování těchto algoritmů bude ovlivněno na základě vstupního XML souboru.

Aplikace bude generovat také tzv. LOD mechanismus. LOD (= levels of details) je základní vlastnost VRML jazyka a určuje stupně zobrazení konkrétního

objektu vzhledem k dané vzdálenosti avatara (=virtuálního návštěvníka) od objektu.

Aplikace ve své základní podobě bude obyčejný XML SAX parser, který upraví přijaté informace do podoby použitelné pro další postup. Dále bude mít několik tříd pro úpravu objektů a podobu jejich zobrazení.

4 REALIZACE

4.1 Postup vývoje

Jelikož jsem na začátku vývoje aplikace neměla žádné zkušenosti s překladovými gramatikami a ani s parsováním XML jazyka, probíhala tvorba spíše způsobem „pokus omyl“. V první fázi vznikla základní kostra aplikace, která uměla n-krát zopakovat model a tím vygenerovat scénu. Ve chvíli, kdy měla být do aplikace zanesena možnost využít náhodně generujících objektů, se ukázala tato implementace jako nevhodná a proto byla z větší části přepracovaná do podoby, ve které je nyní.

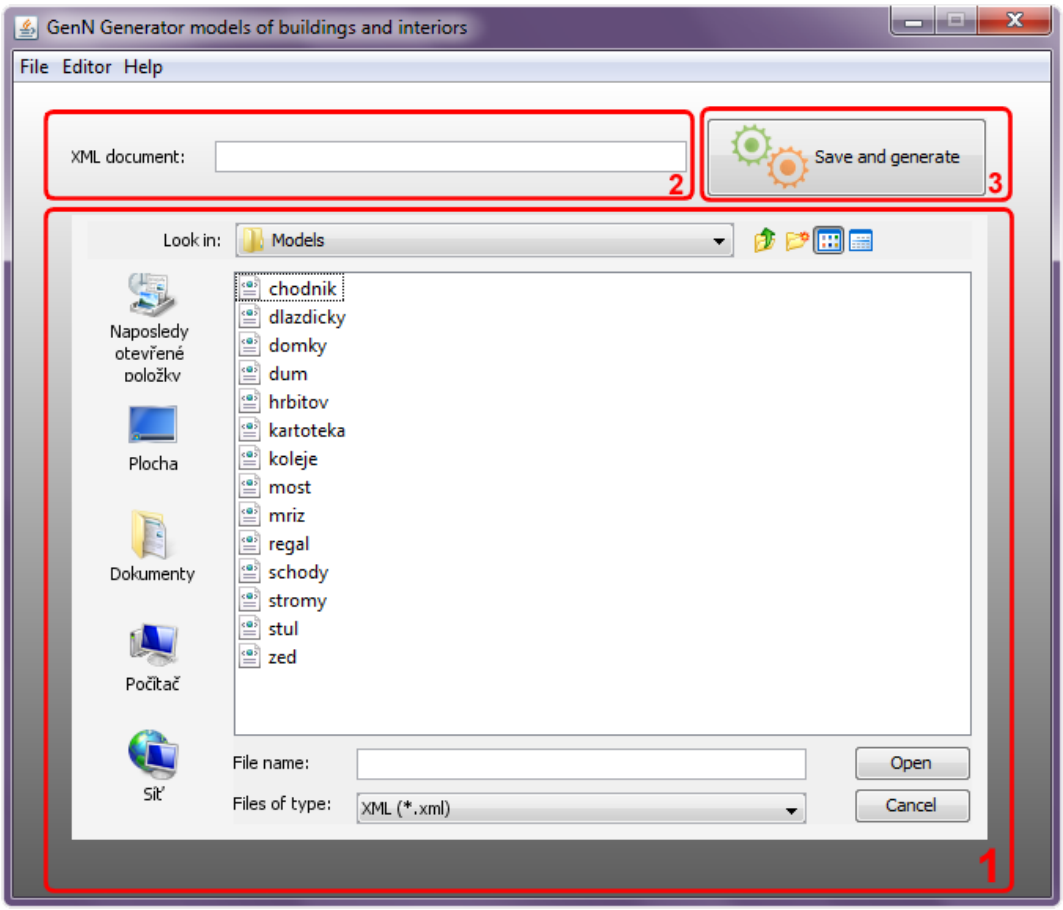
Z počátku se počítalo, kromě toho, že objekty bude možné obarvit, také objekty otexturovat. Nakonec se ukázalo, že pro objekty, jak jsou navrženy nyní, by nebylo mapování textur efektivní, a proto se od textur pro tuto chvíli ustoupilo a aplikace umožňuje objekty pouze obarvit.

4.2 Implementační prostředí

Na základě zadání bakalářské práce, jsem využila výše zmiňovaný jazyk XML jako vstupní jazyk a VRML jako jazyk výstupní. Samotná aplikace pro generování modelů je poté implementována v jazyce JAVA ve vývojovém prostředí NetBeans IDE 6.8.

4.3 Popis grafického rozhraní

Jelikož samotná aplikace slouží jen pro načtení už vytvořeného XML dokumentu a jeho následné zpracování do VRML. Ovládání aplikace je díky jednoduchému grafickému rozhraní velmi jednoznačné *(obr. 4.1)*.



Obrázek 4.1: Hlavní okno aplikace

Podle obrázku vidíme rozdělení hlavního okna na několik částí. První z nich (sekce 1) je jednoduchá tabulka sloužící pro nalezení XML souboru uloženého v počítači. Po kliknutí na tlačítko Open se vybraný soubor připraví ke generování, viz sekce 2 a tlačítko v sekci 3 na obrázku spustí proces generování. V tuto chvíli se vyvolá stejné dialogové okno jako vidíme v sekci 1 s tím rozdílem, že neslouží pro otevření souboru, ale žádá uživatele o vybrání umístění konečného vygenerovaného modelu v počítači. Poté co uživatel vybere cílový adresář se spustí generování. V případě, že zdrojový XML dokument je syntakticky správně je program ukončen a uživatel má k dispozici vygenerovaný 3D model v jazyce VRML.

