Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE

Milan Jaška

Generování obrázků a animací pomocí L-sytémů

Kabinet software a výuky informatiky

Vedoucí bakalářské práce: RNDr. František Mráz, CSc.
Studijní program: Informatika, Programování

2007

Chtěl bych poděkovat mému vedoucímu bakalářské práce za odborné rady, pomoc, ale hlavně trpělivost, které mi pomohli při vypracování této práce.

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

V Praze dne Milan Jaška

Obsah

Název práce: Generování obrázků a animací pomocí L-Systémů
Autor: Milan Jaška
Katedra: Kabinet software a výuky informatiky
Vedoucí bakalářské práce: RNDr. František Mráz, CSc., KSVI MFF UK
e-mail vedoucího: mraz@ksvi.ms.mff.cuni.cz

Abstrakt: Cílem práce je navrhnout systém na editaci a vizualizaci planárních a prostorových scén obsahujících elementární objekty a objekty popsané Lindenamayerovými systémy. Vizualizace planárních scén bude přímá, prostové scény budou vizualizované prostřednictvím popisu v jazyce VRML. K vizualizaci VRML bude použit externí program. Systém by měl umožnit zachytávat vybraná stádia vývoje objektů podle Lindenmayerových systémů tak, aby bylo možné animovat vývoj objektu.

Klíčová slova: Lindenmayer, L-Systém, želví grafika, VRML

Title: Images and animations generating using L-Systems
Author: Milan Jaška
Department: Department of software and computer science education
Supervisor: RNDr. František Mráz, CSc., KSVI MFF UK
Supervisor's e-mail address: mraz@ksvi.ms.mff.cuni.cz

Abstract: Main goal of this thesis is to design system for editing and visualizing planar and cubical scenes consisting of elementary objects and objects described by Lindenmayers systems. Planar scenes visualization will be straight and cubical scenes visualization will be done through VRML language. VRML data will be visualized using external program. System should be able to capture chosen stages of Lindenmayer objects development as pictures so that it is possible to animate the development of object.

Keywords: Lindenmayer, L-System, turtle graphics, VRML

1. Úvod

1.1. Cíl

Cílem této práce je vytvoření softwarového systému k práci s L-Systémy, který bude pro uživatele snadno a intuitivně použitelný i bez toho, aby absolvoval zdlouhavé seznamování se specifikací vstupu. Pokud ví, co jsou to L-Systémy a chce s nimi pracovat, proniknutí do toho systému by mělo být otázkou několika minut. Kromě toho by výsledky, které tento systém vyprodukuje, měly být snadno použitelné pro další účely. Měly by tedy obsahovat komplexní informace o vygenerovaných objektech a zároveň by měly být ve formátu, který bude použitelný jako vstup nějakého freewarového softwaru, a který bude snadno převeditelný a upravitelný.

Zároveň by systém měl programátorům umožňovat snadné rozšiřování jeho funkčních možností, které by mělo spíše představovat přidělávání dalších funkcí či modulů, aniž by bylo nezbytné provádět systémové změny v již hotovém kódu.

1.2. Motivace

V současné době existují jak velmi komplexní systémy - např. L-Studio, tak vcelku jednoduché a pro začátečníka velmi vhodné jako např. LSysMaker. Ať se však jedná o jakýkoliv již existující systém, je vždy poměrně zdlouhavé naučit se, jakým způsobem se L-Systémy do daného systému zadávají, ve srovnání s dobou, kterou člověk potřebuje k seznámení s L-Systémy samotnými. V systémech také postrádám export vytvořených objektů do jiných formátů, které by uživatel mohl dále zpracovávat, než do formátu obrázků.

Motivací této práce je spíše než poskytnutí badateli nad L-Systémy zcela komplexního a složitého systému, poskytnutí konzumentovi jednoduchý systém, s jehož pomocí si vygeneruje data, která mu budou sloužit jako vstupní data do dalších aplikací a pro další zpracování. Příkladem takového použití by mohlo být vygenerování animovaných objektů do textur nebo do pozadí her.

1.3. O čem je tato práce

Tato práce je seznámením s tím, co jsou to L-Systémy, popisem analýzy, vývoje a implementace systému, se závěrečným hodnocením a myšlenkami na další vývoj. Seznámení s L-Systémy je pro tuto práci nezbytné, další kapitoly by bez něj čtenáři nedávaly smysl.

Práce v žádném případě nezastává funkci programátorské nebo uživatelské dokumentace vytvořeného systému. Tyto dokumentace jsou dodávány zvlášť na nosiči s vytvořeným softwarovým systémem.

2. Lindenamayerovy systémy

2.1. Základní definice

K definici L-Systémů je potřeba několik základních znalostí z oblasti gramatik, proto si je zde uveďme. **Abecedou** rozumíme konečnou neprázdnou množinu symbolů. **Slovem** rozumíme konečnou (třeba i prázdnou) posloupnost symbolů z dané abecedy. Máme-li abecedu X, potom jako X^* označme množinu všech slov v abecedě X a X^+ jako množinu všech neprázdných slov v abecedě X. Se slovy lze provádět operaci **zřetězení** (konkatenaci). Nechť máme slova $\chi_1 = a_1...a_m, \chi_2 = b_1...b_n \in X^*$. Potom jejich zřetězením značeným $\chi_1\chi_2$ rozumíme posloupnost délky $m+n$, pro kterou platí $\chi_1\chi_2 = a_1...a_m b_1...b_n$.

Základní L-Systém (označovaný jako **0L-Systém**) je potom uspořádaná trojice $L = (X, \omega, P)$, kde X je abeceda systému, $\omega \in X^+$ je slovo nazývané **axiom** a $P \subset X \times X^*$ je konečná množina tzv. **přepisovacích pravidel**.

Základní myšlenkou L-Systémů je generování řetězců a jejich následná grafická interpretace. Axiom je prvotním řetězcem, který můžeme interpretovat. Obvykle však interpretujeme až řetězce, které z tohoto axiomu pomocí sady přepisovacích pravidel odvodíme.

