VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

GENERÁTOR 3D OBJEKTŮ S VYUŽITÍM L-SYSTÉMŮ

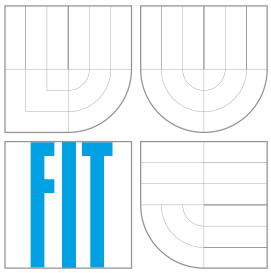
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAKUB KVITA

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

GENERÁTOR 3D OBJEKTŮ S VYUŽITÍM L-SYSTÉMŮ

GENERATOR OF 3D OBJECTS BASED ON L-SYSTEMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JAKUB KVITA
AUTHOR

VEDOUCÍ PRÁCE — Ing. LUKÁŠ VRÁBEL
SUPERVISOR

BRNO 2013

Abstrakt

Cílem této bakalářské práce bylo vytvořit interaktivní systém pro generování 3D modelů. Generátor je založen na L-systémech jako druhu formálních gramatik a želví grafice pro 3D modelování. Aplikace byla vytvořena v Javě SE s pomocí knihovny JOGL jako přístupovým bodem k OpenGL k vykreslování grafiky. Práce postupně rozebírá teoretický základ L-systémů, želví grafiku a vykreslování 3D objektů a poté popisuje vytvoření aplikace pomocí získaných znalostí.

Abstract

The aim of this bachelor thesis was to create an interactive system for generating 3D models. The generator is based on L-systems as a kind of formal grammars and turtle graphics for 3D modeling. The application was created in Java SE using JOGL library as access point for OpenGL and rendering. The thesis analyze the theoretical basis of L-systems, turtle graphics and rendering 3D objects and then describes creation of application using the acquired knowledge.

Klíčová slova

L-systémy, Lindenmayerovy systémy, formální gramatika, želví grafika, fraktály, modelování rostlin, 3D grafika, Java, JOGL

Keywords

L-systems, Lindenmayer systems, formal grammar, turtle graphics, fractals, plant modelling, 3D graphics, Java, JOGL

Citace

Jakub Kvita: Generátor 3D objektů s využitím L-systémů, bakalářská práce, Brno, FIT VUT v Brně, 2013

Generátor 3D objektů s využitím L-systémů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pánů Lukáše Vrábela a Pavla Tišnovského a s uvedením všech použitých zdrojů.

.....
Jakub Kvita
21. dubna 2013

Poděkování

Chtěl bych poděkovat panu Tišnovskému za počáteční impuls u výběru práce a panu Vrábelovi za velmi přínosné konzultace ve všech denních i nočních hodinách.

Obsah

Seznam obrázků

Kapitola 1

Úvod

V současnosti je počítačová grafika masivně využívána ve filmech, počítačových hrách a dalších oblastech a tvůrci požadují vysokou realističnost při nízkých nákladech. Tyto vlastnosti obsahují L-systémy a ukázaly se jako velmi vhodná cesta při modelování rostlinných struktur. Je možné s nimi vytvářet na jedné straně jednoduché modely květin a na druhé obrovské lesy, se stejným stupněm detailů v obou případech. Druhým faktorem je náročnost takových modelů, která je v ostatních případech velmi vysoká, ale to neplatí pro tyto systémy. Jeden příklad za všechny: model lesa do posledního listu zabírá ve zdrojové podobě v počítači asi stejně paměti, jako jeho fotografie.

Mým cílem bylo vytvořit interaktivní systém pro generování 3D modelů s použitím těchto L-systémů. Rozhodl jsem se, že součástí generátoru bude i prohlížeč výsledných modelů s jednoduchými funkcemi, protože s výsledným systémem se bude pracovat lépe a rychleji, když bude mít integrované všechny součásti a uživatel nebude potřebovat další aplikace.

V kapitole 2 popisuji základy L-systémů a formálních jazyků, jak L-systémy vznikly a jejich druhy. Jejich interpretaci pro 2D i 3D modelování vysvětluji v kapitole 3. V další kapitole (4) jsou vyloženy různé možnosti vykreslování 3D objektů. 5. kapitola kombinuje teoretické znalosti z předešlých kapitol a popisuji zde mou aplikaci pro generování objektů, včetně základního grafického rozhraní. Následující kapitola 6 obsahuje příklady modelů vygenerovaných ve vytvořené aplikaci s jejich zdrojovým tvarem, jak vstupoval do generátoru. V poslední kapitole (7) jsem zhodnotil výsledky a další vývoj systému.

Kapitola 2

L-systémy

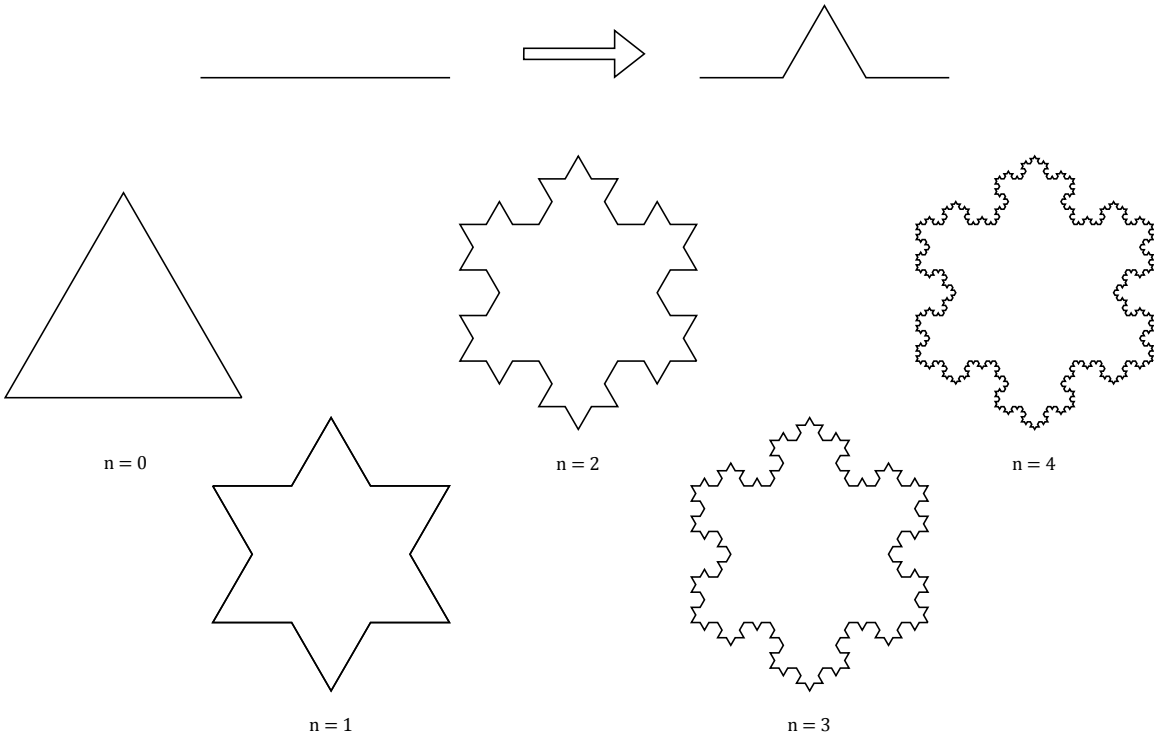
Teorii těchto systémů vytvořil v šedesátých letech maďarský biolog Aristid Lindenmayer a na jeho počest jsou pojmenovány jako Lindenmeyerovy systémy (ve zkratce L-systémy). Zpočátku sloužily pro popis vývoje buněčných struktur jednoduchých rostlin, ale ukázalo se, že jejich možnosti jsou větší a jdou použít i ke složitějším úlohám jako jeden z možných nástrojů, jak popisuje v [34]. L-systémy je možno použít i v jiných oblastech, než je grafika, např. pro generování fraktálové hudby (viz. [30]) nebo *DNA computing* a další (viz. [36]).

L-systém je druh formální gramatiky a základním konceptem je zde přepisování a přepisovací pravidla. Mohou se používat křivky, které se různě mění a vznikají fraktály. Jednou z nejznámějších ukázek je Kochova[1] křivka někdy také nazývaná Kochova vločka [35]. Ta se skládá ze dvou částí - úvodního obrazce - trojúhelníka a pravidla, které používá křivku podobnou stříšce. Oba jsou na obrázku 2.1. Pravidlo vyobrazené v horní části zní takto: Každou úsečku nahraď křivkou, úsečka i křivka mají společný počáteční a koncový bod. Toto pravidlo je možno použít kolikrát chceme - vznikají další generace/iterace modelu. Ve druhé polovině obrázku je vidět několik prvních iterací a limitně zde vzniká fraktál podobný vločce.

Na L-systémy je možno nahlížet jako paralelní protiklad k sekvenční hierarchii Chomského gramatik(viz. [37] a [31]). Jak je vidět u příkladu z Kochovou vločkou, pravidla se aplikují na všechny úsečky(symboly) paralelně, místo postupné aplikace na každou úsečku zvlášť - tzn. kromě derivací zde mluvíme také o iteracích a o jednotlivých stavech celého objektu (obrazce, řetězce, . . .). Stejně jako Chomského gramatik, L-systémů je větší množství, které se liší vlastnostmi a schopnostmi - konkrétněji jsou popsány v podkapitole 2.2. Obě dvě rodiny sdílí základní kameny formálních jazyků, jako je například zavedení pojmů *abecedy* a *řetězce* atp., které jsou potřeba pro jejich definice - ty jsou vysvětleny v podkapitole 2.1.

V paralelním přístupu se blíží L-systémy celulárním automatům - příkladem může být hra Life od Johna Conwaye [24]. Celulární automaty také pracují na biologickém principu, že vše se děje zároveň(paralelně) a můžeme pozorovat stavy, kterými prochází celý model. Největší podobnost lze vysledovat s jednorozměrnými automaty, které mají stejně jako okolí řetězců pouze dva stavy - před prvkem a za prvkem. Tato podobnost je čistě vizuální, základní principy celulárních automatů jsou jiné a používají se k naprosto odlišným případům, které jsou například popsány v [26].

[1] Helge von Koch (1870 - 1924) švédský matematik



Obrázek 2.1: Kochova vločka a její první 4 generace. Převzato z [33].

2.1 Úvod do formálních jazyků

V této podkapitole vysvětlím základní pojmy, na kterých budu stavět při definici L-systémů. Hlavními pojmy jsou *abeceda*, *řetězec* a *jazyk*, kromě nich i zavedení jejich množin, které se běžně používají u formálních jazyků. Řetězec je možné označovat jako *slovo* - budu tyto dva pojmy v dalším textu zaměňovat.

Pro pochopení této a dalších kapitol je třeba základních znalostí o množinách, relacích a funkcích. Potřebné informace lze nalézt například v úvodu [31]. Definice, které jsou zde uvedeny také pochází ze zmíněné knihy společně s jejich výkladem. Jako další zdroj definic jsem použil [37].

Definice 2.1.1 – abeceda a symboly
Abeceda je neprázdná, konečná množina prvků, které jsou nazývány *symboly*.

Se symboly můžeme vytvářet řetězce, které jsou, stejně jako slova, tvořeny posloupností symbolů určité abecedy. Také je potřeba definovat prázdný řetězec, ve kterém není ani jeden symbol. Tento řetězec se označuje znakem ε. Zde je uvedena rekurzivní definice řetězce:

Definice 2.1.2 – řetězec
Nechť Σ je abeceda.

1. ε je řetezec nad Σ.
2. Pokud x je řetězec nad Σ a $a \in \Sigma$, pak xa je řetězec nad Σ.

Délka řetězce je intuitivně řečeno počet symbolů v řetězci.

Definice 2.1.3 – délka řetězce
Nechť x je řetězec nad abecedou Σ. *Délka* řetězce x, $|x|$, je následující:

1. Pokud je $x = \varepsilon$, pak $|x| = 0$.
2. Pokud je $x = a_1 \dots a_n$, pro nějaké $n >= 1$, kde $a_i \epsilon \Sigma$ pro všechna $i = 1, \dots, n$, pak $|x| = n$.

Pro definici jazyka je potřeba nejdříve zavést Σ^* jako množinu všech řetězců nad abecedou Σ a dále

$$\Sigma^+ = \Sigma^* - \varepsilon,$$

neboli Σ^+ je množina všech neprázdných slov nad Σ. Jazyk je určitá množina slov nad abecedou Σ.

Definice 2.1.4 – jazyk
Nechť je abeceda Σ a $L \subseteq \Sigma^*$. Pak je L *jazyk* nad abecedou Σ.

Definice 2.1.5 – konečný a nekonečný jazyk
Nechť L je jazyk. L je *konečný* pokud obsahuje konečný počet slov, jinak je L *nekonečný*.

Jazyky jsou podle definice množiny, takže nad nimi lze provádět množinové operace jako průnik, sjednocení a rozdíl jako s jinými množinami a doplněk je definován takto

$$\overline{L} = \Sigma^* - L,$$

kromě nich jsou nad jazyky definovány i další operace, např. konkatenace a reverzace, které nebudu v této práci potřebovat. Jejich definice jsou uvedeny v [31].

2.2 Druhy L-systémů

Postupem času se vyvinulo několik skupin L-systémů, jedny z hlavních jsou zejména T0L, E0L a ET0L systémy. Jejich jména se řídí pravidly pojmenování, která byla zavedena a zajišťují okamžité porozumění všemi, kteří s nimi pracují. Jména jednotlivých druhů se skládají z písmen a číslic, která mají přesně definovaný význam a popisují vlastnosti L systémů. Například všechny typy L systémů obsahují v názvu písmeno L na znamení toho, že jsou druhem Lindenmayerových systémů. Klíčovou vlastností je bezkontextovost - pravidla nezávisí na okolí přepisovaného symbolu. Při výkladu týkajícím se rostlin je to možné popsat tak, že neexistuje interakce mezi jednotlivými buňkami nebo segmenty rostliny. Bezkontextové L systémy se označují číslicí *0*. Její význam bude jasný po popisu kontextových L-systémů, které se souhrnně označují písmenem I. Nejprve definuji základní 0L systém a derivaci, kterou ukážu na příkladu. Jednotlivé druhy a jejich vlastnosti jsou popsány v dalších částech této části kapitoly. Hlavní rozdělení je na 0L a IL systémy, ale jednotlivé vlastnosti mohou být součástí jedné i druhé skupiny.

Většinu typů jsem převzal z [36] s dodatky z [34]. První kniha se věnuje hlavně L-systémům z pohledu formálních jazyků, zatímco druhá nejvíce těm druhům, které usnadňují a vylepšují modelování rostlin - např. parametrické L-systémy.

Definice 2.2.1 – 0L systém
0L systém je uspořádaná trojice $G = (V, \omega, P)$, kde V je abeceda systému, $\omega \in V^+$ je neprázdné slovo nazývané *axiom* a $P \subset V \times V^*$ je konečná množina pravidel. Pravidlo $(a, \chi) \in P$ se zapisuje ve tvaru $a \to \chi$. Předpokládá se, že pro každé $a \in V$ existuje alespoň jedno pravidlo ve tvaru $a \to \chi$. Pokud neexistuje, zavádí se pravidlo identity ve tvaru $a \to a$.

Definice 2.2.2 – derivace
Nechť $\mu = a_1 \dots a_m$ je řetězec nad V. Řetězec $\nu = \chi_1 \dots \chi_m \in V^*$ je přímo derivován (generován) z μ, psáno $\mu \Rightarrow \nu$, právě tehdy, když existuje pravidlo $a_i \to \chi_i$ pro každé $i = 1, \dots, m$. ν se nazývá *produkt* derivace. Řetězec ν je generován derivací o *délce* n tak, že $\mu = \mu_0$ a $\nu = \mu_n$, pokud existuje n na sebe navazujících derivací ve tvaru $\mu_0 \Rightarrow \mu_1 \Rightarrow \dots \Rightarrow \mu_n$.