V menu je uživateli k dispozici Editor. Při spuštění editoru může využít základní kostru XML struktury s předdefinovanými a okomentovanými základními elementy. Také je mu k dispozici nápověda, kdy se mu spustí kompletní přehled elementů a atributů tak, jak je uvedena v příloze C této práce. Dále si za pomoci editoru může nadefinovat vlastní barvy z RGB nabídky, které dále může využívat při tvorbě modelu. Editor má prozatím jen základní kostru a v budoucnu by bylo dobré ho rozšířit. Tímto rozšířením by mohlo být automatické generování souvisejících

atributů (např. při zadání shape="Box" by se automaticky přiřadil atribut size udávající jeho velikost).

4.4 Popis chodu aplikace

Celá aplikace je implementována v jazyce JAVA a obsahuje několik základních tříd a rozhraní ContentHandler, jak už bylo popsáno v kapitole 3.

Popis chování aplikace je zobrazen na následujícím sekvenčním diagramu. Výpočetní proces začíná načtením XML dokumentu. V tuto chvíli je vytvořena instance třídy `Parser` implementující rozhraní `ContentHandler`. Iterativním způsobem následně probíhá čtení jednotlivých elementů a atributů XML documentu a jejich zpracování, v tuto chvíli to znamená volání příslušných metod třídy `Object`. V případě, že jsou zpracována všechna data z elementu OBJECT a informace elementu ACTION, které tento objekt rozšiřují se objekt uloží pro další použití do instance třídy `Db`.

Třída `Db` je realizována jako návrhový vzor *Singleton*. Singleton (česky unikát) je návrhový vzor, který zajistí, že v programu bude běžet právě jedna jeho instance. Tuto jeho jedinou vlastnost zajistí privátní konstruktor a vytvoření jedné instance uvnitř třídy, která je uložena do statické proměnné. V jiných třídách aplikace se pak dotazujeme, zda tato instance existuje. Jestliže instance neexistuje, můžeme jí vytvořit. V opačném případě toto volání vrátí již existující instanci této třídy pro další zpracování.

Ve chvíli, kdy je objekt uložen, je vytvořena instance třídy `Repeater`, která na základě informací z XML souboru zopakuje právě vytvořený objekt. Třída `Repeater` ke svému běhu využívá několik dalších tříd. První z nich je třída `Transformer` implementující dvě základní metody – `translate()` a `rotate()`. Metoda `translate()` s atributem `Object` slouží pro posouvání objektu v případě, že `Object` má nastavenou pozici jedním z klíčových slov top, bottom, right, left, front nebo back. V tomto případě musí aplikace vypočítat jeho posunutí vhledem k oběktu, ke kterému se toto klíčové slovo vztahuje. Jestliže je pozice zadána pevnými hodnotami pak je pozice nastavena přímo ve třídě `Object` a metoda `translate()` posunutí nevypočítává. Druhou metodou třídy `Transformer` je metoda `rotate()`. Jelikož uživatel nemá možnost v XML dokumentu zadat rotaci objektu, je tato metoda volána opět jen v případě zadání pozice v podobě klíčového slova, kdy je v některých

případech rotování objektu zapotřebí.

Další třídou, kterou využívá třída `Repeater` je třída `RandomGenerator`. Ta implementuje několik metod využívaných v případě nadefinování tzv. *random* objektu neboli náhodného objektu. V konkrétní instanci třídy `Object` se nastaví String type na hodnotu random v případě, že některá z jejich vlastností není zadána pevnou hodnotou, ale buď rozmezím (v případě velikosti objektu) nebo klíčovým slovem (v případě pozice objektu). Nebo je nadefinován atribut visible s hodnotou random. Což signalizuje, že objekt se bude při iteraci zobrazovat náhodně. V takovém případě se o tyto vlastnosti random objectu musí postarat aplikace, neboli právě třída `RandomGenerator`. První z jejích metod je metoda `generateGridPosition()`. Tato metoda vrací pole celočíselných hodnot, tzn. náhodně vygenerované místo na dané mřížce. Co je to mřížka? Mřížka představuje abstraktní rozdělení příslušného objektu na základě informací o velikosti přiřazených objektů. To znamená, že v případě, kdy chceme některému objektu přiřadit jiný objekt, pomocí klíčového slova udávající pozici (top, bottom, right, left, front, back), se tomuto objektu určí mřížka. Ta je vypočítána na základě maximálních velikostí na lokálních osách x, y, z všech objektů, které mají být na danou pozici příslušného objektu vygenerovány. Tyto objekty jsou uloženy v instanci třídy `Db`. Objekt má tedy pro všechna klíčová slova vygenerovanou abstraktní mřížku pro určení pozice přiřazeného objektu. Nic tedy nebrání metodě `generateGridPosition()` určit konkrétní pozici na této mřížce. Další metodou třídy `RandomGenerator` je třída `generateRandomPosition()`. Tato metoda vygeneruje na základě již určené pozice metodou `generateGridPosition()` konkrétní hodnoty pro správné umístění objektu vzhledem k celkové scéně. Poslední důležitou metodou této třídy je metoda `generateRandomSize()`, která určí náhodnou velikost objektu v případě, že atribut size je definován v podobě rozmezí minimální a maximální velikosti daného objektu.

Poslední třídou, využívanou ve třídě `Repeater` je `LODGenerator`. Tato třída, jak už napovídá její název, se stará o LOD mechanizmus. Má jen jednu metodu `setLOD()`, která na základě vstupních atributů vygeneruje konstrukci LOD mechanizmu tak, jak je definována jazykem VRML.