Chceme-li, aby náš systém byl deterministický (**D0L-Systém**), musí pro každý symbol $a \in X$ existovat právě jedno pravidlo $(a, \chi) \in P$. Symbol a nazýváme **přepisovaným symbolem** a slovo χ nazýváme **pravou stranou pravidla**. Pokud není pravidlo pro daný symbol explicitně zadané, předpokládá se existence implicitního identického pravidla $(a, a) \in P$.

Mějme slovo $\mu = \mu_1...\mu_l$. Řekneme, že slovo $\upsilon = \chi_1...\chi_l$ vzniklo **přímým odvozením** (derivací, přepsáním) slova μ (značení $\mu \Rightarrow \upsilon$), jestliže $(\mu_i, \chi_i) \in P$ pro všechna $i = 1,...,l$. O slově υ řekneme, že je generované L-Systémem L **derivací délky** n, jestliže $\upsilon_0 = \omega \Rightarrow \upsilon_1 \Rightarrow ... \Rightarrow \upsilon_n = \upsilon$.

Důležité je, že se při přímém odvození všechny symboly řetězce přepisují zároveň.

2.2. Grafická interpretace řetězce

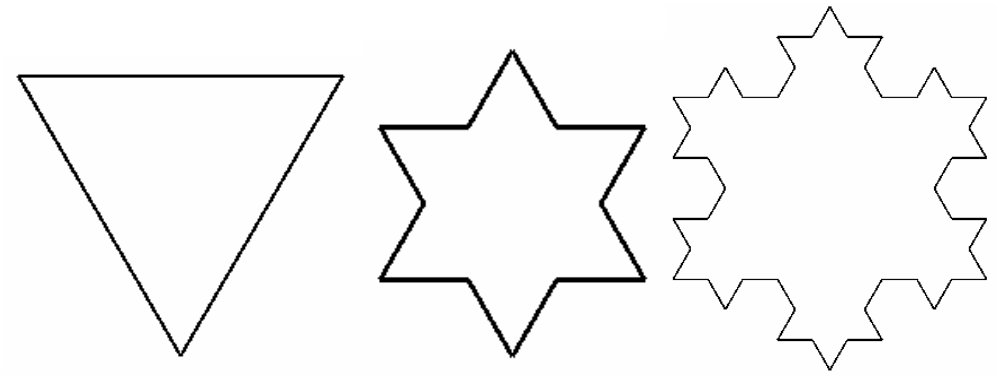
Smyslem generování řetězců pomocí L-Systémů je jeho grafická interpretace. Grafická interpretace se provádí pomocí tzv. **želví grafiky**. Symboly řetězce jsou čteny zleva doprava a předávány jako příkazy želvičce, která takto řízená vykresluje grafický výstup.

Před začátkem grafické interpretace řetězce má želva nastavenu polohu, směr dalšího postupu, délku kroku d a úhel pootočení δ. Aktuální stav želvy ve 2D je zcela popsán trojicí (x, y, α), kde x a y určují aktuální polohu želvy a α určuje úhel - směr dalšího postupu. Ve 3D je situace podobná, musíme si však navíc kromě směru pamatovat také jak je želvička „nakloněna“. Příkazy pro želvičku ve 2D mohou vypadat následovně:

Symbol	Interpretace želvou
F	Želva se posune o krok dopředu a cestou k novému umístění kreslí čáru. Z původního stavu (x, y, α) se želva dostane do stavu $(x + d * \cos\alpha, y + d * \sin\alpha, \alpha)$.
f	Želva se posune o krok dopředu (cestou nekreslí čáru). Z původního stavu (x, y, α) se želva dostane do stavu $(x + d * \cos\alpha, y + d * \sin\alpha, \alpha)$.
+	Želva se pootočí doleva. Nový stav želvy je $(x, y, \alpha + \delta)$.
-	Želva se pootočí doprava. Nový stav želvy je $(x, y, \alpha - \delta)$.

Tabulka 1 - Příklad symbolů interpretovaných želvičkou

Hezkým ukázkovým příkladem použití L-Systémů ve 2D je Kochova vločka. K jejímu zadání stačí axiom $\omega = F--F--F$ a jediné pravidlo $(F, F+F--F+F)$. Před grafickou interpretací je nutné nastavit želvičce úhel pootočení $\delta = 60°$.



Obrázek 1 – Axiom Kochovy vločky, a derivace délky 1 a 2

Výsledkem interpretace L-Systému je vždy lomená čára, jejíž části se mohou protínat nebo mohou být neviditelné.

Předchozí příklad rovněž ukazuje použití L-Systémů ke tvorbě fraktálů. Takovýchto fraktálů, které můžeme vykreslit „jednoduchým“ L-Systémem je mnoho. Původním smyslem L-Systémů však mělo být použití při vykreslování přírodních struktur (především rostlin), jejichž vývin lze zachytit do pravidel L-Systému.

2.3. Parametrické L-Systémy

V **parametrických L-Systémech** je možné přiřadit každému symbolu v množině pravidel **formální parametry**. Symbol A s parametry $a_1,...,a_n$ budeme značit jako $A(a_1,...,a_n)$. **Aktuální (skutečné) parametry**, které se nacházejí v přepisovaném řetězci korespondují s formálními parametry přepisovaného pravidla. Symbol s aktuálními parametry v řetězci vypadá takto $A(r_1,...,r_n)$, kde r_i je jakékoliv reálné číslo pro všechna $i = 1,...,n$.

Parametrické rozšíření nám dává možnost přidat do přepisovacích pravidel **podmínkovou část**. Podmínková část je tvořena **logickým výrazem**. Logické výrazy mohou být spojovány do složitějších logických výrazů logickými operátory ! (logická negace), & (logický součin) a | (logický součet). Nejjednodušší logický výraz vznikne jako spojení dvou **aritmetických výrazů** pomocí některého z **relačních operátorů** <, >, =, <= a >=. Podmínková část má svůj smysl díky tomu, že v těchto aritmetických výrazech můžeme použít názvy formálních parametrů, které v nich reprezentují odpovídající hodnotu skutečného parametru. Prázdný výraz v podmínkové části se vyhodnocuje vždy jako logická pravda.