Klíčový rozdíl u derivací oproti gramatikám Chomského hierarchie je, že zde považujeme za derivaci aplikaci právě jednoho pravidla na každý symbol řetězce, klidně to může být pravidlo identity, a u gramatik jde o aplikaci právě jednoho pravidla na právě jeden symbol.

Příklad 2.2.1
Zde je ukázka jednoduchého 0L systému. První čtyři derivace jsou na obrázku 2.2. Kromě jiného, jsou zde zajímavá automaticky vytvořená pravidla p_{5i} a p_{6i}, která jsou identitou a jediným pravidlem pro symboly e a f.

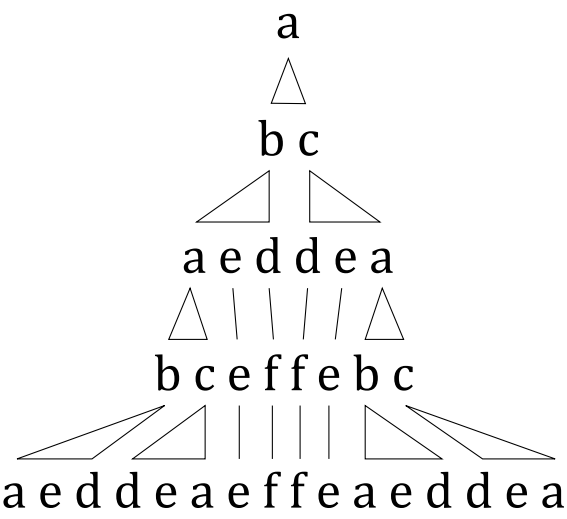
$$
\begin{array}{lcccl}
\omega & : & a & & \\
p_1 & : & a & \to & bc \\
p_2 & : & b & \to & aed \\
p_3 & : & c & \to & dea \\
p_4 & : & d & \to & f \\
p_{5i} & : & e & \to & e \\
p_{6i} & : & f & \to & f
\end{array}
$$

2.2.1 Druhy 0L systémů

Jednou z důležitých vlastností je determinismus systému. Stav systému v budoucnosti se dá odvodit z jeho současného stavu a pevně daných pravidel, protože pro každý stav existuje právě jeden následující stav. Tato vlastnost u L-systému se označuje písmenem U a má následující definici.

Definice 2.2.3 – deterministický L-systém
Systém $G = (V, \omega, P)$ je deterministický, jestli pro každé $a \in V$ existuje nejvýše jedno pravidlo ve tvaru $(a \to \chi) \in P$.

V základní definici 0L systémů je možné mít více pravidel se stejnou levou stranou. Ve *stochastických* L-systémech, lze těmto pravidlům přiřadit pravděpodobnosti, jak často se



Obrázek 2.2: První čtyři derivace ukázkového DOL-systému.

použijí. Tento typ se hodí například pro simulace, protože lze zajistit určitou variabilitu jednotlivých modelů a zároveň společné vlastnosti, ale pro jiné činnosti, např. popis programovacích/přirozených jazyků, je nevhodný nebo nesmyslný. Lze pak generovat modely, které vypadají jako různí jedinci jednoho druhu rostliny. Základní 0L systémy si můžeme představit jako stochastické systémy, ve kterých mají pravidla shodnou pravděpodobnost.

Definice 2.2.4 – stochastický L-systém
Stochastický L-systém je uspořádaná čtveřice $G_\pi = (V, \omega, P, \pi)$. Abeceda V, axiom ω a množina pravidel P jsou definovány stejně jako u 0L-systému. Funkce rozdělení pravděpodobnosti $\pi : P \to (0, 1]$ mapuje jednotlivá pravidla na jejich pravděpodobnosti jak často se použijí. Součet pravděpodobností všech pravidel se stejnou levou stranou musí být 1.

Příklad 2.2.2
Zde je jednoduchá ukázka stochastického 0L systému převzatá z [34].

$$\begin{array}{lcll} \omega & : & F & \\ p_1 & : & F & \overset{.33}{\to} \quad F[+F]F[-F]F \\ p_2 & : & F & \overset{.33}{\to} \quad F[+F]F \\ p_3 & : & F & \overset{.34}{\to} \quad F[-F]F \end{array}$$

Další typ je označovaný písmenem P z anglického *propagating*. Jsou to systémy jejichž pravidla nemají na pravé straně ε. Zjednodušeně řečeno nedochází zde k vymazání a zániku symbolů. V [36] toto označují jako *cell death* a v této knize je také o P0L systémech popsáno více. Tento typ má použití zvláště při vytváření růstových funkcí, kdy nás nezajímá, jaké symboly jsou v řetězci obsaženy, ale pouze jeho délka, kterou je možno modelovat téměř jakoukoli číselnou posloupnost.

Při generování jazyků zatím uvedenými systémy je výrazným faktorem to, že musíme přijmout všechny řetězce vygenerované systémem. Pokud je potřeba určitá slova vyloučit v

D0L systémech to nejde - nemají žádný filtrovací mechanismus. Základní filtrací v Chomského gramatikách je použití neterminálních symbolů. Pokud se nějaký objeví ve slově znamená to, že slovo není možno přijmout a nepatří do jazyka. Stejný princip je možno zavést do L-systémů, rozdělit abecedu na dvě disjunktní množiny terminálů a neterminálů. Pravidla a slova mohou obsahovat symboly z obou množin, ale přijímaný jazyk se skládá pouze ze slov tvořených symboly z množiny terminálů. Tyto systémy se označují písmenem E (*extended*). Musím upozornit na to, že původně bylo cílem L-systémů popisovat vývoj organismů a každý derivovaný řetězec popisoval stav organismu. Je proto nepřirozené, najednou vylučovat určitá slova, s tím, že nepopisují růst organismu. Postupem času se zkoumání těchto typů ukázalo jako velmi plodné a E0L systémy tvoří důležitou skupinu.

Velmi významným prvkem v paralelních systémech je použití tabulek. Druh systémů s tabulkami se označuje pomocí písmene T. Tabulka je množina pravidel, která se použijí při jedné derivaci. Systém obecně obsahuje libovolné množství těchto tabulek, která se v derivacích různě střídají. Tímto je možné modelovat změny při vývoji organismu, například různé vývojové stupně nebo změny okolního prostředí (střídání dne a noci, ročních období, znečištění, ...).

Definice 2.2.5 – T0L systém
T0L systém je uspořádaná trojice $G = (\Sigma, S, \omega)$, kde S je konečná množina množin pravidel taková, že pro každé $\sigma \in S$, trojice (Σ, σ, ω) je 0L systém. Při derivaci je možno použít libovolnou množinu z prvků množiny S, ale ta musí být pro všechny symboly stejná.

Jednotlivé druhy systémů mohou obsahovat více vlastností zároveň, jako jsou např. ET0L systémy, které kombinují neterminály a tabulky, jsou největší podrobně zkoumanou skupinou bezkontextových L-systémů, nebo DE0L systémy, které, jak už název napovídá, jsou bezkontextové, deterministické a používají neterminály. Typy, které jsem uvedl nejsou všechny, existuje mnoho dalších. Několik dalších z nich je uvedeno v podkapitole 2.2.4.

2.2.2 IL systémy

IL systémy jsou kontextové L-systémy neboli systémy s *interakcemi*. Podle tohoto slova jsou obecně pojmenovány písmenem *I*. Při derivaci zde použití pravidla závisí nejen na vstupním symbolu, ale také na jeho kontextu - jaké symboly má kolem sebe. Tento princip je velmi užitečný například při simulaci proudění informací nebo toku živin a vody v rostlinách.

Tyto systémy se označují jako (m, n)L systémy, kde $m, n \geq 0$. V [36] pravidla v těchto systémech vypadají takto

$$(\alpha, a, \beta) \rightarrow \omega, \ |\alpha| = m, \ |\beta| = n$$

a podle [34] následovně

$$\alpha \ < \ a \ > \ \beta \ \rightarrow \ \omega$$

α a β zde jsou *levý* a *pravý kontext* a pravidlo znamená, že symbol a je možné přepsat na řetězec ω, pouze pokud je mezi řetězci α a β. Aby bylo možno dostat dostatečný počet symbolů pro oba kontexty, celý přepisovaný řetězec je umístěn mezi m a n *symbolů prostředí* #.

Označení IL systém je společné jméno pro všechny (m, n)L systémy. $(1, 0)$L a $(0, 1)$L systémy jsou pojmenovány jako *1L systémy*, protože je v nich interakce pouze jednostranná. Tyto systémy můžeme zapisovat následovně:

$$\begin{array}{lllll} \alpha & < & a & \rightarrow & \omega \\ & & a & > \beta & \rightarrow \omega \end{array}$$

Po tomto výkladu konečně začne dávat smysl označení bezkontextových systémů *0L* - není v nich interakce ani s jednou stranou. V [34] jsou systémy používající oba kontexty nazvány *2L* systémy.

Použití písmen D, E, P, ... je úplně stejné jako předtím, ovšem s drobnými změnami. *Determinismus* například znamená, že nemůže být více pravidel se stejnou levou stranou a zároveň se stejným *(m,n)* okolím. Pro kontextové systémy s pravděpodobností platí stejná myšlenka a pravděpodobnost se určuje zvlášť pro každou skupinu pravidel, která mají stejnou nejen levou stranu, ale i kontext.

Při interpretaci kontextových systémů želví grafikou je třeba s kontextem pracovat jiným způsobem, aby systém odrážel reálné chování rostlin. Hlavní myšlenkou je přeskakování symbolů, které sice mají podobné místo v řetězci, ale v modelu spolu vůbec nemusí souviset. Tento princip jsem popsal v části 3.4.

Příklad 2.2.3
Jednoduchý příklad kontextového systému, který může simulovat například průchod signálu řetězcem. Tento systém jsem převzal z [34].

$$\begin{array}{lllllll} \omega & : & baaaaaaaaa & & & & \\ p_1 & : & b & < & a & \rightarrow & b \\ p_2 & : & & & b & \rightarrow & a \end{array}$$

Několik prvních řetězců vygenerovaných tímto systémem nasleduje:

$$\begin{array}{c} baaaaaaaaa \\ abaaaaaaaa \\ aabaaaaaaa \\ aaabaaaaaa \\ aaaabaaaaa \\ \vdots \end{array}$$

Podobně lze modelovat zaplavování systému určitým symbolem, v našem případě b. Systém by měl pouze jedno pravidlo a místo p_2 by se použila identita.

2.2.3 Parametrické L-systémy

Zatím uvedené L-systémy mohou sloužit k rozmanité škále činností, ale jedno mají společné - pracují s diskrétní představou světa. Pokud bychom chtěli reprezentovat veličiny, které jsou v základu spojité, s přijatelnou odchylkou, velikost systémů by velmi narostla, až do stovek symbolů a pravidel, a tím bychom přišli o základní princip L-systémů, kterým je jednoduchost. Podle [34], Lindenmayer přišel s řešením, že symboly systému budou obsahovat číselné parametry, které budou reprezentovat spojitý vývoj rostlin. Prusinkiewicz a Hanan tuto myšlenku dále rozpracovali a Hanan ji předkládá ve své disertační práci, viz. [25]. Některé části této podkapitoly jsou převzaty z [34].

Symbol A z abecedy systému V v parametrickém řetězci se společně s parametry z množiny reálných čísel $\mathbb{R}$ nazývají *modul* a zapisují se $A(a_1, a_2, \ldots, a_n)$. Každý modul patří do množiny $M = V \times \mathbb{R}^*$, kde $\mathbb{R}^*$ je množina všech konečných posloupností parametrů.V řetězcích modulů se tedy objevují *skutečné* parametry, což jsou reálná čísla. Zároveň korespondují s *formálními* parametry, které se objevují v pravidlech. Ty jsou reprezentovány proměnnými. Dále pokud je Σ množina formálních parametrů, pak $C(\Sigma)$ je množina všech *logických výrazů* s parametry z Σ a $E(\Sigma)$ je množina všech *aritmetických výrazů* z Σ. Oba typy výrazů se skládají z formálních parametrů a číselných konstant, aritmetických operátorů, operátorů porovnání a dalších prvků podle standardních principů v matematice. Logické výrazy se vyhodnocují jako 1 pro pravdu a 0 pro nepravdu. Ukázky všech těchto množin jsou na příkladu 2.2.4, společně s pravidly a derivací jak jsou dále definovány.

Definice 2.2.6 – parametrický 0L systém
Parametrický 0L systém je uspořádaná čtveřice $G = (V, \Sigma, \omega, P)$, kde V je abeceda systému, Σ je množina formálních parametrů, $\omega \in (V \times \mathbb{R}^*)^+$ je neprázdný parametrický řetězec nazvaný axiom a $P \subset (V \times \Sigma^*) \times C(\Sigma) \times (V \times E(\Sigma))^*$ je konečná množina pravidel.

V pravidlech se používají symboly : a $\rightarrow$ pro oddělení vstupu pravidla, podmínky a výstupu. Například pravidlo přepisující modul $A(t)$ na $B(t+1)CD(t^{0.5}, t-2)$ za podmínky $t > 5$ se zapíše takto

$$A(t) \quad : \quad t > 5 \quad \rightarrow \quad B(t+1)CD(t^{0.5}, t-2).$$

Pravidlo je aplikováno na symbol řetězce (modul), jestliže platí následující podmínky:

- symbol v řetězci a na levé straně pravidla se shodují,
- oba moduly mají shodný počet parametrů a
- podmínka uvedená v pravidle je vyhodnocena jako pravdivá.

Příklad 2.2.4
Jednoduchý parametrický 0L systém převzatý z [34], společně s několika prvními derivacemi. Nad symbolem derivace jsou pravidla, která se aplikovala na jednotlivé moduly ve stejném pořadí jak jsou napsána, pro lepší pochopení principu.

$$\begin{array}{lcllcl}
\omega & : & B(2)A(4,4) & & & \\
p_1 & : & A(x,y) & : \; y \leq 3 & \rightarrow & A(x*2, x+y) \\
p_2 & : & A(x,y) & : \; y > 3 & \rightarrow & B(x)A(x/y, 0) \\
p_3 & : & B(x) & : \; x < 1 & \rightarrow & C \\
p_4 & : & B(x) & : \; x \geq 1 & \rightarrow & B(x-1)
\end{array}$$

$$B(2)A(4,4) \overset{[p_4 p_2]}{\Rightarrow} B(1)B(4)A(1,0) \overset{[p_4 p_4 p_1]}{\Rightarrow} B(0)B(3)A(2,1) \overset{[p_3 p_4 p_1]}{\Rightarrow} CB(2)A(4,3)$$

Parametrické systémy také lze zkombinovat s kontextovými, pokud potřebujeme interakci mezi jednotlivými symboly. Výsledné systémy mají stejný formát jako ostatní, pouze jsou pravidla komplexnější.

Příklad 2.2.5
Ukázkové pravidlo parametrického IL systému.

$$A(x) \quad < \quad B(y) \quad > \quad C(z) \quad : \quad x + y + z > 10 \quad \rightarrow \quad E(x^2 + y)F((y - 5) * z)$$

2.2.4 Ostatní

Existuje velké množství druhů L-systémů, které jsem zde neuvedl a jejich popis by zabral velké množství místa. Na více místech jsem se už odkazoval na [36] kde je jich uvedeno více společně s dalšími odkazy. Například jsou zde popsány systémy, které místo jednoho počátečního axiomu obsahují konečnou množinu řetězců - axiomů. Takové systémy se označují písmenem F z sousloví *finite set of axioms*.

Druhým příkladem rozvoje L-systémů jsou systémy s fragmentací. Ta umožňuje pomocí řezů blokovat komunikaci a přenos informací mezi segmenty řetězce nebo modelovat rozmnožení organismu. Používá se pro tyto účely speciálního symbolu např. q, který rozděluje řetězec, a derivace může pokračovat z každé části řetězce. Pro tento druh se používá označení písmenem J - *jakautua*. Toto slovo pochází z finštiny, protože při představení tohoto typu, všechna písmena, kterými začínají vhodná slova v angličtině byla zabrána.