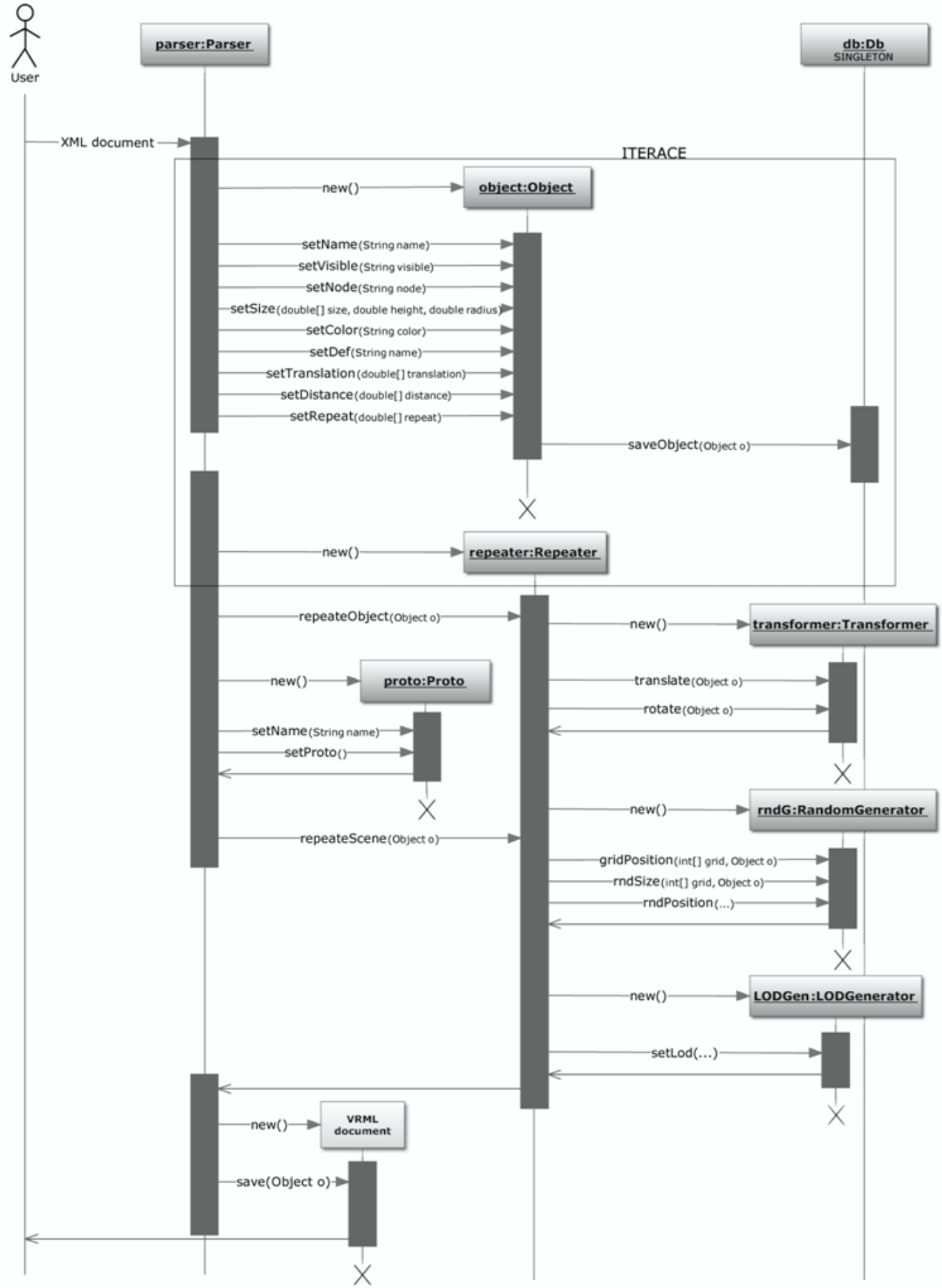
Nyní už má třída `Repeater` veškeré potřebné instance ostatních tříd pro zopakování objektu.

Tento proces probíhá do chvíle dokud parser nenarazí na element SCENE. Element SCENE je posledním elementem XML struktury a na základě jeho atributů se generuje scéna. V tuto chvíli parser ví, že má vytvořený model a může začít

generovat scénu. Jako první vytvoří instanci třídy `Proto` a předá jí veškeré doposud zpracované informace. Třída Proto má metody k vytvoření konstrukce PROTO uzlu používaného ve VRML. Ve chvíli, kdy je vytvořen PROTO uzel je možné ho uložit do VRML souboru.

Poté se zavolá metoda `repeatScene()` třídy `Repeater` a proběhne zopakování po jednotlivých osách stejně tak, jak tomu bylo při opakování jedné části modelu.

Po skončení iterace máme již vygenerovanou scénu a stačí pouze celou scénu uložit k již uloženému PROTO uzlu do VRML souboru. V tuto chvíli proces končí.



Obrázek 4.2: Sekvenční diagram

4.5 LOD mechanizmus

LOD mechanizmus je příslušnou třídou vygenerován do podoby tak, jak je známa z VRML jazyka. Původně se počítalo s navržení do 3 stupňů, ale problém byl s podobou druhého stupně. Jelikož modely vytvořené touto aplikací jsou už tak dost jednoduché bez nějakých velkých detailů, bylo obtížné vůbec druhý level LOD mechanizmu zkonstruovat.

Prvním stupněm je samotný model, tak jak je vygenerován na základě informací z příslušného XML souboru a tento stupeň je logicky vidět, když je avatar nejblíže.

Pro druhý stupeň se v rámci analýzy zprvu přemýšlelo o tzv. bounding boxu, neboli pomocné obálce, která primárně slouží ke zrychlení výpočtu při zobrazování objektů a obklopuje všechny potomky rodičovského uzlu, tedy celý model, tak jak je detailně zobrazen ve stupni jedna. Tato možnost byla vzápětí zavržena z důvodu nezobrazování obálky, tedy obálka se vypočítá, ale fyzicky není vidět. Další možností bylo v druhém stupni zobrazit jen příslušně velký Box. Ani toto řešení se ve výsledku neosvědčilo, jelikož modely se převážně skládají pouze z těchto geometrických tvarů a změna nebyla téměř žádná a u některých modelů bylo toto řešení spíše na škodu.

Posledním stupněm, když je avatar nejdále, je prázdný uzel Group, tedy uživatel už nevidí model vůbec.