Při přepisování řetězců v parametrických L-Systémech musí být shodný nejen přepisovaný symbol, ale také počet formálních a aktuálních parametrů a musí být ještě splněna podmínka uvedená v podmínkové části. Po splnění těchto podmínek může být teprve symbol přepsán podle tohoto pravidla.

Pro každý symbol v řetězci, pro který neexistuje explicitně zadané pravidlo, které by mu odpovídalo symbolem, počtem parametrů, a které by mělo splněnou podmínku, existuje opět implicitní identické pravidlo, které symbol přepíše i s jeho skutečnými parametry.

V pravých stranách pravidel parametrických L-Systémů se mohou objevit pravidla, která obsahují symboly v následující podobě: $S(s_1,...,s_m)$, kde s_i je aritmetický výraz pro všechna $i = 1,...,m$. V těchto aritmetických výrazech se mohou rovněž objevovat názvy formálních parametrů. Aritmetické výrazy jsou při přepisování redukovány na jejich hodnoty – tj. na reálná čísla.

Díky parametrickému rozšíření můžeme také rozšířit sadu symbolů interpretovaných želvičkou, např.:

Symbol	Interpretace želvičkou
$F(l)$	Posun želvičky o délku l, cestou kreslí čáru.
F	Posun želvičky o pevnou délku d, cestou kreslí čáru.
$f(l)$	Posun želvičky o délku l bez kreslení čáry.
f	Posun želvičky o pevnou délku d bez kreslení čáry.
$+(\alpha)$	Pootočení želvičky doleva o úhel α.
$+$	Pootočení želvičky doleva o pevný úhel δ.
$-(\alpha)$	Pootočení želvičky doprava o úhel α.
-	Pootočení želvičky doprava o pevný úhel δ.

Tabulka 2 - Příklad symbolů interpretovaných želvičkou v parametrickém L-Systému

2.4. Stochastické L-Systémy

Pokud by byly použity L-Systémy podle základní definice pro generování většího počtu rostlin než jedna, nastala by potíž s tím, že všechny takto vygenerované rostliny by byly identické. Stochastický mechanismus nabízí možnost jak do generování jednotlivých kusů rostlin vnést náhodný prvek, ale zároveň zanechat topologii všech rostlin nezměněnou.

Modifikujme základní denici na čtveřici $L_\pi = (X, \omega, P, \pi)$, kde $\pi : P \to (0;1]$ se nazývá rozdělení pravděpodobností pravidel. Opět je nutno si říci, jak probíhá přepisování pravidel podle nové definice L-Systému. Při stochastickém odvození $\mu \Rightarrow \upsilon$ je důležité, aby pro každý výskyt symbolu a v řetězci μ byla pravděpodobnost aplikace pravidla $(a, \chi) \in P$ rovna $\pi(a, \pi)$. Znamená to také, že v jednom kroku odvozování může být výskyt stejného symbolu v řetězci nahrazen podle jiného pravidla.

2.5. Závorkové L-Systémy

Pro generování stromových struktur se ukázalo jako výhodné, rozšířit množinu příkazů pro želvičku o příkazy [a], které slouží k uložení k uložení jejího aktuálního stavu (polohy, směru a ve 3D naklonění) do zásobníku a k vyzvednutí tohoto stavu ze zásobníku.

Ve skutečnosti bychom se mohli krkolomným způsobem použití těchto symbolů vyhnout, do některé z předchozích poloh bychom se vždy mohli dostat příslušným natočením a posunutím, při kterém bychom kreslili neviditelnou čáru. Použití těchto symbolů však celou situaci značně zjednodušuje a navíc poskytuje poněkud „přehlednější“ tvar řetězce. Vždy, když je želvička v místě, kde chceme provést větvení, použijeme symbol [k uložení této pozice, vykreslíme větev zakončenou symbolem] pro návrat do uložené pozice, a můžeme pokračovat stejným způsobem ve vykreslování další větve jdoucí z tohoto místa.

2.6. Kontextové L-Systémy

Nové možnosti mohou L-Systémy také získat, pokud přepisovací pravidla budou **závislá na kontextu**, ve kterém se přepisovaný symbol právě nachází. Díky tomuto rozšíření lze například modelovat šíření látek v rostlinách.

Definic kontextových L-Systémů je několik. **2L-Systémy** předepisují, aby pravidlo mělo levý i pravý kontext a aby oba tyto kontexty byly stejně dlouhé (stejný počet symbolů). **1L-Systémy** jsou kontextové L-Systémy, které mají kontexty v pravidlech pouze na jedné straně. Širší třídou kontextových L-Systémů jsou tzv. **(k,l)-Systémy**, které mají levé kontexty délky k a pravé kontexty délky l.

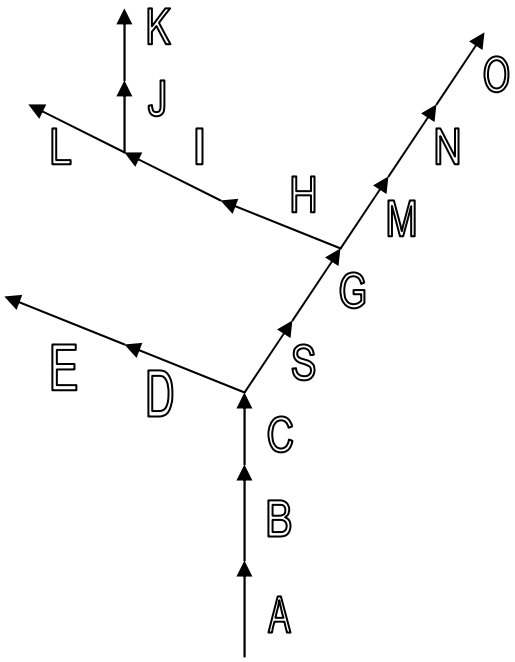
Pro jednoduchost našeho systému budeme užívat definici, kde povolíme v jednom L-Systému existenci pravidel s různými délkami levých i pravých kontextů.

Zároveň povolíme v kontextech přítomnost formálních parametrů u jednotlivých symbolů. Tyto formální parametry se stejně, jako formální parametry přepisovaného symbolu, mohou objevit také v podmínce pravidla a na pravé straně pravidla.