Posledním příkladem jsou *UL systémy*, které používají abecedu obsahující pouze jeden symbol. Jméno mají podle anglického slova *unary*.

Kapitola 3

Želví grafika

Krátce po ustanovení Chomského gramatik začaly vznikat různé návrhy pro práci s grafikou. Prvotní výzkum se věnoval rozeznávání obrazu - symboly zde reprezentovaly grafická primitiva, například přímky, zatáčky doleva a doprava, větve atp. Blíže je tato historie popsána v [32]. Poté se výzkum přesunul do oblasti generování obrazu a modelů a vznik koncept *chain coding* [20], popisující pohyb v prostoru pomocí mřížky. Prusinkiewicz po zveřejnění konceptu L-systémů zkoumal možnosti jejich propojení s želví grafikou [32], která by umožnila generovat obrázky, které nejsou omezovány mřížkou (uvažovalo se primárně o dvourozměrném prostoru).

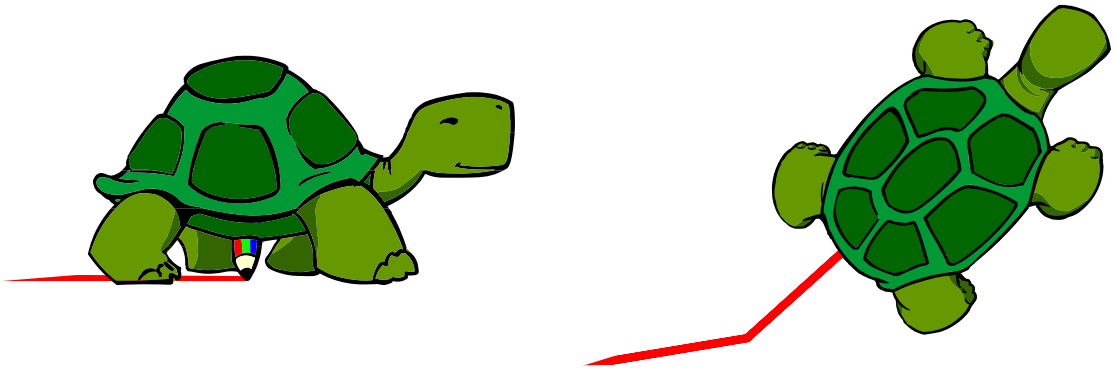
V želví grafice se symboly vygenerovaného řetězce chápou jako příkazy řídící želvu, která nemá ponětí o výsledném obraze. Laická představa želvy je na obrázku 3.1. Formální definice, která je převzata z [34], je následující:

Definice 3.0.7 – želví grafika
Želva ve 2D prostoru je uspořádaná trojice (x, y, α), kde x a y jsou souřadnice pozice želvy v prostoru a α je úhel natočení - směr kam želva kráčí. Základní směr $\alpha = 0°$ je osa X, tzn. ve 2D prostoru obrázku směr doprava, a kladný úhel je proti směru hodinových ručiček. Pro pohyb želvy slouží velikost kroku želvy d a velikost změny úhlu δ.

symbol	příkaz
F	Udělej krok o velikosti d ve směru α. Stav želvy se změní na (x', y', α), kde $x' = x + d\cos\alpha$ a $y' = y + d\sin\alpha$. Nakresli čáru mezi (x, y) a (x', y').
f	Udělej krok o velikosti d ve směru α bez kreslení čáry.
$+$	Otoč se o úhel δ doleva. Další stav želvy je $(x, y, \alpha + \delta)$.
-	Otoč se o úhel δ doprava. Další stav želvy je $(x, y, \alpha - \delta)$.
ostatní	Nedělej nic.

Tabulka 3.1: Základní příkazy želví grafiky z [34].

V tabulce 3.1 jsou uvedeny příkazy, které teď dokáže želva zpracovat. Tyto příkazy se v naprosté většině publikací shodují a dají se považovat za standardní. Kromě nich se používají ještě další, které budu definovat v dalších podkapitolách. V poslední řadě je možno zavést příkazy například pro změnu barvy, otočení želvy dozadu a další, které se liší případ



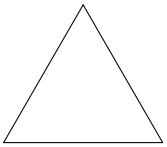
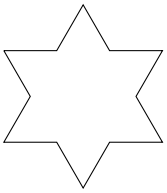
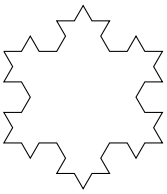
Obrázek 3.1: Reprezentace želvy. Převzato a dodatečně upraveno z [40] a [41].

od případu a je nutné definovat je až u konkrétní implementace.

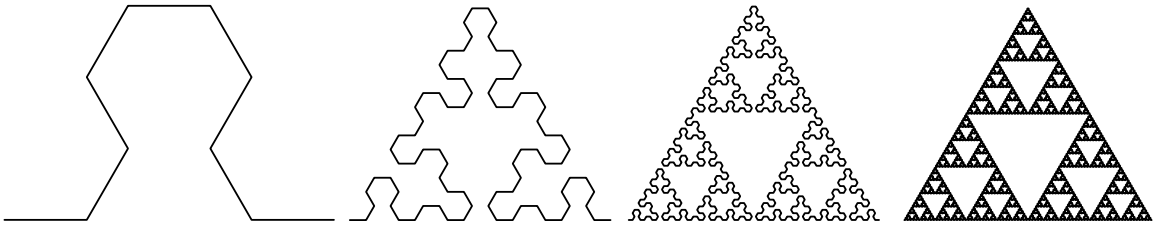
Příklad 3.0.6
Se současnými informacemi mohu definovat Kochovu vločku z obrázku 2.1 znovu pomocí OL-systémů se symboly a želví grafiky. OL-systém je uveden na následujících rovnicích. Axiom vykreslí počáteční trojúhelník. Při několika prvních derivacích je vidět, že vločka se velice rychle zvětšuje - to je způsobeno konstantním parametrem d. Konstantních rozměrů lze dosáhnout zmenšováním d po každé derivaci na 1/3. Ukázka je uvedena v tabulce 3.2.

$$\begin{aligned} \omega &: F++F++F \\ p &: F \rightarrow F-F++F-F \end{aligned}$$

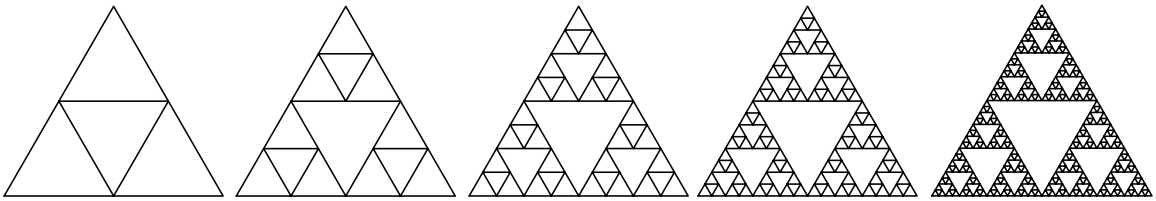
$$\begin{aligned} d &= 1 \\ \delta &= 60^\circ \end{aligned}$$

axiom $d = 1$	$F++F++F$	
1.derivace $d = 1/3$	$F-F++F-F++F-F++F-F++F-F++F-F++F-F$	
2.derivace $d = 1/6$	$F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F++F-F++F-F-F-F++F-F$	

Tabulka 3.2: Kochova vločka s želví grafikou



Obrázek 3.2: Model Sierpinského trojúhelníku z [34]. L-systém je uveden v příkladu 3.0.7.



Obrázek 3.3: Další model Sierpinského trojúhelníku. Viz. příklad 3.0.8.

Dalším příkladem zde je Sierpińského[1] trojúhelník ve dvou variantách. Oba modely zavádí indexování symbolu F. Všechny F bez ohledu na index znamenají posun želvy dopředu a nakreslení čáry. Indexy rozlišují jednotlivé symboly pro pravidla, která jsou pro každý symbol jiná. Pro dosažení stejného efektu by šlo použít několik různých písmen s významem písmene F pro rozlišení pravidel.

Příklad 3.0.7
První varianta je převzata z [34] a má následující L-systém. Několik prvních derivací je na obrázku 3.2.

$$\begin{aligned} \omega &: F_l \\ p_1 &: F_r \rightarrow F_r - F_l - F_r + F_l + F_r + F_l + F_r - F_l - F_r \\ p_2 &: F_l \rightarrow F_l + F_r + F_l - F_r - F_l - F_r - F_l + F_r + F_l \end{aligned}$$

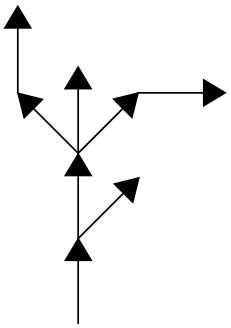
$$\delta = 60^\circ$$

Příklad 3.0.8
Druhá varianta Sierpinského trojúhelníku, tentokrát s přesnými trojúhelníky. První derivace jsou na obrázku 3.3.

$$\begin{aligned} \omega &: F_c - F - F \\ p_1 &: F_c \rightarrow F_c - F + F_c + F - F_c \\ p_2 &: F \rightarrow FF \end{aligned}$$

$$\delta = 120^\circ$$

[1] Wacław Sierpiński (1882 - 1969) polský matematik



F [-F] F [+F -F] [-F -F] F

Obrázek 3.4: Řetězec se závorkami a jeho interpretace ve formě stromu.

Existuje mnoho variant L-systémů, ale naprostá většina z nich nemá na následnou interpretaci řetězce pomocí želví grafiky žádný vliv. Jednou z výjimek jsou parametrické systémy. U nich není potřeba zavádět standardní délku kroku a otočení, protože tyto hodnoty si každý symbol nese sám, ve svých parametrech. V tabulce 3.3 je uvedeno několik symbolů s parametry a jak se interpretují. V této práci jsem převzal interpretaci z [34].

symbol	příkaz
$F(a)$	Udělej krok o velikosti a ve směru α, podobně jako u F bez parametru.
$f(a)$	Udělej krok o velikosti a ve směru α bez kreslení čáry, tak jako u f bez parametru.
$+(a)$	Otoč se o úhel δ. Pokud je úhel kladný želva zatáčí doleva, při záporném úhlu doprava.

Tabulka 3.3: Příkazy želví grafiky parametrického L-systému.

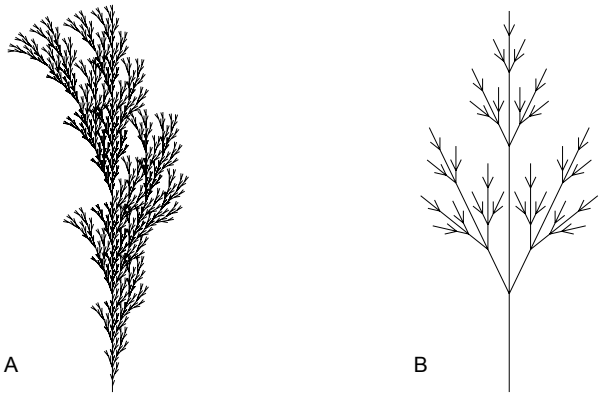
Zápis u parametrických systémů není pro pochopení sémantiky moc vhodný, a proto je potřeba specifikovat, který parametr u symbolu slouží po pohyb a jak ostatní vykládat, na vhodném místě u systému.

U symbolu $+$ je důležité upozornit, že ho budu používat nejen jako symbol abecedy systému, ale také jako aritmetický operátor a jeho význam bude záležet na kontextu. Toto bude platit i u některých dále představených symbolů, zejména u interpretace ve 3D.

3.1 Větvení - Závorkové systémy

Do této chvíle želva umí vytvářet velké množství objektů, různě tvarovaných a s některými částmi neviditelnými, ale pořád jsou všechny modely tvořeny jednou čarou. Mezi rostlinami je ale nejrozšířenějším tvarem strom s větvemi, a proto se studovaly možnosti, jak vykreslovat stromové struktury. Reprezentace stromů má ve výpočetní technice více podob a jednou z nich, která se ujala pro L-systémy jsou závorkové struktury. Ty jsou rozšíření želví grafiky a její interpretace řetězců L-systémů. Lindemayer v [34] zavádí dva nové symboly pro vytvoření větve, které jsou uvedeny v tabulce 3.4.

Jak je vidět z popisu symbolů, zavádí se objekt zásobníku, který je naprosto standardní LIFO fronta. Derivace závorkových systémů probíhají stejně jako u OL-systémů. V drtivé většině případů neexistují pravidla se závorkami na levé straně, takže se používá identické pravidlo. Jak vypadá řetěze se závorkami a vytvořený strom je na obrázku 3.4.



Obrázek 3.5: Modely interpretovaných řetězců se závorkami z příkladů 3.1.1 a 3.1.2.

symbol	příkaz
[	Ulož současný stav želvy na zásobník. K uloženým informacím patří pozice a orientace želvy, a kromě toho další vlastnosti např. barva, kterou se kreslí, a šířka čáry.
]	Vyjmi ze zásobníku stav želvy a ten se stane aktuálním stavem. Nekresli žádnou čáru, i když se mění pozice želvy.

Tabulka 3.4: Želví příkazy pro vytváření větví.

Následuje několik příkladů převzatých z [34]. Na obrázcích, na které příklady odkazují je proměnná n, která určuje počet derivací, které proběhly.

Příklad 3.1.1
Stromová struktura vygenerovaná následujícím systémem je na obrázku 3.5 v části A. Vyobrazena je pátá derivace axiomu.

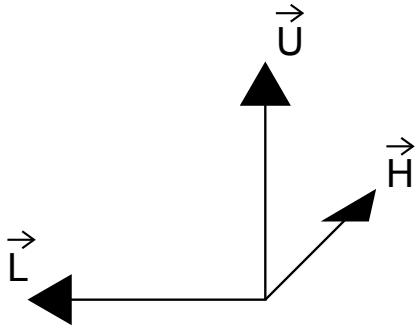
$$\begin{aligned} \omega &: F \\ p_1 &: F \rightarrow F[+F]F[-F][F] \end{aligned}$$

$$\delta = 20^\circ$$

Příklad 3.1.2
Další ukázka systému dvourozměrného stromu, tentokrát s dvěma pravidly. Výsledný model je na obrázku 3.5 v části B a v páté derivaci.

$$\begin{aligned} \omega &: X \\ p_1 &: X \rightarrow F[+X][-X]FX \\ p_1 &: F \rightarrow FF \end{aligned}$$

$$\delta = 25,7^\circ$$



Obrázek 3.6: Orientace želvy ve třech dimenzích.

3.2 Želví grafika ve 3D

Reálné modely bývají ve trojrozměrném prostoru a 2D želví grafika je pro ně nepoužitelná, lze ji ovšem rozšířit do 3D. Jak je uvedeno v [34] základním konceptem je reprezentace aktuálního směru želvy, ne jedním, ale třemi vektory $\vec{H}, \vec{L}, \vec{U}$, které postupně znamenají směr želvy vpřed, písmeno vektoru je z anglického *heading*, směr doleva (písmeno ze slova *left*) a nahoru (*up*). Tyto vektory jsou jednotkové, vzájemně kolmé a platí u nich rovnice vektorového součinu $\vec{H} \times \vec{L} = \vec{U}$. Ukázka těchto vektorů je na obrázku 3.6.