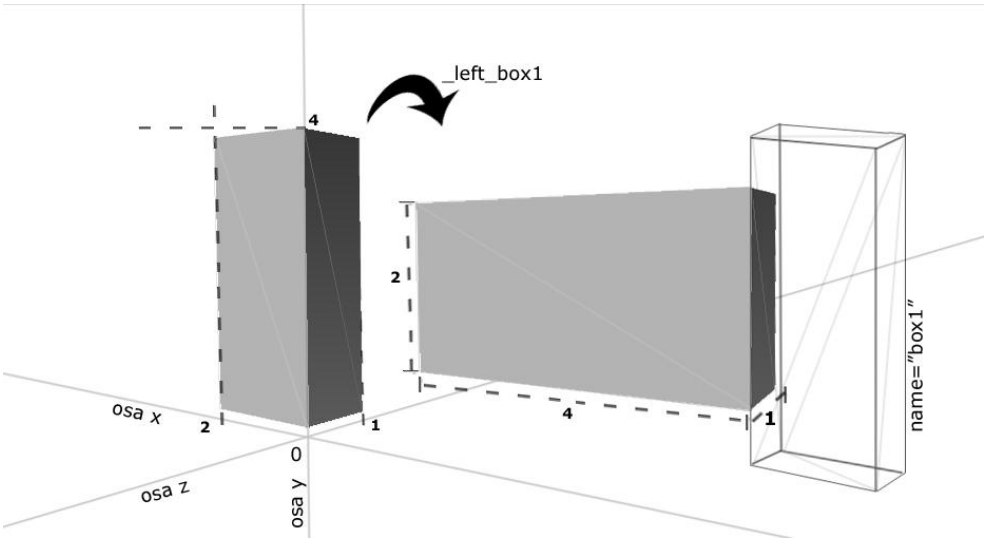
Na základě této analýzy jsem se v rámci této práce rozhodla realizovat LOD mechanizmus pouze ve dvou stupních a generování dalších stupňů, kdy model by byl vytvořen v několika stupních detailů, může být případným dalším vývojem.

4.7 Zobrazování scény

Pro zobrazení vygenerované scény je potřeba mít nainstalovaný VRML prohlížeč. Ty existují buď jako samostatná tzv. stand-alone aplikace nebo jako plugin (=zásuvný modul) do internetových prohlížečů. Jedním z takových pluginů je např. Cortona 3D Viewer nebo BS Contact Player.

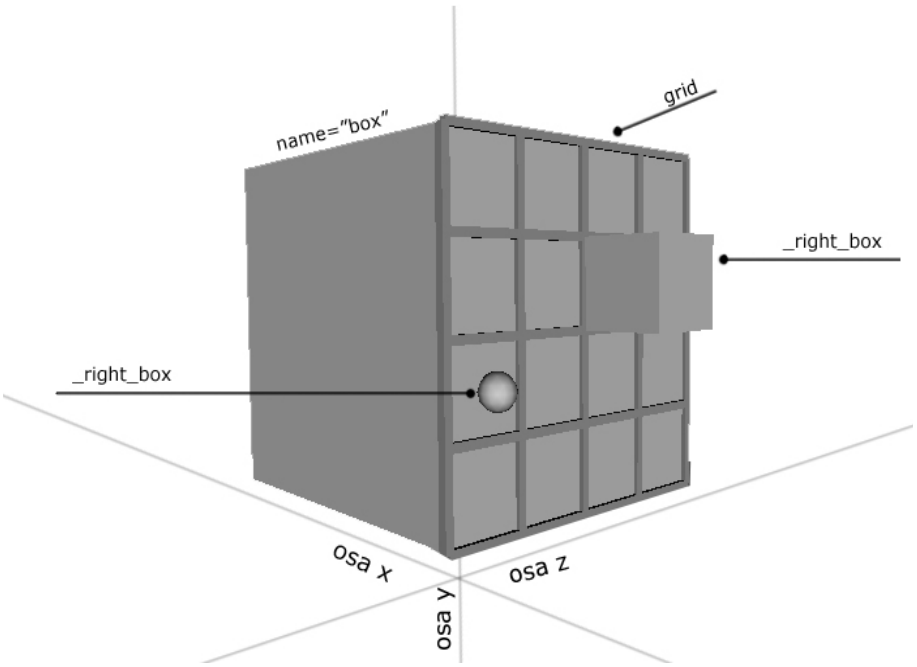
4.7.1 Klíčová slova pro určení pozice

Při využívání klíčových slov na posouvání objektu, tzn. přiřazení k nějakému jinému objektu ve scéně, se při využití jednoho z klíčových slov left, right, front nebo back objekt otočí po příslušné ose. Na to by uživatel neměl zapomenout při zadávání velikosti objektu. Na obrázku 4.3 je vytvořený objekt s velikostí size="2,4,1". Poté v elementu action je pozice daná klíčovým slovem, tzn. position="_left_box1". V tuto chvíli se objekt přichytí na levou stranu objektu box1 a otočí o 90° doleva.



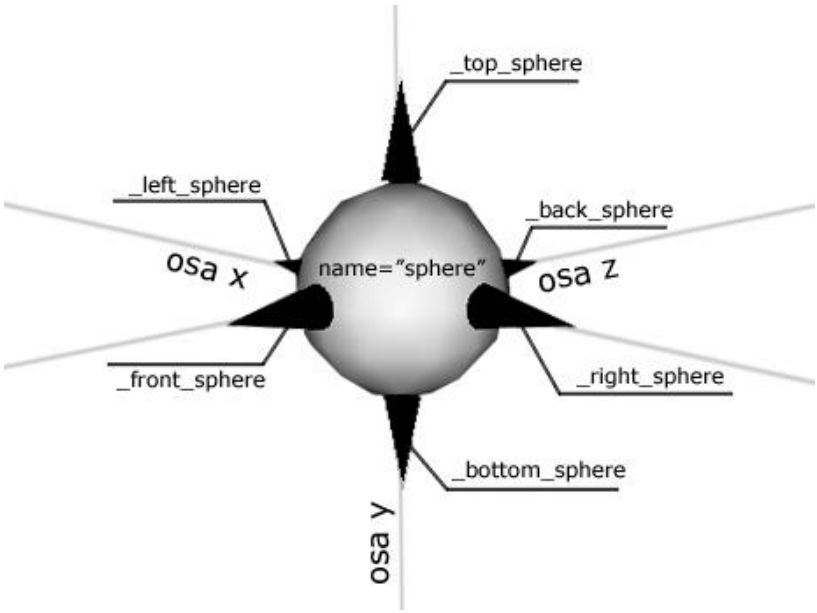
Obrázek 4.3: Klíčové slovo left

Způsob tohoto generování posunutí se řídí na základě mřížky (popsáno výše, kapitola 4.3 Popis chodu aplikace). Na obrázku 4.4 je graficky znázorněn případ, kdy jsou objektu *box* přiřazeny dva další objekty a jeho mřížka je vygenerovaná na základě největšího z nich. Pozice na abstraktní mřížce je generována náhodně. V případě, že by přiřazený objekt měl větší rozměry než objekt, ke kterému je připojený, vygeneruje se speciální mřížka pouze s jedním políčkem a objekt je tedy přiřazen na stejné souřadnice, jako má objekt, ke kterému se vztahuje.