Při testování pravidel pro přepis symbolu v řetězci postupujeme tak, že nejprve ověříme shodnost symbolu v řetězci se symbolem a pravidle, počet skutečných a formálních parametrů, symboly v levém a pravém kontextu včetně počtů skutečných parametrů a ověříme podmínku. Pokud takto vyhovuje více pravidel, má kontextové pravidlo (pravidlo s neprázdnými alespoň jedním kontextem) přednost před aplikací bezkontextového pravidla. Pokud nám zbude více než jedno pravidlo, můžeme ještě použít stochastický mechanismus.

Interpretace řetězců s kontextovými pravidly se ještě komplikuje, protože při modelování rostlin se obvykle používají závorkové L-Systémů se symboly [a].

Mějme řetězec $ABC[DE][SG[HI[JK]L]MNO]$, který reprezentuje strom na obrázku 2. Potom můžeme symbol S přepsat podle pravidla se stejným symbolem, které má levý kontext BC a pravý kontext $G[H]M$, protože při testování levého kontextu nás zajímá, jak jsme k danému symbolu došli cestou od kořene, a tudíž musíme větev $[DE]$ při testování vynechat. S pravým kontextem je to ještě poněkud složitější. Zatímco v levém pouze vynecháváme větve „vyššího řádu“, v pravém kontextu naopak musíme otestovat všechny možné větve, které z daného bodu následují. U pravého kontextu je nutné, aby se alespoň jedna z větví shodovala. Navíc, v pravém kontextu zadaném v pravidle nemusí být větev uvedena celá. Stačí, je-li tam uveden její začátek, který se má shodovat. V našem případě vynecháváme při testování zbytek větve, která končí symboly $I[JK]L$.



Obrázek 2 - Strom

Někdy se nám může hodit, aby se při testování shody kontextu v pravidle proti skutečnému kontextu v řetězci, vynechávali ze skutečného kontextu některé symboly, které jsou v něm navíc. Symboly, které na kterých nám při testování kontextu na shodu nezáleží. Takovéto symboly budeme označovat jako **symboly ignorované při testování kontextu**. Pokud máme L-Systém, ve kterém chceme mít takovéto ignorované symboly, musíme je vždy specifikovat ve vstupu příslušné aplikace.

2.7. Homomorfismus

Výhodným rozšířením pro aplikace zpracovávající L-Systémy je přidání možnosti použití dvou sad pravidel. V takových systémech potom první sada pravidel slouží k vygenerování struktury objektu a druhá sada, která se na řetězec aplikuje vždy maximálně jednou a to při jeho posledním přepisování, slouží k přípravě objektu (řetězce) na jeho grafickou interpretaci. Tato pravidla by neměla měnit logiku struktury řetězce. Druhé sadě pravidel se říká **pravidla homomorfismu**.

Obě sady pravidel mají stejný tvar a platí pro ně stejná pravidla, jako pro pravidla L-Systémů o kterých jsme zde mluvili.

Kromě přípravy řetězce na grafickou interpretaci mají pravidla homomorfismu ještě jeden smysl. Díky nim můžeme totiž ušetřit mnoho času při přepisování řetězce. V běžných L-Systémech se totiž vyskytují pravidla, jejichž pravé strany mají délky i v desítkách symbolů. Uvědomíme-li si, že se při přepisování podle těchto pravidel řetězec prodlužuje exponenciální rychlostí, znamená to, že při několika málo přepsáních na začátku velmi krátkého řetězce, můžeme ve výsledku pracovat s řetězci s délkami mnoho-řádově delšími.

Proto, pokud těchto pravidel využijeme a použijeme při prvních $n-1$ přepisováním normálních pravidel k vygenerování struktury řetězce, a poté aplikujeme při n-tém přepisování pravidla homomorfismu, může k řádovému prodloužení řetězce dojít až při posledním přepisování, zatímco předchozí fáze pracovali s mnohem kratšími řetězci, a tudíž proběhly rychleji.

3. Analýza, návrh a implementace systému

3.1. Počáteční analýza

Při zkoumání stávajících systémů zjistíme, že všechny mají jako vstup prostý textový soubor. Proto je také nutné se naučit, jak má být tento soubor strukturovaný a jaké jsou syntaxe jednotlivých jeho částí. Námi vytvářený systém by se však měl tomuto způsobu zadávání vstupu vyhnout a měl by být pokud možno zprostředkován formulářovými okny.

Při zadávání prostřednictvím formulářových oken se však zcela nevyhneme nutnosti i tak provádět na zadaných datech lexikální a syntaktickou analýzu. Abychom vytvořili rychlé a spolehlivé analyzátory, bylo by nejlépe sáhnout po některých z generátorů kódu. Autor si pro tyto účely zvolil dvojici GNU programů – flex a bison. Tato volba již cosi předesílá o volbě programovacích prostředků, a sice o jazyce C.

Vytvářet formulářové pracovní prostředí v jazyce C však není úplně nejvhodnější volbou, proto se zrodila myšlenka systém rozdělit na dva aplikační programy – jeden, napsaný v jazyce C, který bude umět přepisovat řetězce L-Systémů na základě vstupního textového souboru, a druhý, který bude sloužit jako uživatelské prostředí a bude k části své práci využívat první. Uživatelské prostředí poté bude možné vytvořit pohodlnějším způsobem v některém z dalších programovacích prostředků. Autor si zvolil jazyk C# spolu s platformou .NET.

Takovéto rozdělení má v konečném důsledku i další výhody. Jednak vzniknou dvě menší aplikace, jejichž úpravy, rozšíření a opravy budou snažší, a jednak se tímto dá možnost interpretovat řetězce vygenerované první aplikací ve zcela jiné aplikaci a například je převádět do zcela jiného formátu, než do jakého bude převáděn v našem systému.

Otázkou zůstává volba výstupního formátu. Samozřejmostí je výstup do souborů s obrázky. Pro komplexnější výstup podle našich požadavků však použijeme jazyk VRML. VRML soubory můžeme snadno zobrazovat a převádět do nejrůznějších formátů i pomocí freewarových programů. Hezkou vlastností tohoto formátu je také existence softwaru Cortona VRML Client, který po své instalaci přidá do systému Windows komponentu, pro zobrazování VRML kódu v uživatelské aplikaci.