Rotace želvy tedy mohou probíhat kolem každého vektoru a matematicky jsou vyjádřeny rovnicí

$$\begin{bmatrix} \vec{H}' & \vec{L}' & \vec{U}' \end{bmatrix} = \begin{bmatrix} \vec{H} & \vec{L} & \vec{U} \end{bmatrix} R,$$

kde R je 3×3 rotační matice. Pro použití této matice je potřeba znát úhel otočení, který může být definován v zadání nebo jako parametr symbolu, a kolem kterého vektoru se želva otáčí. Konkrétně pro úhel α se postupně pro vektory $\vec{H}, \vec{L}$ a $\vec{U}$ použijí tyto matice:

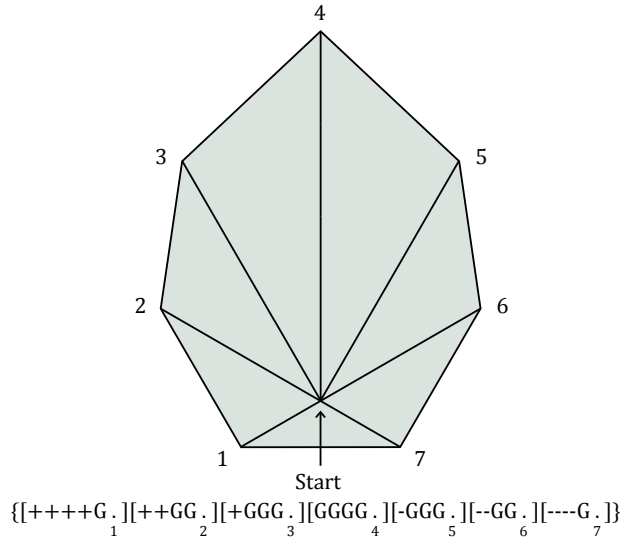
$$R_U(\alpha) = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_L(\alpha) = \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha \\ 0 & 1 & 0 \\ \sin\alpha & 0 & \cos\alpha \end{bmatrix}$$

$$R_H(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}$$

Pro použití těchto rotací se v [34] zavádí interpretace skupiny symbolů uvedených v tabulce 3.5. Jejich prostorové uspořádání je na obrázku 3.6.

Je vidět, že vždy dva symboly jsou kolem stejné osy, pouze na opačnou stranu. Dva symboly místo záporných čísel se používají kvůli danému kroku, který nelze měnit. Poslední



Obrázek 3.7: Vykreslení mnohoúhelníku s interpretací nových symbolů.

lichý symbol svislé čáry se nepoužívá moc často, protože na principu vracení se na pozici fungují hranaté závorky, viz. 3.4.

symbol	příkaz
$+$	Otoč se doleva o úhel δ, s použitím matice $R_U(\delta)$.
-	Otoč se doprava o úhel δ, s použitím matice $R_U(-\delta)$.
&	Skloň se o úhel δ, s použitím matice $R_L(\delta)$.
$\wedge$	Zvedni hlavu o úhel δ, s použitím matice $R_L(-\delta)$.
$\backslash$	Převal se doleva o úhel δ, s použitím matice $R_H(\delta)$.
/	Převal se doprava o úhel δ, s použitím matice $R_H(-\delta)$.
$\vert$	Otoč se čelem vzad s použitím matice $R_U(180°)$.

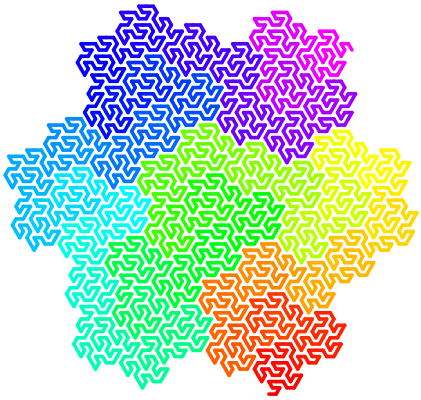
Tabulka 3.5: Příkazy rotace želvy ve 3D.

3.3 Vykreslování ploch, barvy a další příkazy

Jedním z posledních prvků modelu rostliny je tvorba listů a okvětí. Větve a ostatní prvky je možno tvořit křivkami o různé tloušťce, ale na listy je potřeba vytvořit plochu. Existuje několik možností jak specifikovat povrch nebo mnohoúhelník.

První možností je předdefinovat objekt listu a při interpretaci řetězce u příslušného symbolu, v [34] za tento symbol považují $\sim$, vložit tento objekt do modelu. Pro určení pozice modelu jsou potřeba dva vektory, které jsou převzaty z orientace želvy, a pozice želvy, která určuje počáteční bod. Tento princip umožňuje vytvářet velmi krásné modely, ale porušuje elegantnost L-systémů, protože je potřeba připravovat jednotlivé objekty, které se budou vkládat. Předdefinované objekty se také nemění při derivacích a není tak možné modelovat růst rostliny. Tento princip nebudu používat, protože existují vhodnější metody.

Další možností je specifikovat vrcholy mnohoúhelníku a poté vykreslit celou jeho plochu. Tento přístup je složitější, ale odstraňuje nedostatky předchozího, za cenu méně komplex-



Obrázek 3.8: Prostor vyplňující křivka s barevným zvýrazněním cesty.

ních tvarů. V [34] autoři zavádějí několik symbolů, které uvedu i zde. Symbol znamená počátek kreslení nového polygonu. Pokud se už nějaký mnohoúhelník vytvářel, uloží se na jejich zásobník (nemá nic společného se zásobníkem stavů želvy). Další symbol (.) si zapamatuje polohu želvy jako jeden z vrcholů současného polygonu. Závěrečným symbolem je , který ukončí tvorbu mnohoúhelníku, vytvoří ho ze všech dosud přijatých vrcholů, a pokud je nějaký polygon na zásobníku, tak ho vyjme. Doplňkovým symbolem je zde G. Ten umožňuje kreslit dovnitř polygonu, jeho interpretace je stejná jako F vně tvorby mnohoúhelníku. Příkazy z této části kapitoly jsou přehledně uvedeny v tabulce 3.6. Ukázkový polygon i s interpretovaným řetězcem je na obrázku 3.7.

symbol	příkaz
{	Začni vytvářet polygon.
.	Ulož polohu želvy jako vrchol polygonu.
G	Posuň želvu a vykresli čáru dovnitř polygonu.
}	Ukonči vytváření polygonu a vykresli ho.
~	Vlož do modelu předdefinovaný tvar.
!	Zmenši průměr dalších segmentů.
%	Odřízni zbytek aktuální větve.
’	Inkrementuj index barvy o jednu položku.
,	Dekrementuj index barvy o jednu položku.

Tabulka 3.6: Příkazy pro tvorbu polygonů, barev a další méně používané.

Mezi speciální symboly patří *!*, který zmenšuje průměr jednotlivých segmentů rostliny o koeficient $w_r = 0,707$. To umožňuje vytvářet realistické stromy, protože s touto konstantou platí postulát Leonarda da Vinciho o tloušťce větví a kmene. Více informací je v [34, strana 57]. Symbol *%* je možno používat při simulaci odpadávání květů a listů. V systému se odřízne zbytek větve vytvářející okvětní lístek v momentu rozvoje plodu.

Posledním prvkem je zde zavedení barev do modelu. Aktuální barva, kterou želva kreslí, musí být zahrnuta do stavu želvy a její reprezentace musí být i v samotném řetězci, který se interpretuje. Zde je ovšem problém reprezentace spojité palety barev v diskrétním světě symbolů. Při použití parametrických systémů je řešení velmi jednoduché - zavedením symbolu $C(r, g, b)$ s třemi parametry, který nastaví současné barvě kanály (r, g, b). V ostatních

L-systémech je to problém komplikovanější a existuje zde několik přístupů.

Jedním z přístupů je mít předdefinovanou paletu barev seřazenou v seznamu nebo nějakém poli. Do tohoto pole existuje index, který určuje čím se bude vykreslovat aktuální prvek. Pro manipulaci s indexem se zde zavádí nový symbol ', který znamená posun indexu o jednu položku doprava[2]. Tak je to zavedeno v [34]. Já sám jsem pro větší variabilitu zavedl i opačný symbol (,), který posouvá index doleva. Tento způsob práce s barvami není moc vhodný při tvorbě rostlin, ale hodí se při práci s plochu vyplňujícími křivkami (viz. [38]). Seznam barev je zaveden jako cyklický a postupně se mezi segmenty modelu mění barva a lze díky tomu vidět kudy křivka prochází. Viz. obrázek 3.8.

V [28] je tento princip rozšířen o parametr symbolů ' a (,), kterým je následujících několik symbolů. Za parametr se zde pokládají číslice, které určují o kolik položek se má přesunout index doleva nebo doprava. Také zde zavádí symbol c. Ten funguje pouze s parametrem a přesune index na hodnotu rovnou parametru. Toto rozšíření zjednodušuje určité aspekty práce s L-systémy, ale porušuje nezávislost základního stavebního bloku - jednoho symbolu, a proto ho nebudu používat.

3.4 Zpracování kontextu při modelování rostlin

Při zkoumání interpretace kontextových L-systémů želví grafikou se přišlo na několik problémů v rozdílech mezi reprezentací řetězcem a modely (většinou rostlin). Jedním je zařazování do kontextu takových symbolů, které mají význam jen při vykreslování modelu, ale v samotném významu kontextu hrají velmi malou roli. Druhým problémem jsou závorkové struktury a větvení - v takových modelech mohou mít segmenty více kontextů, které nejsou souvisle za sebou v řetězci. Tyto situace jsou popsány v [34, strany 31 a 32].

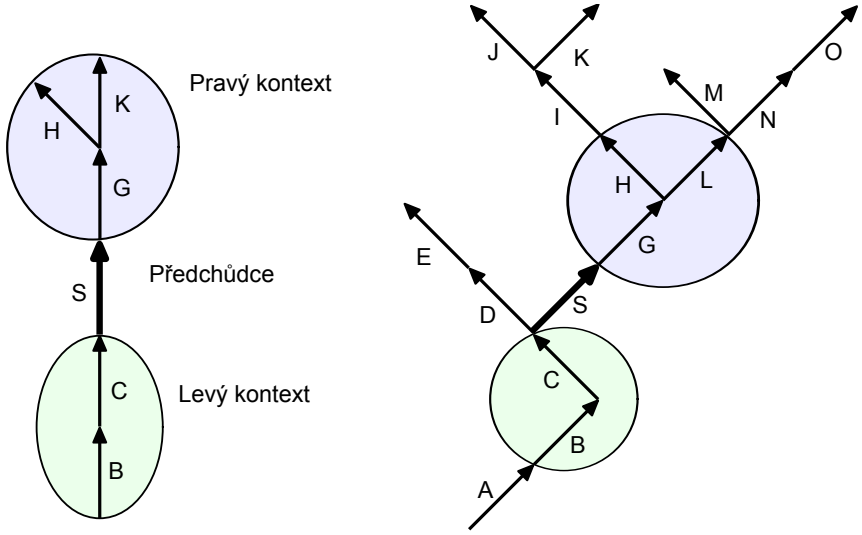
Zpracováním kontextového systému, ve kterém se mohou v kontextu vyskytovat všechny symboly abecedy, je možné dosáhnout jinak nepřístupných modelů, ale s přihlédnutím k cílům, realistického modelování rostlin, se zjistilo, že tyto systémy nejsou potřeba. Základním příkladem kontextových systémů je propagace signálu rostlinou - z pohledu symbolů pak pohyb vykonává F s určitým indexem. Zde nastává problém, že mezi jednotlivými F jsou symboly pro rotaci želvy jako je $+$ a -. V naprosté většině případů nechceme odlišit situace, kdy právě želva zatočila nebo šla kupředu. Tento problém má jedno velmi jednoduché řešení a tím je, ignorování určitých symbolů z pohledu hledání kontextu u pravidel.

Příklad 3.4.1
Ukázka ignorování nežádoucích symbolů. F_b zde označuje segment už dosažený signálem a F_a naopak ještě nedosažený. Direktiva *#ignore* převzatá z [34] zajišťuje přeskakování symbolů $+$ a - při hledání kontextu.

$$\#ignore \quad : \quad +-$$

$$\begin{aligned} \omega \quad &: \quad F_b + F_a - F_a - -F_a + F_a + F_a - F_a \\ p_1 \quad &: \quad F_b \quad < \quad F_a \quad \rightarrow \quad F_b \end{aligned}$$

[2] Zde záleží na tom jak je pole barev definováno. Je možno se posunout doprava nebo dolů, důležité je, že index má o 1 vyšší hodnotu než předtím.



Obrázek 3.9: Kontext o délce 2 symbolů ve stromové struktuře. Převzato z [34].

V rámci stromových modelů dostává kontext úplně nový rozměr, kdy jeden segment může mít více kontextů a ty nemusí být sekvenčně za sebou v interpretovaném řetězci. Lindenmayer a Prusinkiewicz[34] za pravý kontext (následující za segmentem) považují všechny větve vyrůstající z segmentu a za levý kontext pouze část modelu, ze které segment vyrůstá. Tyto principy umožňují modelovat propagaci dvou druhů signálů v rostlinách - od kořene k listům, využívající levý kontext, a od listů ke kořeni, využívající pravý kontext. Oba jsou uvedeny na obrázku 3.9.

Z pohledu řetězců se závorkami je situace mnohem více komplikovanější. Strom z obrázku 3.9 můžeme reprezentovat jako

$$ABC[DE][SG[HI[J]K]L[M]NO].$$

Při zpracování samotné symboly *[* a *]* řadíme mezi ignorované, které ovšem musíme počítat, abychom se dostali ke správným větvím. Při hledání prvních dvou symbolů levého kontextu pro symbol S potřebujeme přeskočit celou větev *[DE]*, která je ukončena před symbolem a dostaneme se k B a C. Pravý kontext, velikost dva symboly, je ještě o něco složitější, protože může mít více podob. Nejprve se dostaneme k symbolu G, který bude součástí každého kontextu a poté k větvi se symbolem H. To ovšem není vše, je nutné zbytek větve přeskočit a hledat další větev - ta začíná symbolem L. Nalezli jsme dva kontexty - GH a GL. Hledání pravého kontextu může být uzavřeno koncem aktuální větve původního symbolu před dosažením požadovaného počtu symbolů.

U zpracování kontextu velmi záleží na množině znaků, která se má ignorovat a většinou je potřeba, aby tvůrce systému specifikoval, které znaky se mají v kontextu zpracovávat. Například u abecedy, kterou definuji v této práci, přeskakování rotačních symbolů, je požadováno skoro vždy, ale symbol { pro vytvoření polygonu, někdy chceme detekovat a někdy ne. Proto neurčuji skupinu ignorovaných znaků a je ji potřeba pokaždé znovu definovat i za cenu nekonzistence s původní definicí, tak jako v [34].

Kapitola 4

Vykreslování 3D objektů

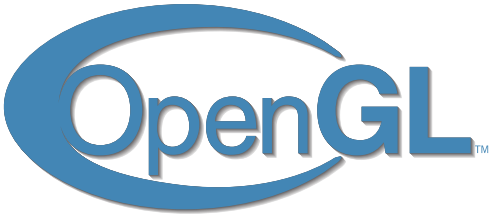
Rozvoj trojrozměrné počítačové grafiky započal v šedesátých letech na více místech současně, převážně ale v USA. Důležitými výzkumnými místy podle [23] byly MIT, Bellovy laboratoře a hlavně University of Utah. Zde objevili základní principy a algoritmy používané v 3D grafice, mezi které patří renderování, mapování textur, stínování objektů a mnoho dalších. Na těchto základech stojí současná 3D grafika a jsou popsány v následujícím odstavci.

Dvě hlavní části 3D grafiky jsou vytváření modelu a jeho reprezentace a převod 3D modelu do 2D zobrazení, většinou na obrazovce. Reprezentovat model lze velkým množstvím metod, například pomocí *konstruktivní geometrie*, kdy model reprezentujeme stromem grafických primitiv, která kombinujeme. Další metodou je *dekompozice* modelu na objemové jednotky - voxely, které mají vlastnosti objektu v daném místě. Velmi oblíbenými jsou také metody popisující hranice modelu pomocí 2D tvarů, například trojúhelníků nebo různých možností popisu 3D ploch. Druhá část, zabývající se vykreslením těchto objektů na průmětnu, nazývaná renderování, je také velmi komplexní, protože se usiluje o co nejrealističtější obrazy. Mezi základní problémy zde patří zabránění aliasu - *antialiasing*, který vzniká při převodu spojité informace na diskrétní, metody *stínování objektů* - vybarvení části objektu odstínem barvy pro iluzi trojrozměrnosti, *texturování* - pokrytí povrchu modelu obrázkem nebo vzorkem. Dále mnoho různých metod *osvětlení* scény a *vrhání stínu* objektů. Tyto pojmy jsou společně s mnoha dalšími popsány v [21]. Je zde také matematické pozadí, jak tyto principy fungují, které jsem použil při programování generátoru L-systémů.