Obrázek 4.4: Generování mřížky

V případě přiřazování objektů k objektům jako je koule, válec nebo kužel se mřížka negeneruje po celé ploše, ale vypočítá se pouze posunutí po příslušné ose vzhledem ke klíčovému slovu pozice, tak aby se objekt přichytil k té správné straně. Na obrázku 4.5 je ukázka, jak se objekty tímto způsobem přichycují k objektu tvaru koule. Tato implementace se v danou chvíli jevila jako nejschůdnější a další algoritmy, které by vypočítávaly speciální mřížku např. u objektu koule spadají do případného budoucího vývoje aplikace.



Obrázek 4.5: Mřížka na objektu tvaru koule

Zdrojový XML soubor obrázku 4.5:

```
<?xml version="1.0" encoding="utf-8" ?>
<model name="front">

<object name="sphere" Shape="Sphere" radius="1" color="white"/>
<action />

<object name="topCone" Shape="Cone" radius="0.2" height="1" color="black"/>
<action position="_top_sphere" />

<object name="bottomCone" Shape="Cone" radius="0.2" height="1" color="black"/>
<action position="_bottom_sphere" />

<object name="frontCone" Shape="Cone" radius="0.2" height="1" color="black"/>
<action position="_front_sphere" />

<object name="backCone" Shape="Cone" radius="0.2" height="1" color="black"/>
<action position="_back_sphere" />

<object name="leftCone" Shape="Cone" radius="0.2" height="1" color="black"/>
<action position="_left_sphere" />

<object name="rightCone" Shape="Cone" radius="0.2" height="1" color="black"/>
<action position="_right_sphere" />

<scene  />
</model>
```

5 TESTOVÁNÍ

Nejlepší cestou, jak zjistit zda je nově vytvořená aplikace vyhovující nejen Vám, ale hlavně lidem, kteří jí budou využívat je vyzkoušet to přímo na nich. A k tomu slouží právě testování.

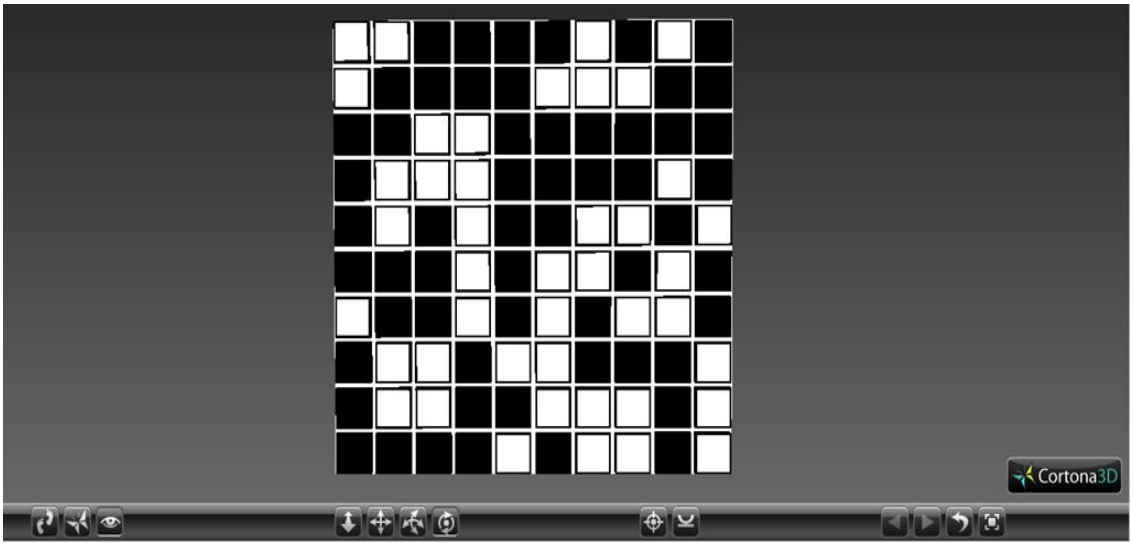
Bylo provedeno usability testování s několika konečnými uživateli. Usability testování patří do skupiny tzv. black-box testování, které řeší vstup a výstup bez ohledu na to, jak vypadá vnitřní struktura systému.

Hlavním úkolem tohoto testování bylo zjistit, zda je daná vytvořená struktura vstupního jazyka pro uživatele dosti srozumitelná a je tedy schopen bez větších problémů vytvořit v relativně krátkém čase požadovaný model.

5.1 Příprava testování

Jelikož je aplikace určena pro projekt Virtual Libra(ry (viz kapitola 1.1 Motivace), tzn. pro osoby zabývající se primárně uměleckou tvorbou, bylo pro účely testování vybráno několik participantů (= osoba, která aplikaci testuje) profilově podobných, tedy neměl by k vytvoření modelu pomocí této aplikace potřebovat žádné zvláštní technické vzdělání.

Každý z participantů měl za úkol napsat příslušnou XML strukturu k zadanému jednoduchému 3D modelu (viz obrázek 5.1). K dispozici mu byla kompletní dokumentace.



Obrázek 5.1: 3D model pro testování

Výstupem každého participanta byla jím vytvořená konkrétní struktura vstupního jazyka ve formátu XML a k tomu příslušná vygenerovaná scéna ve výstupním formátu VRML. U každého z nich se také měřil čas potřebný k naučení a napsání struktury pro zadaný model.

5.2 Výsledky testování

Pro testování aplikace byli ve výsledku vybrání 3 participanti. Všichni měli zpočátku problém představit si elementární část celé scény a tu pak zopakovat. V případě zadaného objektu se jedná o zeď pod dlaždicemi, kdy v rámci modelu je vytvořená jen její část pod jednou dlaždicí. Tento model je pak ve výsledné scéně zopakován a tím je vytvořená celá zeď i s dlaždicemi.