3.2. Dekompozice úlohy

Úlohu rozdělíme na dvě aplikace. Aplikace **lsgen** bude dostávat na vstupu informace o L-Systému – délka derivace, zda se mají použít pravidla homomorfismu, ignorované symboly, samotná pravidla a počáteční řetězec, a jejím výstupem bude textový soubor s cílovým řetězcem.

Druhá aplikace, aplikace **eL**, bude sloužit jako uživatelské prostředí, ve kterém budeme zadávat prostřednictvím formulářů vytvářet stromovou strukturu scény, která bude moci kromě L-Systémů obsahovat i některé další tvary a tělesa, a kterou budeme moci ukládat do soboru XML a opětovně ji načítat. Tuto strukturu budeme moci také převádět do VRML, zobrazovat a exportovat. eL bude k přepisování L-Systémů využívat externí aplikaci lsgen.

3.3. Aplikace lsgen

Nyní se podíváme podrobněji na implementaci aplikace lsgen. Začneme popisem jejího rozhraní, vstupu, poté se podíváme na jeho zpracování, na nejdůležitější algoritmy a datové struktury této aplikace.

Ještě však poznamenejme, že aplikace lsgen je primárně určená pro běh v konzolovém prostředí 32 bitových systémů Windows. Její zdrojový kód by však vzhledem k použitým technologiím neměl působit žádné potíže při překladu i na jiných platformách (zcela bez potíží by měly být systémy UNIX a Linux).

3.3.1. Rozhraní aplikace lsgen

Aplikace lsgen nemá žádné uživatelské prostředí. Při spouštění máme možnost pomocí parametrů spuštění specifikovat vstupní a výstupní soubor. Pokud tak neučiníme, budou použity standardní vstup a výstup. Úspěšnost zpracování vstupního souboru je signalizována návratovou hodnotou programu. Nulová návratová hodnota signalizuje úspěšnost, nenulová neúspěšnost. Při neúspěchu by měl program na chybový výstup vypsat hlášení o tom, k jaké chybě došlo.

Svým rozhraním aplikace připomíná spíše typický UNIXový nástroj.

3.3.2. Formát vstupního souboru aplikace lsgen

Vstup aplikace je rozdělen do 4 sekcí. V první z nich se specifikuje délka derivace, zda se mají použít pravidla homomorfismu, které symboly jsou ignorovány,

definice konstant, apod. V druhé a třetí sekci se definují pravidla L-Systému (normální a pravidla homomorfismu) a ve čtvrté sekci se nachází pouze jeden řádek s počátečním řetězcem.

S aplikací lsgen můžeme používat pravidla veškerých typů L-Systémů, o kterých jsme se v předchozí kapitole zmínili.

3.3.3. Zpracování vstupu

Jelikož se může stát, jak jsme již zmínili, že budeme pracovat s velkými daty, a není tedy možné, abychom vstupní soubor načetli do paměti a poté zpracovali. Práce nad vstupními daty bude tedy probíhat přímo při jejich sekvenčním čtení.

Při parsování prvních tří sekcí se aplikace učí – ukládá si do paměti všechna nastavení a pravidla, která se v těchto sekcích objevila. Tuto část vstupu ještě můžeme uchovávat v paměti, protože s největší pravděpodobností nebude příliš rozměrná. Při čtení čtvrté sekce však budeme muset přečtené symboly co nejdříve přepsat do dočasného nebo výsledného souboru. Do dočasného souboru píšeme tehdy, jestliže toto přepisování není posledním.

3.3.4. Formát výstupního souboru

Po úspěšném provedení všech přepsání programem lsgen bychom měli ve výstupním souboru nalézt jedinou řádku (pravděpodobně velmi dlouhou) s výstupním řetězcem – derivace zadané délky s připadnou aplikací pravidel homomorfismu. Řetězec by neměl obsahovat mezery a měl by obsahovat pouze symboly se skutečnými parametry.

3.3.5. Učení pravidel

Aplikace si pamatuje pravidla pomocí pole, indexovaného od 0 až po počet všech možných symbolů použitelných jako symboly L-Systému -1. Položky tohoto pole jsou ukazatele na první pravidlo lineárního seznamu pravidel pro daný symbol.

Pro snadnou indexaci používá funkci *int sym2tab(char *c)*, která pro daný symbol L-Systému vrátí jeho index v tabulce. To, že jsou pravidla v lineárních seznamech, a že ke každému z nich nemáme okamžitý přístup, nevadí, protože při výběru pravidla, které budeme aplikovat na přepsání symbolu, je potřeba otestovat všechna pravidla pro tento symbol.

Struktury, které jsou pro ukládání pravidel v paměti použity následují, jejich popis je uveden v komentářích ve stylu jazyka C:

```
// jeden formální parametr – jeden prvek spojového seznamu
struct formal_parameter
{
	char *name;				// jméno parametru

	struct formal_parameter *prev;	// propojení seznamu
	struct formal_parameter *next;
};

// struktura zastřešující formální parametry daného symbolu
// a pravidla, obsahuje jejich počet, aby při testování proti
// počtu skutečných parametrů nemusely být parametry spočítány,
// a ukazatel na první parametr v seznamu
struct formal_parameters
{
	int count;				// počet

	struct formal_parameter *first;	// ukazatel na první
};

// jeden symbol v levém nebo pravém kontextu v pravidle;
// spolu se symboly ukládáme rovněž jejich formální parametry;
// struktura obsahuje ukazatele na propojení více symbolů do seznamu
struct context_symbol
{
	char symbol;
	struct formal_parameters *params;

	struct context_symbol *prev;
	struct context_symbol *next;
};

// „hlavička“ seznamu symbolů v kontextu pravidla;
// jelikož se používá předchozí struktura k uložení jak pravého, tak
// levého kontextu, obsahuje hlavička ukazatel jak na první, tak na poslední
// položku seznamu (při testování levého kontextu začneme posledním
// symbolem, při testování pravého prvním);
// zároveň se opět ukázalo jako výhodné, mít v hlavičce uvedenou délku
struct context_symbols
{
	struct context_symbol *first;
	struct context_symbol *last;

	int count;
};
```

```
// jedno pravidlo seznamu; obsahuje ukazatele na „podstruktury“ obsahující
// formální parametry, podmínku, kontexty, pravděpodobnost použití,
// pravou stranu pravidla, a ukazatel na následující pravidlo seznamu
struct rule
{
        struct formal_parameters *params;
        char *condition;
        struct context_symbols *left_context;
        struct context_symbols *right_context;
        double probability;          // pokud nemá pravidlo určenou
        char *right_side;            // pravděpodobnost, je nastavena
                                     // na zápornou hodnotu
        struct rule *next;
};
```