Při původních výzkumech probíhaly grafické výpočty na procesoru společně s ostatními, ale záhy se začaly využívat grafické akcelerátory a GPU[1], které se specializují na vytváření obrazu na monitoru a grafické operace. Použití specializovaného hardwaru pro vykreslování umožnilo masivní navýšení výkonu a uvolnění procesoru. V současnosti začínají být samostatné grafické karty na základní desce vytlačovány řešeními se společným GPU a procesorem v jednom pouzdru.

Komunikace s grafickou kartou je z programátorského hlediska zajištěna aplikační knihovnou (v angličtině se označuje termínem API), která zpracovává jednotlivá volání a smazává rozdíly mezi jednotlivými modely akcelerátorů. Ke dvěma nejpoužívanějším a nejznámějším knihovnám patří OpenGL[39] a Direct3D[7], které dále v této kapitole popíšu (části 4.1 a 4.2). Dále existují i jejich nástavby, které umožňují použití těchto knihoven v dalších programovacích jazycích. Protože jsem se rozhodl naprogramovat aplikaci pro L-systémy v Javě, zmíním v 4.3 několik grafických knihoven pro tento programovací jazyk.

[1] Graphical processing unit - grafický procesor.



Obrázek 4.1: Logo OpenGL.

4.1 OpenGL

OpenGL[4] bylo na počátku vyvinuto firmou SGI a v současné době jej spravuje ARB[2] - konsorcium, jehož členy jsou významní hráči na poli grafiky (nVidia, SGI, ...). Jde o nepoužívanější specifikaci API pro grafiku a jeho implementace existují pro skoro všechny platformy pracující s grafikou. Jak bylo uvedeno výše, OpenGL ulehčuje programátorovi činnost, protože nemusí rozlišovat jednotlivé modely grafických karet a je mu přístupný jejich výkon. V krajním případě, kdy není v systému nějaký grafický akcelerátor obsažen, knihovna použije implementaci v softwaru a nic kritického se nestane.

Knihovna byla navržena jako nezávislá, proto neobsahuje žádné funkce pro vytváření a práci s okny ani zpracování událostí. Pro tyto činnosti je nutné použít další nástroje - součástí OpenGL je proto knihovna GLUT, která tuto funkcionalitu umožňuje. V duchu multiplatformnosti OpenGL také zavádí vlastní základní datové typy, jako *GLint* a *GLfloat*. Standardní implementace knihovny je v jazyce C a základem je hlavičkový soubor pro C/C++. Existují ovšem implementace pro velké množství dalších jazyků - Python, Javu, Fortran, Lisp, atd. Tyto implementace vetšinou obalují zdrojový soubor v C a zpřístupňují volání jeho funkcí. Někdy se tato implementace označuje cizím slovem jako *wrapper* z anglického pojmu pro obalovat nebo zabalovat.

OpenGL je z pohledu programátora stavovým automatem, se svým stavem, a rozhraním knihovny je asi 200-250 funkcí, které tímto stavem manipulují a vytváří interaktivní trojrozměrné aplikace. Při zadávání příkazu tedy lze měnit vlastnosti vykreslovaných objektů, jako je barva, průhlednost, tloušťka čáry, nebo celé scény - například osvětlení a transformace modelu a pohledu. Pomocí OpenGL lze vytvářet modely složené ze základních geometrických útvarů neboli primitiv. Mezi ně patří bod, úsečka, trojúhelník, čtyřúhelník, polygon, obrazy a další, všechny jsou uvedeny např. v [39] odkud jsem čerpal implementační detaily. V této knize je také popsaná struktura názvů funkcí a proměnných v OpenGL, která se skládá z několika částí. Bývá zde předpona *gl* pro jádro OpenGL, samotný název funkce a počet a druh parametrů. Příkladem může být *glColor3f()*, funkce která mění barvu vykreslovaných primitiv a přijímá 3 parametry typu *GLfloat*.

Standardní součástí každé implementace je knihovna utilit OpenGL nazývaná *GLU*. Tato knihovna umožňuje popisovat modely metodami na vyšší úrovni než samotné jádro. Příkladem možností mohou být NURBS[3] křivky a plochy nebo kvadratické povrchy. Funkce z této knihovny používají předponu *glu*, například jako funkce *gluSphere()*, která vykreslí kouli. Můžeme ji použít jako dočasný model při vytváření prostředí pro samotnou práci.

[2] ARB - Architecture Review Board.

[3] NURBS - Nonuniform Rational B-spline - neuniformní racionální b-splajn - velmi často používaný popis křivek v počítačové grafice.

4.2 Direct3D

Direct3D[7] je rozhraní, které se specializuje na 3D grafiku v *DirectX*[8]. DirectX je sada knihoven vyvíjená společností Microsoft pro zjednodušení přístupu k grafickému hardwaru v široké paletě úloh. Knihovny fungují pouze v systému Windows - nepodporují žádné jiné operační systémy a platformy. Lze ji však zprovoznit na unixových operačních systémech přes *Wine*[13]. První verze API byla zveřejněna v roce 1995 a v současnosti je aktuální verze 11.1 z srpna 2012.

Součástí DirectX je mnoho specializovaných knihoven, např. pro práci s 2D a 3D grafikou, fonty, práci se zvuky, obecné výpočty na grafické kartě a další. Direct3D je neznámější částí a její jméno se často zaměňuje s DirectX pro označení celé sady. Je nejznámější, protože se používá hlavně pro počítačové hry a například také na uživatelské rozhraní Windows Aero.

Po implementační stránce je Direct3D podobné OpenGL. Modely jsou složeny z stejných druhů grafických primitiv a probíhají stejné fáze při vykreslování. Protože je Direct3D vyvíjen konkrétně pro jeden operační systém, nabízí se fakt, že je rychlejší než OpenGL, které musí podporovat mnoho platforem, což ale nemusí být vždy pravda. V současnosti jsou hlavním rozdílem systémová volání Direct3D, která přepínají do privilegovaného módu jádra. Tato akce spotřebovává větší množství procesorového času a ztrácí se výkon. Stejný problém je i v OpenGL, ale není tak markantní. Dalším z rozdílů je, že Direct3d pracuje s objektově orientovaným rozhraním. Tuto knihovnu je možno používat v jazyce C/C++ a pro další jazyky jsou k dispozici obalovací knihovny - wrappery.

4.3 3D grafika a Java

Protože je OpenGL při práci s grafikou takřka standardním nástrojem pro většinu platforem, téměř všechny řešení práce s grafikou pro Javu používají tuto knihovnu, více nebo méně obalenou další funkcionalitou.

Jednou z nejznámějších implementací je *JOGL*, ta patří mezi nízkoúrovňové knihovny téměř kopírující OpenGL. Je popsaná v části 4.3.1 společně s další knihovnou *LWJGL*. Zástupcem vysokoúrovňových knihoven a prostředí je *jMonkeyEngine* v části 4.3.2 nebo *Java 3D*[17].

Na přelomu tisíciletí mezi nejpoužívanější vazby OpenGL patřila i knihovna *GL4Java*[10], která byla později vytlačena knihovnou JOGL pracující na stejných principech.

4.3.1 JOGL a LWJGL

JOGL[3] je velmi oblíbené navázání (neboli obalovací knihovna - wrapper) OpenGL na Javu, které se používá chceme-li pracovat ve skoro čistém OpenGL, a zároveň mít přístup k možnostem Javy. V rámci Javy musí být model knihovny objektový, ale rozhraní je velice intuitivní, a proto není problém pro programátory dříve pracující s OpenGL velmi rychle začít pracovat s JOGL. Přístup knihovny je tak podobný OpenGL, že je možné využít knihy a dokumentaci pro jazyk C, jako například [39], kterou jsem použil při implementaci. Základní OpenGL API pro C je zpřístupněno pomocí JNI[4] volání. JOGL umožňuje přístup jak k jádru OpenGL - funkcím s předponou *gl*, tak ke GLU knihovně - předpona *glu*.

[4] JNI - Java Native Interface - framework umožňující aplikacím Javy přístup ke knihovnám napsaným v jiných programovacích jazycích jako C a C++.

Knihovna byla vyvíjena skupinou ve firmě Sun Microsystems a pro prodeji firmy v roce 2009 je od roku 2010 vedena jako nezávislý projekt s BSD licencí. Podle [1] je JOGL referenční implementací vazby OpenGL na Javu.

JOGL má mnoho výhod, protože je velká část kódu generovaná automaticky z C, pomocí nástroje *GlueGen*[9], přidávání změn v OpenGL do JOGL je velmi rychlé a kód má stejnou strukturu. Také je důležité, že je to referenční implementace, to zajišťuje, že autoři se snaží o určitý standard. JOGL má rovněž vazby na *Swing* a *AWT*, API Javy pro práci s okny, to velmi zjednodušuje práci při vytváření aplikací.

Tato knihovna se používá například pro zobrazování grafů v programu *Scilab* nebo v počítačových hrách jako je *RuneScape*. Muže být také použita v *jMonkeyEnginu* a je přítomna ve vysokoúrovňové knihovně *Java3D*.

LWJGL - Lightweight Java Game Library[2] je další knihovna obalující OpenGL pro Javu, která jak už název napovídá se hlavně používá pro vývoj her a je velmi podobná JOGL. Klade důraz na jednoduché rozhraní, výkon a bezpečnost, protože se předpokládá použití na velkém množství platforem. Knihovna umožňuje pracovat nejen s grafikou, ale i se zvukem, paralelními výpočty a nejrůznějšími herními ovladači - gamepady, volanty, atd. LWJGL používá mnoho enginů, jako *jMonkeyEngine*, kde je základním rendererem, a je součástí velmi známé hry *Minecraft*. Zatím poslední verze této knihovny je 2.8.5 z listopadu 2012.

4.3.2 jMonkey Engine

jMonkeyEngine[5] je herní engine[5], který umožňuje práci s 3D grafikou a vývoj aplikací. Byl vytvořen aby odstranil absenci vývojových prostředí pro práci s grafikou v Javě. Pro vykreslování na nižších úrovních používá LWJGL, ale je možno využít i JOGL. V současnosti se blíží vydání verze 3.0 jako stabilní, předpokládá se o ní, že bude plnohodnotným prostředím pro vývoj grafických aplikací.

Samotný engine je skupina knihoven a samostatně jde o nízkoúrovňový nástroj. Po jejich kombinaci s vývojovým prostředím, oficiálním a doporučovaným je *jMonkeyEngine 3 SDK*, získáme vysokoúrovňový nástroj s rozsáhlou paletou možností pro práci s grafikou, nejen na úrovni zdrojového kódu, ale i s uživatelským rozhraním. Kromě toho zde lze pracovat, podobně jako u jiných nástrojů, s základní fyzikou a detekcí kolizí, multimédii, umělou inteligencí, atd. Projekt je šířen s BSD licencí a je možné jej volně využívat.

[5] Engine - systém pro tvorbu a vývoj aplikací, většinou her, pracující s grafikou.

Kapitola 5

Vytvořená aplikace LModeler

Cílem tohoto projektu bylo vytvořit program pro generování 3D modelů a v této kapitole popisuji mou aplikaci, pojmenovanou LModeler, založenou na teorii popsané v předchozích kapitolách 2 a 3. Nejdříve se v části věnuji 5.1 grafickému rozhraní, vizualizaci modelu a s čím vším se uživatel může v programu setkat. Dále popisuji, z pohledu tvůrce a programátora, použité prostředky pro vývoj a zdrojový kód - část 5.2.

5.1 Uživatelská část

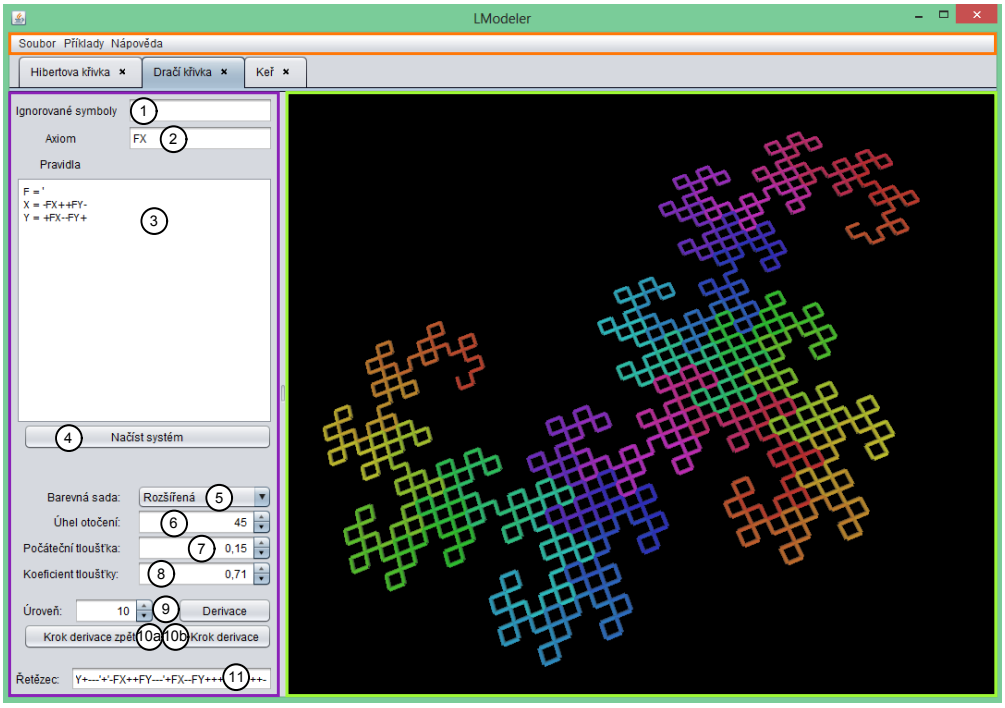
Podle zadání bylo cílem vytvořit interaktivní systém pro generování 3D objektů pomocí L-systémů a želví grafiky, což je velmi stručný popis, proto bylo potřeba nejprve specifikovat co budu vytvářet. Rozhodl jsem se, kromě základních OL-systémů podle definice 2.2.1, také použít stochastické L-systémy ze stejné části podle definice 2.2.4 a kontextové L-systémy z části 2.2.2. Při zpracování stochastických systémů, se vybírá pro každý symbol jiné pravidlo, podle aktuálního rozložení pravděpodobnosti.

Z pohledu uživatele je vždy žádoucí, aby ovládání aplikace bylo maximálně intuitivní. Proto jsem použil jediný široce používaný styl aplikace, který obsahuje menu v horní části okna, sloupec nebo lištu ovládacích prvků a pracovní nebo vizualizační část okna. Častým požadavkem, který je velmi usnadňující, je možnost pracovat paralelně s více modely zároveň a proto bylo součástí rozšířeného zadání použití záložek, které korespondují s intuitivním stylem aplikace. Hlavní okno aplikace je na obrázku 5.1. Jednotlivé části jsou barevně odlišeny, menu - oranžový obdélník, ovládací prvky - fialový a model v světle zeleném obdélníku. Záložky jsou v řádku pod menu a každá z nich má vlastní ovládací panel s aktuálními hodnotami a vizualizaci modelu.

Součástí menu jsou základní akce, které uživatelé v menu očekávají, jako je možnost ukládat svou si tvorbu do souboru a načítat ji a hlavně je přítomen manuál, s přesným popisem akcí, které vyvolávají jednotlivé prvky rozhraní. Také jsem se rozhodl přidat možnost vyvolat ukázkové L-systémy, využívající plně možnosti tohoto programu, které usnadní seznámení s aplikací.