Po chvíli neměl nikdo z nich problém požadovanou strukturu dopsat a vytvořit tak stejný 3D model jako je na obrázku 5.1.

Průměrný čas na vypracování a naučení se jazyka: cca 20min

5.3 Shrnutí testování

Při prvním testování byla odhalena chyba v implementaci. Jeden z participantů použil pro přiřazování objektů klíčové slovo front, které spolu s klíčovým slovem back, bylo přidáno až v průběhu vývoje a zpočátku se s nimi nepočítalo. Pro tyto dvě klíčová slova byla špatně vypočítávána pozice přiřazeného objektu. Na základě tohoto odhalení byla chyba odstraněna a participant mohl pokračovat již s plně funkční aplikací.

Nakonec se všichni testovaní shodli, že jazyk je srozumitelný a po relativně krátkém čase neměl nikdo z nich problém model vytvořit.

6 ZÁVĚR

V úvodu této bakalářské práce jsme se dozvěděli něco málo o tom, proč tato práce vůbec vznikla a pro zajímavost také základní informace o světových projektech, které mají stejné základy jako aplikace GenN. Dále jsem rozebrala několik možných řešení pro implementaci aplikace a seznámili jsme se se základními informacemi o použitých technologiích, které následně posloužili pro samotnou realizaci. Po úspěšném dokončení aplikace proběhlo krátké uživatelské testování, díky němuž byly odhaleny a opraveny některé chyby a pro tuto chvíli je aplikace plně funkční.

V dnešní době moderních technologií je již mnoho možností, jak vytvořit kvalitní 3D model téměř identický s realitou. Avšak hlavním cílem této práce bylo vytvořit jednoduchou aplikaci pro tvorbu 3D modelů s možností opakování již vytvořených objektů a tím vytvoření scény. Tento prvotní záměr byl s dostupnými prostředky dodržen a vznikla tak aplikace GenN.

V rámci této bakalářské práce jsem se seznámila s gramatikami a dozvěděla se více i o již známých jazycích XML a VRML.

6.1 Budoucí vývoj

Pro tuto chvíli byla vytvořena zcela funkční aplikace, která však může mít další následující rozšíření.

1) Algoritmy vypočítávající speciální mřížky pro tvar koule, válec, kužel.

V případě přiřazení pozice pomocí klíčového slova se pro tuto chvíli u tvarů jako je koule, válec nebo kužel pouze posune objekt po příslušné ose, tak aby odpovídal zadanému klíčovému slovu. Příklad je na obrázku 4.5.

2) Zavedení mapování textur.

Aplikace umožňuje objekty pouze obarvit na libovolnou barvu dle RGB hodnot.

3) Rozšíření LOD mechanizmu.

V rámci této práce byl navržen dvoustupňový LOD mechanizmus. I přesto, že jsou modely generované touto aplikací velmi jednoduché, bylo by dobré v další fázi vývoje tento mechanizmus rozšířit o minimálně jeden stupeň.

4) Rozšíření editoru.

Editor má pouze nejzákladnější podobu, kde je uživateli k dispozici základní struktura XML dokumentu. Umožňuje také vytvářet nové barvy na základě RGB hodnot, které je pak možné využít v dané scéně. Editor by mohl mít další rozšíření, např. přiřazování souvisejících atributů k vybranému tvaru, což by uživateli usnadnilo tvorbu modelu (tzn. při zadání shape="Box" by se automaticky přiřadil atribut size udávající jeho velikost).

LITERATURA

[1] A. RAU-CHAPLIN, B. MACKAY-LYONS, P. F. SPIERENBURG. *The LaHave House Project: Towards an Automated Architectural Design Service, 1996*
Published in:
Proceeding
GRAPHITE '06 Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia
ACM New York, NY, USA ©2006
table of contents ISBN:1-59593-564-9 doi>10.1145/1174429.1174501

[2] M. LARIVE, V. GAILDRAT. *Wall grammar for building generation*, 2006
<http://portal.acm.org/citation.cfm?id=1174501>

[3] PROF. RNDR. MILAN ČEŠKA, CSC. *Gramatiky a jazyky.*
Učební texty vysokých škol, VUT Brno, 1992

[4] Server wikipedie: http://cs.wikipedia.org/wiki/Turing%C5%AFv_stroj,
(poslední přístup 27.4.2011)

[5] Server wikipedie:
http://cs.wikipedia.org/wiki/Kone%C4%8Dn%C3%BD_automat
(poslední přístup 27.4.2011)

[6] Seriál o XML pro Softwarové noviny:
http://www.kosek.cz/clanky/swn-xml/index.html
(poslední přístup 28.4.2011)

[7] Server wikipedie: http://cs.wikipedia.org/wiki/VRML
(poslední přístup 28.4.2011)

[8] VRML: Jazyk pro popis virtuální reality:
http://www.root.cz/clanky/vrml-jazyk-pro-popis-virtualni-reality/
(poslední přístup 28.4.2011)

[9] J. ŽÁRA. *Laskavý průvodce virtuálními světy aneb Praktická příručka jazyka VRML 97.* Computer Press, Praha, 1999

[10] A. SKONNARD, M. GUDGIN. *XML pohotová referenční příručka: referenční příručka programátora k XML, XPath, XSLT, XML Schema, SOAP a dalším.*
Grada Publishing a.s., 2006

PŘÍLOHA A – SEZNAM POUŽITÝCH ZKRATEK

3D	Three-dimensional space (trojrozměrný prostor)
CAVE	Cave Automatic Virtual Environment
XML	Extensible Markup Language
SGML	Standard Generalized Markup Language
DTD	Document Type Definition
DOM	Document Object Model
SAX	Simple API for XML
VRML	Virtual Reality Modeling Languge
LOD	Level Of Detail

PŘÍLOHA B - SPECIFIKACE XML STRUKTURY

Podoba XML struktury se měnila spolu s vývojem aplikace, kdy byly některé atributy přidávány, některé upravovány a některé naopak zcela zmizely.

Konečná podoba XML struktury vypadá následujícím způsobem. Tento vzor obsahuje všechny elementy a jejich atributy, které jsou možné využít.