3.3.6. Vícenásobné přepisování

Jestliže aplikace dostane za úkol přepsat počáteční řetězec pouze jednou, píše výstupní řetězec přímo do výstupního souboru. Situace je poněkud zajímavější, pokud máme počáteční řetězec přepsat vícekrát. Nabízí se několik možností, jak vzniklou situaci řešit. Hezké by bylo typické UNIXové řešení, s přepisováním a zápisem nefinálních řetězců do roury další instanci aplikace. To je ale v prostředí Windows jen těžkopádně použitelné, jelikož Windows implementuje roury tak, že vytvoří dočasný soubor, do něhož nechá první aplikaci zapsat, a teprve po jeho uzavření jej předá na vstup aplikaci druhé.

Implementované řešení je však v zásadě hodně podobné tomuto, akorát že všechna přepsání obstará jedna instance aplikace. Aplikace zapisuje nefinální řetězce do dočasných souborů, a vždy, když dojde na konec souboru, přesměruje vstup na předchozí dočasný soubor, sníží počet zbývajících přepsání o 1, a pokračuje s přepisováním. Pouze při posledním přepisování zapisuje přímo do výstupního souboru.

3.3.7. Vlastnosti pravidel a jejich výběr při přepisování

Dokud jsou v zadání pouze pravidla, která jsou bezkontextová, je situace velmi snadná. Při přečtení dalšího symbolu řetězce můžeme v nejjednodušším případě pouze srovnat symbol v pravidle a v řetězci, ve složitějších pak počet formálních a skutečných parametrů, ověřit platnost podmínky, a při zadání více stochastických pravidel vybrat s patřičnou pravděpodobností jedno z nich. S kontextovými pravidly je však potíž.

Jelikož čteme vstup sekvenčně, nezbývá nám, než si pro účely testování shody levého kontextu ukládat jistou část již přečteného a možná i přepsaného řetězce. Co však hůř, pro účely testování pravého kontextu jsme někdy nuceni přečtené symboly nepřepisovat přímo, ale zařadit je do fronty symbolů, čekajících na přepsání, dokud nebudeme mít přečtený dostatečný počet symbolů, které se nachází v řetězci za nimi, abychom mohli správně otestovat shodu pravého kontextu.

Za zmínku asi stojí myšlenka, testovat kontexty přímo „ze souboru“. Kromě složitého pohybu a několika násobného parsování stejné části řetězce, by toto řešení představovalo a neúměrnou zátěž na systém, protože by docházelo k mnohonásobnému přečtení jednoho symbolu v souboru. Cena přístupu k diskové paměti by se jistě projevila a výkonnostně by byla celá aplikace mnohem níže.

Pro vytvoření fronty čekajících symbolů je použit obousměrný lineární spojový seznam, stejně tak jako pro udržení symbolů z kontextu řetězce. Levý kontext totiž testujeme na shodu zprava, pravý zleva.

Algoritmus při čtení symbolů řetězce a jejich přepisování, je-li přítomno aspoň jedno kontextové pravidlo, vypadá takto:

1) Přidej právě načtený symbol i s jeho reálnými parametry do fronty symbolů čekajících na přepsání.
2) Pokud tento symbol není mezi symboly ignorovanými v kontextových systémech, přidej jej také na konec seznamu symbolů pro účely testování kontextu. Pokud symbol není ignorován, podívej se, od konce, jestli mají symboly čekající na přepsání nastavený odkaz na jejich aktuální pravý kontext, a pokud ne (jejich hodnota je NULL), nastav jim jej na tento symbol.
3) Zkus přepsat první ze symbolů čekajících na přepsání.
4) Pokud pro učinění rozhodnutí o tom, které pravidlo má být použito, není v paměti ještě dostatečně dlouhý pravý kontext, načti další symbol a pokračuj krokem 1. Pokud se již ve vstupním řetězci další symbol nenachází (jsme na konci souboru), pokračuj krokem 6.
5) Pokud se krok 3 podařil, odstraň přepsaný symbol z fronty. Zároveň můžeš zkrátit posloupnost uložených symbolů pro účely testování kontextu zleva. Pokračuj krokem 3.
6) Tento krok je konečný pro jednu fázi přepisování. Přepiš všechny symboly čekající na přepsání. Pokud je pravý kontext pro přepsání

kteréhokoliv ze symbolů příliš krátký, použij bez kontextové pravidlo, třeba i implicitní identické. Poté uvolni celou frontu symbolů čekajících na přepsání a také všechny symboly uložené v seznamu pro účely testování kontextu.

Uvolňování uloženého kontextu se dělá tak, že se jednak uvolňují větve vyššího řádu, pokud jsou celé v levém kontextu prvního čekajícího symbolu na přepsání, a jednak že se uvolňují zleva symboly, pokud je jich v libovolné větvi více, než je délka nejdelšího levého kontextu v kterémkoliv z pravidel (počítáno na počet symbolů).