5.1.1 Ovládání generátoru

Na obrázku 5.1 jsou ve fialové části komponenty ovládající model označeny čísly, pomocí kterých je budu popisovat. Prvek s číslem *2* slouží pro vstup axiomu L-systému a pod ním *3* funguje jako vstup pravidel. Oba dva prvky jsou synchronní a je možno je libovolně upravovat. Svůj vstup předají pro generování až při stisku tlačítka označeného číslem *4*.



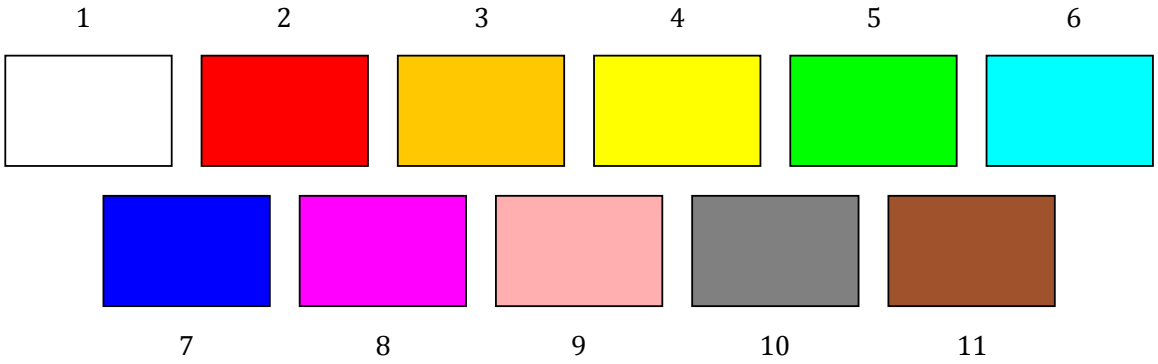
Obrázek 5.1: Hlavní okno aplikace.

Komponenty *2*, *3* a *4* stačí pro vytvoření základních bezkontextových gramatik. Nepotřebuji zde celkovou abecedu symbolů, protože za abecedu systému považuji množinu všech možných znaků. Na symboly, které jsem v systému neočekával, aplikuji pravidla jako na každé jiné, ale při interpretaci řetězce je ignoruji. Lépe řečeno jejich význam je takový, že se nestane nic.

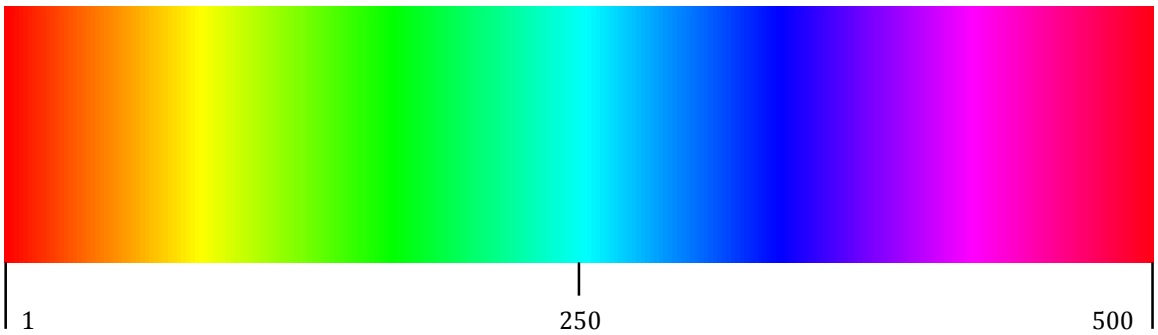
Prvek *1* je určen k použití u kontextových systémů pro specifikaci symbolů, které se mají přeskakovat při hledání kontextu v řetězci. Funguje na stejném principu jako direktiva *#ignore*, z [34], kterou jsem popsal v 3.4. Tak jako prvky *1* a *2* je prvek *1* synchronní s aktivací při stisku tlačítka *4*.

V prvcích *6*,*7* a *8* lze nastavit úhel otočení, který se využívá při interpretaci symbolů rotace želvy, počáteční poloměr kreslených segmentů a koeficient zeštíhlování segmentů u symbolu *!*. Úhel se zadává ve stupních od 0° po 359° a má stejný význam jako v kapitole 3. Poloměr segmentu lze zadat jako číslo od 0 do 10, kde 1,0 je délka segmentu. Při poloměru 1 tedy bude segment dvakrát širší než delší. Koeficientem tloušťky se násobí aktuální tloušťka, takže při velikosti od 0 do 1 bude model postupně tenčí a pokud bude koeficient větší než 1, tak bude model tloustnout.

Lze modelu také nastavit dvě barevná schémata/sady pomocí roletové nabídky prvku číslo *5*. *Základní* sada obsahuje 11 základních barev, které by měly stačit pro modely, které potřebují pro určité části konkrétní barvy, jako jsou například rostliny - hnědý kmen, zelené listy a bílé květy. *Základní* barvy jsou na obrázku 5.2. Druhé schéma je *rozšířená* sada. Ta obsahuje 500 barev v barevném spektru, které se postupně mění, tak jako v realitě od červené, přes zelenou a modrou, k fialové a cyklicky zpět červené. Nejpodobnější barvy jsou tedy u sebe nejblíže. Toto obarvení modelu se využívá při modelování křivek pro zvýraznění kudy křivka vede. Model s tímto principem jsem uvedl na obrázku 3.8. *Rozšířená* barevná sada, barevné spektrum, je na obrázku 5.3.



Obrázek 5.2: *Základní* barevná sada aplikace.



Obrázek 5.3: *Rozšířená* barevná sada aplikace - barevné spektrum.

L-systémy pracují s úrovněmi/derivacemi celého řetězce, kde každý stupeň je součástí simulace růstu modelu. Prvek *9* umožňuje nejprve nastavit, jaký stupeň derivace chce uživatel vykreslit a po stisknutí tlačítka, se tento stupeň vypočítá a vymodeluje. Úroveň se počítá od 0 do kladných čísel, kde 0 je zastoupena axiomem a dále značí kolik derivačních kroků bylo na axiom aplikováno. Lze zobrazit libovolný kladný stupeň, je ale potřeba upozornit, že z fraktálové podstaty L-systémů, jsou výpočetní požadavky velmi časově náročné, při velkém počtu derivací. Na druhou stranu, ze zkušenosti jsem došel k faktu, že krokování mezi derivacemi je častým jevem, a proto jsem do ovládacího panelu doplnil tlačítka pro jeden krok derivace. Prvek *10a* po stisknutí umožní jeden krok zpět směrem k axiomu, a prvek *10b* provede jeden derivační krok nad aktuálním stavem modelu.

Zobrazení aktuálního řetězce, který je interpretován a vykreslen je v poli *11*.

Symboly a pravidla v LModeleru

Z pohledu interpretace mají symboly v LModeleru stejný význam jako v kapitole 3, kde jsem je popsal v několika tabulkách. Jedním z rozdílů je, že aplikace neumí interpretovat symbol $\sim$, vložení předdefinovaného tvaru, protože na to není vytvořena. V L-systémech, které se neohlíží na implementaci na počítači se velmi často vyskytují indexy symbolu F. Takové lze nalézt v příkladech 3.0.7 a 3.4.1. Indexy se ale velmi špatně pracuje a to vede k druhému a poslednímu rozdílu v interpretaci symbolů. Rozšířil jsem význam symbolu F i na další symboly od A po P a stejně tak symbol f lze nahradit kterýmkoli z a až p. Zbytek znaků abecedy má společně s číslicemi význam neznámého symbolu, který je možno použít

v pravidlech, ale neinterpretuje se.

Každé pravidlo uvedené v prvku *3* musí být na samostatném řádku. Mezi pravidly se může vyskytovat libovolný počet prázdných řádků nebo řádku s bílými znaky, ale nesmí obsahovat jiné znaky, protože pak nelze zjistit, jestli se jedná o pravidlo nebo ne. Všechna pravidla mají společný základní tvar

$$symbol = řetězec,$$

kde = je povinný znak, který se v pravidle musí vyskytovat a zastupuje $\rightarrow$ z dosud uvedených pravidel, a jako *řetězec* se počítá i řetězec o délce 0 znaků.

Příklad 5.1.1
Základní pravidlo z příkladu 3.0.6

$$p \ : \ F \ \rightarrow \ F - F + +F - F,$$

lze přepsat do tvaru v LModeleru

$$F \ = \ F - F + +F - F.$$

Pokud uvedeme několik pravidel se stejným symbolem na levé straně, při derivaci je potřeba rozhodnout, které pravidlo se použije. V tento moment mají všechna pravidla, mezi kterými se rozhodujeme stejnou pravděpodobnost, ale pokud chceme určité pravidlo upřednostnit a definovat si vlastní pravděpodobnosti, je potřeba použít stochastické systémy. Pravidlo zde bude mít tvar

$$symbol \ = pravděpodobnost = \ řetězec,$$

kde oba dva znaky = se v pravidle musí vyskytovat a *pravděpodobnost* je desetinné číslo od 0 do 1 a součet pravděpodobností u pravidel se stejnou levou stranou musí být 1. Lze kombinovat pravidla s pravděpodobností a bez ní. Pokud není pravděpodobnost u pravidla dána, vyhledají se všechny pravidla se společnou pravděpodobností a každé dostane stejný díl.

Příklad 5.1.2
Stochastická pravidla z příkladu 2.2.2

$$\begin{aligned} p_1 &: F \overset{.33}{\rightarrow} F[+F]F[-F]F \\ p_2 &: F \overset{.33}{\rightarrow} F[+F]F \\ p_3 &: F \overset{.34}{\rightarrow} F[-F]F \end{aligned}$$

lze přepsat do tohoto tvaru v LModeleru:

$$\begin{aligned} F &= 0,33 = F[+F]F[-F]F \\ F &= 0,33 = F[+F]F \\ F &= F[-F]F \end{aligned}$$

Poslednímu pravidlu bude automaticky dopočítána zbylá pravděpodobnost, která je $1,0 - (2 * 0,33) = 0,34$, takže není potřeba ji uvádět. Pokud bych chtěl pro všechny pravidla stejnou pravděpodobnost, mohu vynechat její zadávání úplně a formát všech pravidel bude s jedním =.

Posledním způsobem, jak upravit pravidla je zadání kontextu, který se zadává ve stejném tvaru, jako v části 2.2.2. Pravidla mají tvar

$$prekontext < symbol > postkontext = řetězec,$$

kde < a > jsou povinné znaky, které musí pravidlo obsahovat pokud má příslušný kontext. *Prekontext* je řetězec definující levý kontext a analogicky *postkontext* definuje pravý kontext. Levý a pravý kontext lze použít každý zvlášť i je kombinovat.

Příklad 5.1.3
Kontextové pravidlo z příkladu 2.2.3

$$p_1 : \quad b < a \rightarrow b$$

lze přepsat do tohoto tvaru v LModeleru:

$$b < a = b$$

Všechny uvedené formy pravidel jsou navzájem kompatibilní a lze je dokonce kombinovat. Například lze zavést pravidlo s pravděpodobností, které kontroluje kontext:

$$prekontext < symbol > postkontext \ = pravděpodobnost = \ řetězec$$

V takovém případě se počítá pravděpodobnost pro pravidla, která mají stejnou nejen levou stranu, ale i kontext.

5.1.2 Manipulace s modelem

Všechny systémy jsou vizualizovány v zelené části aplikace (viz. obr. 5.1) jako 3D modely. I ty, které pracují pouze s jednou rovinou a jedním druhem rotací želvy. Není jednoduché určit vhodný pohled na obecný model, který by zachytil jeho smysl, a někdy to ani není možné, a proto jsem zavedl základní manipulaci pomocí myši, která nemění L-systém, ale pouze pohled na model.

Fungují zde tři druhy manipulace - *rotace*, *translace* a *změna velikosti*. Rotace funguje při stisknutí levého tlačítka myši a jejím tažení do některého směru. Model rotuje v tomtéž směru s přijatelnou rychlostí. Translace funguje na úplně stejném principu, pouze místo levého je potřeba stisknout pravé tlačítko. Poslední funkcí je změna velikosti, která využívá kolečko myši. Při krokování kolečka směrem od uživatele se model zmenšuje a při směru k sobě se stejnou rychlostí zvětšuje.

5.1.3 Export a import XML

Při ukládání L-systému do souboru jsem použil značkovacího jazyka XML[15] kvůli jeho velké podpoře a vlastnostem. Také jsem přihlédl k tomu, že ukládaná data nebudou moc velká a ukládání v binární podobně by ušetřilo pouze zanedbatelné množství paměti, což se jeví jako naprosto zbytečné, oproti možnosti číst soubory pouhým okem a zpracovat je bez potřeby spouštět program a přenášet jej.

Všechny systémy mají v XML stejný tvar. Na úvodu je hlavička s verzí a kódováním, následuje kořenový element *lsystem* a v něj jsou vnořeny jednotlivé části systému. Je zde

pět elementů s hodnotou, které určují vlastnosti systému - *axiom* pro zadání axiomu, *angle* pro určení úhlu zatočení želvy, *thickness* pro tloušťku segmentů, *contextIgnored* pro zadání symbolů ignorovaných při hledání kontextu a *colorset* pro určení použité barevné sady. Colorset může mít dvě hodnoty - *basic* pro základní sadu a *extended* pro rozšířenou. Soubor musí obsahovat každý ze zmíněných elementů právě jednou.

Posledním elementem obsaženým v lsystem je *rules*, element, který obsahuje jednotlivá pravidla, každé uzavřené v elementu *rule*. Rule je element bez hodnoty, který obsahuje atributy s jednotlivými částmi pravidla - *precontext* - levý kontext, *lside* - symbol s levou částí pravidla, *postcontext* - pravý kontext, *rside* - pravá část pravidla a *probability* - pravděpodobnost jako desetinné číslo, splňující podmínky popsané výše.

Příklad 5.1.4
Peano-Gosperova křivka uložená jako XML kompatibilní s LModelerem.

```
<?xml version="1.0" encoding="UTF-8"?>
<lsystem>
        <axiom>FX</axiom>
        <angle>60.0</angle>
        <thickness>0.15</thickness>
        <contextIgnored></contextIgnored>
        <colorset>extended</colorset>
        <rules>
                <rule precontext="" lside="F" postcontext=""
                      probability="1.0" rside="'" />
                <rule precontext="" lside="Y" postcontext=""
                      probability="1.0" rside="+FX-FYFY--FY-FX++FX+FY" />
                <rule precontext="" lside="X" postcontext=""
                      probability="1.0" rside="FX-FY--FY+FX++FXFX+FY-" />
        </rules>
</lsystem>
```

5.2 Vývoj a implementace

Po vzniku návrhu, který odpovídal popisu aplikace z předchozí části kapitoly, byla hlavním problémem otázka výběru programovacího jazyka. Návrh jsem vytvářel s myšlenkou objektů, a proto jsem chtěl použít nějaký objektově orientovaný jazyk. Nakonec jsem zvolil *Java SE*[11], protože jsem s tímto jazykem už pracoval a vyhovovalo mi jeho odstínění od architektury, přenositelnost a zároveň velká robustnost. Také jsem chtěl prozkoumat jeho možnosti, co se týče 3D grafiky a práce s grafickou kartou. Jako základní manuál jsem použil tutoriály[16] od společnosti Oracle, která v současnosti Javu spravuje.

Swing

Dalším důvodem použití Javy je ten, že už v standardní implementaci obsahuje grafickou knihovnu pro vytváření GUI - *Swing*, což je dobře navržené a systematické API. Jeho kvalita je zdůvodněna tím, že v počátcích Javy pro práci s grafikou existovala pouze knihovna AWT, která, podle [27], "byla šita velmi horkou jehlou" a byla cílem mnoha stížností. Brzy

byla vylepšena a vznikl Swing, který odstranil velkou část problémů a získává zpět ztracenou reputaci. Swing je kompletně implementován v Javě, takže přebírá její platformovou nezávislost a silně využívá dědičnosti, která je těžištěm Javy. Dalším z důvodů, proč jsem použil Swing je jeho tutoriál[6], také vytvořený Oraclem, který je velmi kvalitní.