<**MODEL** *name*=”model” *lod*=“yes/no“ colors=“nameOfColor:RGB;“>

<**OBJECT**
name=“abc“
shape=[Box,Cone,Cylinder,Sphere]
size=[x,y,z] / *size*=[x_0-x_1,y_0-y_1,z_0-z_1]
radius=[1] / *radius*=[0-n]
height=[1] / *height*=[0-n]
color=“[red,blue,...]“
visible=“random“
/>

<**ACTION**
position=[x,y,z] /
position=_left/_right/_bottom/_top/_front/_back_nameOfObject
repeat=[x,y,z]
distance=[x,y,z]
/>

<**SCENE** *repeat*=[x,y,z] *distance*=[x,y,z] />

</**MODEL**>

Při tvorbě scén pomocí aplikace GenN je důležité představit si scénu jako posloupnost n-krát se opakující její nejmenší a nejjednodušší části. V případě, že uživatel dokáže určit tuto část, může vytvořit velmi jednoduše celou scénu. Struktura XML zápisu je navržena přesně tak, aby splňovala tuto podmínku.

B.1 Element MODEL

Kořenový element tohoto zápisu MODEL obsahuje povinný **atribut name**, kde uživatel pojmenuje opakující se část celé scény. Bližší specifikace opakování je určena v elementu SCENE.

Druhým atributem je **atribut lod**, který definuje, zda aplikace vygeneruje či nevygeneruje LOD mechanizmus.

V **atributu colors** je možné jednoduše nadefinovat vlastní barvy v podobě RGB, které pak budou ve scéně používány.

B.2 Element OBJECT

Element OBJECT slouží pro nadefinování jednoho z možných objektů a jeho základních vlastností. Obsahuje několik atributů.

Atribut name – jméno objektu

Objekt je možné pojmenovat. Teto atribut není povinný, ale v určitých případech nutný (viz atribut position). V případě, že už se uživatel rozhodne objekt pojmenovat, je nutné, aby jeho název byl v celé scéně unikátní. V opačném případě aplikace vygeneruje individuální řetězec, který sama přiřadí příslušnému objektu do atributu name pro další použití.

Atribut Shape – tvar objektu

Jednou ze základních vlastností při generování modelu je vytvořit scénu jen za pomoci elementárních geometrických tvarů. Právě v tomto povinném atributu shape se nadefinuje konkrétní tvar. Těch je na výběr hned několik. Klíčová slova po jednotlivé tvary jsou Box (= kvádr), Sphere (= koule), Cylinder (= válec) a Cone (= kužel). Každý z uvedených tvarů má další povinné atributy pro jejich bližší specifikaci.

Atribut size – velikost objektu

Size je povinný atribut při nadefinování objektu typu Box. Velikostí objektu se rozumí délka, šířka, hloubka u kvádru zadaná na konkrétních osách x, y, z. Pro úplnost je třeba dodat, že délka objektu je daná osou x, výška osou y a hloubka osou z.

Tuto hodnotu je možné zadat ve dvou variantách. Buď jako konkrétní hodnotu a poté se objekt při opakování vygeneruje vždy se stejnou velikostí. V druhé případě je možné zadat rozmezí „a-b“, kde číslo „a“ udává minimální velikost objektu a číslo „b“ maximální velikost příslušného objektu. Při tomto zadání se pak při opakování vygeneruje náhodná velikost v daném rozmezí.

Atribut radius – poloměr objektu

Jestliže atribut shape udává typ objektu Sphere, Cone nebo Cylinder je radius jeho povinným atributem a určuje poloměr.

Stejně jako v případě size je možné jeho hodnotu zadat ve dvou formátech.

Atribut height – výška objektu

Povinná hodnota pro tvar typu Cone a Cylinder. Také možné zadat ve dvou variantách stejně jako u atributu size a radius.

Atribut color – barva objektu

Objekty se dají jednoduše obarvit, kdy uživatel má možnost z několika základních předdefinovaných barev, které přiřadí objektu pomocí jednoho z klíčových slov. Také může používat vlastní barvy, které jsou nadefinovány v atributu colors elementu model. Tento atribut není povinný a v případě, že není zadaný je objektu přiřazena základní barva.

Atribut visible – zobrazení objektu

Posledním atributem elementu OBJECT je speciální atribut visible. Má jen jedinou možnou hodnotu random a jeho definování není povinné. Tato hodnota udává zobrazení daného objektu při opakování. V případě přiřazení tohoto atributu, aplikace náhodně rozhodne, kdy ve scéně objekt zobrazí a kdy ne. Jestliže visible není nadefinováno objekt se při opakování zobrazí s příslušnými vlastnostmi vždy.

B.3 Element ACTION

Element ACTION je povinný element a je nutné ho zadat i v případě, že by nedefinoval žádné doplňující informace. Zadává se vždy hned po základní specifikaci objektu a má následující atributy.

Atribut position – pozice objektu

Co se týče posouvání objektů tak bude možné si opět vybrat ze dvou možností zápisu. Buď uživatel zadá pevné hodnoty v podobě tří čísel ve tvaru x,y,z, kde každé číslo udává posunutí po příslušné ose. Jako druhý způsob zápisu má

uživatel možnost zvolit jedno z klíčových slov – top(=nahoře), bottom(=dole), left(=vlevo), right(=vpravo), front(=vpředu), back(=vzadu) v konstrukci s názvem objektu, ke kterému se má přiřazení vypočítat. V tomto případě je nutné mít objekt, ke kterému se klíčové slovo vztahuje pojmenovaný. Tedy vyplněný atribut name v elementu OBJECT. Aplikace poté konkrétní posunutí vypočítá sama a objekt umístí dle zadaného klíčového slova a objektu.

Atribut repeat – zopakování objektu

Ve scéně je vhodné objekty nadefinovat jen jednou a poté už jen takto nadefinovaný objekt zavolat a tím vytvořit jeho kopii. Repeat není povinný atribut. Jestliže tedy není zadána jeho hodnota objekt je ve scéně umístěn právě jednou.

Hodnota tohoto atributu se zadává ve formátu x, y, z, kde číslo na příslušné ose udává kolikrát se v příslušném směru objekt zopakuje. Při definování se objektu přiřadí konstrukce DEF. DEF je jedna ze základních vlastností VRML jazyka a v případě jeho použití se objektu přiřadí název určený atributem name v elementu OBJECT. Při vytváření kopie je použit příkaz USE s příslušným názvem v zadaném počtu a se zadaným rozmístěním. Toto rozmístění udává atribut distance.