Struktury, které pro tyto účely používáme jsou následující:

```
// jeden symbol čekající na přepsání; struktura obsahuje informace
// o symbolu, jeho skutečných parametrech v řetězci, ukazatele na
// poslední symbol v uloženém levém kontextu symbolu a na první
// v pravém kontextu a ukazatele na propojení čekajících symbolů
// do spojového seznamu
struct saved_symbol
{
        char symbol;
        struct real_parameters *real;
        struct saved_context_symbol *left_context;
        struct saved_context_symbol *right_context;

        struct saved_symbol *prev;
        struct saved_symbol *next;
};

// symbol uložený jako část skutečného kontextu v řetězci;
// obsahuje symbol, skutečné parametry a ukazatele na propojení
// do spojového seznamu
struct saved_context_symbol
{                           // kontextu
        char symbol;
        struct real_parameters *real;

        struct saved_context_symbol *next;
        struct saved_context_symbol *prev;
};
```

Pro komplexnost si ještě uveďme struktury pro uchování skutečných parametrů, které silně připomínají struktury, pro uchování formálních parametrů, pouze s tím rozdílem, že v nich máme uloženy hodnoty namísto jmen:

```
// jeden skutečný parametru seznamu
struct real_parameter
{
	double value;

	struct real_parameter *prev;
	struct real_parameter *next;
};

// hlavička skutečných parametrů
struct real_parameters
{
	int count;

	struct real_parameter *first;
};
```

3.3.8. Testování podmínky a přepisování pravé strany

Jelikož lsgen ukládá podmínky i pravé strany pravidel v podobě řetězce znaků, vzniká nutnost při jejich zpracování použít opět parsery. Lexikální scannery a parsery jsou v aplikaci oba zastoupeny třikrát. Pro zpracování vstupu, podmínky a pravé strany pravidla.

V obou případech – podmínky i pravé strany - tyto řetězce předáváme jako „vstup“ lexikálnímu scanneru a vlastní práci provede parser. Ten, který testuje platnost podmínky má jako svůj výstup hodnotu 1 nebo 0, představující logickou pravdu nebo opak, kterou uloží do globální proměnné. V druhém případě parser přepisuje pravou stranu tím způsobem, že přepíše vždy symbol, a je-li následován závorkami obsahujícími výrazy, která mají být vyčísleny na skutečné parametry, vyčíslí je, a zapíše v patřičné podobě. K výstupu je opět použita globální proměnná s handlem souboru, do kterého se zapisuje právě přepisovaný řetězec.

Ve výrazech v podmínce a i ve výrazech na pravé straně pravidla se mohou vyskytovat názvy proměnných a formálních parametrů. Řešení práce s konstantami a jejich hodnotami, je v zásadě shodná s řešením práce s formálními parametry. lsgen obsahuje dva moduly – „constv“ a „formalv“, které nabízejí mimo jiné funkce *constant_set(), constant_get(), formal_set(), formal_get() a formal_clear()*. Hodnoty konstant jsou nastaveny již při zpracovávání první sekce vstupu. Hodnoty formálních parametrů jsou nastaveny při testování vhodnosti pravidla. Pro oba parsery je poté získání hodnot literálů obou případů jednoduchým použitím funkce *constant_get()* nebo *formal_get()*. Funkce *formal_clear()* se používá před prací s dalším pravidlem k

„vyčištění“ nastavených hodnot formálních parametrů, aby se nestalo, že při práci s některým z dalších pravidel „budeme mít k dispozici“ hodnoty formálních parametrů z jiných pravidel, který byly použity na jiné symboly a v jiném kontextu.

Moduly pro práci s konstantami a formálními parametry ukládají jejich hodnoty do jednoduchých hashovacích tabulek. Tyto tabulky mají nějakou pevně stanovenou velikost (závislé na definovaném makru). Jedna položka této tabulky je ukazatelem na první položku spojového seznamu s konstantami/formálními parametry, které mají stejný hashovaní kód pro svůj název.

3.4. Aplikace eL

Zdrojové kódy aplikace jsou rozděleny do tříd jazyka C#. Třídy, nebo respektive formuláře, které jsou výsledkem použití Form Designeru Visual Studia, nejsou příliš zajímavým námětem k diskusi. Zajímavější jsou třídy, které se starají o vstup a výstup projektu do soboru typu XML - třída *FileOperation*, pro uchování informací o jednotlivých objektech ve stromu scény - jsou použity struktury, které jsou definovány ve jmenném prostoru *eL.TreeObjects*, a také algoritmus vizualizace, který je ve třídě *Visualization*.

3.4.1. Strom scény

Strom scény se nachází na hlavním formulářovém okně aplikace a je pro něj použitá komponenta .NET Frameworku 1.1 *TreeView*. Jednotlivé uzly představují objekty vložené do scény. Vlastností této komponenty je, že každý z uzlů může mít k sobě připojený *Tag* – instanci libovolné třídy. Těmito tagy jsou připojeny instance tříd ze jmenného prostoru *eL.TreeObjects* s kompletním popisem vložených objektů.

Práce se stromem scény je vcelku intuitivní a taková, jako by člověk očekával. Umožňuje vkládání nových objektů a úpravu a odstranění již existujících. Ke každému typu objektu je zvláštní formulářové okno pro jeho úpravu. Při potvrzení změn v objektu ve formuláři by nemělo dojít k „uložení“ chybných údajů, protože veškeré údaje jsou do objektu ukládány pomocí *Properties* tříd jazyka C#, které údaje zkontrolují. Je-li některý údaj zadán nesprávně, uživatel dostane zprávu pomocí dialogového okna o tom, který to je, a dostane příležitost jej ve formuláři opravit.

3.4.2. Ukládání do XML

Jak již bylo předesláno, v projektu je třída *FileOperation* ze jmenného prostoru *eL.FileOperations*, která se specializuje na ukládání stromu scény do souboru, a jeho zpětnou konstrukci z tohoto souboru.

Algoritmus ukládání je velmi jednoduchý a nachází se v metodě *SaveToFile()*. Prochází se postupně do hloubky všechny uzly stromu, od vrchu dolů, a do XML souboru se zapisují nové XML uzly, které odpovídají typům objektů nalezených ve stromě. Opakem je činnost metody *LoadFromFile()*, která čte uzly XML souboru, vytváří instance příslušných tříd podle typu nalezeného XML uzlu, ukládá do něj hodnoty, a přidává nové uzly i s tagy do stromu scény v komponentě *TreeView*.

XML soubor je při načítání kontrolován proti tzv. systémové definici typu dokumentu v soboru *el01.dtd*, který je součástí projektu.

3.4.3. Vizualizace

Třída *Visualization* obsahuje veřejnou statickou metodu Visualize(), která je volána pro visualizaci stromu scény. Podle toho, zda je kořenem stromu objekt typu 2D nebo 3D svět, zavolá metodu Visualize2D() nebo Visualize3D(). Tyto metody se v principu neliší - rozdílné jsou pouze typy objektů, které umí připravit k zobrazení.