Netbeans a Eclipse

Implementaci aplikace lze provádět ve vývojových prostředích primárně určených pro Javu jako jsou *Netbeans*[19] a *Eclipse*[18], které jsem použil já. Existuje ještě řada dalších, ale nejsou tak rozšířené, hlavně při práci s Java SE. Obě vývojová prostředí mohou být rozšířena širokou škálou doplňků a podporovat i další jazyky jako je C/C++, PHP, atd. Hlavním důvodem mého použití Netbeans byl integrovaný grafický editor GUI pracující se Swingem, který je v Eclipse potřeba doplnit a Eclipse jsem využil pro možnosti exportovat celý projekt do spustitelného JAR[1] souboru, který obsahuje i veškeré potřebné knihovny. Další možnosti jako je například refaktorizace kódu a správa projektů obsahují obě dvě a naprostá většina ostatních vývojových prostředí.

5.2.1 Použité knihovny a prostředky

JOGL

Modely generované z řetězců l-systému jsem vykresloval pomocí OpenGL skrze knihovnu *JOGL*[3], kterou jsem více popsal v předchozí kapitole v části 4.3.1 společně s nástrojem *Gluegen*[9]. Pro práci s touto knihovnou je potřeba importovat do pracovního prostoru (např. ukázat vývojovému prostředí kde se knihovna nachází nebo určit cestu ke knihovnám pro skripty) několik JAR souborů. Některé z nich jsou bohužel systémově závislé, protože pracují na nízké úrovni, a vyskytují se drobné problémy při zajištění multiplatformnosti aplikace.

Vecmath

Vecmath[12] je podprojekt Java 3D[17], vysokoúrovňového API pro práci s 3D grafikou, který se specializuje na vektorovou a maticovou matematiku, pro práci ve dvou nebo třírozměrném prostoru. Obsahuje třídy pro vytvoření 2x2, 3x3 a 4x4 matic, vektorů o 2-4 položkách a také bodů v prostoru o 2-4 dimenzích a zároveň metody pro práci s nimi jako je násobení matic a vektorů a další. V mé práci jsem tuto knihovnu používal při interpretaci řetězce u počítání absolutní pozice bodů segmentu modelu a také u vektorů jednotlivých směrů želvy a jejich transformací.

Dom4j

Dom4j[14] je open source knihovna pro práci s XML[15], XPath a XSLT. Tato knihovna je sice staršího data, ale podporuje zmíněné jazyky ve standardu definovaném *W3C*[2] a je plně kompatibilní s aktuální verzí Javy. Využil jsem ji pro jednoduché vytváření XML a také opačně jeho zpracování, které jsem potřeboval pro ukládání L-systémů do souboru, jak jsem popsal v části 5.1.3. Práce s knihovnou je velmi prostá, celá je implementována v Javě a distribuuje se v jednom JAR souboru.

[1] JAR - **J**ava **Ar**chive - archiv pro soubory s bajtkódem Javy. Některé z nich je možné spouštět.
[2] W3C - World Wide Web Consortium - mezinárodní společnost vyjíjející standardy pro web.

5.2.2 Struktura a velikost kódu

Zdrojový kód je rozdělen do dvou balíků - *lmodeler* a *lsystem*. Lsystem obsahuje třídy pro vytvoření a práci s l-systémem jako gramatikou a zároveň jeho vykreslení jako 3D modelu želví grafikou. Lmodeler se skládá ze tříd, které vytvářejí GUI aplikace a umožňují do sebe zakomponovat l-systém.

V balíku lsystem, třída *LRule* slouží pro vytvoření objektu jednoho pravidla v L-systému, obsahuje pole pro zpracování kontextu i pro pravděpodobnost. *LsysException* je speciální výjimka pro chyby, které nastaly při zpracování l-systému. *Lsystem* je základní třída, které dává dohromady ostatní a její objekty se vytváří při práci s l-systémem. MyPoint3d je třída rozšiřující *Point3d* z Vecmath a obsahuje bod v prostoru, společně s vektorem normály povrchu modelu v tomto bodě. Třída *Turtle* reprezentuje objekt želvy s jeho stavem a akcemi.

Těžištěm balíku lmodeler je třída *MainFrame* obsahující metodu main a vytvářející hlavní okno programu. K dalším samostatným oknům patří třídy *AboutFrame* a *HelpFrame* vytvářející okno informací o aplikaci a manuál ovládání. *LPanel* vytváří část okna pro manipulaci s modelem a jeho vizualizaci, komunikuje s objektem LSystemu a určuje, co se bude vykreslovat. Protože program obsahuje více panelů oddělených záložkami, vytvořil jsem speciální formát záložky, který je ve třídě *LPanelHeadline*. *EListener* slouží jako posluchač událostí renderovaného modelu, který řeší jeho vykreslování a společně s *ArcBall* i transformace pomocí myši.

balík	třída	počet řádků
lsystem		
⌞	LRule	104
	LsysException	23
	LSystem	1358
	MyPoint3d	37
	Turtle	401
lmodeler		
⌞	AboutFrame	63
	ArcBall	252
	EListener	220
	HelpFrame	170
	LPanel	480
	LPanelHeadline	138
	MainFrame	1069
	Celkem	4315

Tabulka 5.1: Rozdělení a velikost zdrojových souborů.

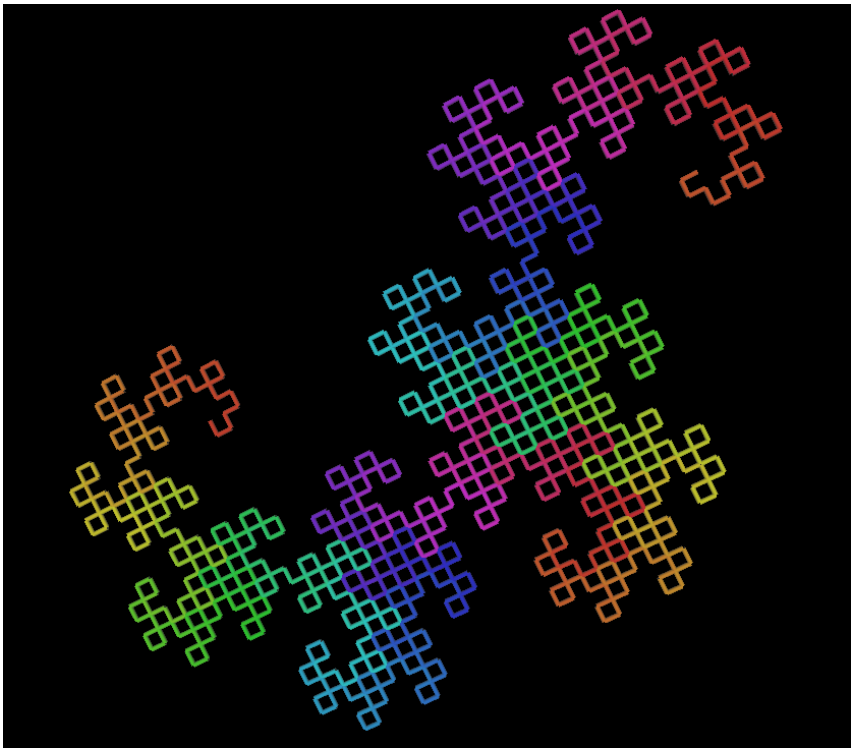
Kapitola 6

Ukázkové modely

6.1 Dračí křivka

Příklad 6.1.1
Dračí křivka - soběpodobná křivka s vlastnostmi fraktálu.

$$\begin{array}{lcll} \omega & : & \text{FX} & \\ p_1 & : & \text{F} & \rightarrow \quad \text{'} \\ p_2 & : & \text{X} & \rightarrow \quad \text{-FX++FY-} \\ p_3 & : & \text{Y} & \rightarrow \quad \text{+FX–FY+} \\ \delta & = & 45° & \end{array}$$

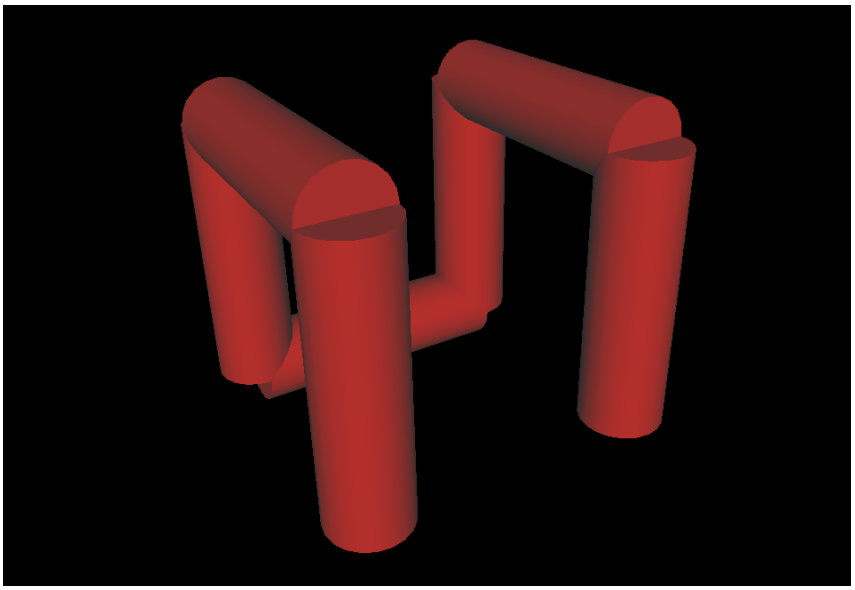


Obrázek 6.1: Dračí křivka v desáté derivaci.

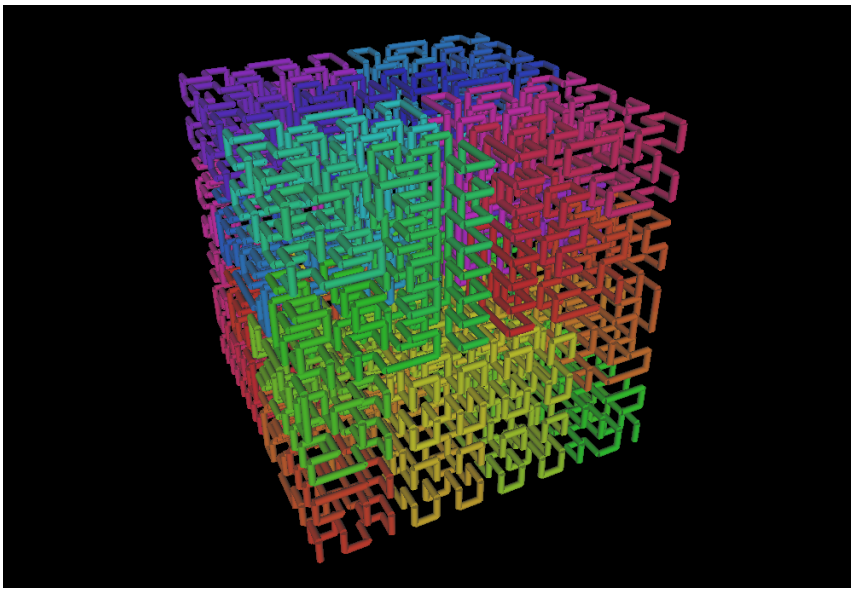
6.2 Trojrozměrná Hilbertova křivka

Příklad 6.2.1
3D varianta Hilbertovy křivky.

$$\begin{array}{lcll} \omega & : & \mathrm{X} & \\ p_1 & : & \mathrm{X} & \rightarrow \quad '' \wedge \backslash XF \wedge \backslash XFX - F \wedge //XFX\&F + //XFX - F/X - / \\ \delta & = & 90^\circ & \end{array}$$



Obrázek 6.2: První derivace Hilbertovy křivky.

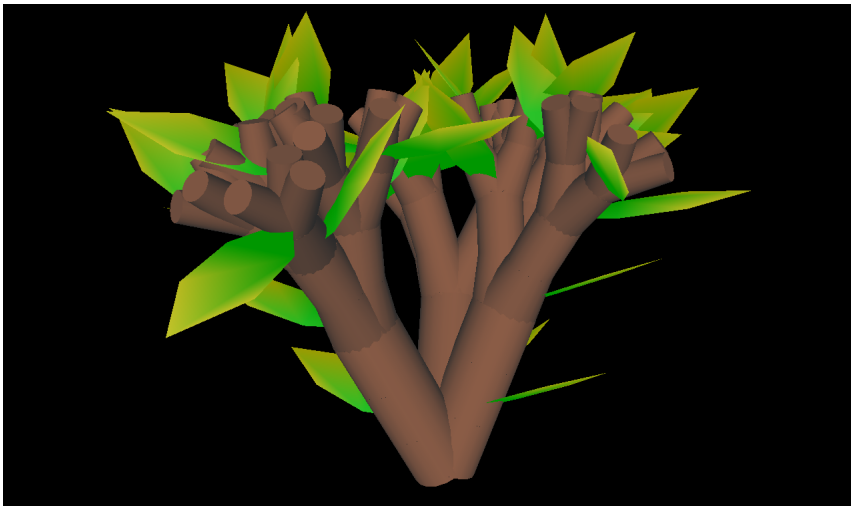


Obrázek 6.3: Hilbertova křivka ve čtvrté derivaci.

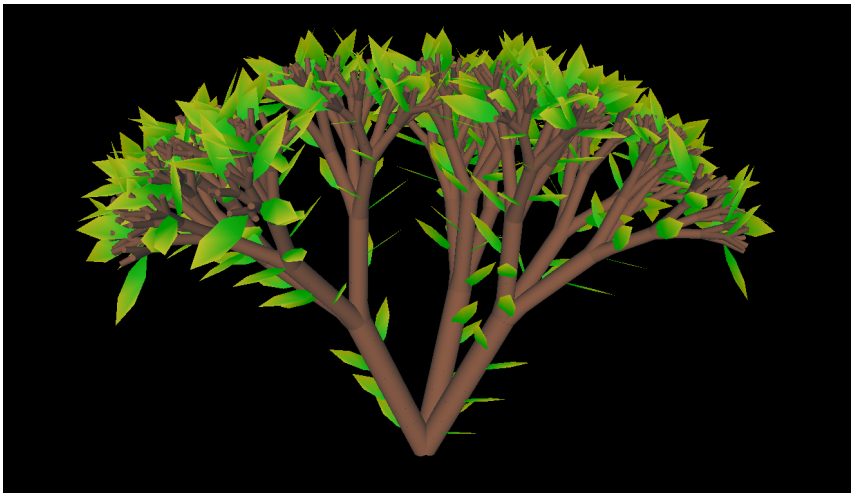
6.3 3D Rostlina 1

Příklad 6.3.1
Model rostliny převzatý z [34, strana 26].

$$
\begin{array}{lcll}
\omega & : & , [\&FY!Z]/////[\&FY!Z]///////[\&FY!Z] & \\
p_1 & : & \mathrm{Z} \rightarrow & [\&FY!Z]/////[\&FY!Z]///////[\&FY!Z] \\
p_2 & : & \mathrm{F} \rightarrow & X/////F \\
p_3 & : & \mathrm{X} \rightarrow & FY \\
p_4 & : & \mathrm{Y} \rightarrow & [''''\wedge\wedge -' f.+, f.+f.-|-f.+' f.+f.] \\
p_5 & : & \mathrm{F} \rightarrow & FF \\
\delta & = & 25^\circ &
\end{array}
$$



Obrázek 6.4: Pohled na třetí derivaci rostliny.