Atribut distance – umístění objektu

Atribut distance existuje ve spojení s atributem repeat. Tedy pokud se objekt má zopakovat, je nutné také zadat, jak se má zopakovat. Toto zajišťuje právě atribut distance, kde opět ve formátu hodnot po jednotlivých osách x, y, z jsou zadány mezery mezi jednotlivým zopakováním daného objektu. Tato vzdálenost se vždy počítá od středu prvního objektu ke středu objektu následujícího.

B.4 Element SCENE

Posledním elementem v dané XML struktuře je element SCENE. SCENE je povinný element a stejně jako element ACTION je nutné ho zavést i v případě, že by model nijak nerozšiřoval. Tento element má dva atributy – repeat a distance. Oba atributy mají stejnou definici jako atribut repeat a distance u elementu ACTION.

PŘÍLOHA C -
PŘEHLED ELEMENTŮ A ATRIBUTŮ VSTUPNÍHO JAZYKA

Následující tabulka udává kompletní přehled elementů a atributů s jejich vlastnostmi. Tento přehled je k dispozici také přímo v aplikaci při spuštění nápovědy.

NÁZEV ELEMENTU	NÁZEV ATRIBUTU	HODNOTA	POVINNÝ	ROZŠÍŘENÍ K...
MODEL ***(párový)***	name	String	Ano	-
	colors	*nameOfColor*:RGB;	Ne	-
	lod	yes no	Ne	-
OBJECT ***(nepárový)***	name	String	Ne	-
	shape	• Box • Cone • Cylinder • Sphere	Ano	-
	size	• x,y,z (double) • od-do (double)	Ano	Box
	radius	• double • od-do (double)	Ano	• Cone • Cylinder • Sphere
	height	• double • od-do (double)	Ano	• Cone • Cylinder
	color	• red • blue • green • yellow • pink • brown • white • black • *nameOfColor*	Ne	-
	visible	random	Ne	-
ACTION ***(nepárový)***	position	• x,y,z (double) • _left_*nameOfObject* • _right_*nameOfObject* • _bottom_*nameOfObject* • _top_*nameOfObject* • _front_*nameOfObject* • _back_*nameOfObject*	Ne	-
	repeat	x,y,z (integer)	Ne	-
	distance	x,y,z (double)	Ne	repeat
SCENE ***(nepárový)***	repeat	x,y,z (integer)	Ne	-
	distance	**x,y,z (double)**	**Ne**	**repeat**

Tabulka B.1: Přehled elementů a atributů XML struktury

Vysvětlivky:

double – hodnota s pohyblivou řádovou čárkou

integer – celočíselná hodnota

String – řetězec znaků

od-do – rozmezí minimální a maximální hodnoty

x,y,z – forma zadávání hodnot; definuje hodnoty na lokálních osách

PŘÍLOHA D - PŘEKLADOVÁ GRAMATIKA

Terminály:

braceL	{
braceR	}
bracketL	[
bracketR	]
qm	“
kw...	keyword... (kličove slovo)
numb	čislo
text	„text“
node	již vytvořeny PROTO uzel
atribut	atribut konkretniho PROTO uzlu

Tabulka C.1: Terminály překladové gramatiky

Přepisovací pravidla:

MODEL → kwProto text bracketL FIELDS bracketR braceL MODEL1 braceR
FIELDS → kwField TYPE text VALUE FIELDS | ε
TYPE → kwSFVec2f | kwSFVec3f | SFString | SFBool
MODEL1 → NAME kwGroup braceL OBJECTS braceR
OBJECTS → OBJECT OBJECTS | ε
NAME → kwDef text
NODE → BOX | CONE | CYLINDER | SPHERE | USE | NEW
NEW → node braceL ATRNODE braceR
ATRNODE → atribut ATRNODE | ε
DIMENSIONS → kwSize VEC3
RADIUS → kwRadius numb
HEIGHT → kwHeight numb
VALUE → VEC2 | VEC3
VEC3 → numb numb numb
VEC2 → numb numb
AXIS → kwTranslation START | kwTranslation DISTANCE
START → VEC3
SCENE → REPEAT
REPEAT → OBJECTS
USE → kwUse text | NODE
DISTANCE → VEC3
OBJECT → kwTransform braceL ATRTRANSFORM braceR
ATRTRANSFORM → AXIS braceL kwChildren bracketL NODE bracketR
MATERIAL → kwAppearance braceL kwMaterial braceL ambientIntensity 0 shininess 0 braceR braceR
BOX → NAME kwShape braceL kwGeometry kwBox braceL kwSize VEC2 braceR MATERIAL braceR
CONE → NAME kwShape braceL kwGeometry kwCone braceL ATRCONE braceR MATERIAL braceR
ATRCONE → kwBottomRadius numb ATRCONE | kwHeight numb ATRCONE | ε
CYLINDER →NAME kwShape braceL kwGeometry kwCylinder braceL ATRCYLINDER braceR MATERIAL braceR
ATRCYLINDER → kwHeight numb ATRCYLINDER | kwRadius numb ATRCYLINDER | ε
SPHERE → NAME kwShape braceL kwGeometry kwSphere braceL kwRadius numb braceR MATERIAL braceR

PŘÍLOHA E - INSTALAČNÍ A UŽIVATELSKÁ PŘÍRUČKA

Aplikaci GenN není třeba nijak instalovat. Stačí z přiloženého CD zkopírovat soubor GenN.exe.

Pro její správné zobrazení je však nutné mít nainstalovanou technologii Java - `http://www.oracle.com/technetwork/java/javase/downloads/index.html`.

A pro zobrazení VRML modelu některý z VRML prohlížečů jako je např.

Cortona 3D Viewer:

`http://www.cortona3d.com/Products/Cortona-3D-Viewer.aspx`

BS Contact Player:

`http://www.bitmanagement.com/en/products/interactive-3d-clients/bs-contact`.

PŘÍLOHA F – OBSAH CD

- Bakalářská práce
- Aplikace GenN
- Složka Models obsahující ukázkové XML struktury pro různé modely
- readme.txt