Tyto metody prochází strom a přepisují vlastnosti nalezených objektů do dočasného souboru ve formátu VRML. Pokud je ve stromě objekt typu L-Systém, volá si na pomoc ještě metodu *VisualizeLSystem2D()* nebo *VisualizeLSystem3D()*. Tato metoda umí interpretovat řetězec, který mu pomůže do dočasného souboru vygenerovat další statická metoda se jménem *ProcessLSGEN()*. Je to právě metoda *ProcessLSGEN()*, která volá externí program lsgen. Tato metoda nachystá dočasný soubor pro vstup lsgenu, zavolá jej, a nechá si opět do dočasného souboru vypsat výstup.

Pokud metoda *Visualize2D()* nebo *Visualize3D()* uspějí s tvorbou dočasného VRML souboru, metoda *Visualize()* předá název tohoto souboru COM objektu Cortona VRML Clienta, která se nachází rovněž na hlavním okně aplikace, k načtení jeho obsahu a jeho zobrazení.

Po zobrazení můžeme exportovat jak VRML soubor scény, tak pořizovat obrázky.

4. Závěr

4.1. Splnění cíle

Cíl, který byl této práci vytýčen, byl splněn. Použití tohoto systému je jednoduché a intuitivní. Výstupní data jsou velmi komplexní a lze použít jako vstupní data dalších aplikací. Můžeme také pořizovat obrázky ze scény, a to dokonce i u planárních objektů pod různými úhly a z různých vzdáleností.

Ať už se jedná o jakékoliv formáty souborů, se kterými se v systému pracuje, jsou vždy „lidsky čitelné“ a tudíž snadněji zpracovatelné.

Zdrojové kódy obou aplikací jsou dostatečně rozděleny na třídy a metody, resp. moduly a funkce, tak, aby vždy tvořily logicky ucelený celek, ale zároveň aby nebyly natolik rozsáhlé, že by jejich případné studium kvůli připravovaným úpravám nebylo přespříliš náročné. Vůbec celá architektura systému umožňuje relativně snadné rozšiřování.

4.2. Získané zkušenosti

Každý projekt s sebou přináší nové zkušenosti, dokonce i tehdy, nepoužijeme-li nějakou dosud neznámou technologii nebo techniku. Celý projekt byl velmi zajímavý, a přinesl nezměrné množství nových zkušeností. Nejzajímavější však byly získány při seznamování se s generováním analyzátorů pomocí GNU flex a GNU bison, s technologií XML a VRML.

4.3. Nápady na rozšíření projektu

Vývoj, který by aplikaci určitě prospěl, by měl směřovat k rychlejší práci s uživatelským prostředím. Například zadávání konstant a pravidel L-Systémů je oproti původnímu očekávání příliš kostrbaté a lehce nepřehledné. Systém formulářů se mi po implementaci zdá příliš pomalý na používání. Zřejmě by bylo vhodné seskupit pravidla a jejich úpravu do jednoho jediného formuláře, ve kterém by se mezi upravovanými pravidly dalo rychle přepínat. Podobná situace je s konstantami a barvami.

Rychlost a jednoduchost práce by mohla také zvýšit nástrojová lišta. Příjemné pro přípravu animovaných sekvencí by jistě bylo vymyslet způsob, jakým by se dala zvyšovat délka derivací L-Systémů bez nutnosti zasahovat do stromu scény.

Výstup do VRML by šel zefektivnit například spojováním na sebe navazujících segmentů. Ušetřilo by se jak prostoru, tak výpočetního výkonu.

Práce aplikace lsgen se zdá být velmi rychlá, přesto by však bylo možné implementovat některá vylepšení. Relativně jednoduchým zrychlením by mohlo být vytvořením „skladišť“ právě nepoužívaných alokovaných struktur, ze kterých by se přidělovalo při pokusu o jejich alokaci. Alokace paměti je vcelku pomalý proces, proto by tato technika (někdy zvaná jako „svépomocný dispose“) mohla program urychlit.

Také testování pravého kontextu by zřejmě bylo možné udělat o něco rychleji. Jedná se sice o algoritmicky trochu komplikovanější část programu, ale testování se provádí často zbytečně. Například by se pokus o přepisování symbolů čekajících na přepsání mohl opakovat až tehdy, bylo-li přečteno tolik dalších symbolů řetězce, kolik chybělo k otestování pravého kontextu při posledním pokusu.

Rozšíření samotných L-Systémů by bylo možno implementovat mnoho. Zajímavým rozšířením by bylo přidání více sad pravidel pro jeden L-Systém a jejich střídání k simulaci vlivu prostředí.

5. Literatura

[1] - S. Lam, S. A. King: Animation of Tree Development. In: Image and Vision Computing, Massey University, Palmerston North, New Zealand, 26-28 November, 2003, pp. 297-302.

[2] – P. Prusinkiewicz, J. Hanan and R. Měch: An L-system-based plant modeling language. In: M. Nagl, A. Schuerr and M. Muench (Eds): Applications of graph transformations with industrial relevance. Proceedings of the International workshop AGTIVE'99, Kerkrade, The Netherlands, September 1999. LNCS 1779, Springer, Berlin, 2000, pp.395-410.

[3] – Nový V.: Lindenmayerovy systémy v počítačové grafice (Diplomová práce, vedoucí: Pelikán J.). MFF UK, Praha.

[4] – Chvál, J.: Lindenmayer Systems as Effective Tool of Plant Morphogenesis Modeling (in Slovak), In: Kelemen, J., Kvasnicka, V., Pospichal, J. (eds.): Kognice a umely zivot, Slezska univerzita v Opave, Opava, Ceska republika, 2001, pp. 59-69. ISBN 80-7248-107-X

[5] – GNU flex manual, http://www.gnu.org/software/flex/manual/

[6] – GNU bison manual, http://www.gnu.org/software/bison/manual/

[7] – Zrzavý J.: VRML tvorba dokonalých WWW stránek – podrobný průvodce. ISBN: 80-7169-643-9