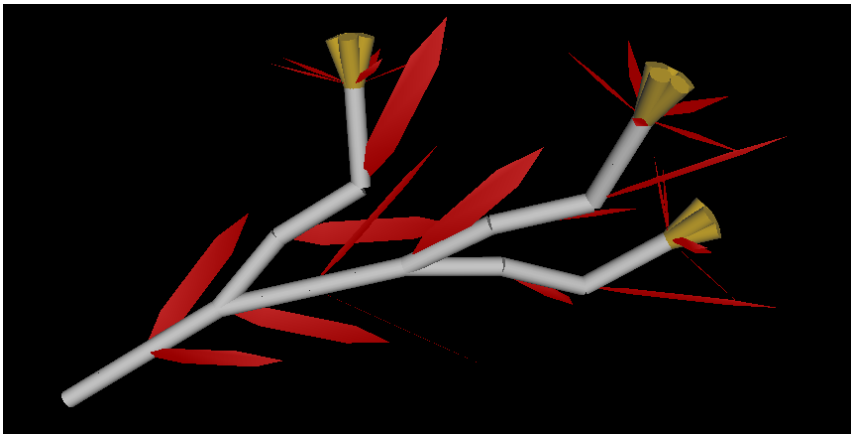


Obrázek 6.5: Pohled pátou derivaci rostliny.

6.4 3D Rostlina 2

Příklad 6.4.1
Model rostliny převzatý z [34, strana 27].

$$
\begin{array}{lcll}
\omega & : & , [\&FY!Z]/////[\&FY!Z]///////[\&FY!Z] & \\
p_1 & : & \mathrm{Q} \rightarrow & R + [Q + U] - -//[- - T]R[+ + T] - [QU] + +QU \\
p_2 & : & \mathrm{R} \rightarrow & FS[//\&\&T][// \wedge \wedge T]FS \\
p_3 & : & \mathrm{S} \rightarrow & SFS \\
p_4 & : & \mathrm{T} \rightarrow & [' + f. - ff. - f. + | + f. - ff. - f.] \\
p_5 & : & \mathrm{U} \rightarrow & [\&\&\&V'/W////W////W////W////W] \\
p_6 & : & \mathrm{V} \rightarrow & FF \\
p_7 & : & \mathrm{W} \rightarrow & [' \wedge F][\&\&\&\& - f. + f.| - f. + f.] \\
\delta & = & 18^\circ &
\end{array}
$$



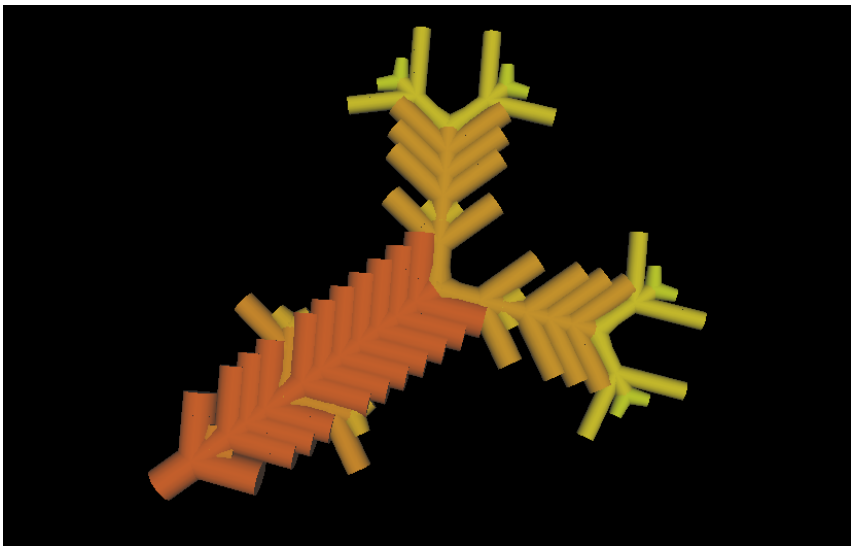
Obrázek 6.6: Pohled na třetí derivaci rostliny.

Obrázek 6.7: Pohled na pátou derivaci rostliny.

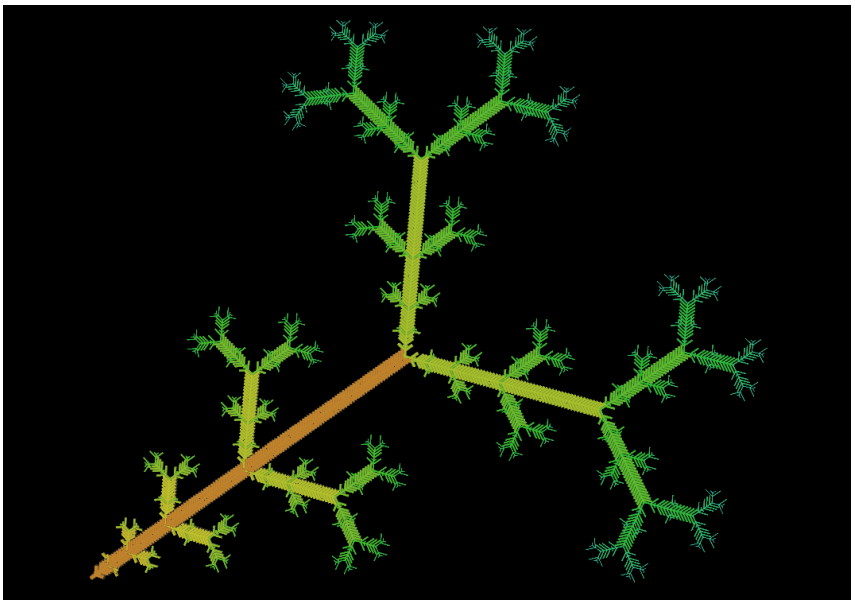
6.5 2D Rostlina

Příklad 6.5.1
Dvourozměrný model rostliny.

$$
\begin{array}{lcll}
\omega & : & Z & \\
p_1 & : & \mathrm{Z} & \rightarrow \quad ''''''ZFX[''''''!+Z][''''''!'-Z] \\
p_2 & : & \mathrm{X} & \rightarrow \quad X[-FFF][+FFF]FX \\
\delta & = & 60^\circ &
\end{array}
$$



Obrázek 6.8: Čtvrtá derivace rostliny.



Obrázek 6.9: Sedmá derivace rostliny.

Kapitola 7

Závěr

Cílem této bakalářské práce bylo vytvořit interaktivní systém pro generování 3D objektů. V práci jsem se nejprve věnoval studiu L-systémů, želví grafiky a možnostem vykreslování 3D objektů jimi generovaných. Poté jsem pomocí získaných znalostí vytvořil aplikaci v jazyce Java, která provádí zmíněné akce s pomocí knihovny JOGL.

Vytvořil jsem plně funkční aplikaci, která může běžet i na slabších strojích, ovšem kvůli časové složitosti zpracování L-systémů a jejich fraktálové podstatě, často je možné dostat se do situace, že pro výpočet vyšších derivací systému nebude stačit ani ten nejvýkonnější stroj. Tomuto jevu se nelze nijak vyhnout.

Při zpracování práce jsem se poučil o možnostech vytváření fraktálů a procedurálního modelování, také o vykreslování trojrozměrné grafiky a její navázání na grafické rozhraní aplikace. Také jsem nastudoval matematické pozadí pro 3D grafiku a různá grafická API a použil jsem je v praxi, pro vykreslování modelů, které jsem vygeneroval.

7.1 Další vývoj projektu

Mezi možnosti vývoje aplikace v části týkající se L-systémů patří rozšíření zpracovávaných systémů o parametrické systémy (viz. 2.2.3) a rozpracování principu náhodných mutací řetězce i samotných pravidel L-systému, jak je zavádí Lapré v *Lparseru*[29]. Z pohledu želví grafiky a vykreslování rostlin lze zavést efekt působení gravitace a dalších jevů na tvar rostliny - tento jev se nazývá *tropismus*[22, strany 806, 823 a 825] a implementovat podporu textur pro věrnější zobrazení kůry a listů. Pro usnadnění práce s aplikací lze doplnit volitelnost barvy pozadí modelu a export modelů do formátů podporovaných zavedenými aplikacemi pro práci s 3D.

Literatura

[1] JSR 231: Java Binding for the OpenGL API [online]. Dostupné z: http://jcp.org/en/jsr/detail?id=231, [cit. 2013-3-11].

[2] LWJGL - Lightweight Java Game Library [online]. Dostupné z: http://www.lwjgl.org/, [cit. 2013-3-11].

[3] JOGL - Java binding for the OpenGL API [online]. Dostupné z: https://jogamp.org/jogl/, [cit. 2013-3-4].

[4] OpenGL - The Industry's Foundation for High Performance Graphics [online]. Dostupné z: http://www.opengl.org/, [cit. 2013-3-4].

[5] jMonkeyEngine 3.0 [online]. Dostupné z: http://jmonkeyengine.com/, [cit. 2013-3-6].

[6] Creating a GUI With JFC/Swing [online]. Dostupné z: http://docs.oracle.com/javase/tutorial/uiswing/index.html, [cit. 2013-4-2].

[7] Direct3D [online]. Dostupné z: http://msdn.microsoft.com/en-us/library/windows/desktop/hh309466.aspx, [cit. 2013-4-2].

[8] DirectX Graphics and Gaming [online]. Dostupné z: http://msdn.microsoft.com/en-us/library/windows/desktop/ee663274.aspx, [cit. 2013-4-2].

[9] Gluegen [online]. Dostupné z: http://jogamp.org/gluegen/www/, [cit. 2013-4-2].

[10] Jausoft GL4Java Home-Page [online]. Dostupné z: http://jausoft.com/gl4java/, [cit. 2013-4-2].

[11] Java [online]. Dostupné z: http://www.java.com/, [cit. 2013-4-2].

[12] Vecmath [online]. Dostupné z: http://java.net/projects/vecmath, [cit. 2013-4-2].

[13] WineHQ [online]. Dostupné z: http://www.winehq.org/, [cit. 2013-4-2].

[14] dom4j - Introduction [online]. Dostupné z: http://dom4j.sourceforge.net/, [cit. 2013-4-3].

[15] Extensible Markup Language (XML) [online]. Dostupné z: http://www.w3.org/XML/, [cit. 2013-4-3].

[16] The Java Tutorials [online]. Dostupné z: http://docs.oracle.com/javase/tutorial/, [cit. 2013-4-3].

[17] Java 3D [online]. Dostupné z: http://java.net/projects/java3d, [cit. 2013-4-5].

[18] The Eclipse Foundation [online]. Dostupné z: http://www.eclipse.org/, [cit. 2013-4-8].

[19] Netbeans IDE [online]. Dostupné z: https://netbeans.org/, [cit. 2013-4-8].

[20] Anderson, D.; Anderson, D.: Introduction to Chain Codes [online]. Dostupné z: http://www.mind.ilstu.edu/curriculum/chain_codes_intro/chain_codes_intro.php, 2010 [cit. 2012-12-29].

[21] Žára, J.; Beneš, B.; Sochor, J.; aj.: *Moderní počítačová grafika*. Computer Press, 2004, ISBN 80-251-0454-0.

[22] Campbell, N.; Reece, J.: *Biologie*. Computer Press, 2008, ISBN 80-251-1178-4.

[23] Carlson, W.: A Critical History of Computer Graphics and Animation [online]. Dostupné z: http://design.osu.edu/carlson/history/lessons.html, 2003 [cit. 2013-3-6].

[24] Gardner, M.: The fantastic combinations of John Conway's new solitaire game 'Life'. *Scientific American*, Říjen 1970.

[25] Hanan, J.: *Parametric L-Systems and their application to the modelling and visualization of plants*. Dizertační práce, Computer Science University of Regina, 1992.

[26] Ilachinski, A.: *Cellular Automata: A Discrete Universe*. World Scientific, 2001, ISBN 978-9812381835.

[27] Jelínek, L.: Java (24) - úvod do grafiky a GUI [online]. Dostupné z: http://www.linuxsoft.cz/article.php?id_article=1184, 2006-4-24 [cit. 2013-4-5].

[28] Jennings, C.: Lindenmayer Systems [online]. Dostupné z: http://cgjennings.ca/toybox/lsystems/index.html, 2002 [cit. 2013-2-12].

[29] Lapré, L.: LParser and Mutation [online]. Dostupné z: http://laurenslapre.nl/lapre_004.htm, [cit. 2013-4-5].

[30] Manousakis, S.: *Musical L-Systems*. Diplomová práce, The Royal Conservatory, 2006.

[31] Meduna, A.: *Automata and Languages: Theory and Applications*. Springer-Verlag, 2000, ISBN 81-8128-333-3.

[32] Prusinkiewicz, P.: Graphical applications of L-systems. In *Proceedings of Graphics Interface '86 - Vision Interface '86*, Canadian Information Processing Society, 1986, s. 247–253.

[33] Prusinkiewicz, P.; Hammel, M.; Měch, R.; aj.: Visual Models of Plant Development. In *Handbook of Formal Languages Vol.3*, Springer-Verlag, 1997, ISBN 3-540-60649-1, s. 535–597.

[34] Prusinkiewicz, P.; Lindenmayer, A.: *The Algorithmic Beauty of Plants*. Springer-Verlag, 1990, ISBN 978-0-387-97297-8.

[35] Riddle, L.: Koch Snowflake [online]. Dostupné z: http://ecademy.agnesscott.edu/~lriddle/ifs/ksnow/ksnow.htm, 2010-1-25 [cit. 2012-12-25].

[36] Rozenberg, G.; Kari, L.; Salomaa, A.: L Systems. In *Handbook of Formal Languages Vol.1*, Springer-Verlag, 1997, ISBN 3-540-60420-0, s. 253–328.

[37] Rozenberg, G.; Salomaa, A.: *Handbook of Formal Languages*. Springer-Verlag, 1997, ISBN 3-540-61486-9.

[38] Sagan, H.: *Space-Filling Curves*. Springer-Verlag, 1994, ISBN 0-387-94265-3.

[39] Shreiner, D.; Woo, M.; Neider, J.; aj.: *OpenGL Průvodce programátora*. Computer Press, 2006, ISBN 80-251-1275-6.

[40] Villarreal, M. R.: KTurtle (side view) - RGB [online]. Dostupné z: http://upload.wikimedia.org/wikipedia/commons/e/ec/Kturtle_side_view_(RGB).svg, 2008-3-2 [cit. 2012-12-29].

[41] Villarreal, M. R.: KTurtle (top view) [online]. Dostupné z: http://upload.wikimedia.org/wikipedia/commons/7/70/Kturtle_top_view.svg, 2008-3-2 [cit. 2012-12-29].

Příloha A

Želví interpretace symbolů v LModeleru

Symbol	Intepretace	Strana
A-*P*	Udělej krok vpřed a vykresli čáru.	14
a-*p*	Udělej krok vpřed.	14
[	Ulož stav na zásobník - začni vytvářet větev.	18
]	Vyjmi stav ze zásobníku - dokonči větev.	18
+	Otoč se doleva.	20
−	Otoč se doprava.	20
∧	Zakloň se.	20
&	Skloň se.	20
/	Převal se doleva.	20
\	Převal se doprava.	20
|	Otoč se čelem vzad.	20
{	Začni vytvářet polygon.	21
.	Ulož pozici jako vrchol současného polygonu.	21
}	Dokonči polygon a vykresli jej.	21
’	Inkrementuj index barvy.	21
,	Dekrementuj index barvy.	21
!	Zmenši poloměr dalších segmentů.	21
%	Odřízni konec aktuální větve.	21
ostatní	Nedělej nic.	

Příloha B

Obsah CD

Adresář	Obsah
LModeler/	Projekt pro Netbeans bez knihoven.
LModeler/src/	Zdrojové soubory LModeleru v Javě.
lib/	Knihovny potřebné pro LModeler.
runnable/	Spustitelný JAR soubor s aplikací.
javadoc/	Vygenerovaná dokumentace.
text/	Zdrojové soubory této práce v LaTeXu.
text/fig/	Obrázky z této práce.